Université de Montréal

Visual Question Answering with Modules and Language Modeling

par

Vardaan Pahuja

Département d' informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures et postdoctorales

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en Informatique

avril 2019

© Vardaan Pahuja, 2019

Université de Montréal

Département d' informatique et de recherche opérationnelle, Faculté des arts et des sciences

Ce mémoire intitulé

Visual Question Answering with Modules and Language Modeling

Présenté par

Vardaan Pahuja

A été évalué par un jury composé des personnes suivantes

Alain Tapp
Président-rapporteur

Christopher J. Pal
Directeur de recherche

Yoshua Bengio
Membre du jury

Résumé

L'objectif principal de cette thèse est d'apprendre les représentations modulaires pour la tâche de réponse visuelle aux questions (VQA). Apprendre de telles représentations a le potentiel de généraliser au raisonnement d'ordre supérieur qui prévaut chez l'être humain. Le chapitre 1 traite de la littérature relative à VQA, aux réseaux modulaires et à l'optimisation de la structure neuronale. En particulier, les différents ensembles de données proposés pour étudier cette tâche y sont détaillés. Les modèles de VQA peuvent être classés en deux catégories en fonction des jeux de données auxquels ils conviennent. La première porte sur les questions ouvertes sur les images naturelles. Ces questions concernent principalement quelques objets/personnes présents dans l'image et n'exigent aucune capacité de raisonnement significative pour y répondre. La deuxième catégorie comprend des questions (principalement sur des images synthétiques) qui testent la capacité des modèles à effectuer un raisonnement compositionnel. Nous discutons de différentes variantes architecturales de réseaux de modules neuronaux (NMN). Finalement nous discutons des approches pour apprendre les structures ou modules de réseau neuronal pour des tâches autres que VQA.

Au chapitre 2, nous décrivons un moyen d'exécuter de manière parcimonieuse un modèle CNN (ResNeXt [110]) et d'enregistrer les calculs effectués dans le processus. Ici, nous avons utilisé un mélange de formulations d'experts pour n'exécuter que les K meilleurs experts dans chaque bloc convolutionnel. Le groupe d'experts le plus important est sélectionné sur la base d'un contrôleur qui utilise un système d'attention guidé par une question suivie de couches entièrement connectées dans le but d'attribuer des poids à l'ensemble d'experts. Nos expériences montrent qu'il est possible de réaliser des économies énormes sur le nombre de FLOP avec un impact minimal sur la performance.

Le chapitre 3 est un prologue du chapitre 4. Il mentionne les contributions clés et fournit une introduction au problème de recherche que nous essayons de traiter dans l'article. Le chapitre 4 contient le contenu de l'article. Ici, nous nous intéressons à l'apprentissage de la structure interne

des modules pour les réseaux de modules neuronaux (NMN) [3, 37]. Nous introduisons une nouvelle forme de structure de module qui utilise des opérations arithmétiques élémentaires et la tâche consiste maintenant à connaître les poids de ces opérations pour former la structure de module. Nous plaçons le problème dans une technique d'optimisation à deux niveaux, dans laquelle le modèle prend des gradients de descente alternés dans l'architecture et des espaces de poids. Le chapitre 5 traite d'autres expériences et études d'ablation réalisées dans le contexte de l'article précédent.

La plupart des travaux dans la littérature utilisent un réseau de neurones récurrent tel que LSTM [33] ou GRU [13] pour modéliser les caractéristiques de la question. Cependant, les LSTM peuvent échouer à encoder correctement les caractéristiques syntaxiques de la question qui pourraient être essentielles [87]. Récemment, [76] a montré l'utilité de la modélisation du langage pour répondre aux questions. Avec cette motivation, nous essayons d'apprendre un meilleur modèle linguistique qui peut être formé de manière non supervisée. Dans le chapitre 6, nous décrivons un réseau récursif de modélisation de langage dont la structure est alignée pour le langage naturel. Plus techniquement, nous utilisons un modèle d'analyse non supervisée (Parsing Reading Predict Network ou PPRN [86]) et augmentons son étape de prédiction avec un modèle TreeLSTM [99] qui utilise l'arborescence intermédiaire fournie par le modèle PRPN dans le but de un état caché en utilisant la structure arborescente. L'étape de prédiction du modèle PRPN utilise l'état caché, qui est une combinaison pondérée de l'état caché du TreeLSTM et de celui obtenu à partir d'une attention structurée. De cette façon, le modèle peut effectuer une analyse non supervisée et capturer les dépendances à long terme, car la structure existe maintenant explicitement dans le modèle. Nos expériences démontrent que ce modèle conduit à une amélioration de la tâche de modélisation du langage par rapport au référentiel PRPN sur le jeu de données Penn Treebank.

Mots clés : Réponse visuelle à une question, raisonnement visuel, réseaux modulaires, optimisation de la structure neuronale, modélisation du langage

Summary

The primary focus in this thesis is to learn modularized representations for the task of Visual Question Answering. Learning such representations holds the potential to generalize to higher order reasoning as is prevalent in human beings. Chapter 1 discusses the literature related to VQA, modular networks and neural structure optimization. In particular, it first details different datasets proposed to study this task. The models for VQA can be categorized into two categories based on the datasets they are suitable for. The first one is open-ended questions about natural images. These questions are mostly about a few objects/persons present in the image and don't require any significant reasoning capability to answer them. The second category comprises of questions (mostly on synthetic images) which tests the ability of models to perform compositional reasoning. We discuss the different architectural variants of Neural Module Networks (NMN). Finally, we discuss approaches to learn the neural network structures or modules for tasks other than VQA.

In Chapter 2, we discuss a way to sparsely execute a CNN model (ResNeXt [110]) and save computations in the process. Here, we used a mixture of experts formulation to execute only the top-K experts in each convolutional block. The most important set of experts are selected based on a gate controller which uses a question-guided attention map followed by fully-connected layers to assign weights to the set of experts. Our experiments show that it is possible to get huge savings in the FLOP count with only a minimal degradation in performance.

Chapter 3 is a prologue to Chapter 4. It mentions the key contributions and provides an introduction to the research problem which we try to address in the article. Chapter 4 contains the contents of the article. Here, we are interested in learning the internal structure of the modules for Neural Module Networks (NMN) [3, 37]. We introduce a novel form of module structure which uses elementary arithmetic operations and now the task is to learn the weights of these operations to form the module structure. We cast the problem into a bi-level optimization technique in which the model takes alternating gradient descent steps in architecture and weight spaces. Chapter 5

discusses additional experiments and ablation studies that were done in the context of the previous article.

Most works in the literature use a recurrent neural network like LSTM [33] or GRU [13] to model the question features. However, LSTMs can fail to properly encode syntactic features of the question which could be vital to answering some VQA questions [87]. Recently, [76] has shown the utility of language modeling for question-answering. With this motivation, we try to learn a better language model which can be trained in an unsupervised manner. In Chapter 6, we discuss a recursive network for language modeling whose structure aligns with the natural language. More technically, we make use of an unsupervised parsing model (Parsing Reading Predict Network or PPRN [86]) and augment its prediction step with a TreeLSTM [99] model which makes use of the intermediate tree structure given by PRPN model to output a hidden state by utilizing the tree structure. The predict step of PRPN model makes use of a hidden state which is a weighted combination of the TreeLSTM's hidden state and the one obtained from structured attention. This way it helps the model to do unsupervised parsing and also capture long-term dependencies as the structure now explicitly exists in the model. Our experiments demonstrate that this model leads to improvement on language modeling task over the PRPN baseline on Penn Treebank dataset.

Keywords: Visual Question Answering, Visual Reasoning, Modular Networks, Neural Structure Optimization, Language Modeling

Contents

List of Tables

List of Figures

List of Abbreviations

VQA	Visual Question Answering
MS-COCO	Microsoft Common Objects in Context
MLP	Multi Layer Perceptron
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
FLOP	Floating Point Operation
conv.	convolutional
NMN	Neural Module Network
GAN	Generative Adversarial Network
MCB	Multimodal Compact Bilinear
BoW	Bag of Words
RGBD	Red Green Blue Depth
NLVR	Natural Language for Visual Reasoning
PRPN	Parsing-Reading-Predict Network
RNN	Recurrent Neural Network

Acknowledgments

I am immensely grateful to a lot of people who have helped me in my research. The most fruitful part during this journey was played by my advisor Prof. Christopher Pal. Under his able supervision, I got familiar with the research methodology for machine learning. He helped me strengthen my basics. His precise insights about potential issues with training machine learning models were crucial to making them work. I acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) under the COHESA network for research funding. I thank my collaborators Jie Fu and Sarath Chander for their contribution in the article titled "Learning Neural Modules for Visual Question Answering". I also thank Zhouhan Lin for mentoring me in the language modeling aspect. I acknowledge the help of Massimo Caccia in translating the summary to French. I thank Evan Racah and Harm de Vries for helpful discussions.

I am thankful to Compute Canada and Calcul Québec for providing me the computational resources for running my experiments. I also acknowledge the support of Nvidia for NVIDIA NGC Deep Learning Containers and the DGX-1 gifted to our lab. Finally, I wish to thank my family especially my mother for her support and encouragement during the course of my research.

Introduction

Visual Question Answering is one of the most interesting tasks in machine learning. It involves the twin challenges of understanding the two modalities involved i.e. language and vision, plus understanding one in the context of the other. Human intelligence is remarkably proficient in capturing the relationship between different objects in a scene and and being able to process this information to answer questions in conversations. However, most neural-network based models of this task have been susceptible to learning biases in datasets than perform actual reasoning [115] and are thus prone to adversarial attacks. A plausible way to increase human trust in deploying such models for practical applications is to make their decision making process explainable to users. One way to achieve this would be to decompose the model into sub-modules which are designed for specific functions. In Chapter 4, we address the task of learning the internal structure of modules for VQA models for answering compositional reasoning based questions. In Chapter 2, we try to learn a modularized version of ResNeXt CNN [110] model for the task of VQA. The key contribution of this work is to efficiently train VQA models by sparsely executing the CNN sub-network. Visual Question Answering models can benefit from language modeling for obtaining a syntax-aware feature representation for the question and also for generating variable length answer tokens. To this end, we propose a language model which can be trained in an unsupervised manner and also capture long-term dependencies in Chapter 6.

Chapter 1

Background

1.1. Visual Question Answering

Visual Question Answering (VQA) has received considerable attention in the machine learning community in recent years. This can be attributed to the fact that it is seen as an *AI-complete* problem i.e. if given an image, a system is capable of answering any reasonable question, then any vision task could be trivially broken into a set of questions about the subject image.

1.2. VQA Datasets

1.2.1. Natural Image datasets

The **DAQUAR** [63] is one of the first datasets which addresses VQA as its core learning task. It is built on top of the NYU-Depth V2 dataset [71] which contains RGBD images of indoor scenes. The questions comprise of both synthetic ones (generated automatically based on some templates and then instantiated with facts from database) and the ones which are crowd-sourced from human workers. It contains 6794 training and 5674 test question-answer pairs in total. The answer-set contains 894 object categories, colors, numbers or sets of those. The main drawback of this dataset is that the answers to the questions are biased towards certain objects like table or chair.

The **COCO-QA** [79] contains questions generated on the images of MS-COCO image dataset [57]. The questions are generated synthetically by converting the image descriptions to questions. It contains 78736 training and 38948 test question-answer pairs. The authors discarded the answers which appear too frequently or too rarely resulting in a more balanced answer distribution.

Visual Genome [54] is one of the largest VQA datasets by size with over 1.77 million question answer pairs. There are to sub-categories of questions - freeform QAs (which are based on

the entire image) and region-based QAs (based on selected regions of the image). This dataset contains annotations for a large number of objects in images, scene graphs and region descriptions in addition to question-answer pairs. It uses the images present in the intersection of YFCC100M [103] and MS-COCO [57] datasets.

VQA-real [4] This is the most widely used VQA dataset for questions on natural images. The images correspond to 123,287 training and validation images and 81,434 test images from the Microsoft Common Objects in Context (MS COCO) dataset. The questions are crowd-sourced from human annotators. It contains 614,163 questions and 10 answers per question from unique workers. Each question can thus possibly have multiple correct answers. The main shortcoming of this dataset is that given a question type, the answer distribution is highly skewed in certain cases. For instance, the answer for 39% of counting questions is '2' and 'tennis' is the answer for questions like "What sport is" in 41% of the cases [26]. In order to alleviate such biases present in the dataset, a second version of the dataset was proposed (called VQA v2 [26]). In this dataset, the authors try to balance the questions by asking human annotators to create a new question $(I^{'},Q,A^{'})$ for an (image, question, answer) triplet (I,Q,A) such that $I \neq I^{'}$ and $A \neq A^{'}$, but I is similar to $I^{'}$. In this way, I and $I^{'}$ are very close in visual feature space, so it is tough for the model to give both answers correctly without properly understanding the visual modalities.

NLVR^2 [92] This is a dataset based on web photographs in which each example consists of a caption paired with two photographs. The task is to predict whether the caption is True or False. This task requires complex reasoning which includes comparative, quantitative and spatial reasoning on the objects present in the images. This is similar to spirit to CLEVR [48] but the images are natural images instead of synthetic ones and the captions are crowd-sourced from human annotators. This dataset poses twin challenges of complex visual reasoning and understanding human specified captions. It contains a total of 107,292 examples.

GQA [41] is a recently proposed dataset for visual reasoning and compositional question answering on real-world images. It uses the images in Visual Genome dataset [54] that have scene graph annotations to create a semantic ontology over the scene graph vocabulary. The questions are created from 524 templates/patterns using the content present in scene graphs. It contains 22.6M questions over 113K images. All the questions have their respective functional programs included in the dataset. This dataset is strategically designed to mitigate the flaws of previous datasets which

lead models to exploit different kinds of biases rather than perform actual reasoning to achieve good accuracy.

1.2.2. Synthetic image datasets

SHAPES [3] is a dataset of synthetic images which emphasizes the understanding of spatial and logical relations among multiple objects. It contains 244 unique questions, pairing each question with 64 different images, resulting in a total of 15616 unique question-answer pairs.

The **CLEVR** [48] dataset proposed on similar lines as SHAPES, contains questions that test visual reasoning abilities such as counting, comparison, logical reasoning based on 3D shapes like cubes, spheres and cylinders of varied shades. Though it is in similar spirit to SHAPES, the CLEVR dataset is much more complex than SHAPES both in terms of complexity and variety of questions and images. Notably the number of questions in CLEVR dataset is around 1 million (compared to 15K in SHAPES), which has contributed it to became a standard benchmark for a variety of visual reasoning models.

VQA abstract scenes In order to make VQA systems focus on the task of visual reasoning rather than recognition, a new dataset named *VQA abstract scenes* was proposed in [4] along with the main natural image version. It contains scenes of paperdoll models in a variety of indoor and outdoor scenes. The complexity of this dataset is enhanced by models of different ages, gender and races, with different face expressions and pose variations. It contains a total of 50K scenes and 150K questions. [26] introduces a complementary abstract scenes dataset such that in a given example, the two scenes have opposite answers to the same boolean question. The motivation for this dataset is to prevent the model from exploiting language biases for performance but instead focus on visual reasoning.

NLVR The Cornell Natural Language Visual Reasoning (NLVR) [93] is a dataset aimed at testing the ability of VQA systems to answer compositional questions using a visual context. Unlike CLEVR [48] and SHAPES [3] which use synthetic questions, here the questions are crowd-sourced from human annotators. The question is a description of the image and there are two images associated with it. The description is true for only one of the given images and the task of the model is to guess the correct image for the given description. It contains 3,962 unique descriptions and 92,244 examples.

1.3. Evaluation metrics

Some VQA models like [64] output natural language answers to questions. For such models, the evaluation is difficult as it amounts to paraphrase detection, which is itself a topic of active research. [63] proposes the use of Wu-Palmer Set (WUPS) score. This is inspired from Fuzzy Sets literature as many objects in the set of answers tend to be similar (e.g. 'cup' and 'mug'), so all incorrect (as per ground truth) answers should not be penalized equally. Here, $WUP(a, b)$ [109] represents the similarity between two words a and b based on their depth in the taxonomy tree. The VQA-real dataset [4] contains answers from 10 human annotators for each question. A predicted answer is deemed 100 $\%$ correct if 3 or more annotators gave that answer.

$$\text{accuracy} = min\left(\frac{\text{no. of annotators who gave that answer}}{3}, 1\right)$$

[41] introduces some new evaluation metrics like consistency, validity, plausibility and grounding for measuring the coherent reasoning capabilities of VQA models.

1.4. Neural Architectures for VQA

1.4.1. Attention-based Neural Architectures

The span of attention-based neural architectures can be broadly classified into two categories, mainly based on the datasets they are suitable for. The first category of architectures is designed primarily for questions based on real images which are mostly crow-sourced from human annotators. In such cases, the questions can be open-ended and mostly concern objects present in the image, but the compositional reasoning ability required to answer questions present in such datasets, is fairly limited. COCO-QA [79], DAQUAR [63], Visual Genome [54] and VQA-real [4] are some examples in the first category. The second category of model architectures is primarily designed for datasets containing questions which test the compositional reasoning ability of models and are mostly synthetically generated. This category includes datasets like SHAPES [3], CLEVR [48] and NLVR [93].

1.4.1.1. *Models for Open-ended questions*

CNN + BoW [118] is a very naive baseline model for VQA in which the question representation is obtained as a Bag of Words [81] of embeddings of individual question words. The visual features are obtained from GoogLeNet [95].

CNN + LSTM In this baseline model [63], the image features are encoded using a CNN like VGGNet [88], ResNet [31] etc. The final hidden state of LSTM [33] which takes the question word embeddings as input, is used for question representation. Both the image and question representations are concatenated and then passed through an MLP to get the distribution over the set of answers. [79] proposes an approach to handle answers of different word lengths in which an encoder LSTM takes the concatenation of image and question feature vectors as input. A decoder LSTM decodes the encoder representation to produce a variable sized string of words for the answer until a special token indicating the end of sentence is predicted.

Multimodal Compact Bilinear Pooling (MCB) [23] In this model, the question and image features are obtained from LSTM and CNN respectively like previous models. It uses Multimodal Compact Bilinear pooling (MCB) [24] to fuse the two modalities instead of concatenation or element-wise sum or product. This fusion takes place at each location of the spatial grid of features obtained from the CNN. Subsequently, two convolutional layers followed by softmax are used to predict the soft attention score for each location. The weighted sum of visual features using the attention map is again fused with the question representation using MCB. The features thus obtained are passed through an MLP followed by softmax to get the distribution over the set of answers. Figure 1.1 illustrates the VQA architecture which uses MCB.

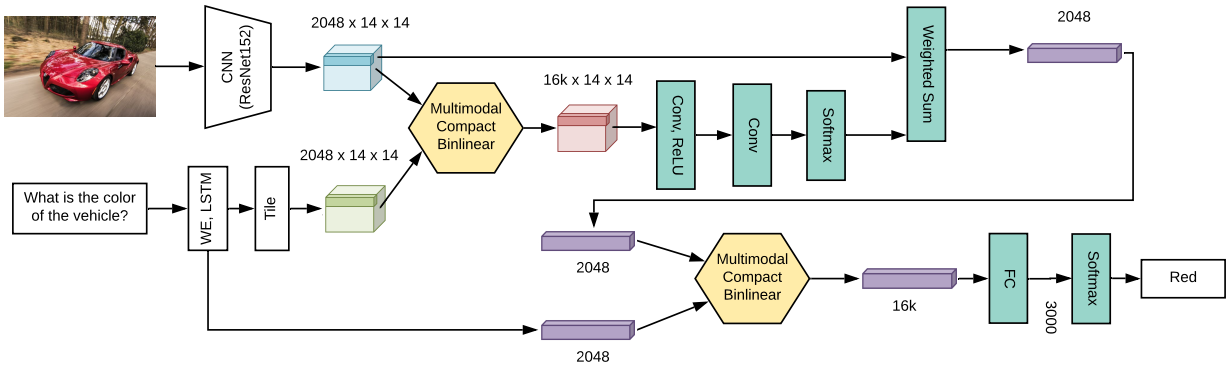

Figure 1.1. Architecture diagram for Multimodal Compact Bilinear (MCB) with Attention (adapted from [23]).

Stacked Attention Networks (SAN) [111] In this model, the model first generates an attention over the image pixels by fusing the spatial image features $\boldsymbol{v_I}$ and question features $\boldsymbol{v_Q}$.

$$\boldsymbol{h_A} = \tanh(\boldsymbol{W_{I,A}}\boldsymbol{v_I} \oplus (\boldsymbol{W_{Q,A}}\boldsymbol{v_Q} + \boldsymbol{b_A}))$$

$$\boldsymbol{p^I} = \text{softmax}(\boldsymbol{W_P}\boldsymbol{h_A} + \boldsymbol{b_P})$$

where $\boldsymbol{v_I} \in \mathbb{R}^{d \times m}$, $\boldsymbol{v_Q} \in \mathbb{R}^d$, $\boldsymbol{W_{I,A}}$, $\boldsymbol{W_{Q,A}} \in \mathbb{R}^{r \times d}$, $\boldsymbol{b_A} \in \mathbb{R}^r$, $\boldsymbol{W_P} \in \mathbb{R}^{1 \times r}$ and $\boldsymbol{b_P} \in \mathbb{R}^m$. $\boldsymbol{p^I} \in \mathbb{R}^m$ corresponds to the attention probability assigned to m image regions. Here, d denotes the feature dimension of each image region in the final conv. layer and r is a hyper-parameter for the size of a parameter matrix. $\oplus$ denotes a matrix-vector summation operation. The weighted sum of visual features using the attention map (denoted by $\tilde{\boldsymbol{v}}_{\boldsymbol{I}}$) is added to the question feature vector $\boldsymbol{v_Q}$ to form the query vector $\boldsymbol{u}$.

$$\tilde{\boldsymbol{v}}_{\boldsymbol{I}} = \sum_i p_i^I \boldsymbol{v_i}$$

$$\boldsymbol{u} = \tilde{\boldsymbol{v}}_{\boldsymbol{I}} + \boldsymbol{v_Q}$$

The attention process described above is repeated for K attention layers. The updated query vector u^k denotes the more fine-grained query in the multimodal space after the k^{th} attention step.

$$\boldsymbol{h_A^k} = \tanh(\boldsymbol{W_{I,A}^k}\boldsymbol{v_I} \oplus (\boldsymbol{W_{Q,A}^k}\boldsymbol{u^{k-1}} + \boldsymbol{b_A^k}))$$

$$\boldsymbol{p^{I,k}} = \text{softmax}(\boldsymbol{W_P^k}\boldsymbol{h_A^k} + \boldsymbol{b_P^k})$$

$$\tilde{\boldsymbol{v}}_{\boldsymbol{I}}^{\boldsymbol{k}} = \sum_i p_i^{I,k} \boldsymbol{v_i}$$

$$\boldsymbol{u^k} = \tilde{\boldsymbol{v}}_{\boldsymbol{I}}^{\boldsymbol{k}} + \boldsymbol{u^{k-1}}$$

The final query vector $\boldsymbol{u_K}$ is fed to the classifier to predict the distribution over the set of answers.

1.4.1.2. *Models for compositional reasoning based questions*

Relational Networks [82] This model incorporates the explicit use of relational features between every pair of objects to predict the answer distribution.

$$\boldsymbol{v_{IQ}} = \sum_{i,j} g([\boldsymbol{v_i}, \boldsymbol{v_j}, \boldsymbol{q}])$$

Here, $\boldsymbol{q}$ denotes the question representation, $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$ denote the spatial features (obtained from CNN) corresponding to the i^{th} and j^{th} image regions and g denotes an MLP. The feature representation $\boldsymbol{v_{IQ}}$ is passed through a classifier to get the answer distribution. A schematic diagram for the Relational Networks model architecture is shown in Figure 1.2.

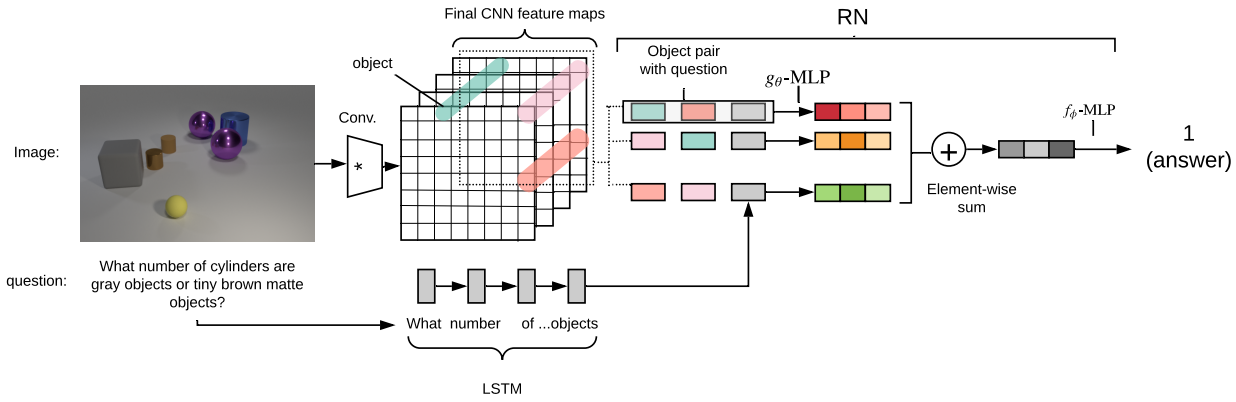

Figure 1.2. Relational Networks model architecture (adapted from [82])

Feature-wise Linear Modulation (FiLM) [75] uses Conditional Batch Normalization [17, 19] (CBN) conditioned on the question features to modulate the visual features for predicting the answer. More specifically, FiLM learns the parameters of Batch Normalization of each channel as affine functions of the GRU [15] representation of the question.

$$\gamma_{i,c} = f_c(\boldsymbol{x}_i)$$

$$\beta_{i,c} = h_c(\boldsymbol{x}_i)$$

$$FiLM(\boldsymbol{F}_{i,c}|\gamma_{i,c}, \beta_{i,c}) = \gamma_{i,c}\boldsymbol{F}_{i,c} + \beta_{i,c}$$

Here, $\boldsymbol{x}_i$ denotes the i^{th} input example's question representation and ($\gamma_{i,c}$,$\beta_{i,c}$) denotes the BN parameters for the i^{th} input and c^{th} channel. The model architecture consists of a CNN (either pre-trained or trained from scratch) followed by four residual blocks. Each residual block consists of a 1×1 convolution layer followed by a 3×3 convolution layer and a batch-normalization layer whose (γ,β) parameters are controlled by FiLM. The classifier (comprising of a 2-layer MLP followed by softmax) uses the features output by FiLM network to output a distribution over the answers. A schematic representation of different components of the FiLM model is shown in Figure 1.3.

Memory, Attention, and Composition (MAC) Network [40] The MAC network consists of an input unit, a recurrent unit comprising of a number of MAC cells stacked together and an output unit. A MAC cell consists of a control unit, read unit and write unit.

The *input unit* ingests the two modalities i.e. the question and image and converts them into their respective distributed feature representations. For the question, a word-embedding is trained from scratch and the features are obtained using a Bidirectional LSTM [33]. The model makes

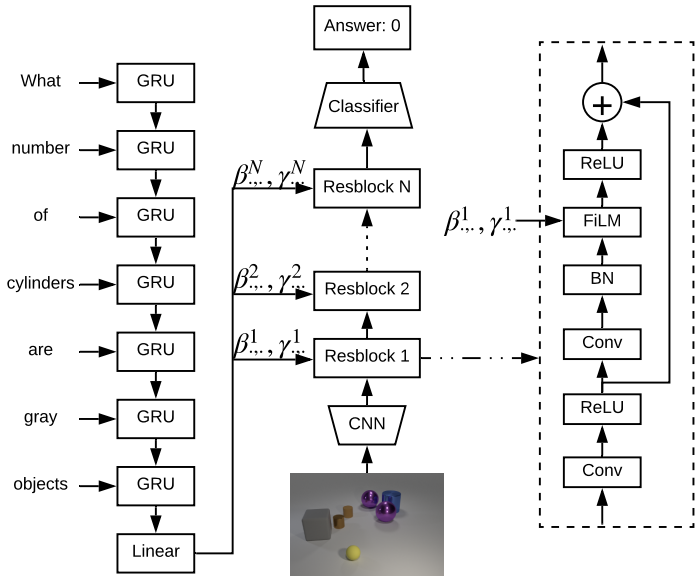

Figure 1.3. Schematic representation of FiLM based model architecture for Visual Reasoning (adapted from [75]).

use of the LSTM outputs at each time-step of the question. For the image, a ResNet101 [32] (pre-trained on ImageNet [18]) is used to extract *conv-4* spatial features from the image.

The function of the *control unit* is to perform an attention on question words and subsequently guide the reasoning operation. Let $[\boldsymbol{cw_1}, \boldsymbol{cw_2}, ..., \boldsymbol{cw_S}]$ denote the output of Bi-LSTM at each time-stamp of the input sequence of question words. Let $\boldsymbol{q}$ denote the concatenation of final hidden state of Bi-LSTM during the forward and backward passes. $\boldsymbol{q_i}$ denotes the question's transformed representation for the i^{th} reasoning step. At first, $\boldsymbol{q_i}$ and $\boldsymbol{c_{i-1}}$ are concatenated and combined through a linear transform to form cq_i. Then, $ca_{i,s}$ is computed as the similarity coefficient of cq_i with the Bi-LSTM feature representation of question words. Finally, $\boldsymbol{c_i}$ represents the weighted feature representation of Bi-LSTM outputs obtained using these similarity values.

$$\boldsymbol{q_i} = \boldsymbol{W}_i^{d\times 2d}\boldsymbol{q} + b_i^d$$

$$cq_i = \boldsymbol{W}^{d\times 2d}[\boldsymbol{c_{i-1}}, \boldsymbol{q_i}] + \boldsymbol{b}^d$$

$$ca_{i,s} = \boldsymbol{W}^{1\times d}(cq_i \odot \boldsymbol{cw_s}) + b^1$$

$$cv_{i,s} = \text{softmax}(ca_{i,s})$$

$$\boldsymbol{c_i} = \sum_{s=1}^{S} cv_{i,s} \cdot \boldsymbol{cw}_s$$

The image features represent the knowledge base $\boldsymbol{K}^{H\times W\times d} = \{\boldsymbol{k}^{\boldsymbol{d}}_{\boldsymbol{h},\boldsymbol{w}}|^{H,W}_{h,w=1,1}\}$ where d is the number of output channels of the CNN. H and W denote the height and width of the visual feature maps respectively. In the above equations, the superscript with the parameter tensors denotes their dimension. The function of the *read unit* is to retrieve the information required for the i^{th} reasoning step. The function of the *write unit* is to compute the result of i^{th} reasoning step and store it in the memory state $\boldsymbol{m_i}$.

1.4.2. Modular Neural Architectures

Neural Module Networks [2] is a technique for answering complex questions which involve spatial, set-theoretic reasoning and the ability to recognize attributes of shapes and objects. Each "module" is a neural network whose structure is pre-specified in terms of layers and whose parameters are learnt during training. The universal dependency parse representation is obtained using Stanford Parser [53]. Next, the set of dependencies is converted into structured queries each of which consists of a function and a short word phrase which the function operates on. These functions are then mapped to modules based on the location in the parse tree, of the node they operate on. Thus, a custom layout of modules is obtained for each question. This model obtains excellent results on SHAPES dataset (introduced in this paper only). The main contribution of this model is that it uses the syntax tree efficiently to generalize well to complex questions involving compositional reasoning. The drawback is that the layout predictor is heavily hand-engineered for this particular dataset and it has to be re-specified for use on other datasets.

End-to-End Module Networks for Visual Question Answering (N2NMN) [37] This model builds upon the approach of Neural Module Networks (NMNs) but improves it in many ways. The layout predictor is not hand-engineered but a policy network with a sequence-to-sequence RNN which outputs a sequence of tokens (denoted by $\left\{m^{(t)}\right\}$) corresponding to the Reverse Polish notation [10] of the syntax tree of modules. The input question (of T words) is encoded using an encoder LSTM to obtain a set of encoder states $[\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots, \boldsymbol{h}_T]$. The decoder is an Attentional Recurrent Neural Network [5]. At timestep t, the decoder predicts attention weights corresponding

to each word of the input sequence:

$$u_{ti} = \boldsymbol{v}^T \tanh(\boldsymbol{W_1 h}_i + \boldsymbol{W_2 h}_t^{'})$$

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

$$\boldsymbol{c}_t = \sum_{i=1}^{T} \alpha_{ti} \boldsymbol{h}_i$$

Here, $\boldsymbol{h}_t^{'}$ denotes the hidden state of decoder LSTM at timestep t, $\boldsymbol{c}_t$ denotes the context vector and $\boldsymbol{q}$ denotes the question representation. $\boldsymbol{v}$, $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ are learnable parameter tensors. The probability of the next token of the layout sequence is predicted as:

$$p(m^{(t)}|m^{(1)}, \cdots, m^{(t-1)}, \boldsymbol{q}) = \text{softmax}(\boldsymbol{W_3 h}_t + \boldsymbol{W_4 c}_t)$$

The probability of a particular layout l in the space of layouts can be written as

$$p(l|\boldsymbol{q}) = \prod_{m^{(t)} \in l} p(m^{(t)}|m^{(1)}, \cdots, m^{(t-1)}, \boldsymbol{q})$$

$$\boldsymbol{x}_{txt} = \sum_{i=1}^{T} \alpha_{ti} \boldsymbol{h}_i$$

The policy samples a high probability layout according to the probability $p(l|\boldsymbol{q})$ and a network of modules is thus assembled. At test time, the maximum probability layout is selected using beam-search. The neural modules take 0, 1 or 2 attention maps as input along with visual features $\boldsymbol{x}_{vis}$ and textual features $\boldsymbol{x}_{txt}$. The implementation details of these modules are shown in Table 1.1. The $\boldsymbol{x}_{vis}$ is obtained from the *conv-4* layer of ResNet-101 (similar to MAC network). The textual feature for module m is computed as the weighted combination of embedding vectors of the question words, weighted by attention vector α_{ti} computed by the decoder LSTM. Since the layout sampling is non-differentiable, it is trained using a combination of backpropagation and policy gradient.

Stack-NMN Stack-NMN [36] is an end-to-end differentiable version of Neural Module Networks. It eliminates the major limitation of previous approaches in this direction [2, 37] that hand engineer the layout or learn it by policy gradient. It uses soft program execution to train the entire model and gradually makes the layout hard towards the end of training by adding an entropy loss term. Please refer to Section 4.2 for more details on this model.

Transparency by Design network [65] builds upon the End-to-End Module Networks [37] but makes an important change in that the proposed modules explicitly utilize attention maps passed

Module name	Att-inputs	Features	Output	Implementation details
`find`	(none)	x_{vis}, x_{txt}	att	$a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W x_{txt}\right)$
`relocate`	a	x_{vis}, x_{txt}	att	$a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W_1 \text{sum}(a \odot x_{vis}) \odot W_2 x_{txt}\right)$
`and`	a_1, a_2	(none)	att	$a_{out} = \text{minimum}(a_1, a_2)$
`or`	a_1, a_2	(none)	att	$a_{out} = \text{maximum}(a_1, a_2)$
`filter`	a	x_{vis}, x_{txt}	att	$a_{out} = \texttt{and}(a, \texttt{find}[x_{vis}, x_{txt}]())$
`[exist, count]`	a	(none)	ans	$y = W^T \text{vec}(a)$
`describe`	a	x_{vis}, x_{txt}	ans	$y = W_1^T \left(W_2 \text{sum}(a \odot x_{vis}) \odot W_3 x_{txt}\right)$
`[eq_count, more, less]`	a_1, a_2	(none)	ans	$y = W_1^T \text{vec}(a_1) + W_2^T \text{vec}(a_2)$
`compare`	a_1, a_2	x_{vis}, x_{txt}	ans	$y = W_1^T \left(W_2 \text{sum}(a_1 \odot x_{vis}) \odot W_3 \text{sum}(a_2 \odot x_{vis}) \odot W_4 x_{txt}\right)$

Table 1.1. Implementation details of neural modules (adapted from [37])

as inputs instead of learning whether or not to use them. This results in more interpretability of the modules since they perform specific functions. It improves the state of art on the CLEVR CoGenT dataset. [48].

Neural Compositional Denotational Semantics for Question Answering [29] constructs a parse-chart over the entire space of parse trees by using a parser similar to CRF-based PCFG parser which uses the CYK algorithm [30, 49, 113]. Each node in the parse chart represents a span of the question. Each token in the question can be an entity, relation, ungrounded semantic type, a truth value or a semantically vacuous word. The composition modules transform the input semantic types into output tokens. It reports good accuracy on CLEVRGEN, a new dataset over CLEVR images which specifically tests semantic operators like conjunction, negation, count etc.

ReasonNet [42] uses two types of modules: visual classification modules and visual attention modules, similar to [37]. The modules are hand-engineered to output specific visual features e.g. object detection, object classification, scene classification, face detection etc. It uses a bilinear model [100] to pool the module's representation and the question representation.

1.5. Neural Structure Optimization

Neural Architecture Search (NAS) is a technique to automatically learn the structure and connectivity of neural networks rather than training human-designed architectures. In [120], a recurrent neural network (RNN) based controller is used to predict the hyper-parameters of a CNN such as number of filters, stride, kernel size etc. They used REINFORCE [108] to train the controller

with validation set accuracy as the reward signal. Among other approaches that use reinforcement learning to perform architecture search, [121] uses proximal policy optimization (PPO) [83] instead of REINFORCE. In [117], Q-learning [106] is used to construct network blocks which are then stacked together to form the complete network. As an alternative to reinforcement learning, evolutionary algorithms [91] have been used to perform architecture search in [77, 68, 60, 78]. Recently, [59] proposed a differentiable approach to perform architecture search and reported success in discovering high-performance architectures for both image classification and language modeling.

1.6. Modular Networks for other tasks

[52] proposes an EM style algorithm to learn black-box modules and their layout for image recognition and language modeling tasks. [1] proposes an approach called modular meta-learning to learn modules (parameterized as MLPs) and a compositional scheme to compose these modules in order to re-use them for new tasks. Progressive Module Networks [50] is a model for multi-task learning for a variety of visual reasoning tasks such as image captioning, counting and visual question answering. It builds modules which are compositional i.e. utilize lower-level modules to produce their output. [116] makes use of modules for training Generative Adversarial Networks (GANs) [25]. The main advantage of this is that modules can be re-used and composed to form GANs for generating images in a specific domain at test time.

Chapter 2

Learning Sparse Mixture of Experts for Visual Question Answering

2.1. Introduction

Deep learning has enabled commendable progress in many machine learning tasks. The ability to train bigger networks (in terms of parameters) with improved infrastructure has enhanced the performance in several tasks like object recognition, object detection, machine translation, face recognition, question-answering etc. However, this poses a serious challenge in terms of huge computational resources that are required to train and deploy such huge models. We aim to tackle this issue for the specific task of Visual Question Answering (VQA). A Convolutional Neural Network (CNN) is an integral part of the visual processing pipeline of a VQA model (assuming the CNN is trained along with entire VQA model). In this project, we propose an efficient and modular neural architecture for the VQA task with focus on the CNN module. Our experiments demonstrate that a sparsely activated CNN based VQA model achieves comparable performance to a standard CNN based VQA model architecture.

2.2. Related Work

Mixture of Experts Mixture of Experts [45] is a formulation in machine learning which employs the divide-and-conquer principle to solve a complex problem by dividing the neural network into several expert networks. In the Mixture of MLP Experts (MME) method [105, 72, 21], a gating network is used to assign a weight g_i to the output of the corresponding expert network. The

final output of the ensemble is a weighted sum of outputs of each of the individual expert networks.

$$\boldsymbol{O} = \sum_{i=1}^{N} \boldsymbol{o}_i g_i$$

Here, there are are total of N experts and $\boldsymbol{o}_i$ denotes the output of the i^{th} expert. The gating network is an MLP followed by softmax operator. [47] uses a mixture of experts for visual reasoning tasks in which each expert is a stack of residual blocks.

Conditional Computation Conditional Computation is the technique of activating only a subportion of the neural network depending on the inputs. For instance, if a visual question answering system has to count the number of a specified object *vs* if it has to tell the color of an object, the specific features needed to give the correct answer in either case are different. Hence, there is a potential to reduce the amount of computation the network has to perform in each case, which can be especially useful to train large deep networks efficiently. The use of stochastic neurons with binary outputs to selectively turn-off experts in a neural network has been explored in [8]. [16] uses a low-rank approximation of weight matrix of MLP to compute the sign of pre-nonlinearity activations. In case of ReLU activation function, it is then used to optimize the matrix multiplication for the MLP layer. [7] uses Policy gradient to sparsely activate units in feed-forward neural networks by relying on conditional computation. [85] proposes Sparsely-Gated Mixture-of-Experts layer (MoE) which uses conditional computation to train huge capacity models on low computational budget for language modeling and machine translation tasks. It makes use of noisy top-K mechanism in which a random noise is added to gating weights and then top-K weights are selected. Another line of work makes use of conditional computation in VQA setting. DPPNet [73] makes use of a dynamic parameter layer (fully-connected) conditioned on the question representation for VQA. ABC-CNN [11] predicts convolutional kernel weights using an MLP which takes question representation as input. Here, the advantage is that the question conditioned convolutional kernels can filter out unrelated image regions in the visual processing pipeline itself.

Computationally efficient CNNs [39] proposes Multi-Scale Dense Convolutional Networks (MSDNet) to address two key issues in (i) budgeted classification- distribute the computation budget unevenly across easy and hard examples, (ii) anytime prediction: the network can output the prediction result at any layer depending on the computation budget without significant loss in accuracy. The optimization of computational complexity of CNNs at inference-time has been studied in [9] in which an adaptive early-exit strategy is learned to by-pass some of the network's layers in

order to save computations. MobileNet [34] uses depth-wise separable convolutions to build light weight CNNs for deployment on mobile devices.

CNN architectures The invention of Convolutional Neural Networks (CNNs) has led to a remarkable improvement in performance for many computer vision tasks [55, 61, 80]. In recent years, there has been a spate of different CNN architectures with changes to depth [89], topology [32, 96], etc. The use of *split-transform-merge* strategy for designing convolutional blocks (which can be stacked to form the complete network) has shown promise for achieving top performance with a lesser computational complexity [96, 97, 98, 110]. The ResNeXt [110] CNN model proposes cardinality (size of set of transformations in a convolutional block) as another dimension apart from depth and width to investigate, for improving the performance of convolutional neural networks. Squeeze-and-Excitation Networks [35] proposes channel-wise attention in a convolutional block and helped improve the state of art in ILSVRC 2017 classification competition.

Grouped Convolution In grouped convolution, each filter convolves only with the input feature maps in its group. The use of grouped convolutions was first done in AlexNet [55] for training a large network on 2 GPUs. A recently proposed CNN architecture named CondenseNet [38] makes use of learned grouped convolutions to minimise superfluous feature-reuse and achieve computational efficiency.

2.3. Model Architecture

Our main goal is to optimize the convolutions performed by the Convolutional Neural Network (CNN) in a conditional computation setting. We use an off-the-shelf neural architecture for both VQA v2 [26] and CLEVR [48] datasets and just replace the CNN with a modularized version of the ResNeXt [110] CNN as we describe below. The details of convolutional architecture used in the VQA v2 model and CLEVR model are illustrated in Table 2.1 and Table 2.2 respectively. For VQA v2, we used the model architecture proposed in [102]. A schematic diagram to illustrate the working of this model is given in Figure 2.1. For CLEVR dataset, we use the Relational Networks [82] model because it is one of the few models which is fully-supervised and trains the CNN in the main model pipeline. A diagram to illustrate the working of this model is shown in Figure 2.2.

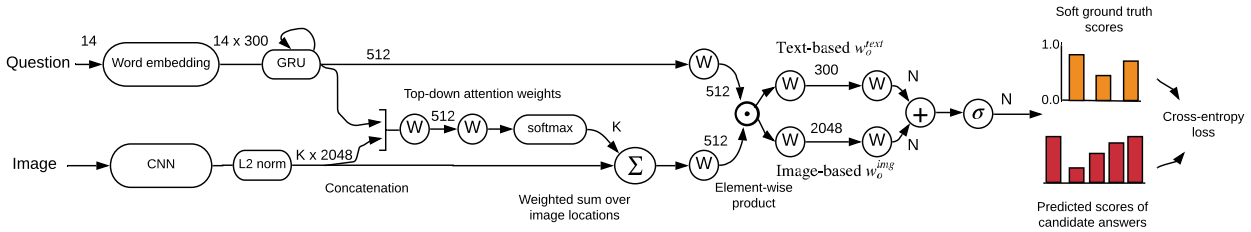

Figure 2.1. Model architecture for VQA v2 dataset (adapted from [102])

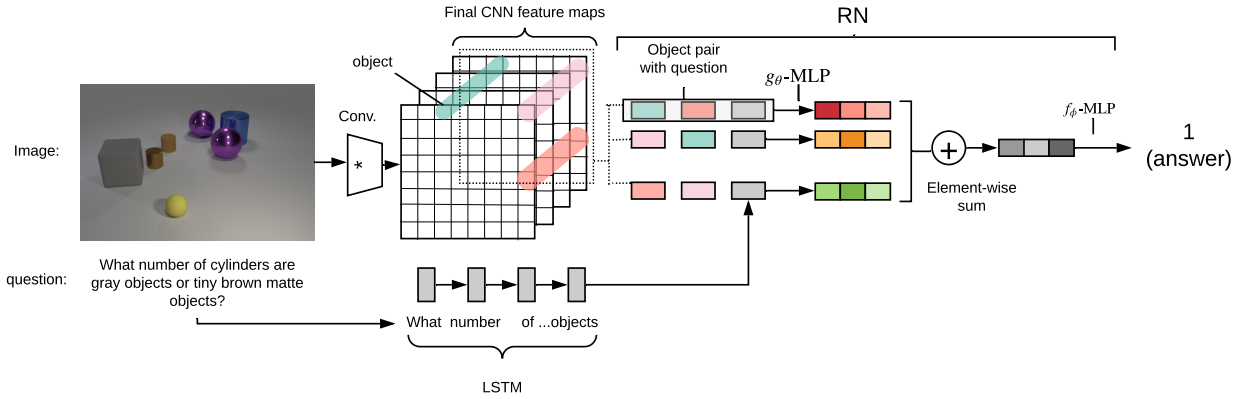

Figure 2.2. Model architecture for Relational Networks (adapted from [82])

2.3.1. Bottleneck convolutional block

The architecture of a residual block of ResNeXt-101 $(32 \times 4d)$ is shown in Figure 2.3. Using algebraic manipulations, it can be shown that Figure 2.3(a), (b) and (c) are equivalent in terms of the function computed by it. Using grouped convolution (with number of groups=32), Figure 2.3(c) is an optimum way of implementing it in standard libraries. We introduce a gating mechanism to assign weights to each of the paths (which equal 32 in the example shown). We treat each path as a convolutional module which should potentially be used for a specific function. The gate values are normalized to sum to unity and are conditioned on the LSTM based feature representation of the question. The working of gate controller is detailed in section 2.3.2.

In order to optimize the computation of a ResNeXt residual block, we execute just the top-k (out of 32) paths and zero out the contribution of others. This is based on the hypothesis that the gate controller shall determine the most important modules (*aka* paths) to execute by assigning higher weights to more important modules. Figure 2.4 (a-d) shows the transformation of modularized ResNeXt-101 $(32 \times 4d)$ block to its grouped convolution form. This technique is similarly applicable for the convolutional block used for CLEVR dataset (shown in Table 2.2). In our efficient

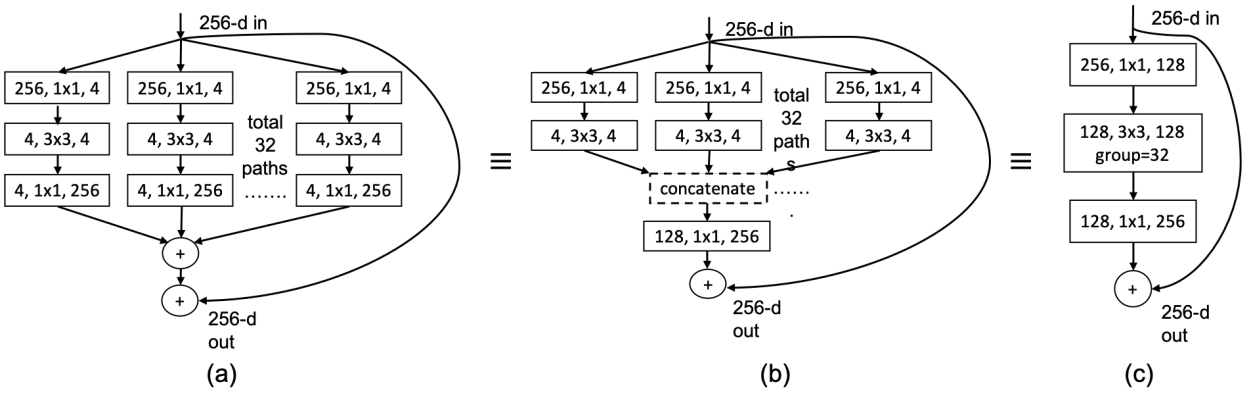

Figure 2.3. Architecture of a sample block of ResNeXt-101 ($32 \times 4d$)

stage	output	Description
conv1	112×112	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2 [1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 256 3×3, 256, C=32 1×1, 512] ×4
conv4	14×14	[1×1, 512 3×3, 512, C=32 1×1, 1024] ×23
conv5	7×7	[1×1, 1024 3×3, 1024, C=32 1×1, 2048] ×3

Table 2.1. Modular CNN for VQA v2 model

stage	output	Description
conv1	64×64	3×3, 64, stride 2
conv2	32×32	3×3 max pool, stride 2 [1×1, 48 3×3, 48, C=12 1×1, 48] ×1
conv3	16×16	[1×1, 48 3×3, 48, C=12 1×1, 48] ×1
conv4	8×8	[1×1, 48 3×3, 48, C=12 1×1, 48] ×1
conv5	8×8	1 × 1 conv. layer with 24 o/p channels

Table 2.2. Modular CNN for Relational Networks Model

implementation, we avoid executing the groups which don't fall in top-k. More technically, we aggregate the non-contiguous groups of the input feature map, which fall in top-k, into a new feature

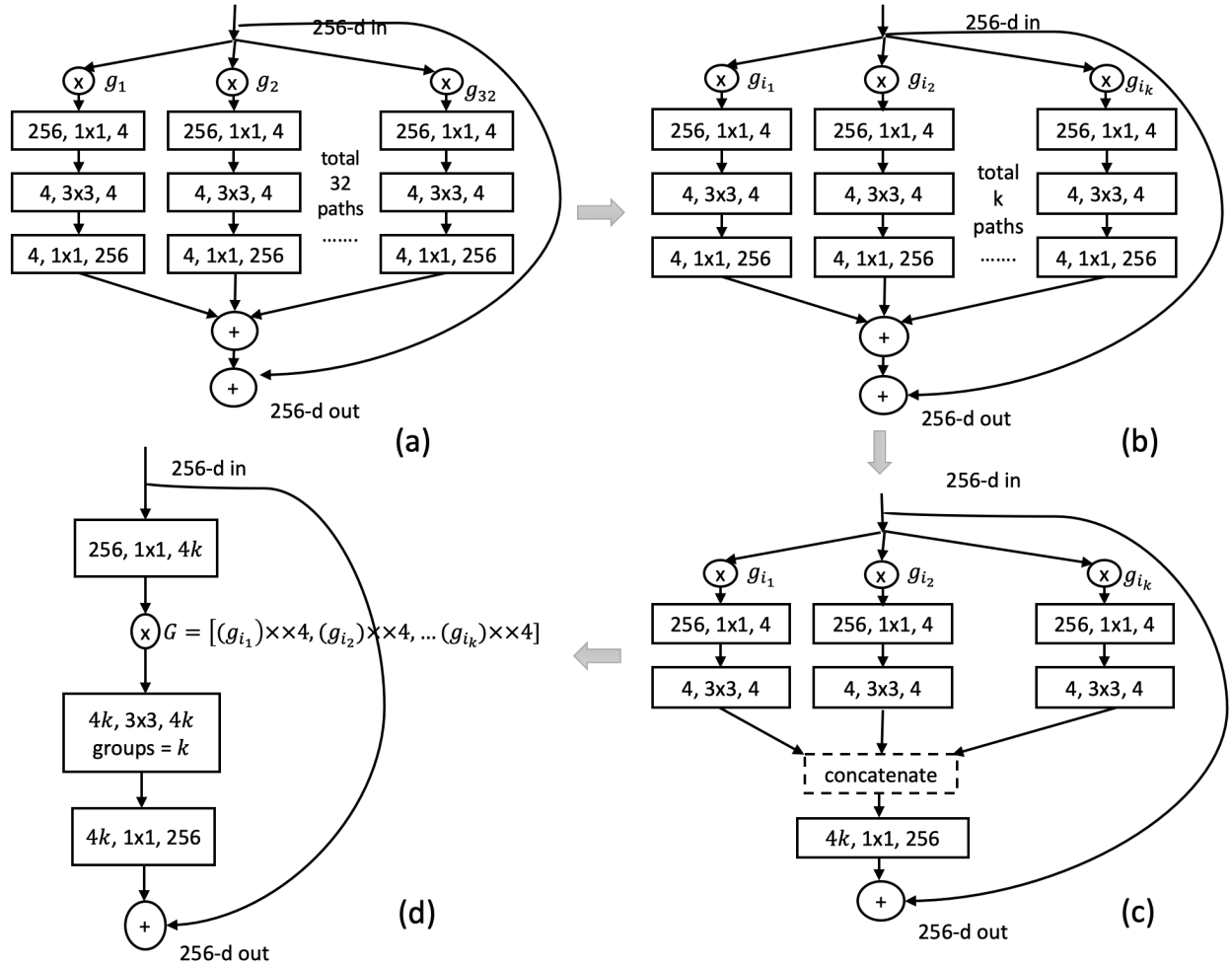

Figure 2.4. Architecture of corresponding block of conditional gated ResNeXt-101 $(32 \times 4d)$ assuming we choose to turn ON paths with $top - k$ gating weights. Here $i_1, \cdots, i_k$ denote the indices of groups in $top - k$.

map. We perform the same trick for the corresponding convolutional and batch-norm weights and biases.

Computational complexity of the ResNeXt convolutional block (in terms of floating point operations) *=

*. No. of FLOPS of a convolutional block (no grouping) = $C_{in} * C_{out} * p * p * H_o * W_o$, No. of FLOPS of a convolutional block (with grouped convolution) = $\frac{C_{in} * C_{out} * p * p * H_o * W_o}{k}$ where C_{in}, C_{out}, p, k, H_o, W_o denote the number of input channels, no. of output channels, kernel size, no. of groups, output feature map height and width respectively

$$= \text{complexity(}\textit{conv-reduce}\text{)} + \text{complexity(}\textit{conv-conv}\text{)}+$$

$$\text{complexity(}\textit{conv-expand}\text{)} + \text{complexity(}\textit{bn-reduce, bn, bn-expand}\text{)}$$

$$= \text{feature-dim} * (k*d) * 1 * 1 * H_o^1 * W_o^1 + \frac{(k*d)*(k*d)*3*3}{k} * H_o^2 * W_o^2$$

$$+ (k*d) * \text{feature-dim} * 1 * 1 * H_o^3 * W_o^3 + O(k)$$

$$= k*d*\text{feature-dim} * H_o^1 * W_o^1 + k*d*\text{feature-dim} * H_o^3 * W_o^3$$

$$+ 9 * d^2 * k * H_o^2 * W_o^2 + O(k)$$

Notation: *conv-reduce*, *conv-conv* and *conv-expand* denote the 1×1, 3×3 and 1×1 convolutional layers in a ResNeXt convolutional block (in that order).

Here, d denotes the size of group in group convolution (equals 4 for Figure 2.4) and 'feature dim' denotes the no. of channels in the feature map input to the convolutional block. The implementation of modularized ResNeXt block is more efficient than the regular implementation in the case when $k < 32$. The overhead of the gate controller can be minimized by appropriate selection of hyper-parameters. The comparison of FLOPS with varying values of the hyper-parameter k is shown in Table 2.3 for the VQA v2 model and Table 2.4 for the CLEVR model.

2.3.2. Gate controller

The gate controller takes as input the LSTM based representation of the question and the intermediate convolutional map which is the output of the previous block. Given the image features $\boldsymbol{v_I}$ and question features $\boldsymbol{v_Q}$, we perform the fusion of these features, followed by a linear layer and softmax to generate the attention $\boldsymbol{p^I}$ over the pixels of the image feature input.

$$\tilde{\boldsymbol{v}}_{\boldsymbol{I}} = \sum_i p_i^I \boldsymbol{v_i}$$

$$\boldsymbol{u_{query}} = \tilde{\boldsymbol{v}}_{\boldsymbol{I}} + \boldsymbol{v_Q}$$

$$\boldsymbol{g} = \boldsymbol{W_g u_{query}} + \boldsymbol{b_g}$$

$$\boldsymbol{g}' = \frac{ReLU(\boldsymbol{g})}{||ReLU(\boldsymbol{g})||_1}$$

Notation: $\boldsymbol{W_I} \in \mathbb{R}^{A \times B}$, $\boldsymbol{W_{I,A}} \in \mathbb{R}^{C \times B}$, $\boldsymbol{W_{Q,A}} \in \mathbb{R}^{B \times C}$, $\boldsymbol{W_P} \in \mathbb{R}^{1 \times C}$, $\boldsymbol{b_P} \in \mathbb{R}^{1 \times D}$, $\boldsymbol{W_g} \in \mathbb{R}^{B \times E}$, $\boldsymbol{b_g} \in \mathbb{R}^{B \times 1}$, $A =$ Feature dim. of each img. region in final conv. layer, $B =$ question feature dim. (from LSTM/GRU), $C =$ Hidden layer dim., $D =$ No. of img. regions,

E = No. of modules/residual blocks. We add an additional loss term which equals the square of coefficient of variation (CV) of gate values for each convolutional block.

$$CV(\boldsymbol{g}^{'}) = \frac{\sigma(\sum_{i=1}^{N} \boldsymbol{g}^{'}_{\boldsymbol{i},.})}{\mu(\sum_{i=1}^{N} \boldsymbol{g}^{'}_{\boldsymbol{i},.})}$$

In the above equation, $\boldsymbol{g}^{'}_{\boldsymbol{i},.} \in \mathbb{R}^{E}$ denotes the gating weight vector for the i^{th} batch example. This helps to balance out the variation in gate values [85] otherwise the weights corresponding to the modules which get activated initially will increase in magnitude and this behavior reinforces itself as the training progresses.

2.4. Experiments

2.4.1. VQA v2 dataset

We use the Bottom-up attention model for VQA v2 dataset as proposed in [101] as our base model and replace the CNN sub-network with our custom CNN. The corresponding results are shown in Table 2.3. The results show that there is a very minimal loss in accuracy from 0% sparsity to 50% sparsity. However, with 75% sparsity [†], there is a marked 3.62% loss in overall accuracy.

2.4.2. CLEVR dataset

We use the Relational Networks model [82] and replace the Vanilla CNN used in their model with our modularized CNN and report the results on the CLEVR v1 dataset. The CNN used for this model has four layers with one residual ResNeXt block each followed by a 1×1 convolutional layer. The results for these experiments are shown in Table 2.4. The results show that with a slight dip in performance, the model which uses 50% sparsity has comparable performance with the one which doesn't have sparsity in the convolutional ResNeXt block.

†. Here, 75% sparsity means that 75% of the modules/paths in the ResNeXt convolutional block are turned off.

‡. The FLOPS calculation assumes an input image of size $224 \times 224 \times 3$

§. The baseline model doesn't use R-CNN based features, so the accuracy is not directly comparable with state of the art approaches.

¶. The FLOPS calculation assumes an input image of size $128 \times 128 \times 3$

Architecture for CNN	FLOPS (CNN) [‡]	Accuracy (%)
ResNeXt-32 (101 x 32d) (baseline) [§]	156.04E+09	54.51
Modular ResNeXt-32 (101 x 32d) k = 32 (0 % sparsity)	181.39E+09	54.90
Modular ResNeXt-32 (101 x 32d) k = 16 (50 % sparsity)	77.72E+09	54.47
Modular ResNeXt-32 (101 x 32d) k = 8 (75 % sparsity)	45.94E+09	51.28

Table 2.3. Results on VQA v2 validation set

CNN Model description	FLOPS (CNN) [¶]	Val. Acc. (%)
Modular CNN, k=12 (baseline)	5.37E+07	94.05
Modular CNN, k=6, 50 % sparsity	3.21E+07	92.23

Table 2.4. Results on CLEVR v1.0 validation set (Overall accuracy)

2.5. Conclusion

We presented a general framework for utilizing conditional computation to sparsely execute a subset of modules in a convolutional block of ResNeXt model. The amount of sparsity is a user-controlled hyper-parameter which can be used to turn off the less important modules conditioned on the question representation, thereby increasing the computational efficiency. Future work may include studying the utility of this technique in other multimodal machine learning applications which support use of conditional computation.

Chapter 3

Prologue to First Article

3.1. Article details

Learning Neural Modules for Visual Question Answering by **Vardaan Pahuja**, Jie Fu, Sarath Chandar, Christopher Pal was submitted to Association for Computational Linguistics (ACL) 2019 conference and is currently under review.

Personal Contribution: The initial idea for the alternating optimization of module structure and module parameters was jointly proposed by Vardaan Pahuja and Jie Fu. I was responsible for coming up with the generalized module structure and the idea of using elementary arithmetic operations to form the module operations. The idea of using Integrated Gradients [94] for measuring module sensitivity and other post-analysis techniques were suggested by me. The initial draft of the paper was written by me. I was also responsible for the entire code implementation in PyTorch [74] library and running different experiments related to training, post-analysis and hyper-parameter optimization.

Contribution of other authors: Jie Fu and Prof. Christopher Pal helped in initial brainstorming about the project and model structure in particular. Sarath Chander helped to accelerate the progress by giving insights on appropriate selection of hyper-parameters. He also suggested the idea of adding a sparsity loss to get a sparse set of module weights at each node in module. Finally, everyone contributed to improving the draft of the paper.

3.2. Context

In this paper, we are interested in the task of Visual Reasoning, which is an important component of Visual Question Answering. Section 1.4.1.2 discusses monolithic black-box architectures

for visual reasoning like FiLM [75], Relational Networks [82] and MAC Network [40]. The second class of architectures [3, 37] (mentioned in Section 1.4.2) consist of modules which are neural networks (trained from scratch) specialized to perform a particular function.

3.3. Contributions

A major limitation of the modular neural architectures discussed in the previous section is that the modules being used were hand-engineered. In this work, we try to learn the structure of modules in terms of elementary arithmetic operations. Our results show that we achieve comparable performance as the model with hand-designed modules. We present a detailed analysis of the degree to which each module influences the prediction function of the model, the effect of each arithmetic operation on overall accuracy and the analytical expressions of the learned modules.

Chapter 4

Learning Neural Modules for Visual Question Answering

Abstract

Visual question answering involves both visual recognition and reasoning grounded in language. Broadly speaking, there are two popular neural approaches to this problem domain. One family of approaches tends to be based on more black-box architectures that perform the fusion of vision and language through multiple steps of attention. Another family of approaches involves human-specified neural modules, each specialized in a specific form of reasoning, where the order of execution of the underlying modules is sometimes learned. In this work, we further expand the second approach and learn the underlying internal structure of modules in terms of simple and elementary arithmetic operators. Our results show that one is indeed able to simultaneously learn both internal module structure and module sequencing, without extra supervisory signals for module execution sequencing. With this approach, we report performance comparable to the models using hand-designed modules.

4.1. Introduction

Visual question answering requires a learning model to answer sophisticated queries about visual inputs. Such reasoning is considered the hallmark of human intelligence and allows us to interpret and draw plausible inferences from our daily interaction with objects present in the environment. Significant progress has been made in this direction to design neural networks which can answer queries about images. However, recent studies [10, 11] have shown that the neural networks tend to exploit biases in the datasets without learning how to actually reason.

There are two broad classes of approaches proposed in the literature for the task of visual reasoning. The first class of approaches propose end-to-end black-box architectures. Examples include FiLM [18] and MAC [8] where the fusion of representations of the two modalities (image and text) is performed by using a variety of neural network architectures. The second class of

approaches are modular networks where each module is designed to focus on a sub-component of the reasoning process.

[2] propose Neural Module Network (NMN) where they compose neural network modules (with shared parameters) for each input question based on the layout predicted by the syntactic parse of the question. The modules take as input the images or the attention maps and return attention maps or labels as output. In [7], the layout prediction is relaxed by learning a layout policy with a sequence-to-sequence RNN. This layout policy is jointly trained along with the parameters of the modules. The model proposed in [6] avoids the use of reinforcement learning to train the layout predictor, and uses soft program execution to jointly learn both layout and module parameters.

A fundamental limitation of these previous modular approaches to visual reasoning is that the modules need to be hand-specified. This might not be feasible when one has limited knowledge of the kinds of questions or associated visual reasoning required to solve the task. In this work, we present an approach to learn the module structure, along with the parameters of the modules in an end-to-end differentiable training setting. Our proposed model, Learnable Neural Module Network (LNMN), learns the structure of the module, the parameters of the module, and the way to compose the modules based on just the regular task loss. Our results show that we are able to learn the structure of the modules automatically and still perform comparable to hand-specified modules. We would like to highlight the fact that our goal in this paper is not to beat the performance of the hand-specified modules since they are specifically engineered for the task. Instead, our goal is to explore the possibility of designing general purpose reasoning modules in a completely data-driven fashion.

4.2. Background

In this section, we describe the working of the *Stack-NMN* model [6] as our proposed LNMN model uses this as the base model. The *Stack-NMN* model is an end-to-end differentiable model for the task of Visual Question Answering and Referential Expression Grounding [21]. It addresses a major drawback of prior visual reasoning models in literature that compositional reasoning is implemented without need of supervisory signals for composing the layout at training time. It consists of several hand-specified modules (namely Find, Transform, And, Or, Filter, Scene, Answer, Compare and NoOp) which are parameterized, differentiable, and implement common routines needed

in visual reasoning and learns to compose them without strong supervision. The implementation detail of these modules is given in Appendix (see Table 4.B.1). The different sub-components of the Stack-NMN model are described below.

4.2.1. Module Layout Controller

The structure of the controller is similar to the one proposed in [8]. The controller first encodes the question using a bi-directional LSTM [5]. Let $[\boldsymbol{h_1}, \boldsymbol{h_2}, ..., \boldsymbol{h_S}]$ denote the output of Bi-LSTM at each time-step of the input sequence of question words. Let $\boldsymbol{q}$ denote the concatenation of final hidden state of Bi-LSTM during the forward and backward passes. $\boldsymbol{q}$ can be considered as the encoding of the entire question. The controller executes the modules iteratively for T times. At each time-step, the updated query representation $\boldsymbol{u}$ is obtained as:

$$\boldsymbol{u} = \boldsymbol{W_2}[\boldsymbol{W_1^{(t)}}\boldsymbol{q} + \boldsymbol{b_1}; \boldsymbol{c_{t-1}}] + \boldsymbol{b_2}$$

where $\boldsymbol{W_1^{(t)}} \in \mathbb{R}^{d \times d}$, $\boldsymbol{W_2} \in \mathbb{R}^{d \times 2d}$, $\boldsymbol{b_1} \in \mathbb{R}^d$, $\boldsymbol{b_2} \in \mathbb{R}^d$. $\boldsymbol{c_{t-1}}$ is the textual parameter from the previous time step. The controller has two outputs viz. the textual parameter at step t (denoted by $\boldsymbol{c_t}$) and the attention on each module (denoted by vector $\boldsymbol{w^{(t)}}$). The controller first predicts an attention on each of the words of the question and then uses this attention to do a weighted average over the outputs of the Bi-LSTM.

$$cv_{t,s} = softmax(\boldsymbol{W_3}(\boldsymbol{u} \odot \boldsymbol{h_s}))$$

$$\boldsymbol{c_t} = \sum_{s=1}^{S} cv_{t,s} \cdot \boldsymbol{h_s}$$

where, $\boldsymbol{W_3} \in \mathbb{R}^{1 \times d}$. The attention on each module $\boldsymbol{w^{(t)}}$ is obtained by feeding the query representation at each time-step to a Multi-layer Perceptron (MLP).

$$\boldsymbol{w^{(t)}} = softmax(MLP(\boldsymbol{u}; \theta_{MLP}))$$

4.2.2. Operation of Memory Stack for storing attention maps

In order to answer a visual reasoning question, the model needs to execute modules in a tree-structured layout. In order to facilitate this sort of compositional behavior, a differentiable memory pool to store and retrieve intermediate attention maps is used. The memory stack (with length denoted by L) stores $H \times W$ dimensional attention maps, where H and W are the height and width of image feature maps respectively. Depending on the number of attention maps required as

input by the module, it pops them from the stack and later pushes the result back to the stack. The model performs soft module execution by executing all modules at each time-step. The updated stack and stack pointer at each subsequent time-step are obtained by a weighted average of those corresponding to each module using the weights $\boldsymbol{w^{(t)}}$ predicted by the module controller. This is illustrated by the equations below:

$$(A_m^{(t)}, p_m^{(t)}) = \text{run-module}(m, A^{(t)}, p^{(t)})$$
$$A^{(t+1)} = \sum_{m \in M} A_m^{(t)} \cdot w_m^{(t)}$$
$$p^{(t+1)} = softmax(\sum_{m \in M} p_m^{(t)} \cdot w_m^{(t)})$$

Here, $A_m^{(t)}$ and $p_m^{(t)}$ denote the stack and stack pointer respectively, after executing module m at time-step t. $A^{(t)}$ and $p^{(t)}$ denote the stack and stack pointer obtained after the weighted average of those corresponding to all modules at previous time-step $(t-1)$. The working of module layout controller and its interfacing with memory stack is illustrated in Algorithm 1. The internal functioning of a module is shown in Appendix (see Algorithm 3).

4.2.3. Final Classifier

At each time-step of module execution, the weighted average of output of the *Answer* modules is called memory features (denoted by $f_{mem}^{(t)} = \sum_{m \in \text{ans. module}} o_m^{(t)} w_m^{(t)}$). Here, $o_m^{(t)}$ denotes the output of module m at time t. The memory features are given as one of the inputs to the *Answer* modules at the next time-step. The memory features at the final time-step are concatenated with the question representation, and then fed to an MLP to obtain the logits.

4.3. Learnable Neural Module Networks

In this section, we introduce Learnable Neural Module Networks (LNMNs) for visual reasoning, which extends Stack-NMN. However, the modules in LNMN are not hand-specified. Rather, they are generic modules as specified below.

4.3.1. Structure of the Generic Module

The *cell* (see Figures 4.1, 4.2) denotes a generic module which can span all the required modules for a visual reasoning task. Each cell contains a certain number of nodes. The function of a node (denoted by O) is to perform a weighted sum of outputs of different arithmetic operations

Data: Question (string), Image features ($\mathcal{I}$)

Encode the input question into d-dimensional sequence $[\boldsymbol{h_1}, \boldsymbol{h_2}, ..., \boldsymbol{h_S}]$ using Bidirectional LSTM.

$A^{(0)} \leftarrow$ Initialize the memory stack $(A; p)$ with uniform image attention and set stack the pointer p to point at the bottom of the stack (one-hot vector with 1 in the 1^{st} dim.).

for *each time-step t = 0, 1, ..., (T-1)* **do**

$\boldsymbol{u} = \boldsymbol{W_2}[\boldsymbol{W_1^{(t)}}\boldsymbol{q} + \boldsymbol{b_1}; \boldsymbol{c_{t-1}}] + \boldsymbol{b_2}$;

$\boldsymbol{w^{(t)}} = softmax(MLP(\boldsymbol{u}; \theta_{MLP}))$;

$cv_{t,s} = softmax(W_3(\boldsymbol{u} \odot \boldsymbol{h_s}))$;

$\boldsymbol{c_t} = \sum_{s=1}^{S} cv_{t,s} \cdot \boldsymbol{h_s}$

for *every module* $m \in M$ **do**

Produce updated stack and stack pointer:

$(A_m^{(t)}, p_m^{(t)}) =$ run-module$(m, A^{(t)}, p^{(t)}, \boldsymbol{c_t}, \mathcal{I})$;

end

$A^{(t+1)} = \sum_{m \in M} A_m^{(t)} \cdot w_m^{(t)}$;

$p^{(t+1)} = softmax(\sum_{m \in M} p_m^{(t)} \cdot w_m^{(t)})$

end

Algorithm 1: Operation of Module Layout Controller and Memory Stack.

applied on the input feature maps $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$. Let $\boldsymbol{\alpha}^{'} = \sigma(\boldsymbol{\alpha})$ denote the output of softmax function applied to the vector $\boldsymbol{\alpha}$ such that

$$\boldsymbol{\alpha}_i^{'} = \sigma(\boldsymbol{\alpha})_i = \frac{exp(\boldsymbol{\alpha}_i)}{\sum_{j=1}^{6} exp(\boldsymbol{\alpha}_j)}$$

$$O(\boldsymbol{x_1},\boldsymbol{x_2}) = \alpha_1^{'} * min(\boldsymbol{x_1}, \boldsymbol{x_2}) + \alpha_2^{'} * max(\boldsymbol{x_1}, \boldsymbol{x_2}) + \alpha_3^{'} * (\boldsymbol{x_1} + \boldsymbol{x_2}) + \alpha_4^{'} * (\boldsymbol{x_1} \odot \boldsymbol{x_2}) + \alpha_5^{'} * choose_1(\boldsymbol{x_1}, \boldsymbol{x_2}) + \alpha_6^{'} * choose_2(\boldsymbol{x_1}, \boldsymbol{x_2})$$

The last two non-standard functions are defined as below:

$$choose_1(\boldsymbol{x_1}, \boldsymbol{x_2}) = \boldsymbol{x_1}$$

$$choose_2(\boldsymbol{x_1}, \boldsymbol{x_2}) = \boldsymbol{x_2}$$

We consider two broad kinds of modules: (i) *Attention modules* which output an attention map (ii) *Answer modules* which output features to be stored in memory. Among each of these two

categories, there is a finer categorization:

4.3.1.1. *Generic Module with 3 inputs*

This module type receives 3 inputs (i.e. image features, textual parameter, and a single attention map) and produces a single output. The first node receives input from the image feature ($\mathcal{I}$) and the attention map (popped from the memory stack). The second node receives input from the textual parameter followed by a linear layer ($W_1 c_{txt}$), and the output of the first node.

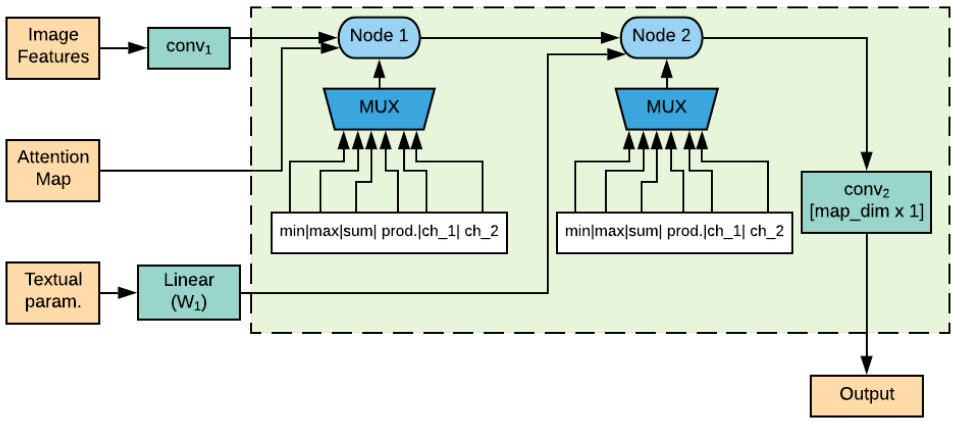

Figure 4.1. Attention Module schematic diagram (3 inputs).

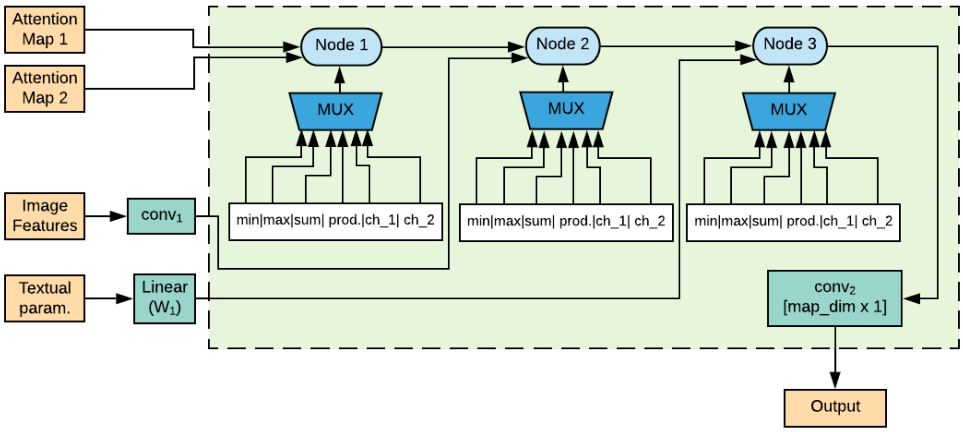

Figure 4.2. Attention Module schematic diagram (4 inputs).

4.3.1.2. *Generic Module with 4 inputs*

This module type receives 4 inputs (i.e. image features, textual parameter and two attention maps) and produces a single output. The first node receives the two attention maps, each of which are popped from the memory stack, as input. The second node receives input from the image features along with the output of the first node. The third node receives input from the textual parameter followed by a linear layer, and the output of the second node.

For the *Attention* modules, the output of the final node is converted into a single-channel attention map using a 1×1 convolutional layer. For the *Answer* modules, the output of the final node is summed over spatial dimensions, and the resulting feature vector is concatenated with memory features of previous time-step and textual parameter features, fed to a linear layer to output memory features. The schematic diagrams of the *Answer* modules are given in the Appendix section (see Figures 4.A.1, 4.A.2).

4.3.2. Overall structure

The structure of our end-to-end model extends *Stack-NMN* in that we specify each module in terms of the generic module (defined in Section 4.3.1). We experiment with three model ablations in terms of number of modules for each type being used. See Table 4.1 for details [*]. We train the module network parameters (denoted by $\boldsymbol{\alpha} = \left\{\alpha_i^{m,k}\right\}_{i=1}^{6}$ for k^{th} node of module m) and the weight parameters ($\mathcal{W}$) by adopting alternating gradient descent steps in architecture and weight spaces respectively. For a particular epoch, the gradient step in weight space is performed on each training batch, and the gradient step in architecture space is performed on a batch randomly sampled from the validation set. This is done to ensure that we find an architecture corresponding to the modules which has a low validation loss on the updated weight parameters. This is inspired by the technique used in [15] to learn monolithic architectures like CNNs and RNNs in terms of basic building blocks (or *cells*). Algorithm 2 illustrates the training algorithm. Here, $\mathcal{L}_{train}(\mathcal{W}, \boldsymbol{\alpha})$ and $\mathcal{L}_{val}(\mathcal{W}, \boldsymbol{\alpha})$ denote the training loss and validation loss on the combination of parameters $(\mathcal{W}, \boldsymbol{\alpha})$ respectively. For the gradient step on the training batch, we add an additional loss term to initially maximize the entropy of $\boldsymbol{w^{(t)}}$ and gradually anneal the regularization coefficient (λ_w) to opposite sign (which minimizes the entropy of $\boldsymbol{w^{(t)}}$ towards the end of training). The value of λ_w varies linearly from 1.0 to 0.0 in the first 20 epochs and then steeply decreases to -1.0 in next

$*$. 1 NoOp module is included by default in all ablations.

10 epochs. The trend of variation of λ_w is shown in Appendix (see Figure 4..5). For the gradient steps in the architecture space, we add an additional loss term ($\frac{l^2}{l^1} = \frac{\|\boldsymbol{\sigma}(\boldsymbol{\alpha})\|_2}{\|\boldsymbol{\sigma}(\boldsymbol{\alpha})\|_1}$)[9] to encourage the sparsity of $\boldsymbol{\alpha}$ parameters after softmax activation.

Model	Attn. modules (3 input)	Attn. modules (4 input)	Ans. modules (3 input)	Ans. modules (4 input)
LNMN (9)	4	2	1	1
LNMN (11)	4	2	2	2
LNMN (14)	5	4	2	2

Table 4.1. Number of modules of each type for different model ablations.

while *not converged* **do**

1. Update weights $\mathcal{W}$ by descending $\nabla_{\boldsymbol{w}}\left[\mathcal{L}_{train}(\mathcal{W}, \boldsymbol{\alpha}) - \frac{\lambda_w}{T}\sum_{t=1}^{T}\mathcal{H}(\boldsymbol{w^{(t)}})\right]$
2. Update architecture $\boldsymbol{\alpha}$ by descending
$\nabla_{\boldsymbol{\alpha}}\left[\mathcal{L}_{val}(\mathcal{W}, \boldsymbol{\alpha}) - \lambda_{op}\sum_{m=1}^{M}\sum_{k=1}^{p}\frac{\left\|\boldsymbol{\sigma}(\boldsymbol{\alpha}^{m,k})\right\|_2}{\left\|\boldsymbol{\sigma}(\boldsymbol{\alpha}^{m,k})\right\|_1}\right]$

end

Algorithm 2: Training Algorithm for LNMN Modules. Here, $\boldsymbol{\alpha}$ denotes the collection of module network parameters i.e. $\left\{\alpha_i^{m,k}\right\}_{i=1}^{6}$ for k^{th} node of module m, $\mathcal{W}$ denotes the collection of weight parameters of modules and all other non-module parameters.

4.4. Experiments

We train our model on the CLEVR visual reasoning task. CLEVR [11] is a synthetic dataset for visual reasoning containing around 700K examples, and has become the standard benchmark to test visual reasoning models. It contains questions that test visual reasoning abilities such as counting, comparing, logical reasoning based on 3D shapes like cubes, spheres, and cylinders of varied shades. A typical example question and image pair from this dataset is given in Appendix (see Figure 4..4). The results on CLEVR test set are reported in Table 4.2. Some ablations of the model are shown in Table 4.3. We use the pre-trained CLEVR model to fine-tune the model on

Model	**CLEVR Overall**	Count	Exist	Compare Numbers	Query Attribute	Compare Attribute	**CLEVR Humans**
Human [12]	92.6	86.7	96.6	86.5	95.0	96.0	-
Q-type baseline [12]	41.8	34.6	50.2	51.0	36.0	51.3	-
LSTM [12]	46.8	41.7	61.1	69.8	36.8	51.8	36.5
CNN+LSTM [12]	52.3	43.7	65.2	67.1	49.3	53.0	43.2
CNN+LSTM+SA+MLP [11]	73.2	59.7	77.9	75.1	80.9	70.8	57.6
N2NMN* [7]	83.7	68.5	85.7	84.9	90.0	88.7	-
PG+EE (9K prog.)* [12]	88.6	79.7	89.7	79.1	92.6	96.0	-
PG+EE (18K prog.)* [12]	95.4	90.1	97.3	96.5	97.4	98.0	66.6
PG+EE (700K prog.)* [12]	96.9	92.7	97.1	98.7	98.1	98.9	-
CNN+LSTM+RN‡ [22]	95.5	90.1	97.8	93.6	97.9	97.1	-
CNN+GRU+FiLM [18]	97.7	94.3	99.1	96.8	99.1	99.1	75.9
CNN+GRU+FiLM [18]	97.6	94.3	99.3	93.4	99.3	99.3	-
MAC [8]	98.9	97.1	99.5	99.1	99.5	99.5	81.5
Stack-NMN (9 modules)†[6]	91.41	81.78	95.78	85.23	95.45	95.68	68.06
LNMN (9 modules)	89.88	84.28	93.74	89.63	89.64	94.84	66.35
LNMN (11 modules)	90.52	84.91	95.21	91.06	90.03	94.97	65.68
LNMN (14 modules)	90.42	84.79	95.52	90.52	89.73	95.26	65.86

Table 4.2. CLEVR and CLEVR-Humans Accuracy by baseline methods and our models. (*) denotes use of extra supervision through program labels. (‡) denotes training from raw pixels. † Accuracy figures for our implementation of Stack-NMN. The original paper reports 93% validation accuracy, test accuracy isn't reported.

CLEVR-Humans dataset. The latter is a dataset of challenging human-posed questions based on a much larger vocabulary on the same CLEVR images. The corresponding results are shown in Table 4.2 (see last column).

Model	Overall	Count	Exist	Compare number	Query attribute	Compare Attribute
Original setting ($T = 5, L = 5$, map_dim $= 384$)	89.78	84.54	93.46	88.70	89.59	94.87
Use hard-max for operation weights (for inference only) ($T = 5, L = 5$, map_dim $= 384$)	87.99	81.53	94.11	87.70	88.27	91.55
$T = 9, L = 9$, map_dim $= 256$	89.96	84.03	93.45	89.98	90.75	93.10
Concatenate all inputs followed by conv. layer	47.03	42.5	61.15	68.64	38.06	49.43

Table 4.3. Model Ablations for LNMN (CLEVR Validation set performance). The term 'map_dim' refers to the dimension of feature representation obtained at the input or output of each node of cell.

We use Adam [13] as the optimizer for the weight parameters with a learning rate of $1\mathrm{e}{-4}$, $(\beta_1, \beta_2) = (0.9, 0.999)$ and no weight decay. For the module network parameters, we use the same optimizer with a different learning rate $3\mathrm{e}{-4}$, $(\beta_1, \beta_2) = (0.5, 0.999)$ and a weight decay of $1\mathrm{e}{-3}$. The value of λ_{op} is set to $1\mathrm{e}{-1}$. We uploaded the code for implementation of our model in the supplementary material.

4.4.1. Results

The comparison of CLEVR overall accuracy shows that our model (LNMN (9 modules)) receives only a slight dip (1.53%) compared to the Stack-NMN model. We also experiment with other variants of our model in which we increase the number of *Answer* modules (LNMN (11 modules)) and/or the *Attention* modules (LNMN (14 modules)). The LNMN (11 modules) model performs better than the other two ablations (0.89% accuracy drop w.r.t. the Stack-NMN model). For the 'Count' and 'Compare Numbers' sub-category of questions, all of the 3 variants perform consistently better than Stack-NMN model. In case of CLEVR-Humans dataset, the accuracy drop

Module ID	Module type	**min**	**max**	**sum**	**product**	**choose_1**	**choose_2**
0	Attn. (3 input)	6.26E+04	2.73E+04	3.63E+04	1.07E+05	5.10E+04	1.56E+04
1	Attn. (3 input)	4.37E+04	1.80E+04	6.17E+04	1.43E+04	2.85E+04	1.66E+05
2	Attn. (3 input)	7.01E+04	3.28E+04	3.85E+04	1.13E+05	5.22E+04	1.47E+04
3	Attn. (3 input)	8.61E+03	6.15E+04	1.74E+04	1.80E+04	4.73E+04	2.96E+04
4	Attn. (4 input)	4.46E+04	3.19E+04	7.65E+04	1.71E+04	3.55E+04	2.06E+05
5	Attn. (4 input)	1.14E+05	5.62E+05	2.27E+05	8.55E+03	2.81E+04	1.84E+05
6	Ans. (3 input)	2.13E+06	4.25E+06	4.42E+06	8.32E+06	2.26E+06	4.89E+05
7	Ans. (4 input)	1.15E+05	5.78E+04	1.72E+05	5.19E+03	1.04E+05	4.51E+05

Table 4.4. Analysis of gradient attributions of α parameters corresponding to each module (LNMN (9 modules)), summed across all examples of CLEVR validation set.

Operator Name	Overall	Count	Exist	Compare number	Query attribute	Compare Attribute
min	86.64	77.98	86.79	87.89	88.77	93.10
max	45.54	35.92	55.25	63.66	40.52	51.83
sum	82.67	69.89	80.25	85.22	87.69	90.05
product	34.65	14.55	51.49	48.79	30.31	49.92
choose_1	89.74	84.24	93.81	89.02	89.59	94.67
choose_2	79.45	64.77	76.07	82.96	86.78	84.94
Original Model	89.88	84.28	93.74	89.63	89.64	94.84

Table 4.5. Analysis of performance drop with removing operators from a trained model (LNMN 9 modules) on CLEVR validation set.

is a modest 1.71%. This clearly shows that the modules learnt by our model (in terms of elementary arithmetic operations) perform approximately as well as the ones specified in the Stack-NMN model (that contains hand-designed modules which were tailor-made for the CLEVR dataset). The

results from the ablations in Table 4.3 show that a naive concatenation of all inputs to a module (or *cell*) results in a poor performance (around 47 $\%$). Thus, the structure we propose to fuse the inputs plays a key role in model performance. When we replace the $\boldsymbol{\alpha}$ vector for each node by a one-hot vector during inference, the drop in accuracy is only 1.79% which shows that the learned distribution over operation weights peaks over a specific operation which is desirable.

4.4.2. Measuring the role of individual arithmetic operators

Each module (*aka* cell) contains nodes which involve use of six elementary arithmetic operations (i.e. *min*, *max*, *sum*, *product*, *choose_1* and *choose_2*). We zero out the contribution to the node output for each of the arithmetic operations for all nodes in all modules and observe the degradation in the CLEVR validation accuracy †. The results of this study are shown in Table 4.5. The trend of overall accuracy shows that removing *max* and *product* operators results in maximum drop in overall accuracy ($\sim 50\%$). Other operators like *min*, *sum* and *choose_1* result in minimal drop in overall accuracy.

4.4.3. Measuring the sensitivity of modules

We use an attribution technique called Integrated Gradients [24] to study the impact of module structure parameters (denoted by $\left\{\alpha_i^{m,k}\right\}_{i=1}^{6}$ for k^{th} node of module m) on the probability distribution in the last layer of LNMN model. Let $F(\text{image}, \text{question}, \boldsymbol{\alpha})$ denote the function that assigns the probability corresponding to the correct answer index in the softmax distribution. Let $\mathcal{I}_j$ and $\boldsymbol{q}_j$ denote the (image, question) pairs for the j^{th} example. $\alpha_i^{m,k}$ denotes the module network parameter for the i^{th} operator in k^{th} node of module m. Then, the attribution of $[\alpha_1^m, \alpha_2^m, \alpha_3^m, \alpha_4^m, \alpha_5^m, \alpha_6^m]$ (summed across all nodes $k = 1,...,p$ for a particular module m) is defined as:

$$\begin{aligned}
\text{IG}(\alpha_i^m) &= \sum_{j=1}^{N}\sum_{k=1}^{p}\left[(\alpha_i^{m,k} - (\alpha_i^{m,k})')\times\int_{\xi=0}^{1}\frac{\partial F(\mathcal{I}_j, \boldsymbol{q}_j, (1-\xi)\times(\alpha_i^{m,k})' + \xi\times\alpha_i^{m,k})}{\partial\alpha_i^{m,k}}\right] \\
&= \sum_{j=1}^{N}\sum_{k=1}^{p}\left[\alpha_i^{m,k}\times\int_{\xi=0}^{1}\frac{\partial F(\mathcal{I}_j, \boldsymbol{q}_j, \xi\times\alpha_i^{m,k})}{\partial\alpha_i^{m,k}}\right]
\end{aligned}$$

Please note that attributions are defined relative to an uninformative input called the baseline. We use a vector of all zeros as the baseline (denoted by $(\alpha_i^{m,k})'$). Table 4.4 shows the results for this

†. The CLEVR test set ground truth answers are not public, so we use the validation set instead. However, Table 4.2 shows results for CLEVR test set (evaluated by the authors of CLEVR dataset).

experiment. The network parameters (α parameters) of the *Answer* modules have their attributions to the final probability around 1-2 orders of magnitudes higher than rest of the modules. The higher influence of Answer modules can be explained by the fact that they receive the memory features from the previous time-step and the classifier receives the memory features of the final time-step. The job of *Attention* modules is to utilize intermediate attention maps to produce new feature maps which are used as input by the *Answer* modules.

4.4.4. Visualization of module network parameters

In order to better interpret the individual contributions from each of the elementary operators to the modules, we plot them as color-maps for each type of module. The resulting visualizations are shown in Figure 4.3 for LNMN (11 modules). It is clear from the figure that the operation weights (or α' parameter) are approximately one-hot for each node. This is necessary in order to learn modules which act as composition of elementary operators on input feature maps rather than a mixture of operations at each node. The corresponding visualizations for LNMN (9 modules) and LNMN (14 modules) are given in Figure 4.C.1 and Figure 4.C.2 respectively (all of which are given in the Appendix section). The analytical expressions of modules learned by LNMN (11 modules) is shown in Table 4.6. The diversity of modules as given in their equations indicates that distinct modules emerge from training.

4.5. Related Work

Visual Reasoning Models: Among the end-to-end models for the task of visual reasoning, FiLM [18] uses Conditional Batch Normalization (CBN) [3, 4] to modulate the channels of input convolutional features in a residual block. [8] obtain the features by iteratively applying a Memory-Attention-Control (MAC) cell that learns to retrieve information from the image and aggregate the results into a recurrent memory. [22] constructs the feature representation by taking into account the relational interactions between objects of the image. With regards to the modular approaches, [2] propose to compose neural network modules (with shared parameters) for each input question based on layout predicted by syntactic parse of the question. [1] extends this approach to question-answering in a database domain. In [7], the layout prediction is relaxed by learning a layout policy with a sequence-to-sequence RNN. This layout policy is jointly trained along with

Module type	Module implementation
Attention (3 inputs)	$O(img, a, c_{txt}) = conv_2(choose_2(conv_1(\mathcal{I}), a) \odot W_1 c_{txt}) = conv_2(a \odot W_1 c_{txt})$
	$O(img, a, c_{txt}) = conv_2(choose_2(choose_1(conv_1(\mathcal{I}), a), W_1 c_{txt})) = conv_2(W_1 c_{txt})$
	$O(img, a, c_{txt}) = conv_2(choose_2(min(conv_1(\mathcal{I}), a), W_1 c_{txt})) = conv_2(W_1 c_{txt})$
	$O(img, a, c_{txt}) = conv_2(max(conv_1(\mathcal{I}), a) + W_1 c_{txt}))$
Attention (4 inputs)	$O(img, a_1, a_2, c_{txt}) = conv_2(choose_1(max(a_1, a_2), conv_1(\mathcal{I})) \odot W_1 c_{txt}))$ $= conv_2(max(a_1, a_2) \odot W_1 c_{txt})$
	$O(img, a_1, a_2, c_{txt}) = conv_2(max(choose_2(a_1, a_2), conv_1(\mathcal{I})) \odot W_1 c_{txt}))$ $= conv_2(max(a_2, conv_1(\mathcal{I})) \odot W_1 c_{txt}))$
Answer (3 inputs)	$O(img, a, c_{txt}) = W_2[\sum min(conv_1(\mathcal{I}), a) \odot W_1 c_{txt}, W_1 c_{txt}, f_{mem}]$
	$O(img, a, c_{txt}) = W_2[\sum min((conv_1(\mathcal{I}) \odot a), W_1 c_{txt}), W_1 c_{txt}, f_{mem}]$
Answer (4 inputs)	$O(img, a_1, a_2, c_{txt}) = W_2[\sum min((min(a_1, a_2) \odot conv_1(\mathcal{I})), W_1 c_{txt}), W_1 c_{txt}, f_{mem}]$
	$O(img, a_1, a_2, c_{txt}) = W_2[\sum((min(a_1, a_2) + conv_1(\mathcal{I})) \odot W_1 c_{txt}), W_1 c_{txt}, f_{mem}]$

Table 4.6. Analytical expression of modules learned by LNMN (11 modules). In the above equations, $\sum$ denotes sum over spatial dimensions of the feature tensor.

the parameters of modules. In [12], the modules are residual blocks (convolutional), they learn the program generator separately and then fine-tune it along with the modules.

Neural Architecture Search: Neural Architecture Search (NAS) is a technique to automatically learn the structure and connectivity of neural networks rather than training human-designed architectures. In [26], a recurrent neural network (RNN) based controller is used to predict the hyper-parameters of a CNN such as number of filters, stride, kernel size etc. They used REINFORCE [25] to train the controller with validation set accuracy as the reward signal. As an alternative to reinforcement learning, evolutionary algorithms [23] have been used to perform architecture search in [19, 17, 16, 20]. Recently, [15] proposed a differentiable approach to perform architecture search and reported success in discovering high-performance architectures for both image classification and language modeling. [14] proposes an EM style algorithm to learn black-box modules and their layout for image recognition and language modeling tasks.

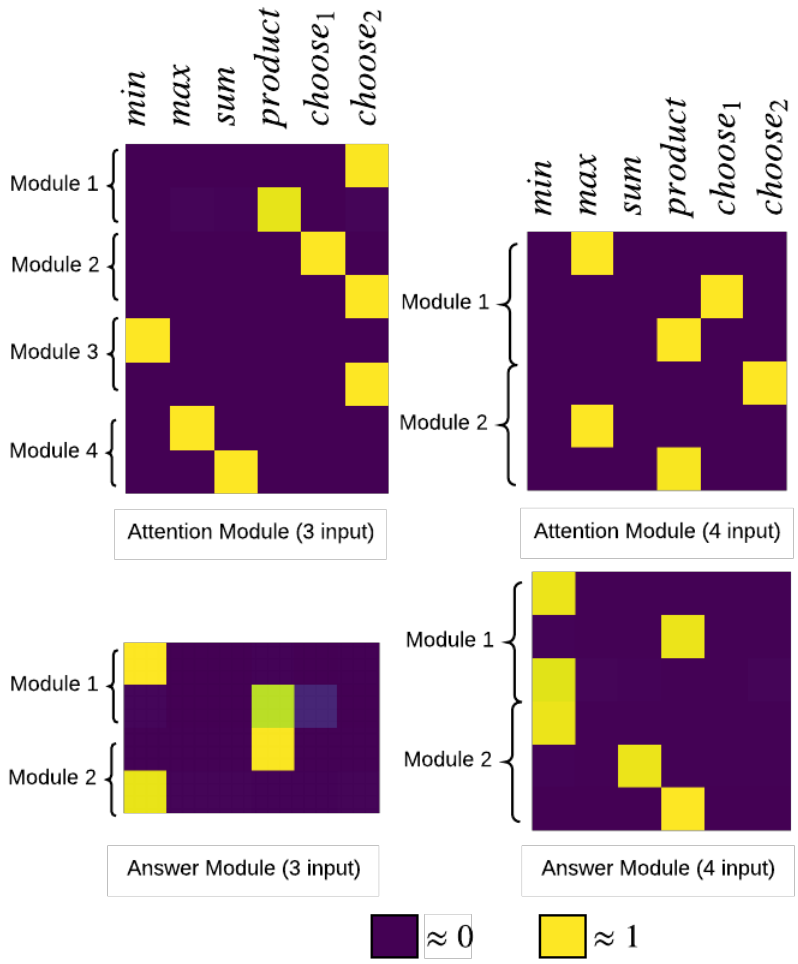

Figure 4.3. Visualization of module structure parameters (LNMN (11 modules)). For each module, each row denotes the $\boldsymbol{\alpha}' = \boldsymbol{\sigma}(\boldsymbol{\alpha})$ parameters of the corresponding node.

4.6. Conclusion

We have presented an approach to automatically learn the modules needed in a visual reasoning task. With this approach we obtain results comparable to an analogous model in which modules are hand-specified for a particular visual reasoning task. In addition, we present an extensive analysis of the degree to which each module influences the prediction function of the model, the effect of each arithmetic operation on overall accuracy and the analytical expressions of the learned modules. In the future, we would like to benchmark this generic learnable neural module network with various other visual reasoning and visual question answering tasks.

Bibliography for First Article (Chapter 4)

[1] Jacob Andreas et al. “Learning to compose neural networks for question answering”. In: *arXiv preprint arXiv:1601.01705* (2016).

[2] Jacob Andreas et al. “Neural module networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 39–48.

[3] Harm De Vries et al. “Modulating early visual processing by language”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6594–6604.

[4] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. “A learned representation for artistic style”. In: *Proc. of ICLR* (2017).

[5] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[6] Ronghang Hu et al. “Explainable neural computation via stack neural module networks”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 53–69.

[7] Ronghang Hu et al. “Learning to Reason: End-to-End Module Networks for Visual Question Answering”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[8] Drew A Hudson and Christopher D Manning. “Compositional attention networks for machine reasoning”. In: *arXiv preprint arXiv:1803.03067* (2018).

[9] Niall Hurley and Scott Rickard. “Comparing measures of sparsity”. In: *IEEE Transactions on Information Theory* 55.10 (2009), pp. 4723–4741.

[10] Allan Jabri, Armand Joulin, and Laurens van der Maaten. “Revisiting visual question answering baselines”. In: *European conference on computer vision*. Springer. 2016, pp. 727–739.

[11] Justin Johnson et al. “Clevr: A diagnostic dataset for compositional language and elementary visual reasoning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2901–2910.

[12] Justin Johnson et al. “Inferring and Executing Programs for Visual Reasoning”. In: *ICCV*. 2017.

[13] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[14] Louis Kirsch, Julius Kunze, and David Barber. “Modular Networks: Learning to Decompose Neural Computation”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 2414–2423.

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “Darts: Differentiable architecture search”. In: *arXiv preprint arXiv:1806.09055* (2018).

[16] Hanxiao Liu et al. “Hierarchical representations for efficient architecture search”. In: *arXiv preprint arXiv:1711.00436* (2017).

[17] Risto Miikkulainen et al. “Evolving deep neural networks”. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[18] Ethan Perez et al. “FiLM: Visual Reasoning with a General Conditioning Layer”. In: *CoRR* abs/1709.07871 (2017). arXiv: 1709.07871. URL: http://arxiv.org/abs/1709.07871.

[19] Esteban Real et al. “Large-scale evolution of image classifiers”. In: *arXiv preprint arXiv:1703.01041* (2017).

[20] Esteban Real et al. “Regularized evolution for image classifier architecture search”. In: *arXiv preprint arXiv:1802.01548* (2018).

[21] Anna Rohrbach et al. “Grounding of textual phrases in images by reconstruction”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 817–834.

[22] Adam Santoro et al. “A simple neural network module for relational reasoning”. In: *Advances in neural information processing systems*. 2017, pp. 4967–4976.

[23] Kenneth O Stanley. *Neuroevolution: A different kind of deep learning*. 2017.

[24] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic attribution for deep networks”. In: *arXiv preprint arXiv:1703.01365* (2017).

[25] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[26] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578* (2016).

Appendix for First Article (Chapter 4)

---

procedure RUN-MODULE($m, A, p, \boldsymbol{c_t}, \mathcal{I}$)

$a_1 \leftarrow \sum_{i=1}^{L} A_i \cdot p_i$ // Read from stack

$p \leftarrow$ 1D-conv$(p, [0, 0, 1])$ // decrement the stack pointer

if *no. of inputs==4* **then**

$a_2 \leftarrow \sum_{i=1}^{L} A_i \cdot p_i$ // Read from stack

$p \leftarrow$ 1D-conv$(p, [0, 0, 1])$ // decrement the stack pointer

$o_m \leftarrow m(\mathcal{I}, \boldsymbol{c_t}, a_1, a_2)$

end

else

$o_m \leftarrow m(\mathcal{I}, \boldsymbol{c_t}, a_1)$

end

$p \leftarrow$ 1D-conv$(p, [1, 0, 0])$ // increment the stack pointer

for $i = 1,...,L$ **do**

$A \leftarrow A \cdot (1 - p_i) + o_m \cdot p_i$ // Write to stack

end

return A, p

Algorithm 3: Operation of a module

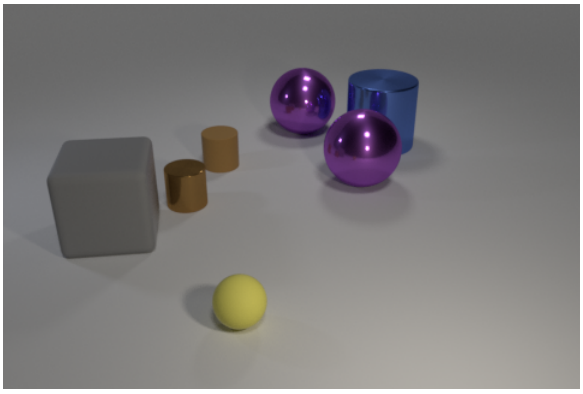

Figure 4..4. ***Q1****: What number of cylinders are gray objects or tiny brown matte objects?* ***A****: 1* ***Q2****: Is the number of brown cylinders in front of the brown matte cylinder less than the number of brown rubber cylinders?* ***A****: no*

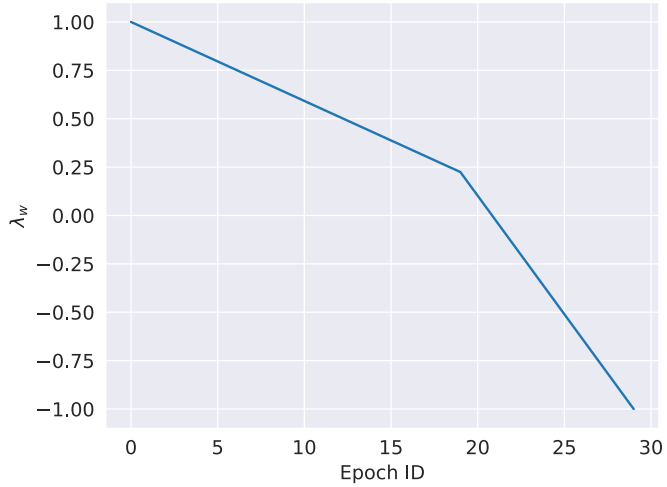

Figure 4..5. Plot of variation of λ_w with epochs.

4.A. Answer Module schematic diagrams

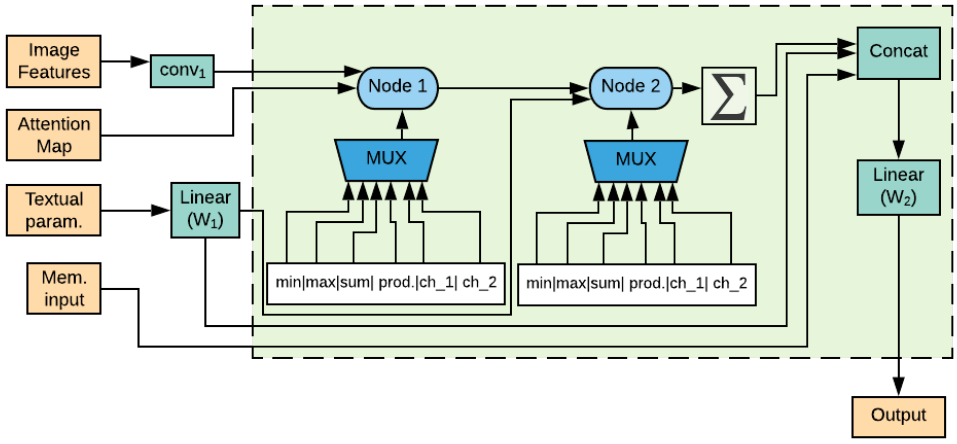

Figure 4.A.1. Answer Module schematic diagram (3 inputs)

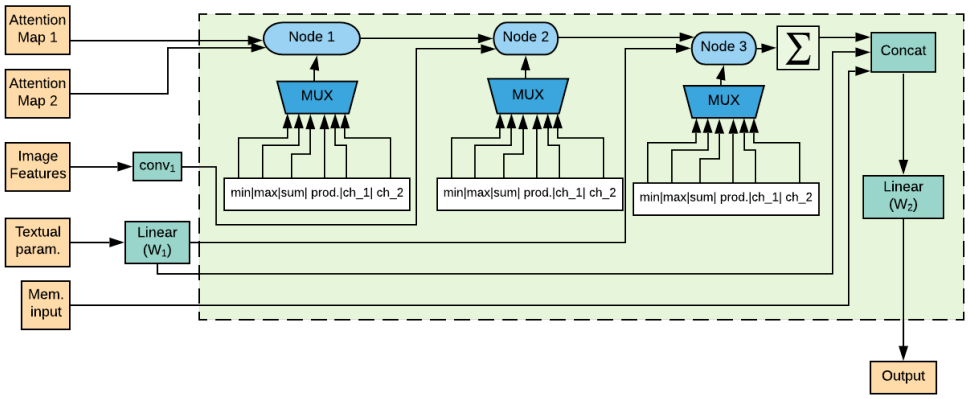

Figure 4.A.2. Answer Module schematic diagram (4 inputs)

4.B. Hand-crafted modules of Stack-NMN

module name	input attention	output type	implementation details (x: image feature map, c: textual parameter)
`Find`	(none)	attention	$a_{out} = \mathrm{conv}_2\left(\mathrm{conv}_1(x) \odot Wc\right)$
`Transform`	a	attention	$a_{out} = \mathrm{conv}_2\left(\mathrm{conv}_1(x) \odot W_1 \sum(a \odot x) \odot W_2 c\right)$
`And`	a_1, a_2	attention	$a_{out} = \mathrm{minimum}(a_1, a_2)$
`Or`	a_1, a_2	attention	$a_{out} = \mathrm{maximum}(a_1, a_2)$
`Filter`	a	attention	$a_{out} = \texttt{And}(a, \texttt{Find}())$, i.e. reusing `Find` and `And`
`Scene`	(none)	attention	$a_{out} = \mathrm{conv}_1(x)$
`Answer`	a	answer	$y = W_1^T \left(W_2 \sum(a \odot x) \odot W_3 c\right)$
`Compare`	a_1, a_2	answer	$y = W_1^T \left(W_2 \sum(a_1 \odot x) \odot W_3 \sum(a_2 \odot x) \odot W_4 c\right)$
`NoOp`	(none)	(none)	(does nothing)

Table 4.B.1. Neural modules used in [36]. The modules take image attention maps as inputs, and output either a new image attention a_{out} or a score vector y over all possible answers ($\odot$ is elementwise multiplication; $\sum$ is sum over spatial dimensions).

4.C. Visualization of module structure parameters

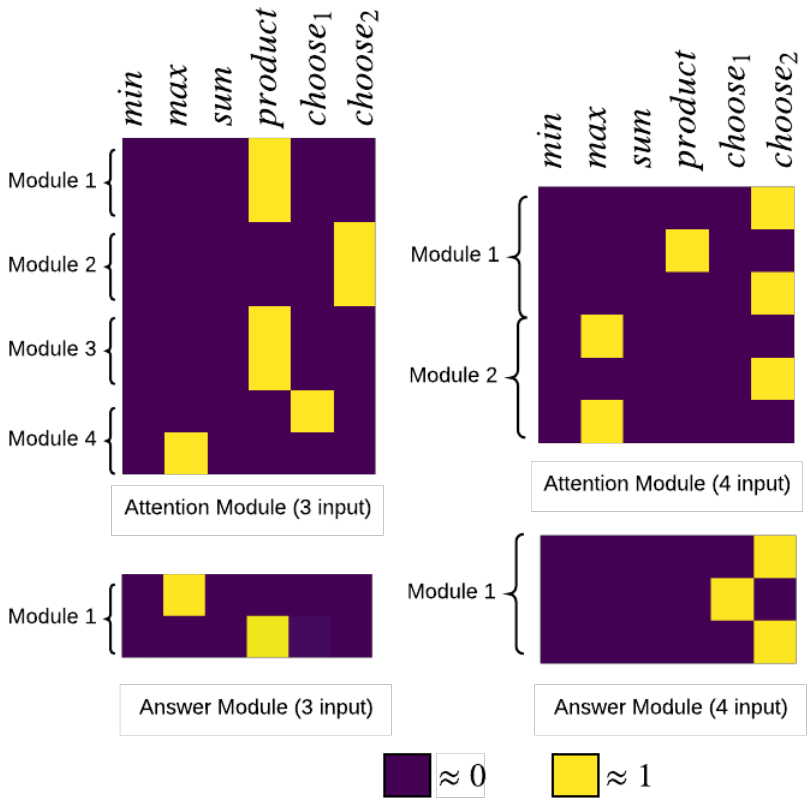

Figure 4.C.1. Visualization of module structure parameters (LNMN (9 modules)). For each module, each row denotes the $\boldsymbol{\alpha}' = \boldsymbol{\sigma}(\boldsymbol{\alpha})$ parameters of the corresponding node.

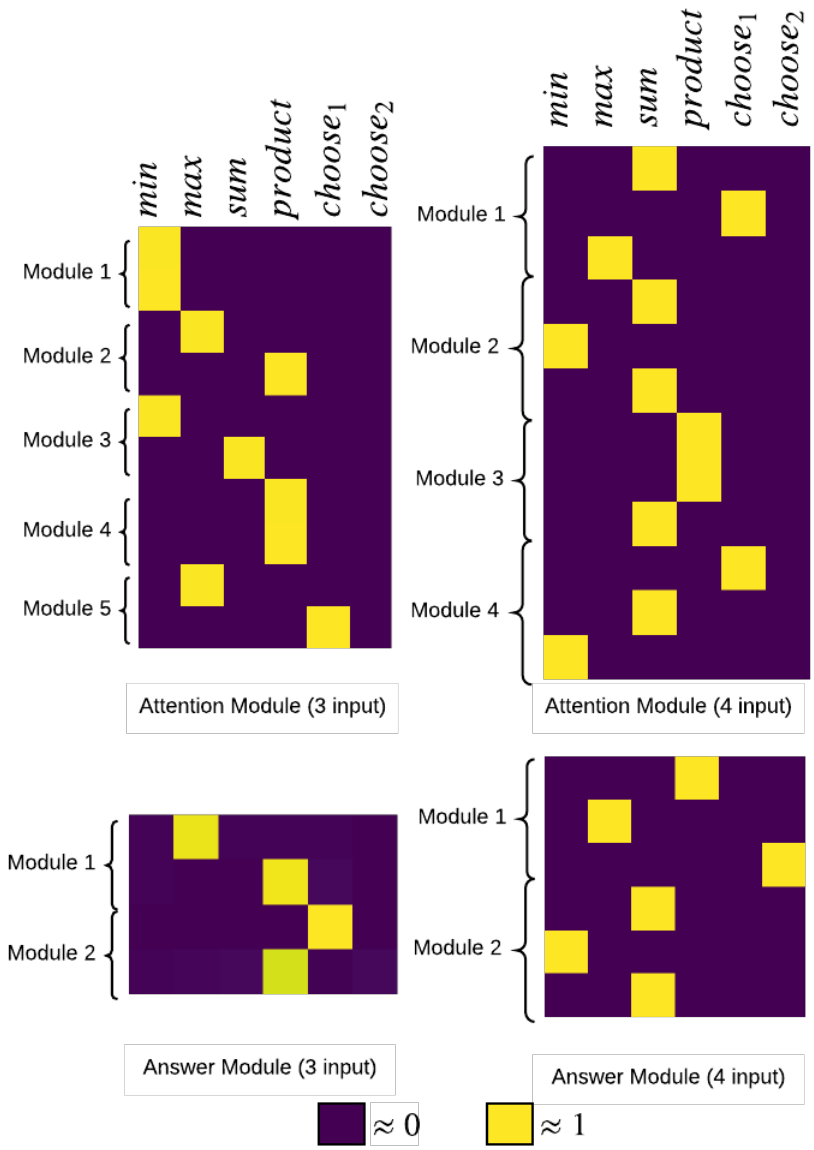

Figure 4.C.2. Visualization of module structure parameters (LNMN (14 modules)). For each module, each row denotes the $\boldsymbol{\alpha}' = \boldsymbol{\sigma}(\boldsymbol{\alpha})$ parameters of the corresponding node.

Chapter 5

Additional Experiments for *Learning Neural Modules for Visual Question Answering*

5.1. Experiments for a different variant of model structure

In this section, we introduce a different version of the cell structure. The cell structure shown in Figures 5.1 and Figure 5.2, for *3 input* and *4 input* cases respectively, is designed to be more expressive in terms of the type of modules it can learn. For n inputs to a module, it forms ${}^{n}C_{2}$ combinations of pairs of inputs and fuses them at one node each. The resultant features are then subject to linear combinations at two additional nodes to form the final output. All the modules in Table 4.B.1 can be realised as special cases of weight initializations of this cell structure (see Figures A.1, A.2, A.3, A.4, A.5, A.6, A.7 and A.8 in Appendix A for details). The new model which uses *v2* modules is referred to as *LNMN v2*. The version of cell structure in the article is now referred to as *LNMN v1*. The results for this model are shown in Table 5.1.

5.2. Experiments for sparsity at node level

The aim of these experiments is to induce sparsity in the operation vector ($\boldsymbol{\alpha} \in \mathbb{R}^k$ where $k = 6$). Our goal is to make the $\boldsymbol{\alpha}^{'}$ vector approximately one-hot towards the end of training. The different model variations for this task are described below:

(a) **Baseline Model**: The baseline model uses no special transformation function on the $\boldsymbol{\alpha}$ vector.

$$\boldsymbol{\alpha}^{'} = \boldsymbol{\alpha}$$

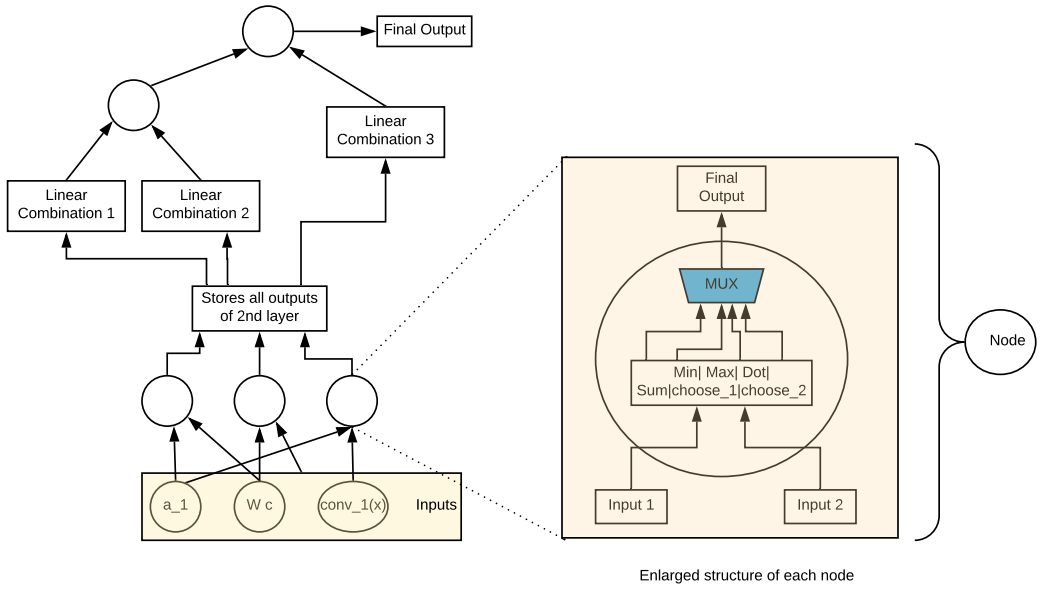

Figure 5.1. Attention Module schematic diagram version 2 (3 inputs).

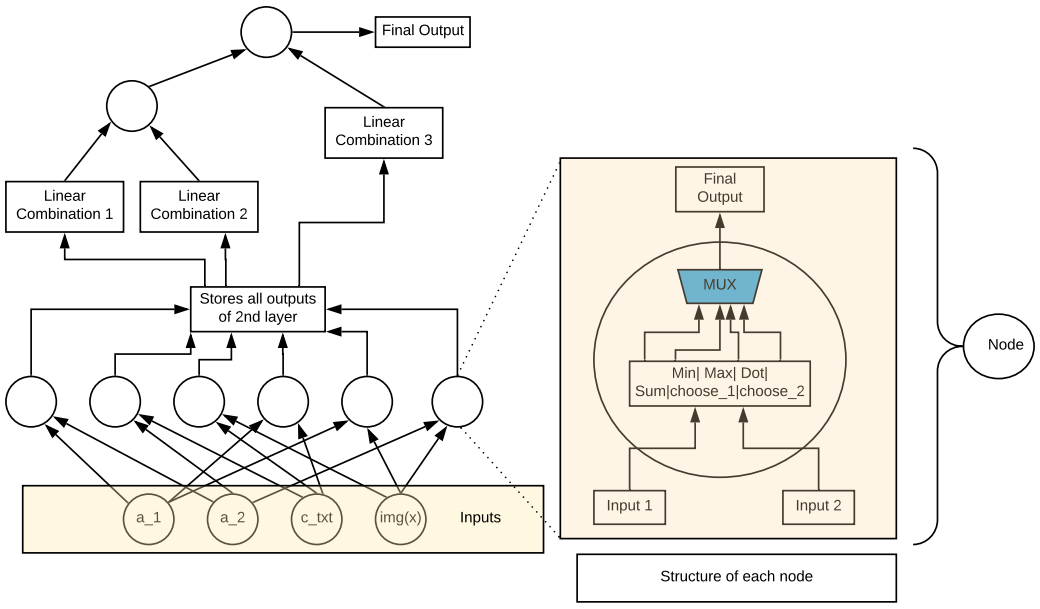

Figure 5.2. Attention Module schematic diagram version 2 (4 inputs).

$$O(\boldsymbol{x_1},\boldsymbol{x_2}) = \alpha_1^{'} * min(\boldsymbol{x_1}, \boldsymbol{x_2}) + \alpha_2^{'} * max(\boldsymbol{x_1}, \boldsymbol{x_2}) + \alpha_3^{'} * (\boldsymbol{x_1} + \boldsymbol{x_2}) + \alpha_4^{'} * (\boldsymbol{x_1} \odot \boldsymbol{x_2}) + \alpha_5^{'} * choose_1(\boldsymbol{x_1}, \boldsymbol{x_2}) + \alpha_6^{'} * choose_2(\boldsymbol{x_1}, \boldsymbol{x_2})$$

(b) **Straight-Through (ST) Gumbel Softmax with temperature=1.0**: In this variation, we use the Straight-Through (ST) Gumbel Softmax estimator [46] to make $\boldsymbol{\alpha}^{'}$ one-hot in the

forward pass whereas in the backward pass we use a differentiable approximation with τ (=1.0) as the temperature parameter.

$$\boldsymbol{\alpha}^{'} = \text{one-hot}(\text{argmax}_i(g_i + log(\alpha_i)))$$

$$g_i \sim \text{Gumbel(0,1)}$$

For the backward pass, we use the continuous and differentiable approximation to argmax (with a temperature parameter).

$$\boldsymbol{\alpha_i^{'}} = \frac{exp((log(\alpha_i) + g_i)/\tau)}{\sum_{j=1}^{k} exp((log(\alpha_j) + g_j)/\tau)}$$

Here, τ is fixed to 1.0

(c) **Straight-Through (ST) Gumbel Softmax with learned temp.**: This model ablation is same as **Straight-Through (ST) Gumbel Softmax with temperature=1.0** except for the fact that temperature is a learnt parameter, which receives gradients from the final loss term.

(d) **Straight-Through (ST) Gumbel Softmax with annealing**: It also uses the Gumbel (straight through) formulation and the value of τ varies linearly from 1.0 to 0.1 from epoch 1 to 21.
$\tau = max(0.1, (1.0 - 0.9 * \frac{n}{20}))$ for the n^{th} epoch where n is zero initially.

(e) **Softmax with annealing**: In this variation, we use softmax with temperature τ and anneal the value of τ in order to make $\boldsymbol{\alpha_i^{'}}$ more peaky towards the end of training.

$$\boldsymbol{\alpha_i^{'}} = \frac{exp(\alpha_i/\tau)}{\sum_{j=1}^{k} exp(\alpha_j/\tau)}$$

The value of τ varies linearly from 1.0 to 0.1 from epoch 1 to 21. $\tau = max(0.1, (1.0 - 0.9 * \frac{n}{20}))$ for the n^{th} epoch where n is zero initially.

(f) **Softmax normalization with annealing**: In this variation, we apply softmax on the L1 normalized version of $\boldsymbol{\alpha}$. The value of τ is annealed as in previous approaches.

$$\boldsymbol{\alpha_i^{'}} = \frac{exp(\frac{1}{\tau}\frac{\alpha_i}{\|\boldsymbol{\alpha}\|_1})}{\sum_{j=1}^{k} exp(\frac{1}{\tau}\frac{\alpha_j}{\|\boldsymbol{\alpha}\|_1})}$$

The rationale behind L1 normalization is that even if the absolute magnitude of $\boldsymbol{\alpha}$ decreases, the L1 normalization restores it and now the temperature parameter τ can give the desired level of sparsity.

(g) **Mix of soft and hard attention (sampling)**: In this variation, we sample from either soft attention (softmax) and hard attention (Gumbel softmax). The sampling probability varies such that soft attention is chosen most often initially and gradually it changes to hard attention towards the end of training.

$$\boldsymbol{\alpha_i'} = \pi * \left(\frac{exp(\alpha_i)}{\sum_{j=1}^{k} exp(\alpha_j)} \right) + (1 - \pi) * (\text{one-hot}(\text{argmax}_i(g_i + log(\alpha_i))))$$

$$\pi \sim \text{Bernoulli}(p)$$

For the n^{th} epoch, $p = \frac{1}{\sqrt{1+n}}$ (where n is zero initially).

(h) **Weighted Softmax with mix of soft and hard attention**: In this variation, we use a weighted mixture of soft attention (softmax) and hard attention (Gumbel softmax). The coefficient varies such that soft attention gets most weight initially and it gradually changes to hard attention towards the end of training.

$$\boldsymbol{\alpha_i'} = \pi * \left(\frac{exp(\alpha_i)}{\sum_{j=1}^{k} exp(\alpha_j)} \right) + (1 - \pi) * (\text{one-hot}(\text{argmax}_i(g_i + log(\alpha_i))))$$

$\pi = max(0, (1.0 - 0.9 * \frac{n}{20}))$ for the n^{th} epoch where n is zero initially.

(i) **Softmax with $\frac{l^2}{l^1}$ loss term**: In this variation, we apply softmax on $\boldsymbol{\alpha}$ and add an additional loss term ($\frac{l^2}{l^1}(\boldsymbol{\alpha})$) for sparsity.

$$\boldsymbol{\alpha_i'} = \boldsymbol{\sigma}(\boldsymbol{\alpha}) = \frac{exp(\alpha_i)}{\sum_{j=1}^{k} exp(\alpha_j)}$$

$$\frac{l^2}{l^1}(\boldsymbol{\alpha}) = \frac{\|\boldsymbol{\sigma}(\boldsymbol{\alpha})\|_2}{\|\boldsymbol{\sigma}(\boldsymbol{\alpha})\|_1}$$

5.3. Results

The results for the new version of cell structure (Table 5.1) show a significant reduction ($\sim 10\%$) in overall validation accuracy. This could be attributed to the presence of two different kinds of parameters present in the module structure - the node weights ($\boldsymbol{\alpha}$) and the linear combination weights. In our training procedure, both of these parameters are treated at par in terms

of learning rate and initialization. However, the linear combination weights may benefit from a different training strategy.

The experiments for inducing sparsity in node weights (Table 5.2) show that the model variation corresponding to *softmax with annealing* and *softmax with* $\frac{l^2}{l^1}$ *loss term* perform well in terms of overall accuracy. The model variations which uses ST Gumbel Softmax suffer from reduced accuracy mainly because of the randomness in α' parameters introduced by adding the Gumbel noise. The trend of mean entropy of the node weights (α') for these two variations and the baseline model is shown in Figure 5.1. Though the entropy decreases for *softmax with annealing*, it is not sufficient to make the node weights one-hot. A more fine-grained analysis shows that the absolute magnitude of the α scalars reduces during the course of training which negates the effect of reduced temperature, so that the weights don't turn one-hot. On the other hand, the $\frac{l^2}{l^1}$ loss term works perfectly in giving the desired level of sparsity throughout the course of training. Hence, we chose this option for our experiments in the article.

Model	**CLEVR Overall**	Count	Exist	Compare Numbers	Query Attribute	Compare Attribute
LNMN v2 (9 modules)	79.24	67.56	84.11	76.53	86.50	77.94
LNMN v1 (9 modules)	89.78	84.54	93.46	88.70	89.59	94.87

Table 5.1. CLEVR Accuracy (val. set) for different versions of LNMN models

Model	**CLEVR Overall**	Count	Exist	Compare Numbers	Query Attribute	Compare Attribute
Baseline	89.36	84.23	95.02	84.65	89.36	94.27
Softmax with annealing	87.05	79.81	91.94	83.32	88.71	91.50
Softmax normalization with annealing	55.84	47.65	69.26	70.09	54.97	51.22
ST Gumbel Softmax (T=1.0)	51.35	45.63	65.82	68.51	45.84	50.47
ST Gumbel Softmax with temp. learned	51.42	46.69	65.80	68.69	45.48	50.11
ST Gumbel Softmax with annealing	64.17	54.42	72.33	71.18	69.24	57.33
Mix of soft and hard attention	54.93	49.03	67.16	67.85	52.57	51.79
Softmax normalization with annealing	67.57	52.32	71.04	69.20	82.64	54.27
Softmax with $\frac{l^2}{l^1}$ loss term	**89.78**	84.54	93.46	88.70	89.59	94.87

Table 5.2. CLEVR Accuracy (val. set) for different versions of LNMN models for node sparsity

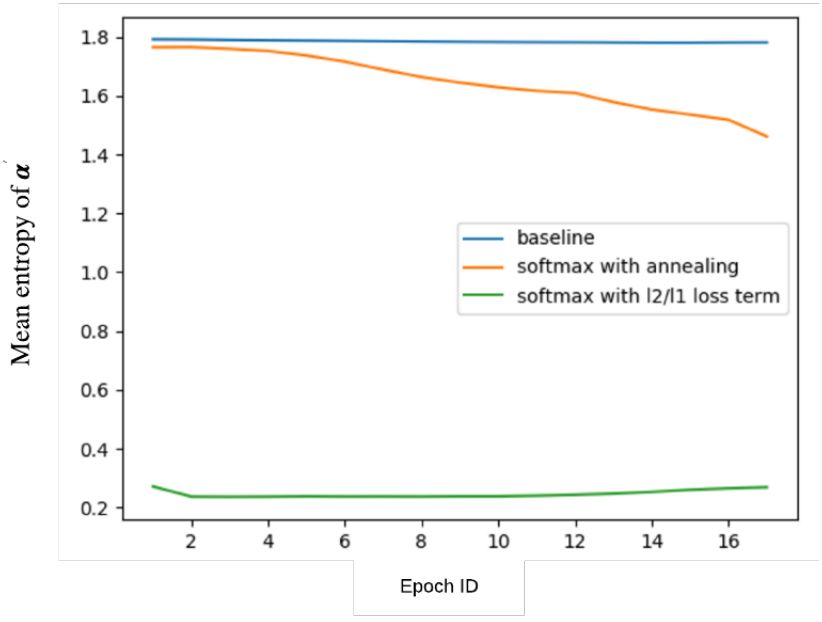

Figure 5.1. Comparison of variation of mean entropy of α'

Chapter 6

Learning a recursive network whose structure aligns with natural language

6.1. Introduction and Related Work

Most visual question answering models use sequence based recurrent models like LSTM [33] or GRU [13] to obtain a feature representation for the question [63, 23]. However, [87] shows that a naive bag of words or a custom feature representation which aggregates word embeddings trained on language modeling tasks, can outperform LSTM based models on VQA. [76] shows the utility of language modeling for zero-shot transfer to natural language processing tasks like reading comprehension, summarization and question-answering. In this ongoing work, our goal is to learn a language model which trains in an unsupervised manner and also captures long-term dependencies effectively. This has the potential to encode syntactic features of question which can improve the performance of VQA models.

The use of LSTM [33] has become ubiquitous in many natural language processing tasks like machine translation [13, 6], image caption generation [104], speech recognition [28], etc. [58] investigates the ability of LSTM to capture the structural dependencies in sentences by proposing a Number Prediction task, which involves predicting whether the verb corresponding to the main subject is singular or plural. Their results show that although LSTMs perform fairly well for sentences from Wikipedia corpus, the baseline LSTM model which takes as input only the nouns of the sentence (devoid of any syntactic structure) also achieves 95% accuracy. They further show that increase in the number of attractors (nouns intervening between the subject and the verb) resulted in some deterioration in accuracy (17.6% error rate). This shows that sequential models like LSTM can capture structure-sensitive dependencies to a certain degree. [56] shows that LSTM

can perform quite well on number agreement task given enough capacity. [56] shows that language models which use syntactic structure to determine model structure like [20] perform much better than LSTMs on the number agreement task (even with multiple attractors). This is supported by the fact that RNNGs [20] use RNNs conditioned on syntactic clues to determine actions for transition-based parsing.

Recursive models like [99, 90] use syntactic tree structure from external parsers to produce sentence representations and show improved results on sentiment classification and semantic relatedness tasks. Latent tree learning models [112, 62, 14] learn a parse tree for the sentences using indirect supervision from a downstream task like sentiment classification or natural language inference. RRNet [44] creates a tree on-the-fly while reading the input sentence and trains it using either explicit supervision or policy gradient. However, the grammar learned by these models is not consistent with recognized syntactic principles [107].

In this work, our aim is to learn a parser for natural language in an unsupervised manner which can also capture long-term dependencies effectively. We use an Parsing-Reading-Predict Network (PRPN) [86] as the unsupervised parser in conjunction with a model that can leverage the intermediate syntactic structure to produce the hidden representation at the next time-step. We choose TreeLSTM [99] for this purpose.

6.2. Overview of PRPN model

The PRPN model [86] induces a tree structure for the input set of tokens $x_0, \cdots, x_n$. Let $\{y_i\}$ denote the set of non-leaf nodes of the latent tree structure learned by the model. The node y_j represents the meaning of all the leaf nodes in its subtree $x_{l(y_j)}, \cdots, x_{r(y_j)}$, where $l(\cdot)$ and $r(\cdot)$ denote the leftmost descendent leaf and rightmost descendent leaf respectively. Let the latent variable l_t represent the structural context of x_t. If x_t is not the left-most child of any subtree, then l_t is x_t's left-most sibling. Otherwise, if x_t is the left-most child of subtree rooted at y_i, then l_t is the left-most child in the subtree rooted at the left-most sibling of y_i. The gates which control skip connections are defined as:

$$g_i^t = \begin{cases} 1, & l_t \leq i < t \\ 0, & 0 < i < l_t \end{cases}$$

Overall, the PRPN model is composed of the Parsing Network, Reading Network and Predict Network. The function of *Parsing Network* is to infer the tree structure (represented by $\{g_i^t\}$) using

the syntactic distance. The syntactic distance is output by applying a convolutional kernel over the embeddings of a set of consecutive previous tokens. The function of *Reading Network* is to summarize the previous recurrent states using structured attention and then perform a recurrent update of the hidden and cell states. The function of *Predict Network* is to predict the probability distribution of the next word using the hidden states and the gates $\{g_i^t\}$.

Next, we discuss the 3 sub-networks which represent the components of this model, in detail.

(a) **Parsing Network** The latent variable l_t is modelled as a stick-breaking process [22, 84].

$$p(l_t = i|x_0, ...,x_t) = (1 - \alpha_i^t) \prod_{j=i+1}^{t-1} \alpha_j^t$$

Let the latent variable d_i denote the syntactic distance between adjacent words (x_{i-1}, x_i). α_i^t is parameterized as:

$$\alpha_j^t = \frac{\text{hardtanh}\left((d_t - d_j) \cdot \tau\right) + 1}{2} \tag{6.2.1}$$

where $\text{hardtanh}(x) = \max(-1, \min(1, x))$ and τ is a temperature parameter. The expectation of gate value is given as (see [86] for details):

$$g_i^t = \mathbf{P}(l_t \leq i) = \prod_{j=i+1}^{t-1} \alpha_j^t \tag{6.2.2}$$

The syntactic distance d_i between a token e_i and the previous token e_{i-1} is calculated by applying a convolutional kernel over the embeddings of the corresponding tokens:

$$\boldsymbol{h}_i = \text{ReLU}(\boldsymbol{W_c} \begin{bmatrix} \boldsymbol{e}_{i-L} \\ \boldsymbol{e}_{i-L+1} \\ \cdots \\ \boldsymbol{e}_i \end{bmatrix} + \boldsymbol{b_c})$$

$$\boldsymbol{d}_i = \text{ReLU}\left(\boldsymbol{W_d}\boldsymbol{h}_i + \boldsymbol{b_d}\right)$$

Here, $\boldsymbol{W_c}$, $\boldsymbol{b_c}$, $\boldsymbol{W_d}$ and $\boldsymbol{b_d}$ are convolutional kernel parameters. L denotes the lookback range i.e. how many previous tokens the model should consider while calculating the syntactic distance. The tree structure can be inferred from the syntactic distances (see Appendix C in [86] for details).

(b) **Reading Network** The reading network uses structured attention [86, 12] to model the dependency relations. The structured intra-attention weight $\tilde{s}_i^t$ is calculated as:

$$\boldsymbol{k}_t = \boldsymbol{W_h}\boldsymbol{h}_{t-1} + \boldsymbol{W_x}\boldsymbol{x}_t$$

$$\tilde{s}_i^t = \mathrm{softmax}(\frac{\boldsymbol{h}_i \boldsymbol{k}_t^{\mathrm{T}}}{\sqrt{\delta_k}})$$

Here, δ_k denotes the dimension of the hidden state $\boldsymbol{h}_i$. The structured intra-attention weight $\tilde{s}_i^t$ is then modulated by the gate g_i^t:

$$s_i^t = \frac{g_i^t \tilde{s}_i^t}{\sum_i g_i^t}$$

The Reading Network maintains a list of the last N_m hidden states and cell states similar to [12]. The memory state $\boldsymbol{m}_i$ is a tuple of the hidden state and cell state $(\boldsymbol{h}_i, \boldsymbol{c}_i)$). A weighted sum of the hidden states corresponding to previous timesteps is computed using the structured intra-attention weights s_i^t as follows:

$$\begin{bmatrix} \tilde{\boldsymbol{h}}_t \\ \tilde{\boldsymbol{c}}_t \end{bmatrix} = \sum_{i=1}^{t-1} s_i^t \cdot \boldsymbol{m}_i = \sum_{i=1}^{t-1} s_i^t \cdot \begin{bmatrix} \boldsymbol{h}_i \\ \boldsymbol{c}_i \end{bmatrix}$$

The final hidden states for the reading network $(\boldsymbol{h}_t, \boldsymbol{c}_t)$ are obtained from an LSTM [33] update on $\boldsymbol{x}_t$ with $(\tilde{\boldsymbol{h}}_t, \tilde{\boldsymbol{c}}_t)$ as the initializations for the hidden and cell states respectively.

$$\begin{bmatrix} \boldsymbol{i}_t \\ \boldsymbol{f}_t \\ \boldsymbol{o}_t \\ \hat{\boldsymbol{c}}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \boldsymbol{W} \cdot [\tilde{\boldsymbol{h}}_t,\ \boldsymbol{x}_t]$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \tilde{\boldsymbol{c}}_t + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

(c) **Predict Network** The function of the Predict network is to predict the probability of next token x_{t+1}. An approximation of the syntactic distance d_{t+1} is done by applying a convolutional kernel over the hidden state $\boldsymbol{h}_t$.

$$\boldsymbol{d}'_{t+1} = \mathrm{ReLU}(\boldsymbol{W}'_{\boldsymbol{d}}\boldsymbol{h}_t + \boldsymbol{b}'_{\boldsymbol{d}})$$

The values of $\{\alpha^{t+1}\}$ and $\{g_i^{t+1}\}$ are then calculated using Eqn.(6.2.1) and Eqn.(6.2.2) respectively. The probability of the next token conditioned on previous tokens is given by:

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_0,...,\boldsymbol{x}_t) \approx p(\boldsymbol{x}_{t+1}; f(\boldsymbol{m}_0,...,\boldsymbol{m}_t,g_0^{t+1},...,g_t^{t+1}))$$

Here, the function f is modelled as:

$$f(\boldsymbol{m}_0,...,\boldsymbol{m}_t,g_0^{t+1},...,g_t^{t+1}) = \hat{f}([\boldsymbol{h}_{l:t-1},\boldsymbol{h}_t]) \tag{6.2.3}$$

where $\boldsymbol{h}_{l:t-1}$ is the collection of hidden states obtained from the structured attention in the reading network and $\hat{f}$ is an MLP.

6.3. Proposed Model

PRPN [86] learns an unsupervised parser for natural language sentences in using a soft tree based learning scheme. TreeLSTM [99] uses the tree structures obtained from external parsers to learn syntax-aware representations for sentences. In our proposed model, at each intermediate time-step of PRPN model, the TreeLSTM model uses the intermediate tree structure (obtained from the $\{d_i\}$) to compute a hidden representation for the next time-step.

Let $(\boldsymbol{h}_t, \boldsymbol{c}_t)$ denote the final hidden states obtained from the reading network. Let $\boldsymbol{h}_t^{tree}$ denote the hidden representation at root of intermediate parse tree obtained from the TreeLSTM model. Our proposed model uses a weighted combination of $\boldsymbol{h}_t$ and $\boldsymbol{h}_t^{tree}$ to get the hidden representation which is utilized by the *predict* step in Eqn.(6.2.3). The rationale behind this modification is to enable the direct use of syntactic structure to predict the next token as long-term dependencies are better captured if structure explicitly exists in the model.

$$\boldsymbol{h}_t^{'} = \alpha * \boldsymbol{h}_t + (1-\alpha) * \boldsymbol{h}_t^{tree}$$

Here, α is a hyper-parameter which is annealed from 1.0 to 0.5 during the first 10 epochs. In the beginning of training, the contribution of the TreeLSTM's hidden state is small but it is gradually increased so that both models have equal influence on the prediction of the next token.

6.4. Experiments

We test our model on word-level language modeling task on Penn Treebank dataset [70]. A pre-trained PRPN model is used to initialise the common weights of the proposed model. The

corresponding results are shown in Table 6.1. The training curve showing the trend of validation perplexity is given in Figure 6.1.

Model	Perplexity
RNN-LDA + KN-5 + cache [69]	92.0
LSTM [114]	78.4
Variational LSTM [51]	78.9
CharCNN [51]	78.9
Pointer Sentinel-LSTM [67]	70.9
LSTM + continuous cache pointer [27]	72.1
Variational LSTM (tied) + augmented loss [43]	68.5
Variational RHN (tied) [119]	65.4
NAS Cell (tied) [120]	62.4
4-layer skip connection LSTM (tied) [66]	**58.3**
PRPN [86]	61.98
PRPN + TreeLSTM(Ours)	59.88

Table 6.1. Perplexity on the Penn Treebank test set

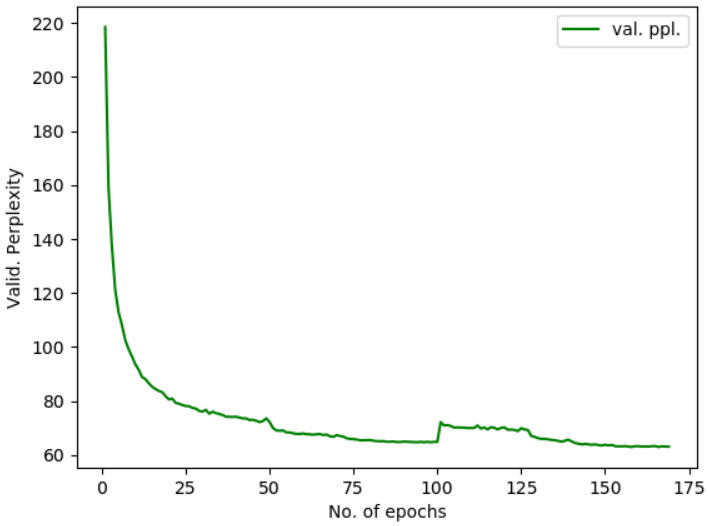

Figure 6.1. Training curve for PRPN + TreeLSTM model

6.5. Conclusion

We presented a model for unsupervised parsing which seeks to take advantage of explicit discrete structure to better model long-term dependencies. Our experiments show that it leads to improvements over the PRPN baseline in a word-level language modeling task. The future work will include analysis of performance of this model on other datasets for language modeling and other natural language tasks which benefit could from the syntactic structure present in sentences and incorporating this language model to construct better question feature representations for VQA models.

General Conclusion

In this thesis, we study different aspects of Visual Question Answering (VQA) which has gained considerable traction in the machine learning community recently. In Chapter 2, we present a method to optimize the computation in a CNN while training a VQA model. The proposed approach leads to significant savings in no. of floating point operations (or FLOPS) with a minimal degradation in performance.

In Chapter 4, we address the issue of learning the internal structure of modules used in Neural Module Networks (NMNs) and its variants. This is important because hand-specifying modules can work well for certain datasets for which the attributes of different objects and the relations between them are already known. Our proposed algorithm is a bi-level optimization algorithm in which the module structure and module parameters are optimized alternately. Taking a cue from the the hand-crafted modules, we propose a general module structure which makes use of elementary arithmetic operations to build complex modules. Our model is able to perform comparably with the one using hand-designed modules while learning the structure of these modules at the same time. Future work in this direction would involve exploring the transferability of these modules to other tasks that involve visual reasoning.

The question feature representation plays a key role in a VQA system. However, most works in literature make use of LSTM/GRU to encode the question features which is not always sufficient to model the syntactic structure of the question. We try to address this issue by learning a better language model so as to learn better question feature representations. In Chapter 6, we build upon an existing language model (PRPN) and augment it with a TreeLTSM in order to explicitly incorporate syntactic structure in its prediction step. Our results shows that it helps the model improve over the PRPN baseline on a word-level language modeling task.

Bibliography

[1] Ferran Alet, Tomás Lozano-Pérez, and Leslie P Kaelbling. "Modular meta-learning". In: *arXiv preprint arXiv:1806.10166* (2018).

[2] Jacob Andreas and Dan Klein. "How much do word embeddings encode about syntax?" In: *ACL (2)*. The Association for Computer Linguistics, 2014, pp. 822–827.

[3] Jacob Andreas et al. "Neural module networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 39–48.

[4] Stanislaw Antol et al. "VQA: Visual Question Answering". In: *International Conference on Computer Vision (ICCV)*. 2015.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv e-prints* abs/1409.0473 (Sept. 2014). URL: https://arxiv.org/abs/1409.0473.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2015).

[7] Emmanuel Bengio et al. "Conditional computation in neural networks for faster models". In: *arXiv preprint arXiv:1511.06297* (2015).

[8] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation". In: *arXiv preprint arXiv:1308.3432* (2013).

[9] Tolga Bolukbasi et al. "Adaptive neural networks for efficient inference". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 527–536.

[10] Arthur W. Burks, Don W. Warren, and Jesse B. Wright. "An Analysis of a Logical Machine Using Parenthesis-Free Notation". In: *Mathematical Tables and Other Aids to Computation*

8.46 (1954), pp. 53–57. ISSN: 08916837. URL: http://www.jstor.org/stable/2001990.

[11] Kan Chen et al. "ABC-CNN: An Attention Based Convolutional Neural Network for Visual Question Answering". In: *CoRR* abs/1511.05960 (2015).

[12] Jianpeng Cheng, Li Dong, and Mirella Lapata. "Long Short-Term Memory-Networks for Machine Reading". In: *EMNLP*. 2016.

[13] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[14] Jihun Choi, Kang Min Yoo, and Sang-goo Lee. "Learning to compose task-specific tree structures". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[15] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[16] Andrew Davis and Itamar Arel. "Low-rank approximations for conditional feedforward computation in deep neural networks". In: *arXiv preprint arXiv:1312.4461* (2013).

[17] Harm De Vries et al. "Modulating early visual processing by language". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6594–6604.

[18] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[19] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. "A learned representation for artistic style". In: *Proc. of ICLR* (2017).

[20] Chris Dyer et al. "Recurrent neural network grammars". In: *arXiv preprint arXiv:1602.07776* (2016).

[21] Reza Ebrahimpour et al. "View-independent face recognition with mixture of experts". In: *Neurocomputing* 71.4-6 (2008), pp. 1103–1107.

[22] Thomas S. Ferguson. "A Bayesian Analysis of Some Nonparametric Problems". In: 1973.

[23] Akira Fukui et al. "Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding". In: *EMNLP*. 2016.

[24] Yang Gao et al. "Compact Bilinear Pooling". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 317–326.

[25] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *CoRR* abs/1406.2661 (2014).

[26] Yash Goyal et al. “Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[27] Edouard Grave, Armand Joulin, and Nicolas Usunier. “Improving neural language models with a continuous cache”. In: *arXiv preprint arXiv:1612.04426* (2016).

[28] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. “Speech recognition with deep recurrent neural networks”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), pp. 6645–6649.

[29] Nitish Gupta and Mike Lewis. “Neural Compositional Denotational Semantics for Question Answering”. In: *EMNLP*. 2018.

[30] David G. Hays. “Automatic language-data processing”. In: 1962.

[31] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[32] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[33] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[34] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[35] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-excitation networks”. In: *arXiv preprint arXiv:1709.01507* 7 (2017).

[36] Ronghang Hu et al. “Explainable neural computation via stack neural module networks”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 53–69.

[37] Ronghang Hu et al. “Learning to Reason: End-to-End Module Networks for Visual Question Answering”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[38] Gao Huang et al. “Condensenet: An efficient densenet using learned group convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2752–2761.

[39] Gao Huang et al. “Multi-scale dense networks for resource efficient image classification”. In: *arXiv preprint arXiv:1703.09844* (2017).

[40] Drew A Hudson and Christopher D Manning. “Compositional attention networks for machine reasoning”. In: *arXiv preprint arXiv:1803.03067* (2018).

[41] Drew A. Hudson and Christopher D. Manning. “GQA: a new dataset for compositional question answering over real-world images”. In: *CoRR* abs/1902.09506 (2019).

[42] Ilija Ilievski and Jiashi Feng. “Multimodal Learning and Reasoning for Visual Question Answering”. In: *NIPS*. 2017.

[43] Hakan Inan, Khashayar Khosravi, and Richard Socher. “Tying word vectors and word classifiers: A loss framework for language modeling”. In: *arXiv preprint arXiv:1611.01462* (2016).

[44] Athul Paul Jacob et al. “Learning hierarchical structures on-the-fly with a recurrent-recursive model for sequences”. In: *Proceedings of The Third Workshop on Representation Learning for NLP*. 2018, pp. 154–158.

[45] Robert A Jacobs et al. “Adaptive mixtures of local experts.” In: *Neural computation* 3.1 (1991), pp. 79–87.

[46] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical Reparameterization with Gumbel-Softmax”. In: *CoRR* abs/1611.01144 (2017).

[47] Jason Jo, Vikas Verma, and Yoshua Bengio. “Modularity Matters: Learning Invariant Relational Reasoning Tasks”. In: *arXiv preprint arXiv:1806.06765* (2018).

[48] Justin Johnson et al. “Clevr: A diagnostic dataset for compositional language and elementary visual reasoning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2901–2910.

[49] Tadao Kasami. “An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages”. In: 1965.

[50] Seung Wook Kim, Makarand Tapaswi, and Sanja Fidler. “Visual Reasoning by Progressive Module Networks”. In: (2018).

[51] Yoon Kim et al. “Character-aware neural language models”. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[52] Louis Kirsch, Julius Kunze, and David Barber. "Modular Networks: Learning to Decompose Neural Computation". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2414–2423.

[53] Dan Klein and Christopher D. Manning. "Accurate Unlexicalized Parsing". In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*. ACL '03. Sapporo, Japan: Association for Computational Linguistics, 2003, pp. 423–430. DOI: 10.3115/1075096.1075150. URL: https://doi.org/10.3115/1075096.1075150.

[54] Ranjay Krishna et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations". In: *International Journal of Computer Vision* 123.1 (2017), pp. 32–73.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[56] Adhiguna Kuncoro et al. "LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 1426–1436.

[57] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[58] Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. "Assessing the ability of LSTMs to learn syntax-sensitive dependencies". In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 521–535.

[59] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search". In: *arXiv preprint arXiv:1806.09055* (2018).

[60] Hanxiao Liu et al. "Hierarchical representations for efficient architecture search". In: *arXiv preprint arXiv:1711.00436* (2017).

[61] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[62] Jean Maillard, Stephen Clark, and Dani Yogatama. "Jointly learning sentence embeddings and syntax with unsupervised tree-lstms". In: *arXiv preprint arXiv:1705.09189* (2017).

[63] Mateusz Malinowski and Mario Fritz. “A multi-world approach to question answering about real-world scenes based on uncertain input”. In: *Advances in neural information processing systems*. 2014, pp. 1682–1690.

[64] Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. “Ask your neurons: A neural-based approach to answering questions about images”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1–9.

[65] David Mascharka et al. “Transparency by Design: Closing the Gap Between Performance and Interpretability in Visual Reasoning”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 4942–4950.

[66] Gábor Melis, Chris Dyer, and Phil Blunsom. “On the state of the art of evaluation in neural language models”. In: *arXiv preprint arXiv:1707.05589* (2017).

[67] Stephen Merity et al. “Pointer sentinel mixture models”. In: *arXiv preprint arXiv:1609.07843* (2016).

[68] Risto Miikkulainen et al. “Evolving deep neural networks”. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[69] Tomas Mikolov and Geoffrey Zweig. “Context dependent recurrent neural network language model”. In: *2012 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2012, pp. 234–239.

[70] Tomas Mikolov et al. “SUBWORD LANGUAGE MODELING WITH NEURAL NETWORKS”. In: 2011.

[71] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. “Indoor Segmentation and Support Inference from RGBD Images”. In: *ECCV*. 2012.

[72] Minh Ha Nguyen. *Cooperative coevolutionary mixture of experts: a neuro ensemble approach for automatic decomposition of classification problems*. University of New South Wales, Australian Defence Force Academy, School of . . ., 2006.

[73] Hyeonwoo Noh, Paul Hongsuck Seo, and Bohyung Han. “Image Question Answering using Convolutional Neural Network with Dynamic Parameter Prediction”. In: *arXiv preprint arXiv:1511.05756* (2015).

[74] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[75] Ethan Perez et al. “Film: Visual reasoning with a general conditioning layer”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[76] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *URL https://openai. com/blog/better-language-models* (2019).

[77] Esteban Real et al. “Large-scale evolution of image classifiers”. In: *arXiv preprint arXiv:1703.01041* (2017).

[78] Esteban Real et al. “Regularized evolution for image classifier architecture search”. In: *arXiv preprint arXiv:1802.01548* (2018).

[79] Mengye Ren, Ryan Kiros, and Richard Zemel. “Image question answering: A visual semantic embedding model and a new dataset”. In: *Proc. Advances in Neural Inf. Process. Syst* 1.2 (2015), p. 5.

[80] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[81] Zellig S. Harris. “Distributional Structure”. In: *Word* 10 (Aug. 1954), pp. 146–162. DOI: 10.1007/978-94-009-8467-7_1.

[82] Adam Santoro et al. “A simple neural network module for relational reasoning”. In: *Advances in neural information processing systems*. 2017, pp. 4967–4976.

[83] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[84] Jayaram Sethuraman. “A Constructive Definition of the Dirichlet Prior”. In: *Statistica Sinica* 4 (Jan. 1994), pp. 639–650.

[85] Noam Shazeer et al. “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer”. In: *arXiv preprint arXiv:1701.06538* (2017).

[86] Yikang Shen et al. “Neural language modeling by jointly learning syntax and lexicon”. In: *arXiv preprint arXiv:1711.02013* (2017).

[87] Kevin J. Shih, Saurabh Singh, and Derek Hoiem. “Where to Look: Focus Regions for Visual Question Answering”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 4613–4621.

[88] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2015).

[89] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[90] Richard Socher et al. “Recursive deep models for semantic compositionality over a sentiment treebank”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642.

[91] Kenneth O Stanley. *Neuroevolution: A different kind of deep learning*. 2017.

[92] Alane Suhr et al. “A corpus for reasoning about natural language grounded in photographs”. In: *arXiv preprint arXiv:1811.00491* (2018).

[93] Alane Suhr et al. “A corpus of natural language for visual reasoning”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2017, pp. 217–223.

[94] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic attribution for deep networks”. In: *arXiv preprint arXiv:1703.01365* (2017).

[95] Christian Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 1–9.

[96] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[97] Christian Szegedy et al. “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[98] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[99] Kai Sheng Tai, Richard Socher, and Christopher D Manning. “Improved semantic representations from tree-structured long short-term memory networks”. In: *arXiv preprint arXiv:1503.00075* (2015).

[100] Joshua B. Tenenbaum and William T. Freeman. “Separating Style and Content with Bilinear Models”. In: *Neural Computation* 12 (2000), pp. 1247–1283.

[101] Damien Teney et al. “Tips and tricks for visual question answering: Learnings from the 2017 challenge”. In: *arXiv preprint arXiv:1708.02711* (2017).

[102] Damien Teney et al. “Tips and tricks for visual question answering: Learnings from the 2017 challenge”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4223–4232.

[103] Bart Thomee et al. “YFCC100M: The new data in multimedia research”. In: *arXiv preprint arXiv:1503.01817* (2015).

[104] Oriol Vinyals et al. “Show and tell: A neural image caption generator”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 3156–3164.

[105] Steven Richard Waterhouse. “Classification and regression using mixtures of experts”. PhD thesis. Citeseer, 1998.

[106] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[107] Adina Williams, Andrew Drozdov, and Samuel R Bowman. “Learning to parse from a semantic objective: It works. is it syntax”. In: *arXiv preprint arXiv:1709.01121* 4 (2017).

[108] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[109] Zhibiao Wu and Martha Palmer. “Verbs semantics and lexical selection”. In: *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 1994, pp. 133–138.

[110] Saining Xie et al. “Aggregated residual transformations for deep neural networks”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 5987–5995.

[111] Zichao Yang et al. “Stacked attention networks for image question answering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 21–29.

[112] Dani Yogatama et al. “Learning to compose words into sentences with reinforcement learning”. In: *arXiv preprint arXiv:1611.09100* (2016).

[113] Daniel H. Younger. “Recognition and parsing of context-free languages in time n3”. In: *Information and Control* 10.2 (1967), pp. 189–208. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(67)80007-X.

[114] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. “Recurrent neural network regularization”. In: *arXiv preprint arXiv:1409.2329* (2014).

[115] Peng Zhang et al. “Yin and yang: Balancing and answering binary visual questions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5014–5022.

[116] Bo Zhao et al. “Modular Generative Adversarial Networks”. In: *ECCV*. 2018.

[117] Zhao Zhong, Junjie Yan, and Cheng-Lin Liu. “Practical network blocks design with q-learning”. In: *arXiv preprint arXiv:1708.05552* (2017).

[118] Bolei Zhou et al. “Simple Baseline for Visual Question Answering”. In: *CoRR* abs/1512.02167 (2015).

[119] Julian Georg Zilly et al. “Recurrent highway networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 4189–4198.

[120] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578* (2016).

[121] Barret Zoph et al. “Learning transferable architectures for scalable image recognition”. In: *arXiv preprint arXiv:1707.07012* 2.6 (2017).

Appendix A

Figures showing realization of basic modules using *v2* of cell structure

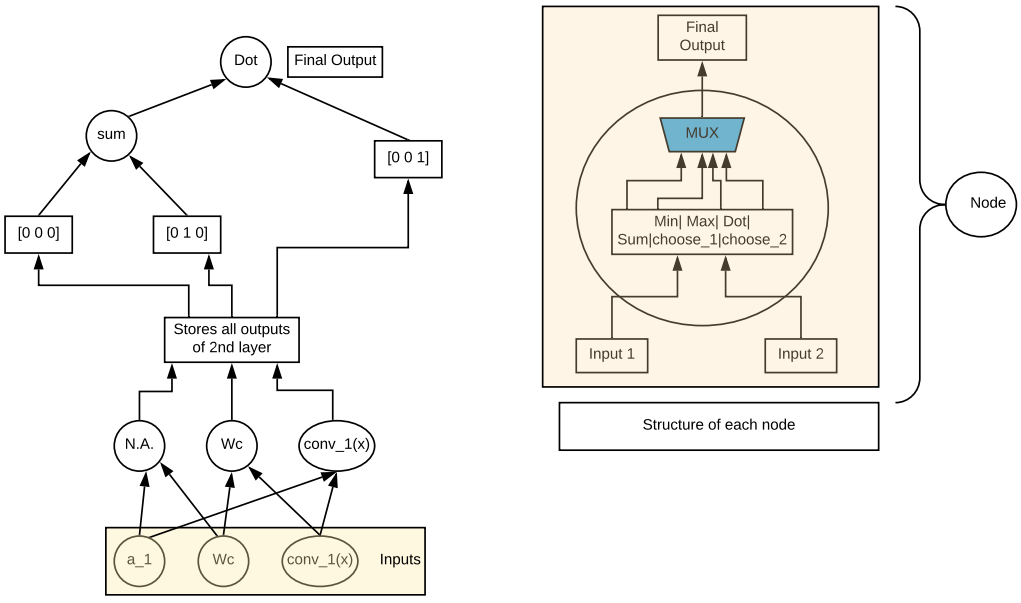

Figure A.1. Schematic diagram for realization of Find Module.

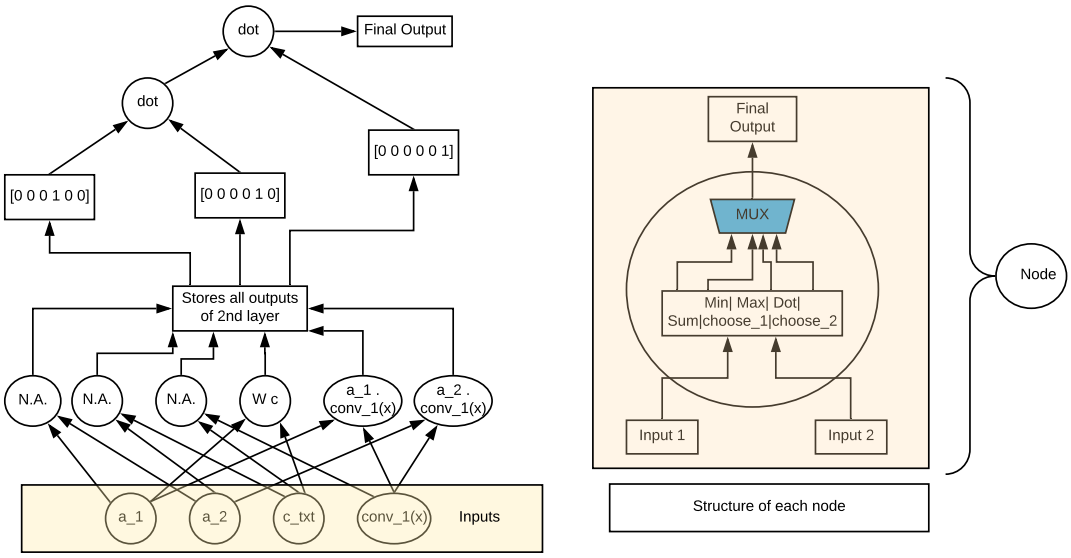

Figure A.2. Schematic diagram for realization of Compare Module.

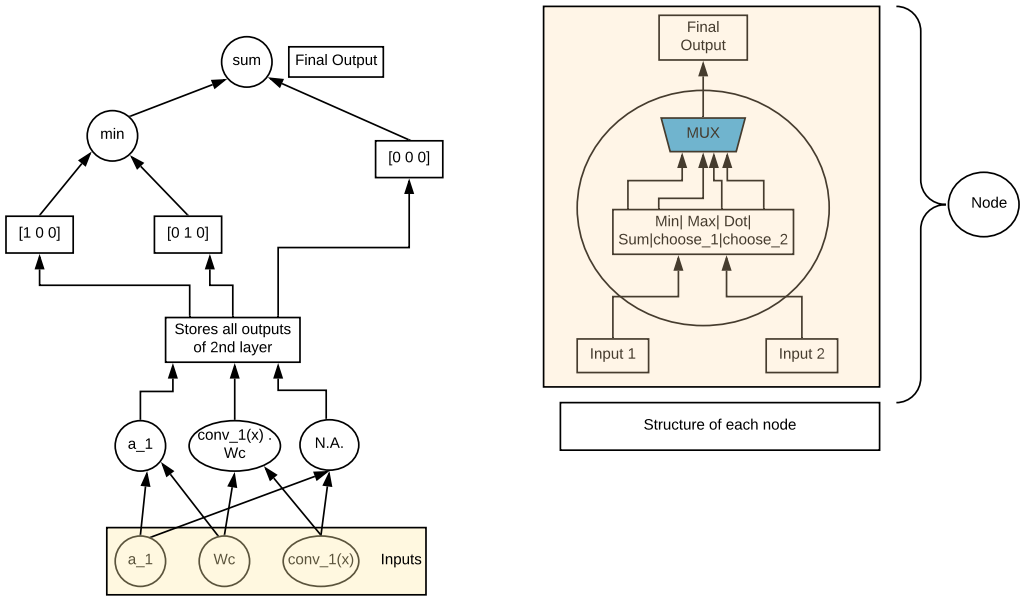

Figure A.3. Schematic diagram for realization of Filter Module.

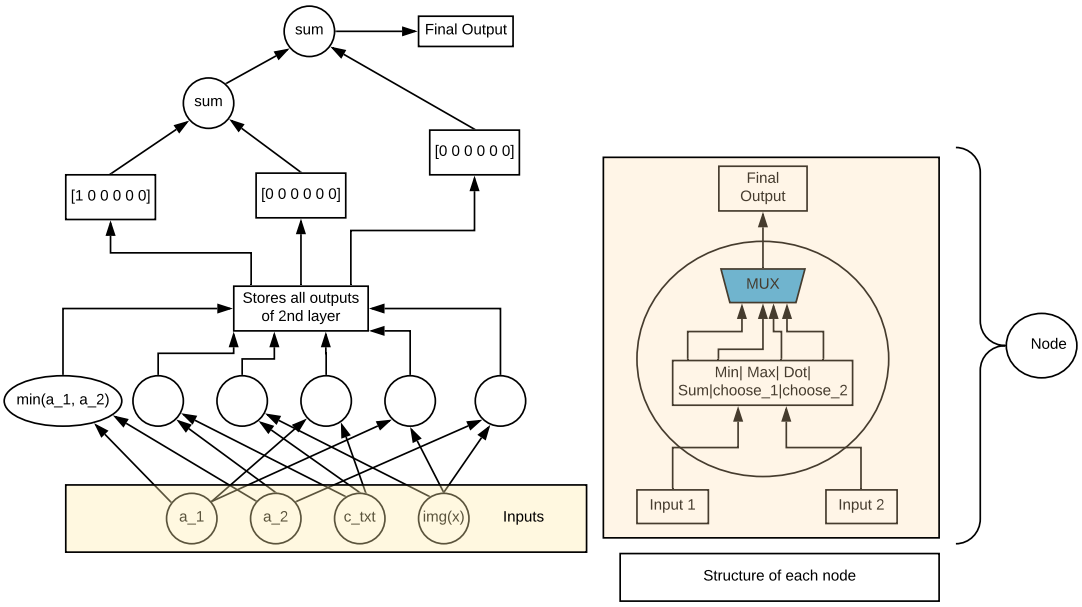

Figure A.4. Schematic diagram for realization of And Module.

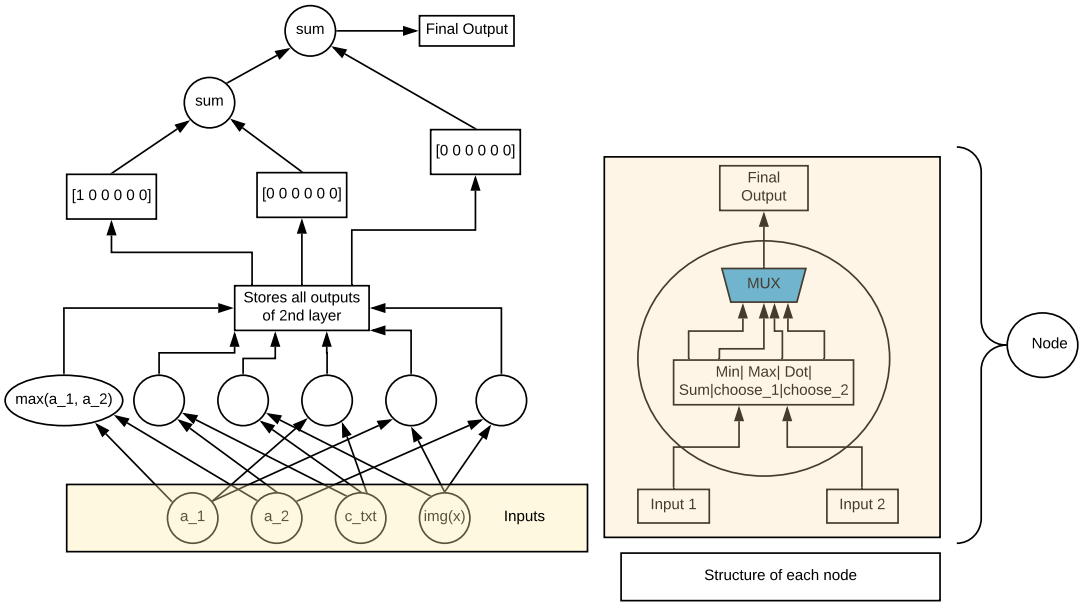

Figure A.5. Schematic diagram for realization of Or Module.

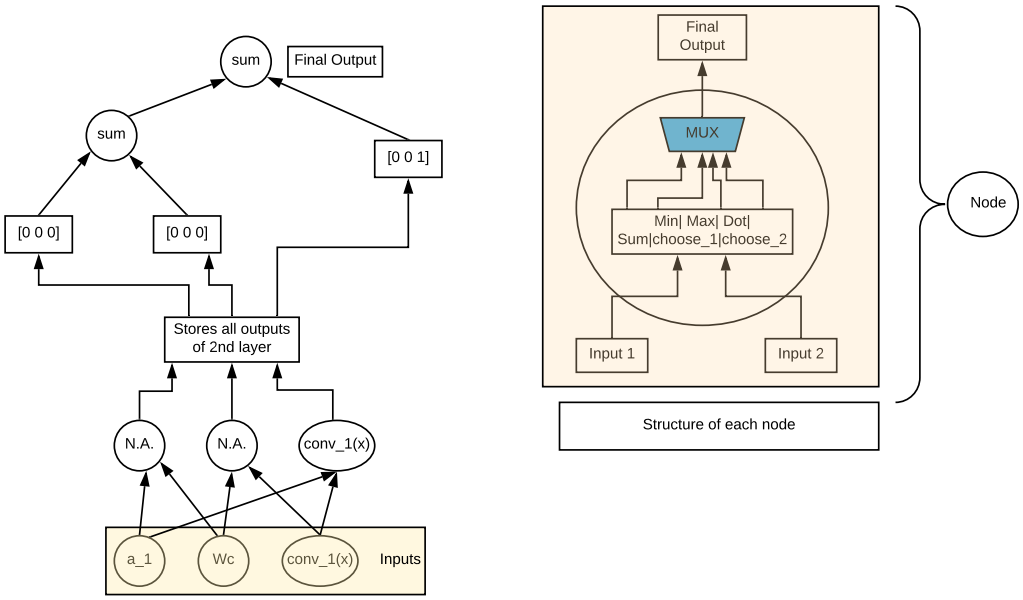

Figure A.6. Schematic diagram for realization of Scene Module.

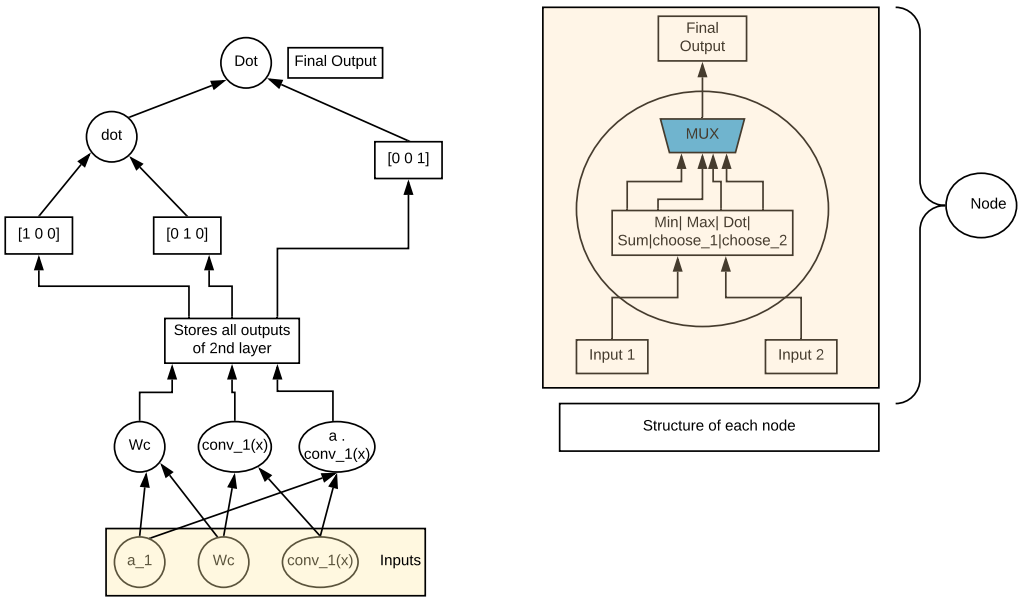

Figure A.7. Schematic diagram for realization of Transform Module.

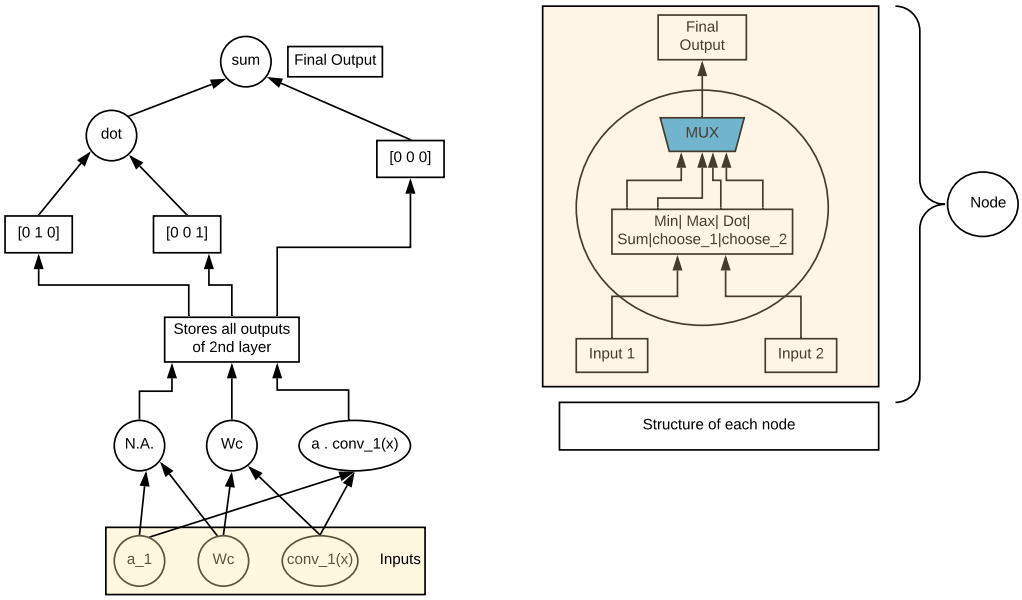

Figure A.8. Schematic diagram for realization of Answer Module.