



HAL
open science

Neural Module Networks for Compositional Visual Reasoning

Wafa Aissa

► **To cite this version:**

Wafa Aissa. Neural Module Networks for Compositional Visual Reasoning. Neural and Evolutionary Computing [cs.NE]. HESAM Université, 2023. English. NNT : 2023HESAC033 . tel-04538813

HAL Id: tel-04538813

<https://theses.hal.science/tel-04538813v1>

Submitted on 9 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ÉCOLE DOCTORALE Sciences des Métiers de l'Ingénieur
Centre d'études et de recherche en informatique et communications

THÈSE

présentée par : **M^{me} Wafa AISSA**

soutenue le : **18 décembre 2023**

pour obtenir le grade de : **Docteur d'HESAM Université**

préparée au : **Conservatoire national des arts et métiers**

Discipline : **Informatique**

Spécialité : **Informatique**

NEURAL MODULE NETWORKS FOR COMPOSITIONAL VISUAL REASONING

**Réseau de modules neuronaux pour un raisonnement visuel
compositionnel**

THÈSE dirigée par :

M. Michel CRUCIANU Professeur, Cnam

et co-encadrée par :

M. Marin FERECATU Maître de conférences, Cnam

M. Souheil HANOUNE Chief technology officer, XXII Group

Jury

M. Camille KURTZ

M. Désiré SIDIBE

M^{me} Valérie GOUET-BRUNET

M^{me} Anissa MOKRAOUI

M. Michel CRUCIANU

M. Marin FERECATU

M. Souheil HANOUNE

Professeur, LIPADE, Univ. Paris Cité

Professeur, IBISC, Univ. d'Evry

Directrice de recherche, LaSTIG, IGN

Professeur, L2TI, Univ. Sorbonne Paris Nord

Professeur, CEDRIC, Cnam

Maître de conférences, CEDRIC, Cnam

Chief technology officer, XXII Group

Rapporteur

Rapporteur

Examinatrice

Examinatrice

Directeur de thèse

Co-encadrant

Co-encadrant

Acknowledgements

Je tiens à exprimer ma profonde gratitude envers mes encadrants du CEDRIC, M. Michel Crucianu et M. Marin Ferecatu, pour leurs conseils éclairés qui ont guidé mes travaux de recherche, leur soutien et leur disponibilité constante. Un merci particulièrement chaleureux à mon directeur de thèse, M. Michel Crucianu, pour sa sagesse et sa bienveillance.

Mes remerciements vont également à M.Souheil Hanoune pour m'avoir offert l'opportunité de rejoindre XXII. Une opportunité qui m'a permis de découvrir le monde de l'entreprise. J'exprime ma reconnaissance envers XXII et l'ANRT pour avoir soutenu financièrement mes trois années de thèse.

Je suis reconnaissante envers Mme Valérie GOUET-BRUNET, Mme Anissa MOKRAOU, M. Camille KURTZ et M. Désiré SIDIBE d'avoir accepté d'être membres du jury lors de ma soutenance de thèse.

Un sincère remerciement à mes collègues du CEDRIC qui ont grandement contribué à enrichir mon expérience au sein du laboratoire, faisant de cette période une opportunité d'épanouissement professionnel et personnel.

Un sincère merci à mes amis que j'aime beaucoup.

Je remercie aussi ma famille.

Mes remerciements s'étendent à toutes les personnes qui, de près ou de loin, m'ont inspirée dans ma trajectoire pour devenir chercheuse en intelligence artificielle.

Enfin, un grand merci à tous ceux qui oeuvrent pour la paix.

Abstract

The context of this PhD thesis is compositional visual reasoning. When presented with an image and a question pair, our objective is to have neural networks models answer the question by following a reasoning chain defined by a program. We assess the model’s reasoning ability through a Visual Question Answering (VQA) setup. Compositional VQA breaks down complex questions into modular easier sub-problems. These sub-problems include reasoning skills such as object and attribute detection, relation detection, logical operations, counting, and comparisons. Each sub-problem is assigned to a different module. This approach discourages shortcuts, demanding an explicit understanding of the problem. It also promotes transparency and explainability. Neural module networks (NMN) are used to enable compositional reasoning. The framework is based on a generator-executor framework, the generator learns the translation of the question to its function program. The executor instantiates an NMN where each function is assigned to a specific module. We also design a neural modules catalog and define the function and the structure of each module. The training and evaluations are conducted using the pre-processed GQA dataset, which includes natural language questions, functional programs representing the reasoning chain, images, and corresponding answers. The research contributions revolve around the establishment of an NMN framework for the VQA task. One primary contribution involves the integration of vision and language pre-trained (VLP) representations into modular VQA. This integration serves as a “warm-start” mechanism for initializing the reasoning process. The experiments demonstrate that cross-modal vision and language representations outperform uni-modal ones. This utilization enables the capture of intricate relationships within each individual modality and the alignment between different modalities, consequently enhancing the overall accuracy of our NMN. Moreover, we explore various training techniques to enhance the learning process and improve cost-efficiency. In addition to optimizing the modules within the reasoning chain to collaboratively produce accurate answers, we introduce a teacher-guidance approach to optimize the intermediate

ABSTRACT

modules in the reasoning chain. This ensures that these modules perform their specific reasoning sub-tasks without taking shortcuts or compromising the reasoning process’s integrity. We propose and implement several teacher-guidance techniques, one of which draws inspiration from the teacher-forcing method commonly used in sequential models. Comparative analyses demonstrate the advantages of our teacher-guidance approach for NMNs.

We also introduce a novel Curriculum Learning (CL) strategy tailored for NMNs to reorganize the training examples and define a start-small training strategy. We begin by learning simpler programs and progressively increase the complexity of the training programs. We use several difficulty criteria to define the CL approach. Our findings demonstrate that by selecting the appropriate CL method, we can significantly reduce the training cost and required training data, with only a limited impact on the final VQA accuracy.

Keywords: deep learning, visual reasoning, computer vision, natural language processing, multi-modal representations.

Résumé

Cette thèse de doctorat porte sur le raisonnement visuel compositionnel. Lorsqu'on présente une paire image-question à un modèle de réseau de neurones, notre objectif est que le modèle réponde à la question en suivant une chaîne de raisonnement définie par un programme. Nous évaluons la capacité de raisonnement du modèle dans le cadre de la Question Réponse Visuelle (QRV). La QRV compositionnelle décompose les questions complexes en sous-problèmes modulaires plus simples. Ces sous-problèmes incluent des compétences de raisonnement telles que la détection d'objets et d'attributs, la détection de relations, les opérations logiques, le dénombrement et les comparaisons. Chaque sous-problème est attribué à un module différent. Cette approche décourage les raccourcis, exigeant une compréhension explicite du problème. Elle favorise également la transparence et l'explicabilité. Les réseaux de modules neuronaux (NMN) sont utilisés pour permettre un raisonnement compositionnel. Ils sont basés sur un cadre de générateur-exécuteur, le générateur apprend la traduction de la question vers son programme de fonctions. L'exécuteur instancie un NMN où chaque fonction est attribuée à un module spécifique. Nous développons également un catalogue de modules neuronaux et définissons leurs fonctions et leurs structures. Les entraînements et les évaluations sont effectués sur l'ensemble de données GQA, qui comprend des questions, des programmes fonctionnels, des images et des réponses. L'une des principales contributions implique l'intégration de représentations pré-entraînées multi-modales dans la QRV modulaire. Cette intégration sert à initialiser le processus de raisonnement. Les expériences démontrent que les représentations multimodales surpassent les unimodales. Ceci permet de capturer des relations complexes intra-modales tout en facilitant l'alignement entre les différentes modalités, améliorant ainsi la précision globale du NMN. De plus, nous explorons différentes techniques d'entraînement pour améliorer le processus d'apprentissage et l'efficacité du coût de calcul. En plus d'optimiser les modules au sein de la chaîne de raisonnement pour produire collectivement des réponses correctes, nous introduisons une approche d'apprentissage guidé pour optimiser les modules

RÉSUMÉ

intermédiaires de la chaîne de raisonnement. Ceci garantit que ces modules effectuent leurs sous-tâches de raisonnement spécifiques sans prendre de raccourcis ou compromettre l'intégrité du processus de raisonnement. L'une des techniques proposées s'inspire de la méthode "teacher-forcing" couramment utilisée dans les modèles séquentiels. Des analyses comparatives démontrent les avantages de cette approche pour les NMN. Nous introduisons également une nouvelle stratégie d'apprentissage par Curriculum (CL) adaptée aux NMN pour réorganiser les exemples d'entraînement et définir une stratégie d'apprentissage progressif. Nous commençons par apprendre des programmes plus simples et augmentons progressivement la complexité des programmes d'entraînement. Nous utilisons plusieurs critères de difficulté pour définir l'approche du CL. Nos résultats montrent qu'en sélectionnant la méthode de CL appropriée, nous pouvons réduire considérablement le coût de l'entraînement et la quantité de données d'entraînement requise, avec un impact limité sur la précision finale de la QRV.

Mots-clés : apprentissage profond, raisonnement visuel, vision par ordinateur, traitement automatique de langage naturel, représentations multi-modales.

Contents

Acknowledgements	ii
Abstract	iii
Résumé	v
List of Tables	xi
List of Figures	xiii
Chapters	
1 Introduction	1
2 Related work	6
2.1 Introduction	7
2.2 Visual reasoning architectures	9
2.2.1 Monolithic approaches	9
2.2.2 Compositional approaches	11
2.2.2.1 Multi-step approaches	11
2.2.2.2 Neural module networks	12
2.3 Executor training strategies	18
2.3.1 Knowledge guidance	18
2.3.2 Teacher forcing	19
2.3.3 Curriculum learning	20
2.4 Datasets for compositional visual reasoning	21

CONTENTS

2.4.1	VQA and VQA 2.0	21
2.4.2	CLEVR	22
2.4.3	GQA	24
3	Multi-modal Neural Module networks	29
3.1	Introduction	30
3.2	Multi-modal representations	31
3.3	Neural modules	34
3.3.1	Pre-processing	35
3.3.2	Attention modules	38
3.3.3	Boolean modules	39
3.3.4	Answer modules definition	41
3.4	Program examples	42
3.5	Generator	48
3.5.1	From questions in natural language to functional programs	48
3.5.2	Arguments prediction	49
3.5.3	Generator optimisation	50
3.5.4	Program inference	50
3.5.5	Experimental validation	52
3.6	Executor	52
3.6.1	Module initialisation	52
3.6.2	Weight sharing	53
3.6.3	Modular network instantiation	54
3.6.4	Reasoning process	54
3.6.5	Answer prediction	55
3.7	Evaluations	55
3.7.1	Experimental settings	56
3.7.2	Unimodal vs crossmodal representations	56
3.8	Conclusion	58
4	Guided-Training of neural module networks	59

CONTENTS

4.1	Introduction	59
4.2	Input guidance	60
4.3	Output feedback	62
4.3.1	Attention loss	62
4.3.2	Boolean loss	63
4.4	Intermediate targets coding	63
4.4.1	Attention targets	63
4.4.2	Boolean targets	66
4.5	Experiments	66
4.5.1	Evaluated methods	66
4.5.2	Results analysis	67
4.5.3	Implementation details	69
4.5.4	Modules training evolution	70
4.5.5	Qualitative analysis of the modular approach	71
4.6	Conclusion	73
5	Curriculum learning for neural module networks	75
5.1	Introduction	76
5.2	Curriculum Learning setup	76
5.2.1	Difficulty criterion	76
5.2.2	Scheduler	78
5.2.3	Sampling function	78
5.2.4	Performance evaluator	79
5.3	Experiments	79
5.3.1	Evaluated methods	80
5.3.2	Results analysis	81
5.3.3	Modules performances	84
5.3.4	Implementation details	85
5.3.5	Additional experiments	86
5.4	Conclusion	87

CONTENTS

6 Conclusion and Perspectives	88
Bibliography	92
Synthèse de la thèse en français	104

List of Tables

2.1	GQA dataset partitioning	26
2.2	GQA dataset function counts and percentage of frequencies in descending order.	27
2.3	Accuracy results on the GQA dataset validation and testdev splits from [Nguyen et al., 2022].	28
3.1	Module functions counts and frequencies after preprocessing.	36
3.2	Attention module definitions. σ : Activation function, r : RELU, \mathbf{W} : weight matrix, \mathbf{a} : attention vector (36×1), \mathbf{V} : visual features (768×36), \mathbf{t} : text features (768×1), \mathbf{o} : attention vector (36×1) \odot : Hadamard product, min : element-wise minimum.	38
3.3	Boolean module definitions. σ : Sigmoid, r : RELU, \mathbf{W} : weight matrix, \mathbf{a} : attention vector (36×1), \mathbf{b} : boolean scalar, \mathbf{V} : visual features (768×36), \mathbf{t} : text features (768×1), \odot : Hadamard product, $[a b]$: concatenation, min : element-wise minimum.	40
3.4	Answer module definitions. S : softmax, r : RELU, \mathbf{W} : weight matrix, \mathbf{a} : attention vector (36×1), \mathbf{b} : boolean scalar, \mathbf{V} : visual features (768×36), \mathbf{t} : text features (768×1), \odot : Hadamard product.	41
3.5	Program examples	48
3.6	Language and vision representation comparison on <code>testdev-all</code>	57
4.1	Performance of various training methods and encodings on the <code>testdev-all</code> set.	67
5.1	Results on <code>testdev-all</code> for several CL strategies.	81
5.2	Results on <code>testdev-all</code> with program length as a refinement for the CL difficulty measure. Computation cost is the number of seen examples per iteration times the number of iterations.	82
5.3	Comparison of our CL model (CL+W.a+P+R) with no-CL models (Unbalanced, Balanced, and Random) on the <code>testdev-all</code> set.	83

LIST OF TABLES

6.1 Définitions de certains modules. σ : fonction d'activation non-linéaire, r : RELU, \mathbf{W} : matrice de poids, \mathbf{a} : vecteur d'attention (36×1), \mathbf{b} : scalaire booléen (1×1), \mathbf{V} : représentations visuelles (768×36), \mathbf{t} : représentations textuelles (768×1), \odot : produit matriciel de Hadamard.	108
---	-----

List of Figures

1.1	Image from the GQA dataset [Hudson and Manning, 2019b].	3
2.1	VQA task: Given an image and a question, a model learns to predict an answer. . . .	7
2.2	LXMERT architecture [Tan and Bansal, 2019], self: self-attention, FF: feed forward, cross: cross-attention	10
2.3	Modules architectures from [Andreas et al., 2016b].	14
2.4	Modules architecture from [Li et al., 2019a].	16
2.5	Model from [Johnson et al., 2017b]. Generator on the right and executor on the left. .	17
2.6	RNN without and with teacher forcing.	19
2.7	VQA dataset examples.	22
2.8	CLEVR Example: Image on the Left, Functional Program and Question in the center and function catalog in the right.	23
2.9	Top: Statistics for CLEVR per split. Bottom left: Comparison of question lengths for different VQA datasets. Bottom right: Distribution of question types in CLEVR. . . .	24
2.10	GQA Example: Image on the Left, Functional Program and Question on the middle and image graph on the right.	25
2.11	GQA Examples based on their structural and semantic types	25
2.12	Dataset statistics from [Hudson and Manning, 2019b]: structural types, semantic types, number of reasoning steps and questions length distribution compared to other datasets.	26
3.1	The proposed modular VQA framework: The (question, image) pair is used by a transformer model to generate aligned cross-modal embeddings for words and objects (3.2). These are used by a Program Generator (3.5) module to produce a program (represented as a sequence of sub-task modules), which will be then applied by the Program Executor (3.6) module to the image to answer the question.	31

LIST OF FIGURES

3.2	A-taxonomy-of-word-embeddings, from [Wang et al., 2021]	32
3.3	LXMERT architecture [Tan and Bansal, 2019], self: self-attention, FF: feed forward, cross: cross-attention.	33
4.1	The teacher guidance for the program execution process related to the question ‘On what is the animal to the right of the laptop sleeping?’. Plain arrows represent input guidance 4.2, dotted arrows represent the output feedback 4.3 and bounding boxes represent the GT intermediate targets.	60
4.2	Comparison between the ground-truth bounding boxes and the LXMERT bounding boxes.	64
4.3	Raw program structure.	64
4.4	Object identifiers, names and coordinates from the image graph data.	65
4.5	Modules’ training loss curves plotted every 10 epochs.	71
4.6	Visualization of the reasoning process.	73
4.7	Visualization of the reasoning process for some incorrect answers.	74
5.1	GQA dataset samples with different difficulty levels.	77
5.2	Accuracy histograms for the structural types of the questions.	84
5.3	Accuracy histograms for the semantic types of the questions.	85
5.4	The program (represented as a sequence of sub-task modules) is applied by the Program Executor module to the image to answer the question. The proposed work focuses on improving the Program Executor by using several curriculum learning (CL) strategies.	86
6.1	Image extraite de la base de données GQA [Hudson and Manning, 2019b].	106
6.2	Architecture du modèle LXMERT [Tan and Bansal, 2019].	107
6.3	Le cadre QRV modulaire proposé : La paire (question, image) est utilisée par un modèle de Transformer pour générer des représentations alignés pour les mots et les objets. Celles-ci sont utilisées par un générateur pour produire un programme (une séquence de modules de sous-tâches), qui sera ensuite appliqué par un exécuteur de programmes sur l’image pour répondre à la question.	109

6.4 L'apprentissage guidé pour l'exécution du programme relatif à la question ' Sur quoi l'animal à droite de l'ordinateur portable est en train de dormir? '. Les flèches continues représentent l'échantillonnage planifié (voir Section 4.2), les flèches discontinues représentent l'erreur multi-tâches (voir Section 4.3), et les boîtes englobantes représentent les cibles intermédiaires. 110

Chapter 1

Introduction

Artificial intelligence (AI) aims to make machines smarter, so they can do things like humans do, such as learning, thinking, and behaving. This pursuit has encountered challenges, particularly in tasks that are intuitive for humans but challenging to formally describe—those that appear automatic, such as recognizing spoken words or discerning faces in images. This frontier in AI has been greatly influenced by the advent of connectionist neural networks, which excel in pattern recognition and learning, as mentioned in [Goodfellow et al., 2016].

Visual reasoning in the context of AI encompasses the ability of machines to interpret and make sense of visual data, much like how humans do. It involves the comprehension, analysis and manipulation of visual information to derive meaningful insights or make informed decisions. Visual reasoning goes beyond mere object recognition; it entails the capacity to understand complex scenes, infer relationships between objects, and answer questions based on visual input. This multifaceted cognitive process often involves tasks such as object localization, attribute identification, spatial reasoning, and even higher-level reasoning involving context and semantics. In essence, visual reasoning aims to bridge the gap between raw visual data and meaningful understanding, enabling AI systems to interact with the visual world in a manner akin to human perception and cognition.

A notable application of visual reasoning is Visual Question Answering (VQA) [Goyal et al., 2017, Hudson and Manning, 2019b], a field that merges computer vision (CV) with natural language processing (NLP). VQA systems are designed to comprehend complex visual scenes and respond to questions about them, bridging the gap between visual data and human-like understanding. However, comprehending the intricate reasoning processes of such systems is challenging, as demonstrated by

the case of Clever Hans—an early 20th-century horse that appeared to answer arithmetic questions. A closer investigation unveiled that Hans was, in reality, responding to subtle cues from human observers, rather than engaging in genuine arithmetic calculations [Pfungst and Hans, 1965].

To enable machines to perform a transparent reasoning about the visual world, our focus lies in achieving explicit reasoning, mirroring the symbolic reasoning processes evident in humans. This entails breaking down complex reasoning tasks into a sequence of defined program steps, akin to the modular processes humans employ [Fodor, 1983, Clune et al., 2013]. For that purpose, we draw inspiration from the field of compositional semantics in natural language understanding [Partee et al., 1984]. Just as compositional semantics explores how sentence meaning emerges from the meanings of its parts and their structure, compositional Visual Question Answering (VQA) [Hudson and Manning, 2019b] aims to dissect complex visual reasoning problems into more digestible components. In VQA, these components are elements like objects, attributes, and relations within a question related to an image. By following compositional semantics principles, we seek to understand how these visual elements are combined to reason about the visual world, ultimately allowing us to provide meaningful answers to image-related questions.

For compositional reasoning, various modular techniques have been proposed. One prominent and central to our work is the Neural Module Network (NMN) [Andreas et al., 2016b, Johnson et al., 2017b, Li et al., 2019a]. In NMN, the reasoning process is divided into smaller, specialized modules that work collaboratively to solve complex tasks. Each module is designed for a specific sub-task and modules can be combined to address diverse questions and problems. For example, consider the task of answering questions about an image containing objects, attributes and relations. NMN employs different modules for sub-tasks like object recognition, attribute identification, and relation understanding. These modules are chained in a sequential step-by-step manner to answer questions.

The power of NMN lies in its flexibility and modularity. It is a dynamic architecture that allows to define specialized modules that can be flexibly combined to tackle a wide array of questions, making it a valuable tool in compositional reasoning tasks. For example, given the question “What color is the fruit on the right side, red or green?” about the image presented in Fig. 1.1, to successfully answer this question in a compositional manner we need three different modules: one to focus on the right side of the image, another to recognize the fruit, and an additional module to decide between red or green as the fruit’s color.



Figure 1.1: Image from the GQA dataset [Hudson and Manning, 2019b].

The NMN framework operates within a generator-executor framework, a two-component process that enables efficient reasoning. For the first component, the generator learns to translate a given question into its corresponding functional program. This program represents a series of operations to execute to answer the question accurately. For the second component, the executor instantiates an NMN, where each function within the program is assigned to a specific module. These functions are derived from a predefined neural modules taxonomy, which categorizes various reasoning sub-tasks into distinct modules. Importantly, each function corresponds to a neural network block, designed to adhere to the structure and requirements of the specific input data for that function. This structured approach ensures that the NMN can effectively execute the functions within the program in a sequential manner, ultimately leading to predicting answers to the posed questions.

The research contributions revolve around the establishment of an NMN framework for the VQA task. One primary contribution involves the integration of vision and language pre-trained (VLP) representations [Tan and Bansal, 2019] into modular VQA [Andreas et al., 2016b]. This integration serves as a “warm-start” mechanism for initializing the reasoning process. It also addresses a notable weakness of the NMN framework, which has been observed in prior works. Specifically, NMN has faced challenges in effectively modeling prior knowledge about answers, as highlighted in earlier studies [Andreas et al., 2016a, Andreas et al., 2016b]. The experiments demonstrate that cross-modal vision and language representations outperform uni-modal ones. This utilization enables the capture of intricate relationships within each individual modality while also facilitating alignment between different modalities, consequently enhancing overall accuracy of the NMN.

Moreover, we explore various training techniques to enhance the learning process and improve cost-efficiency. In addition to optimizing the modules within the reasoning chain to collaboratively produce accurate answers, we introduce a teacher-guidance approach to optimize the intermediate modules in the reasoning chain. This ensures that these modules perform their specific reasoning sub-tasks without taking shortcuts or compromising the reasoning process’s integrity. We propose and implement several teacher-guidance techniques, one of which draws inspiration from the teacher-forcing method commonly used in sequential models. Comparative analyses demonstrate the advantages of our teacher-guidance approach for NMNs, as detailed in our paper [Aissa et al., 2023b].

We also introduce a novel Curriculum Learning (CL) strategy tailored for NMNs to reorganize the training examples and define a start-small training strategy. We begin by learning simpler programs and progressively increase the complexity of the training programs. We use several difficulty criteria to define the CL approach. Our findings demonstrate that by selecting the appropriate CL method, we can significantly reduce the training cost and required amount of training data, with only a limited impact on the final VQA accuracy. This significant contribution forms the core of our paper [Aissa et al., 2023a].

Visual reasoning occupies an important position in academia on a global level. Efforts are made to bridge the gap between academic advancements and practical industry applications. This aims to unleash the potential of acquired skills in real-world scenarios. Such practical implementations span diverse domains, exemplified by the application of visual reasoning in video analysis for urban settings. This includes functions like crowd analysis [Tomar et al., 2022], efficient parking management [Singh et al., 2018], and robust intrusion detection systems [Zabłocki et al., 2014]. Furthermore, the scope extends to scene analysis for autonomous vehicles, where visual reasoning contributes to essential tasks such as vehicles recognition [Tsai et al., 2018, Arinaldi et al., 2018], trajectory prediction [Wang et al., 2022a], and environmental understanding [Palanisamy, 2020], ultimately fostering advancements in the field of self-driving cars. Enhancing the transparency of AI model decision-making through a modular approach significantly improves explainability and helps model debugging. Employing extra supervision for the model components enables the monitoring of their behavior, ensuring they adhere to their specific tasks. Furthermore, the incorporation of a curriculum learning training strategy ensures gradual model training, paving the way for a more effective continual learning roadmap.

This thesis is structured as follows: we commence with this introduction chapter, followed by an exploration of related work in Chapter 2, encompassing both monolithic and compositional techniques in Visual Question Answering (VQA). We delve into the various training strategies employed in Neural Module Networks (NMN) and discuss the available datasets pertinent to VQA. In Chapter 3, we present and motivate the NMN architecture that we have designed and implemented. We then proceed to the first significant contribution of this thesis, centered around teacher forcing for NMN in Chapter 4. Subsequently, we delve into curriculum learning for NMN in the dedicated Chapter 5. Finally, we summarize our findings and insights in the concluding Chapter 6.

Chapter 2

Related work

Contents

2.1	Introduction	7
2.2	Visual reasoning architectures	9
2.2.1	Monolithic approaches	9
2.2.2	Compositional approaches	11
2.3	Executor training strategies	18
2.3.1	Knowledge guidance	18
2.3.2	Teacher forcing	19
2.3.3	Curriculum learning	20
2.4	Datasets for compositional visual reasoning	21
2.4.1	VQA and VQA 2.0	21
2.4.2	CLEVR	22
2.4.3	GQA	24

2.1. INTRODUCTION



1. Is the **tray** on top of the **table** black or light brown? light brown
2. Are the **napkin** and the **cup** the same color? yes
3. Is the small **table** both oval and wooden? yes
4. Is the **syrup** to the left of the **napkin**? yes



1. Is there a **door** or a **window** that is open? no
2. Do you see any white **numbers** or **letters**? yes
3. What is the large **container** made of? cardboard
4. What **animal** is in the **box**? **bear**



1. Which side of the image is the **plate** on? right
2. Are there any **lamps** on the **desk** to the right of the **rug**? yes
3. What type of **furniture** are the **flowers** on, a **bed** or a **table**? **table**

Figure 2.1: VQA task: Given an image and a question, a model learns to predict an answer.

2.1 Introduction

Visual scene understanding transcends mere visual recognition, object detection, and segmentation [Redmon and Farhadi, 2016, Anderson et al., 2018, Wu et al., 2019, Kirillov et al., 2023]. It requires a model’s ability to emulate human-like reasoning processes to attain a profound comprehension of complex scenes. This entails identifying not only the visual elements but also comprehending the intricate relationships and properties of objects within the scene. In contrast to the established domain of visual recognition, visual reasoning remains in its early stages, offering a ground for exploration and innovation. The challenge lies in enabling machines to engage in nuanced reasoning, akin to human cognition, to extract meaningful insights from visual data.

At the forefront of visual reasoning tasks stands Visual Question Answering (VQA) [Ren et al., 2015a, Goyal et al., 2017, Hudson and Manning, 2019b], a benchmark that tests a model’s ability to perform complex reasoning about the visual world. In VQA, a model is tasked with comprehending the content of an image and responding accurately to a posed question, effectively bridging the gap between textual and visual data, examples extracted from [Hudson and Manning, 2019b] are given in Figure 2.1. This task not only underscores the complexity of visual reasoning but also underscores the potential of AI systems to understand, interpret, and interact with the visual world.

To effectively engage in visual reasoning with deep neural networks, two fundamental components are imperative: first, robust vision and language representations and, second, a transparent reasoning

process.

Robust vision and language representations imply the capacity to seamlessly fuse and interpret diverse forms of data, ensuring that the model comprehends the patterns embedded within visual scenes and textual queries. Significant progress has been made in the development of multi-modal Vision-Language Pretrained (VLP) models, which serve as foundational frameworks for extracting features from both images and textual questions in various downstream tasks. These models are integral in bridging the gap between vision and language, enabling the fusion of visual and textual information for better understanding. One category of such models consists of Cross-Attention Transformer models, which have demonstrated interesting capabilities. Within this category, some models employ a Region-based Cross-Attention approach. These models begin by passing the image through an object detector, such as Faster R-CNN [Girshick, 2015] or Vinvl [Zhang et al., 2021], to identify and localize objects. Subsequently, they learn region-based features that encapsulate information about the objects. Examples of such models include LXMERT [Tan and Bansal, 2019], OSCAR [Li et al., 2020], and UNITER [Chen et al., 2020]. In contrast, another set of models divides the image into smaller patches and employs a Vision Transformer (ViT) [Dosovitskiy et al., 2021] to encode these patches. This approach has gained prominence and is implemented in models like ViLT [Kim et al., 2021], AlBef [Li et al., 2021], and TCL [Yang et al., 2022]. Furthermore, a noteworthy model in this domain is CLIP [Radford et al., 2021], which adopts a dual-encoder transformer architecture. CLIP represents a significant advancement as it simultaneously encodes both images and text, enabling cross-modal understanding and reasoning. In the subsequent section we focus on VLP models that are designed for the specific task of VQA.

Transparency in the reasoning process is equally important [Andreas et al., 2016b, Mascharka et al., 2018, Li et al., 2019a]. It signifies the model’s ability to provide clear and interpretable steps in its decision-making process. This transparency empowers not only end-users but also researchers to understand, analyze, and trust the outputs of these neural networks, ensuring that the AI-driven conclusions align with human intuition and expectations. Incorporating these foundational elements is pivotal in advancing the capabilities of deep neural networks for complex reasoning tasks.

This chapter is structured as follows: We begin by elucidating various visual reasoning approaches, encompassing integrated multi-modal approaches [Li et al., 2019b, Lu et al., 2019a, Tan and Bansal, 2019]

and compositional approaches [Chen et al., 2021, Johnson et al., 2017b, Li et al., 2019a]. As the focus of this research centers on compositional reasoning, we delve into the training strategies specific to these approaches. Lastly, we discuss the datasets utilized in the VQA context.

2.2 Visual reasoning architectures

The first category embodies the monolithic approach [Li et al., 2019b, Lu et al., 2019a, Tan and Bansal, 2019], where both the image and the question are cohesively encoded within a static architecture through the use of attention and fusion mechanisms. This approach seeks to integrate visual and linguistic information within a fixed framework for scene understanding.

In contrast, the second category, which is the primary focus of our research, adopts a compositional approach and views reasoning as a multi-step process. In this dynamic paradigm, we utilize Neural Module Networks (NMNs) [Andreas et al., 2016b] as the underlying architecture. Here, the model’s structure adapts to the specific demands of each question, enabling it to assemble and reconfigure its components for problem-specific reasoning processes.

2.2.1 Monolithic approaches

Among the recent advancements in deep learning techniques applied to the VQA task, monolithic networks, notably transformers [Vaswani et al., 2017, Devlin et al., 2019, Dosovitskiy et al., 2021], have emerged as exceptional performers in both natural language processing (NLP) and computer vision (CV) domains. These models harness attention mechanisms to effectively capture long-range dependencies and contextual relationships.

Prior to the emergence of Transformers, VQA models like [Antol et al., 2015, Goyal et al., 2017, Kim et al., 2018, Kim et al., 2016] typically encoded questions using LSTM-like models and images using CNN-like models and used fusion modules or attention mechanisms to assess the correlation among the two information sources. Transformer-based methods take the form of a one stream self-attention combining the visual and textual inputs like in [Li et al., 2019b] or two streams co-attention [Tan and Bansal, 2019, Lu et al., 2019a], to align the elements between the two sources of information, combining visual and textual inputs, while accommodating the distinct processing requirements for each modality.

2.2. VISUAL REASONING ARCHITECTURES

The training strategy employed in models like [Li et al., 2019b, Lu et al., 2019a, Tan and Bansal, 2019] draws inspiration from BERT pre-training [Devlin et al., 2019]. It involves pre-training on a vast amount of data collated from several language and/or vision pretraining tasks to obtain task-agnostic representations and subsequently fine-tuning the model by using the data of the downstream task.

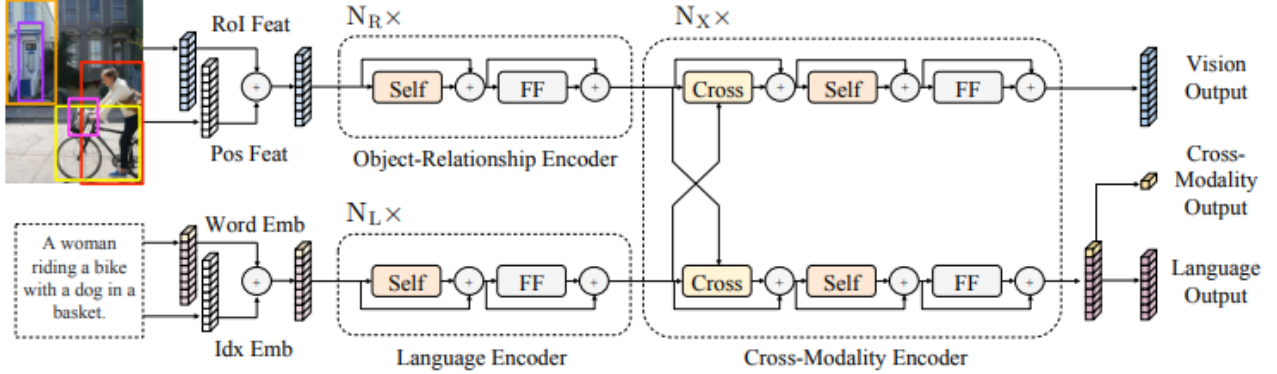


Figure 2.2: LXMERT architecture [Tan and Bansal, 2019], self: self-attention, FF: feed forward, cross: cross-attention

To provide a more detailed understanding of a monolithic architecture, we focus on LXMERT (Figure 3.3). This architecture implements BERT-like encoder blocks [Devlin et al., 2019] and utilizes self-attention mechanisms for intra-modal relationships and cross-attention mechanisms for inter-modal alignment. In LXMERT, images are represented through region-based features, which are extracted via an object detector from [Ren et al., 2015b]. These regions are represented by their bounding box coordinates (Pos Feat) and spatial features (RoI Feat), and are subsequently passed through the object-relationship encoder. On the other hand, the textual input, which is the question, is represented using word embeddings (Word Emb) along with their positional embeddings (Idx Emb). These embeddings are processed by the language encoder. The outputs from these single-modal encoders are then directed to the Cross-modality encoder, which uses two unidirectional cross-attention mechanisms: one from language to vision and one from vision to language. This serves to contextualize the features in one modality based on the information from the other, enabling a better understanding of the relationships between visual and textual data. This model serves as the multi-modal encoder for our work, details about LXMERT usage are in Section 3.2

Despite the benefits of the integrated approaches, these models have notable drawbacks. One prominent limitation is their lack of interpretability, making it challenging to understand—and

2.2. VISUAL REASONING ARCHITECTURES

debug, when necessary—the underlying reasoning process. Moreover, these models often rely on “shortcuts” in the reasoning, which means learning biases present in the training data. Consequently, their performance tends to suffer when confronted with out-of-distribution data, as shown on GQA-OOD [Kervadec et al., 2021], where the authors perform a distribution shift in the validation and test sets of a common VQA dataset [Hudson and Manning, 2019b] and observed that state of the art VQA models still learn from biases in the training data and fail to generalize on rare question-answer pairs.

2.2.2 Compositional approaches

Compositionality in reasoning is defined by the intrinsic structure of the reasoning problems, where a complex problem is composed of multiple smaller, interconnected sub-problems. Compositional VQA takes this approach by breaking down complex questions into simpler, modular sub-problems. This allows the model to focus on basic tasks like recognizing objects, attributes, relationships, counting, and making comparisons. Importantly, this method discourages taking shortcuts and requires a clear understanding of each sub-problem. It also makes the model’s decision-making process more explicit and transparent.

Compositional VQA approaches can be categorized into two groups. The first group utilizes implicit reasoning and is characterized by recurrent approaches. These methods leverage attention-based mechanisms to perform iterative reasoning. Notably, these multi-step approaches [Yang et al., 2016, Perez et al., 2018, Hudson and Manning, 2018] maintain the same recurrent unit for each reasoning step. The second group, which includes Neural Module Networks (NMN), embraces explicit reasoning. In this approach, reasoning is guided by a program that dictates the different reasoning steps, providing a more structured and programmatic way of addressing the VQA task.

2.2.2.1 Multi-step approaches

Stacked attention [Yang et al., 2016] leverages a multi-step soft attention mechanism [Bahdanau et al., 2015, Xu et al., 2015], which locates at each step the image regions that are relevant to the question for answer prediction. When trained on the VQA dataset, the model typically employs a low number of attention layers, often one or two stacked attention layers.

Augmented CNN approaches, exemplified by [Perez et al., 2018], employ Feature-wise Linear Modulation (FiLM) to modify image feature maps. This modulation involves scaling and shifting

2.2. VISUAL REASONING ARCHITECTURES

these feature maps according to the specific question, akin to a generalized form of convolutional normalization. The scaling and shifting parameters used in this feature-wise affine operation are predicted based on the continuous representation of the question, and they are unique for each input and feature map. The model is structured as a stack of residual blocks, and a FiLM layer is integrated into each of these blocks to enable feature map manipulation. The model exhibits robustness in terms of the number of residual blocks required, typically falling within the range of 2 to 12 blocks.

MAC network [Hudson and Manning, 2018] breaks down the problem by decomposing the question into a series of attention-based reasoning operations derived directly from the question. The recurrent MAC cell segregates control and memory functions. Control determines the reasoning operation by selecting relevant question words via soft attention. Memory stores intermediate results, combining past memories with image-derived information. The model’s number of reasoning steps is a hyperparameter set to 20 during training. Questions exhibit varying complexities and demand different numbers of reasoning steps. To accommodate this, a gating mechanism is incorporated, enabling the model to bypass unnecessary reasoning steps.

While these approaches offer the advantage of implicit multi-step reasoning, they lack task-specialized modules because they rely on a shared recurrent reasoning block for all the different sub-tasks. This drawback is effectively addressed by the Neural Module Networks (NMNs) presented in the following sub-section.

2.2.2.2 Neural module networks

Neural Module Networks (NMNs) represent a distinctive class of models tailored for VQA. They leverage the innate compositional structure [Partee et al., 1984] found in linguistic questions and the potent capabilities of neural networks for representation learning. NMNs rely on the functional program format of a question to construct a dynamic NMN architecture, subsequently employing it to perform reasoning tasks on the associated image data in a step-by-step manner.

Originally introduced by [Andreas et al., 2016b], NMN emerged from the premise that visual reasoning fundamentally involves compositionality. NMN seeks to establish a framework for modular, composable, and jointly trained neural networks. Over time, NMNs have undergone various developments and evolutions, encompassing changes in its definition, functional structure, and training techniques. An example of the NMN architecture is presented in the Figure 2.5.

2.2. VISUAL REASONING ARCHITECTURES

At a high level, an NMN comprises three key components:

- **Modules taxonomy:** NMN relies on a set of predefined modules, each representing a different function or operation that can be applied to process the input information.
- **Generator:** This component is responsible for transforming a natural language question into a program format that is used to build the NMN structure.
- **Executor:** The executor takes the program generated by the generator and instantiates it into an NMN. This NMN is then executed on the provided image to perform the required visual reasoning and answer the question.

In the following subsections, we delve into the related work that laid the foundation for NMNs and contributed to their evolution. This section is organized based on the different components of NMNs, where we introduce related work and various approaches for each component, and also present their various training techniques.

Module Taxonomy

Module taxonomy encompasses the primitive operations that represent various reasoning sub-tasks within NMN. Defining a universal module taxonomy is a complex challenge, with most NMN papers, such as [Andreas et al., 2016b, Li et al., 2019a, Chen et al., 2021], creating their own unique taxonomies. Notably, [Mascharka et al., 2018] is an exception, as it adopts the module taxonomy used in [Johnson et al., 2017b].

Several factors contribute to the diversity in module taxonomy. First, it's influenced by the dataset type. Synthetic datasets like Shapes [Andreas et al., 2016b] or CLEVR [Johnson et al., 2017a] necessitate a smaller set of primitives as they primarily involve questions about a small set of geometric objects and their relations. In contrast, real-world datasets like [Goyal et al., 2017, Hudson and Manning, 2019b] present a broader range of objects and relations, demanding a more extensive module taxonomy. The taxonomy is also subject to the specific design choices of the respective conceptors, given the absence of a universally perfect way to define module taxonomy. Lastly, the generalization capacity of chosen operations to different objects, attributes and relations significantly influences the structure of module taxonomy. This capacity is closely tied to the architectural design

2.2. VISUAL REASONING ARCHITECTURES

of the modules and has contributed to the ongoing evolution of NMN, as elaborated upon in the following.

Specific modules. Based on the methodology outlined in [Andreas et al., 2016b], the primitive sub-tasks are organized into a set of modules, each module being designed to address a specific sub-task. The module types encompass several attention and classification functions and each module type has several instances that describe the precise context or entity associated with the module’s function. For example, within the scope of an `attend` module, instances could represent the object or property deserving of attention, as in `attend[cat]` or `attend[dog]`. The architectures of the various modules from the taxonomy are intentionally distinct from one another. This architectural diversity allows each module to be tailored to the unique characteristics of its inputs, such as image features or outputs from other modules, enabling them to perform heterogeneous computations. The [Andreas et al., 2016b] work presented different sets of modules for the Shapes dataset and for the VQA dataset [Antol et al., 2015]. An alternative approach, as presented in [Johnson et al., 2017b], adopts a more uniform module structure for all modules, which are based on residual blocks [He et al., 2016]. However, these modules still exhibit variations in their instantiations based on the specific instances similar to [Andreas et al., 2016b, Mascharka et al., 2018]. The paper primarily conducts experiments on the CLEVER dataset [Johnson et al., 2017a] and features modules customized to this particular

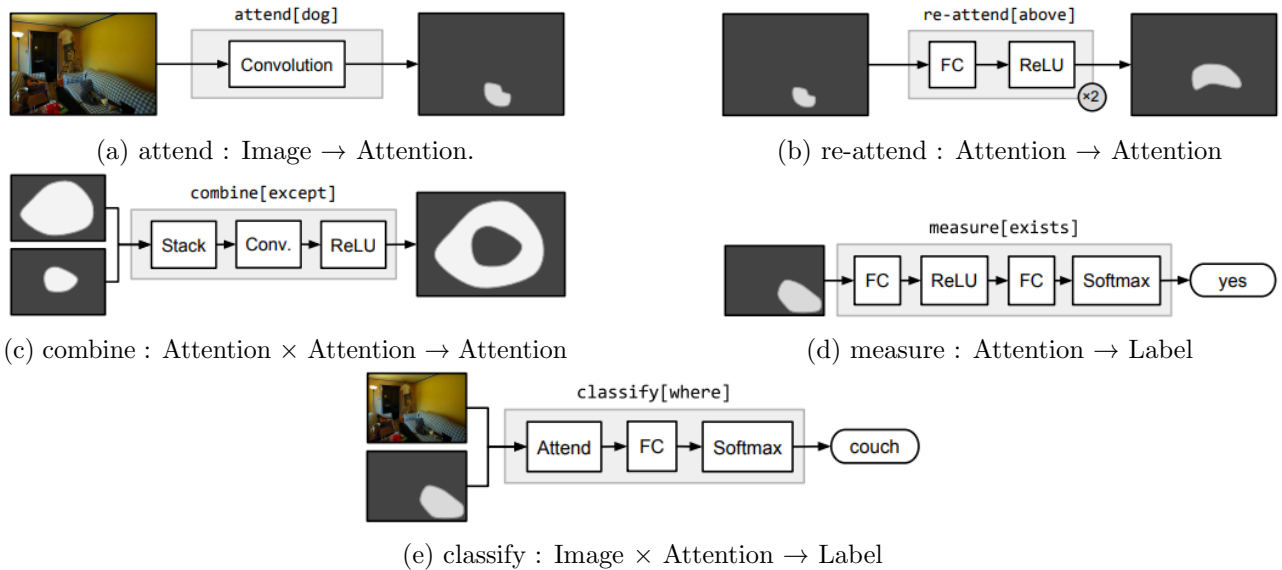


Figure 2.3: Modules architectures from [Andreas et al., 2016b].

2.2. VISUAL REASONING ARCHITECTURES

dataset. While designing specific modules tailored to their respective functions and instances offers unparalleled adaptability, it simultaneously engenders a large number of module instances. Consequently, the total number of distinct module instances can fluctuate in correspondence with the array of instances encountered in the dataset. According to [Andreas et al., 2016b], this variance results in a substantial number of unique module instances, reaching a count of 1995 during the experiments on the real-world VQA dataset [Antol et al., 2015]. Figure 2.3 shows the different modules architectures from [Andreas et al., 2016b]. The modules process the input image features, or attention maps from previous modules. They are composed of non-linear layers or convolutional layers based on the modules sub-tasks.

Generic modules. In contrast to the specific modules approach, there has been a shift towards designing more versatile modules capable of addressing real-word reasoning problems. For example, a single `attend` module can now be flexibly used to attend to various objects, eliminating the need for a dedicated `attend` module for each individual object. This enhanced versatility is achieved by introducing “textual arguments” into the module definitions, allowing the modules to adapt their behavior as needed. In works such as [Hu et al., 2017] and [Li et al., 2019a], the generator not only learns the mapping between a question and its functional program but also predicts a textual argument for each module token within the functional program. This is achieved through the attention mechanism of the seq2seq model. At each module prediction step, the attention LSTM generates an attention map over the question words, identifying the most relevant words from the question to serve as arguments for the NMN’s modules.

To illustrate a sample taxonomy of generic modules, we reference Figure 2.4 from the [Li et al., 2019a] paper. In this taxonomy, modules are categorized into four groups and operate with three distinct data types: *att* for attention type, *bool* for boolean type and *ans* for answer type. Modules that accept textual parameters (x_{txt}) are indicated with a checkmark (\checkmark). Additionally, modules utilize image features (x_{vis}) and question features (x_q) during their execution. The \odot operation represents element-wise multiplication, and *vec* is an operation that flattens attention maps while introducing extra dimensions (e.g., *max*, *min*, *average* over attention maps). $P(b)$ denotes the probability of a given statement b being true. Lastly, the *padding* operation transforms logical outputs into answers.

In [Chen et al., 2021], the modules are attention based Transformers where the “queries” are the mixed embedding of each module function name with its argument to leverage the attention based

2.2. VISUAL REASONING ARCHITECTURES

Module Type	Module Name	x_{txt}	Inputs	Output	Implementation Details
Attention	Find	✓	-	att	$a_{out} = \text{conv}_2(\text{conv}_1(x_{vis}) \odot Wx_{txt})$
	Filter[Attr Pos]	✓	att	att	$a_{out} = \text{minimum}(a_1, \text{conv}_2(\text{conv}_1(x_{vis}) \odot Wx_{txt}))$
	FilterDiff		[att, att]	att	$a_{out} = \text{minimum}(a_1, a_1 - a_2)$
	Rel2[subj obj] And	✓	att [att, att]	att att	$a_{out} = \text{conv}_2(\text{conv}_1(x_{vis}) \odot W_1 \sum(a \odot x_{vis}) \odot W_2 x_{txt})$ $a_{out} = \text{minimum}(a_1, a_2)$
Logic	Exist		att	bool	$b_{out} = W_b^T \text{vec}(a)$
	Verify[Attr Pos]	✓	att	bool	$b_{out} = W_b^T (W_1 \sum(a \odot x_{vis}) \odot W_2 x_{txt})$
	VerifyRel, Same	✓	[att, att]	bool	$b_{out} = W_b^T (W_1 \sum(a_1 \odot x_{vis}) \odot W_2 \sum(a_2 \odot x_{vis}) \odot W_3 x_{txt})$
	SameAll	✓	att	bool	$b_{out} = W_b^T [W_1 \sum(a \odot x_{vis}) \odot W_2 x_{txt}, W_3 x_q]$
Inference	LogicNot		bool	bool	$P(b_{out}) = 1 - P(b)$
	LogicAnd		[bool, bool]	bool	$P(b_{out}) = P(b_1)P(b_2)$
	LogicOr		[bool, bool]	bool	$P(b_{out}) = 1 - (1 - P(b_1))(1 - P(b_2))$
Answer	AnswerLogic		bool	ans	$y_{out} = \text{padding}(b)$
	QueryName		att	ans	$y_{out} = W_y^T (W_1 \sum(a \odot x_{vis}) \odot W_2 x_q)$
	QueryPos		att	ans	$y_{out} = W_y^T (W_1 \sum(a \odot x_{vis}))$
	QueryAttr, Choose	✓	att	ans	$y_{out} = W_y^T (W_1 \sum(a \odot x_{vis}) \odot W_2 x_{txt})$
	ChooseRel	✓	[att, att]	ans	$y_{out} = W_y^T [W_1 \sum(a_1 \odot x_{vis}), W_2 \sum(a_2 \odot x_{vis})]$
	Common		[att, att]	ans	$y_{out} = W_y^T [W_1 \sum(a_1 \odot x_{vis}) \odot W_2 \sum(a_2 \odot x_{vis}), W_3 x_q]$

Figure 2.4: Modules architecture from [Li et al., 2019a].

mechanism. The experiments were conducted on the GQA dataset, and the set of modules in this context comprises a total of 48 modules.

Generator

Let’s now provide an overview of the various generators employed in NMN architectures.

Parser. The initial generator component from [Andreas et al., 2016b] relies on semantic linguistic parsers like the Stanford Parser [Klein and Manning, 2003] and the dependency parser [De Marneffe and Manning, 2008] to extract an abstract structure from the input question. This symbolic representation of the question then undergoes a sequence of pre-processing operations to shape it into a suitable network structure. For example, a question such as “What color is the cat?” is converted into a logical format like `color(cat)`, which is subsequently transformed into a network structure like `classify[color](attend[cat])`.

RNN. In an alternative approach to the parser-based generator, the generator employs a sequence-to-sequence (seq2seq) generation mechanism to learn a mapping of questions to module functions. This method leverages recurrent deep networks, akin to those utilized in natural language processing tasks like language translation. Specifically, the works of [Hu et al., 2017, Johnson et al., 2017b, Mascharka et al., 2018, Li et al., 2019a] implement an attention LSTM [Sutskever et al., 2014] to

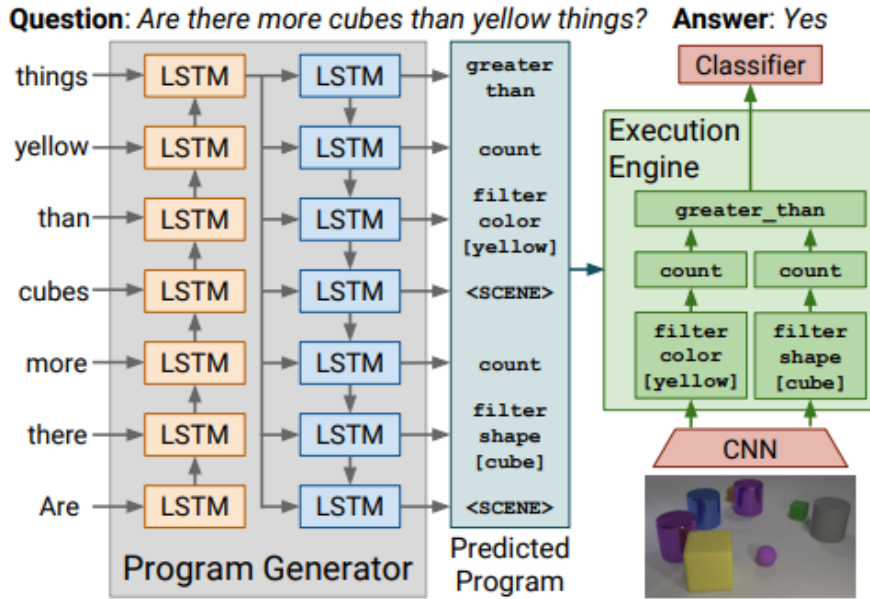


Figure 2.5: Model from [Johnson et al., 2017b]. Generator on the right and executor on the left.

convert the question into its corresponding function program sequence. Figure 2.5 represents the NMN model from [Johnson et al., 2017b]. On the left side of the image, the program generator, which is an LSTM model that inputs the question and decodes it into the program.

Transformer. In the research conducted by [Chen et al., 2021], they use a Transformer [Vaswani et al., 2017]. The set of argument tokens and the set of module tokens are merged to form the output domain vocabulary of the decoding process, and the program tokens and the arguments are inferred by the Transformer generator.

Executor

Across various proposals [Andreas et al., 2016b, Hu et al., 2017, Li et al., 2019a], the execution process follows a consistent pattern. The function program is used to call the different modules instantiation and assemble an NMN. This network is subsequently executed in a sequential manner on the image. Notably, the modules communicate with each other by passing messages, wherein the output of one module becomes the input for subsequent modules, creating a directed graph-like flow of information. This process culminates in the network delivering an answer to the posed question. Detailed NMN training strategies are discussed in the following section.

2.3 Executor training strategies

Jointly training the modules of a NMN for VQA using only the answer loss has shown promising results on synthetic datasets like CLEVR [Johnson et al., 2017a] and moderately complex questions from VQA datasets [Antol et al., 2015, Goyal et al., 2017]. However, it faces challenges when dealing with high-complexity and compositional questions, particularly those found in the GQA dataset [Hudson and Manning, 2019b]. To address these challenges, additional supervision and enhanced training processes are required. These training techniques will be discussed in the following subsections.

2.3.1 Knowledge guidance

To ensure that the modules within an NMN align with their specific sub-tasks, stay true to the reasoning process dictated by the functional program, and ultimately enhance the NMN’s performance, techniques involving knowledge guidance have been incorporated [Li et al., 2019a, Chen et al., 2021]. These techniques provide additional supervision to the modules during the **joint-training** process, enabling them to learn distinct functionalities while collaboratively executing complex reasoning sub-tasks.

The modules in NMN serve different roles, ranging from attention modules that pinpoint relevant image regions to boolean modules performing logical operations and answer modules responsible for classification. These modules are sequentially interconnected to form the NMN, allowing information to flow through this reasoning chain to arrive at a final answer. Knowledge guidance intervenes by offering targets for the modules at intermediate steps of the NMN, enabling the modules to optimize their parameters to better align with their respective subtasks.

For instance, in works like [Li et al., 2019a] and [Chen et al., 2021], knowledge guidance is derived from the scene graph and the question’s answer provided by the dataset. These methods involve pre-executing modular networks symbolically to identify intermediate targets.

In the case of [Li et al., 2019a], knowledge guidance involves matching target regions to the outputs of attention modules using a grid-based matching technique. Attention modules are then optimized to minimize the discrepancy between their outputs and the matched knowledge guidance. For boolean modules, their knowledge guidance comes from subsequent answer modules that are optimized to predict correct answers.

2.3. EXECUTOR TRAINING STRATEGIES

In [Chen et al., 2021], the forms of attention and boolean module outputs are unified, and knowledge guidance relies on Intersection over Union (IOUs) to measure the alignment between target regions and module outputs. A multi-layer fully connected network is employed to align the projections of module outputs with the knowledge guidance.

2.3.2 Teacher forcing

Teacher forcing (TF) [Williams and Zipser, 1989] is a widely used technique in sequence prediction or generation tasks, especially in RNNs with an encoder-decoder architecture. It involves training the model using the true output as novel input (2.6b), which helps improve prediction accuracy. However, during inference, the model relies on its own predictions without access to ground-truth information (2.6a), leading to a discrepancy known as “exposure bias”.

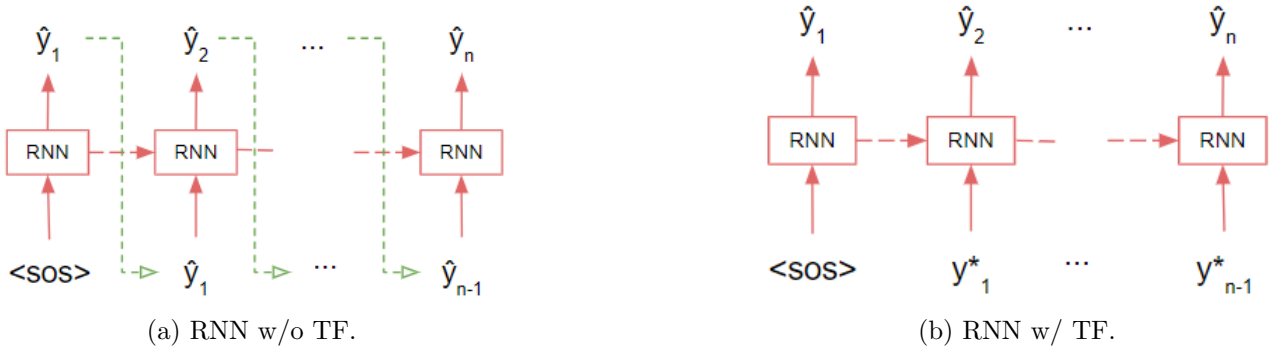


Figure 2.6: RNN without and with teacher forcing.

Scheduled sampling (SS) is a notable approach to mitigating the train-test discrepancy in sequence generation tasks [Bengio et al., 2015]. It introduces randomness during training by choosing between using ground truth tokens or the model’s predictions at each time step. This technique, initially developed for RNN architectures, has also been adapted for transformer networks [Mihaylova and Martins, 2019], aiding to align the model’s performance during training and inference.

NMNs, on the other hand, are trained using only the output of a module as input for the next module, which has drawbacks. Errors made by an intermediate module can propagate to subsequent modules, leading to cumulative bad predictions. This effect is particularly prominent during the early stages of training when the model’s predictions are close to random.

In Chapter 4 we show that NMNs can leverage the TF strategy to enhance their training process.

2.3. EXECUTOR TRAINING STRATEGIES

Initially, training begins with a fully guided schema, where the true previous targets are used as input. As training progresses, the model gradually transitions to a less guided scheme, relying more on the predicted outputs from previous steps as input. This gradual reduction in guidance and increased reliance on the model’s own predictions, named “decaying TF”, helps NMNs better learn and adapt to the complexity of the task. With decaying TF, modules can conform to their expected behavior for their respective sub-tasks.

2.3.3 Curriculum learning

Curriculum learning was introduced in [Elman, 1993] where the author shows that successful learning may depend on “starting small” by first learning a simple grammar with a recurrent network and then gradually learning more complex tasks such as relative clauses, number agreement, etc. CL was later applied to various machine learning tasks and recently adapted to textual question answering (QA) in [Liu et al., 2018]. The authors use a sampling function that gives higher selection weights to simple QA pairs and then, as the training advances, it selects more complex QA pairs. A *term frequency* selector and a *grammar* selector assess the difficulty of the training examples. In [Sachan and Xing, 2016], CL is reframed as a self-paced learning (SPL) algorithm [Kumar et al., 2010] and the question loss is taken as the measure of difficulty. The authors implement several heuristics reminding of active learning [Ren et al., 2021] in order to improve the SPL performance.

Curriculum Learning for VQA. The definition of relevant difficulty criteria to order the training examples of VQA is challenging and this may explain why there is little work on the use of CL for the VQA field.

Preliminary experiments conducted by [Lechat, 2021] on incremental VQA have underscored the challenge of establishing a clear and effective difficulty criterion for VQA tasks. While question length might appear as an intuitive measure of difficulty, their experiments utilizing the BUTD model [Anderson et al., 2018], trained on the GQA dataset, revealed an intriguing inverse correlation. Specifically, they found that on the longer questions, measured by the number of reasoning steps required, the model’s performance is better than the models’ performance on shorter questions. It’s worth noting that the BUTD model used in their experiments is not compositional, in contrast to the one employed in our work. Nevertheless, these findings emphasize the complexity of defining a robust and reliable CL strategy for VQA tasks.

The recent work in [Askarian et al., 2021] applies CL in a modular VQA context to the synthetic CLEVR dataset [Johnson et al., 2017a]. The base model is from [Johnson et al., 2017b], with an LSTM generator and generic residual blocks for the executor modules. The experiments were conducted on the executor alone, using as input the ground-truth programs directly. Several difficulty criteria were evaluated, including program length, answer hierarchy, and question loss. The results demonstrated that CL with a question loss difficulty criterion has a positive impact in a low data setting. However, the study in [Askarian et al., 2021] was focused on the CLEVR dataset [Johnson et al., 2017a] consisting of synthetic images of simple 3D objects, with a limited number of object classes and attributes. In our work (Chapter 5), we employ the GQA dataset that is based on real-world images with many object classes and attributes, as well as more complex relations and more challenging object detection. We thus have to completely redefine the candidate CL strategies.

2.4 Datasets for compositional visual reasoning

Training neural networks in a supervised manner demands a substantial collection of annotated training data. Various datasets have been introduced to serve as benchmarks for assessing a model’s capacity to answer questions related to images and perform visual reasoning tasks. These datasets include [Antol et al., 2015, Goyal et al., 2017, Kaffe and Kanan, 2017a, Malinowski and Fritz, 2014, Krishna et al., 2017, Johnson et al., 2017a, Hudson and Manning, 2019b, Ren et al., 2015a].

This thesis concentrates on compositional questions that demand a multi-step reasoning process, particularly for complex questions. In this context, we will provide details about several notable datasets: an early VQA dataset [Antol et al., 2015], a synthetic image dataset known as CLEVR [Johnson et al., 2017a], and a dataset encompassing real-world images [Hudson and Manning, 2019b], which we utilize in our research.

2.4.1 VQA and VQA 2.0

The VQA dataset [Antol et al., 2015, Goyal et al., 2017] was one of the early datasets created for the task of VQA. It consists of real images from the COCO (Common Objects in Context) dataset [Lin et al., 2014] paired with questions about the content of those images. The goal is to develop models that can understand both the visual content of an image and the textual content of a question in

2.4. DATASETS FOR COMPOSITIONAL VISUAL REASONING

order to provide accurate answers. Figure 2.7 provides examples of images and their related questions that can be found in the VQA dataset.

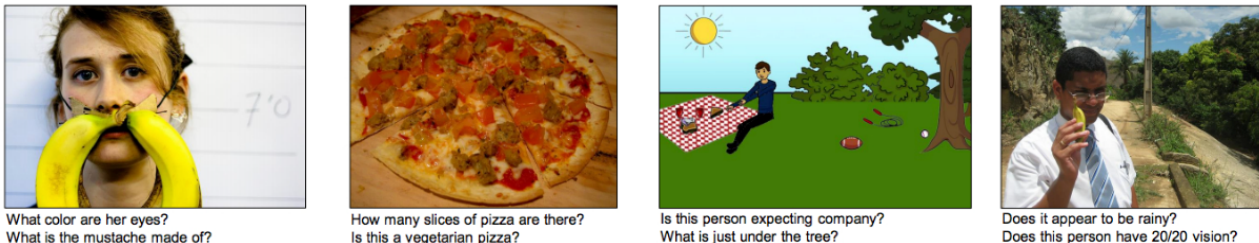


Figure 2.7: VQA dataset examples.

VQA 2.0 [Goyal et al., 2017] is an extended version of the original VQA dataset [Antol et al., 2015]. It addresses some of the limitations of the first dataset, including language biases in the questions that allowed models to answer correctly without understanding the images. For instance, questions that start with the n-gram “Do you see a...” often result in models blindly answering “yes” without considering the rest of the question or even looking at the image. This alone could lead to a high VQA accuracy of 87%, which clearly indicated a problem. Another example highlighting this bias is the question, “What covers the ground?”. A model could easily answer correctly not because it comprehended the scene but because the dataset frequently posed questions about the ground when it was snow-covered. The issue of dataset bias was raised by several works including [Agrawal et al., 2016, Kafle and Kanan, 2017b, Cadene et al., 2019]. VQA 2.0 contains more balanced and less biased questions, making it a more challenging benchmark for VQA systems. It also has a larger number of questions and answers. While datasets like VQA and VQA 2.0 are widely used in the field of VQA, they are not ideally suited for compositional reasoning approaches like ours. This is because our approach necessitates a structured program that explicitly represents the sequence of the reasoning steps required to answer questions in a modular and interpretable manner. These traditional datasets typically lack such fine-grained programmatic representations of the reasoning processes.

2.4.2 CLEVR

The CLEVR (Compositional Language and Elementary Visual Reasoning) dataset, introduced in [Johnson et al., 2017a], is a popular dataset used for evaluating the visual reasoning and the question-answering capabilities of models. This dataset consists of synthetic images containing simple 3D objects and scenes, along with questions that require reasoning about the objects and their

2.4. DATASETS FOR COMPOSITIONAL VISUAL REASONING

relationships.

The CLEVR dataset stands as a pioneering initiative, being the first public dataset to introduce functional programs as a means to explicitly represent the required reasoning steps for answering questions. Its primary objective is to assess the reasoning capabilities of VQA systems and their capacity to really understand a scene, rather than relying on dataset biases for answers. CLEVR’s questions encompass a broad spectrum of visual reasoning challenges, including tasks like attribute identification, counting, comparison, handling multiple attention, and logical operations. The dataset features three object types: cubes, spheres, and cylinders, each characterized by two distinct sizes (“small” and “large”), two material properties (shiny “metal” and matte “rubber”), and a palette of eight colors. Moreover, objects within CLEVR scenes exhibit spatial relationships defined by four key orientations: “left”, “right”, “behind”, and “in front”. The images, questions and functional programs are generated based on randomly sampled scene graphs. Figure 2.8 showcases an example image from the CLEVR dataset and two questions with their functional programs. A function catalog on the right of the image, represents the set of the atomic operations used for the reasoning skills.

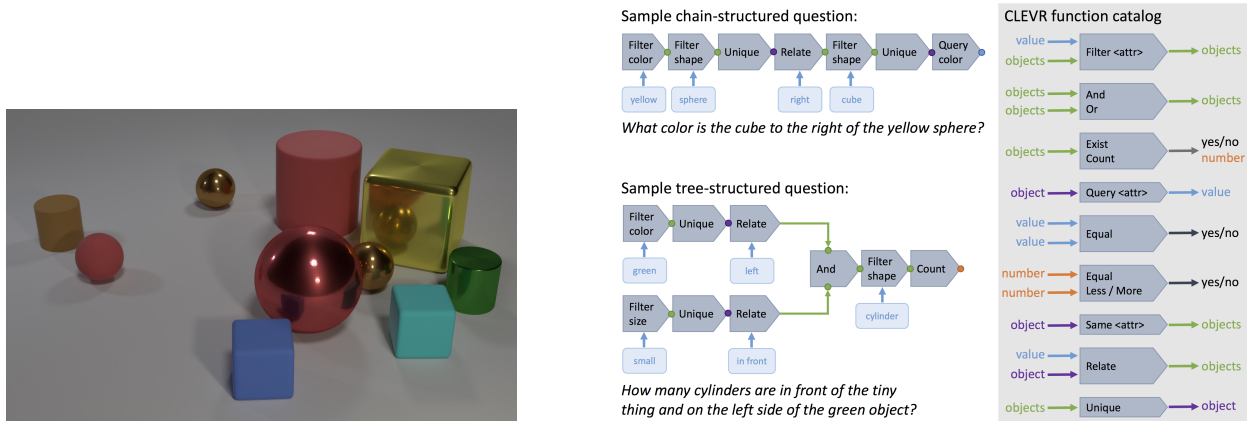


Figure 2.8: CLEVR Example: Image on the Left, Functional Program and Question in the center and function catalog in the right.

As shown in the top table of Figure 2.9, the dataset is composed of various key elements. This includes a training set that contains 70,000 images, accompanied by 699,989 questions. Additionally, there is a validation set comprising 15,000 images with 149,991 corresponding questions. The test set, used for evaluation, consists of 15,000 images and 14,988 questions. Answers are provided for all questions in both the training and validation sets. Furthermore, scene graph annotations are

2.4. DATASETS FOR COMPOSITIONAL VISUAL REASONING

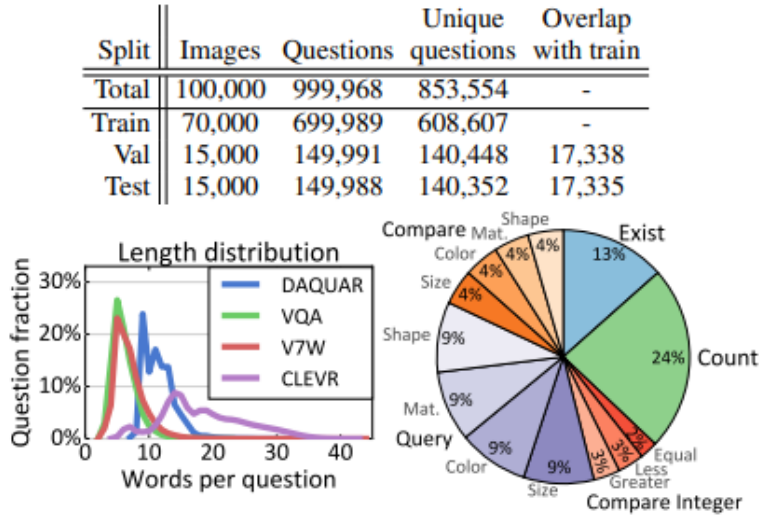


Figure 2.9: Top: Statistics for CLEVR per split. Bottom left: Comparison of question lengths for different VQA datasets. Bottom right: Distribution of question types in CLEVR.

available for the training and validation images, providing ground-truth details about object locations, attributes, and relationships. Notably, functional program representations are included for all training and validation images, although this information is not provided for the test set.

2.4.3 GQA

The Question Answering on Image Scene Graphs (GQA) dataset [Hudson and Manning, 2019b] features compositional questions over real-world images. GQA is widely used for benchmarking and advancing the capabilities of VQA models, particularly those that employ compositional reasoning and complex language understanding. It’s considered one of the standard datasets in the field of VQA.

The GQA dataset builds upon the Visual Genome dataset [Krishna et al., 2017], refining the scene graphs associated with the images. Visual Genome provides a structured representation of images based on their objects, attributes, and their relations. Object locations are delineated using bounding boxes, contributing to a richer understanding of the image content.

The image graphs serve as inputs for a question engine, which generates questions and functional programs. These programs outline a sequence of steps representing the reasoning process required to answer the questions. Figure 2.10, extracted from [Hudson and Manning, 2019b], illustrates an example of the data provided by the GQA dataset. In the center of the figure, the question engine combines scene graphs with predefined structural patterns and program templates. Additionally, it

2.4. DATASETS FOR COMPOSITIONAL VISUAL REASONING

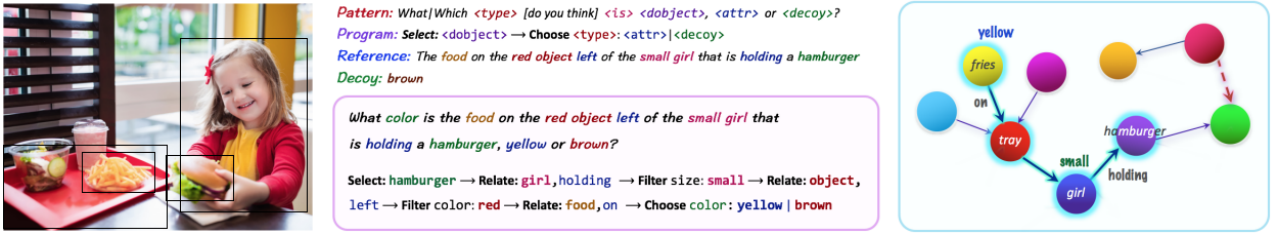


Figure 2.10: GQA Example: Image on the Left, Functional Program and Question on the middle and image graph on the right.

utilizes synonyms and alternate expressions to generate a diverse set of questions.

Type	Open/Binary	Semantic	Structural	Form	Example
queryGlobal	open	query	global	select: scene/query: type	How is the weather in the image?
verifyGlobal	binary	verify	global	select: scene/verify type: attr	Is it cloudy today?
chooseGlobal	open	query	global	select: scene/choose type: a b	Is it sunny or cloudy?
queryAttr	open	query	attribute	select: obj/.../query: type	What color is the apple?
verifyAttr	binary	verify	attribute	select: obj/.../verify type: attr	Is the apple red?
verifyAttrs	binary	logical	attribute	select: obj/.../verify t1: a1/verify t2: a2/and	Is the apple red and shiny?
chooseAttr	open	choose	attribute	select: obj/.../choose type: a b	Is the apple green or red?
exist	binary	verify	object	select: obj/.../exist	Is there an apple in the picture?
existRel	binary	verify	relation	select: subj/.../relate (rel): obj/exist	Is there an apple on the black table?
logicOr	binary	logical	object	select: obj1/.../exist/select: obj2/.../exist/or	Do you see either an apple or a banana there?
logicAnd	binary	logical	obj/attr	select: obj1/.../exist/select: obj2/.../exist/and	Do you see both green apples and bananas there?
queryObject	open	query	category	select: category/.../query: name	What kind of fruit is on the table?
chooseObject	open	choose	category	select: category/.../choose: a b	What kind of fruit is it, an apple or a banana?
queryRel	open	query	relation	select: subj/.../relate (rel): obj/query: name	What is the small girl wearing?
verifyRel	binary	verify	relation	select: subj/.../verifyRel (rel): obj	Is she wearing a blue dress?
chooseRel	open	choose	relation	select: subj/.../chooseRel (r1 r2): obj	Is the cat to the left or to the right of the flower?
chooseObjRel	open	choose	relation	select: subj/.../relate (rel): obj/choose: a b	What is the boy eating, an apple or a slice of pizza?
compare	binary	compare	object	select: obj1/.../select: obj2/.../compare type	Who is taller, the boy or the girl?
common	open	compare	object	select: obj1/.../select: obj2/.../common	What is common to the shirt and the flower?
twoSame	verify	compare	object	select: obj1/.../select: obj2/.../same	Does the shirt and the flower have the same color?
twoDiff	verify	compare	object	select: obj1/.../select: obj2/.../different	Are the table and the chair made of different materials?
allSame	verify	compare	object	select: allObjs/same	Are all the people there the same gender?
allDiff	verify	compare	object	select: allObjs/different	Are the animals in the image of different types?

Figure 2.11: GQA Examples based on their structural and semantic types

To showcase the diversity of the generated compositional questions, Figure 2.11 presents examples of questions along with their type annotations. These questions possess both semantic and structural types. The structural type is inferred from the final operation in the question’s functional program, while the semantic type pertains to the primary subject of the question. Additionally, each question has a detailed type, which combines its semantic and structural types, as represented in the first column of the table in Figure 2.11.

GQA features over 18M compositional questions and 113K real-world images. These questions encompass a multitude of reasoning skills and demand various multi-hop reasoning steps. They also exhibit varying lengths, determined by the number of words used in their composition. Relevant

2.4. DATASETS FOR COMPOSITIONAL VISUAL REASONING

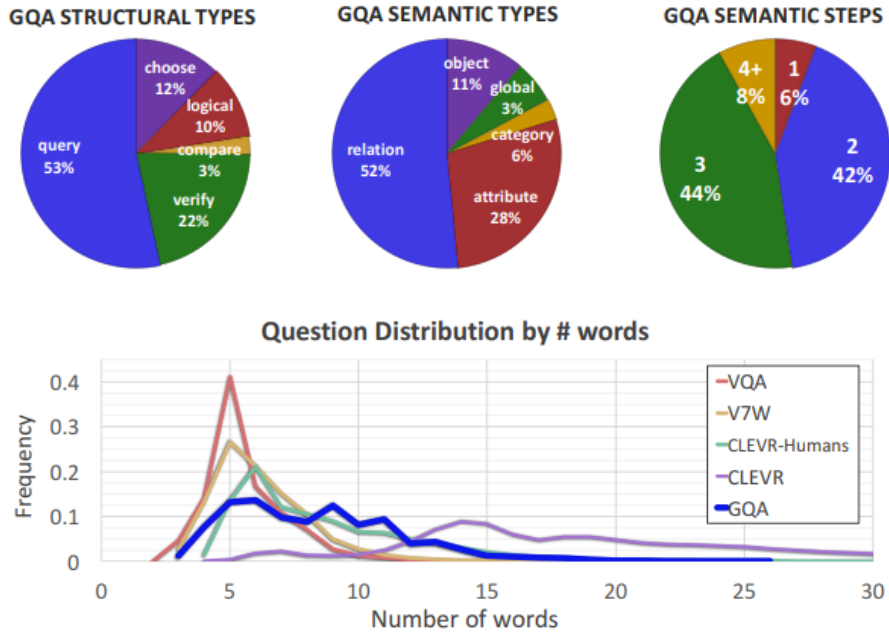


Figure 2.12: Dataset statistics from [Hudson and Manning, 2019b]: structural types, semantic types, number of reasoning steps and questions length distribution compared to other datasets.

Split	Train	Val	Testdev	Test	Challenge
Balanced	943.000	132.062	12.578	95.336	50.726
Unbalanced	14.305.356	2.011.853	172.174	1.340.048	713.449

Table 2.1: GQA dataset partitioning

statistics regarding these questions are available in Figure 2.12. The dataset has two versions: an unbalanced version and a more balanced one resulting from the application of a tunable smoothing technique. This technique works to equalize the answer distribution across question groups, resulting in a more uniform dataset. Table 2.1 reports the number of examples per split. For the unbalanced version, the `train-all` split has over 14M examples and the `testdev-all` has 172,174 examples, while the balanced version has over 943,000 examples for `train`, 132,062 for `val` and a `testdev` of 12,578 examples.

Regarding the provided programs for the balanced training and validation splits, there are a total of 136 distinct functions, with a combined total occurrence count of 3,365,899. Table 2.2 displays the module counts in descending order. The most frequently occurring module is `select`, with 1,207,994 instances in the dataset, accounting for 35.9% of the total module occurrences. It’s responsible for identifying the bounding box of a specified object.

2.4. DATASETS FOR COMPOSITIONAL VISUAL REASONING

Function	Count	Freq	Function	Count	Freq	Function	Count	Freq
select	1207994	35.9	filter pattern	1560	0.05	filter depth	118	0.004
query	556119	16.5	choose place	1557	0.05	choose thickness	112	0.003
relate	555482	16.5	different	1434	0.04	filter room	108	0.003
exist	288378	8.6	verify pattern	1343	0.04	verify race	103	0.003
filter color	130407	3.9	filter sportActivity	1281	0.04	filter liquid	100	0.003
or	77120	2.3	verify place	1221	0.04	verify gender	100	0.003
verify rel	64456	1.9	verify cleanliness	945	0.03	verify realism	93	0.003
and	53005	1.6	choose weather	890	0.03	verify weight	79	0.002
verify color	52559	1.6	filter cleanliness	726	0.02	verify company	73	0.002
choose rel	42939	1.3	verify state	722	0.02	choose age	69	0.002
filter	23870	0.7	choose younger	712	0.02	filter realism	63	0.002
filter material	23492	0.7	verify activity	673	0.02	filter weight	62	0.002
filter hposition	22402	0.7	verify thickness	654	0.02	choose width	61	0.002
choose vposition	19413	0.6	filter sport	653	0.02	filter company	57	0.002
choose color	18895	0.6	choose older	647	0.02	choose face expression	55	0.002
filter size	17099	0.5	choose length	636	0.02	filter orientation	49	0.001
verify hposition	16710	0.5	choose activity	583	0.02	choose tone	45	0.001
filter vposition	16390	0.5	choose pose	549	0.02	same shape	44	0.001
choose hposition	16230	0.5	filter weather	546	0.02	different shape	42	0.001
choose name	15450	0.5	filter state	519	0.02	choose depth	34	0.001
verify size	10787	0.3	choose height	438	0.01	choose gender	32	0.001
same color	10068	0.3	verify age	435	0.01	verify type	26	0.001
different color	10049	0.3	verify tone	429	0.01	choose race	25	0.001
verify	10033	0.3	filter thickness	343	0.01	filter event	23	0.001
filter pose	9794	0.3	choose healthier	312	0.01	choose fatness	23	0.001
verify vposition	7176	0.2	verify sportActivity	303	0.01	verify flavor	16	0.0005
common	5670	0.2	filter tone	293	0.01	choose company	13	0.0004
verify material	5419	0.2	filter flavor	293	0.01	choose weight	13	0.0004
choose	5391	0.2	choose less healthy	285	0.01	verify texture	9	0.0003
verify location	4590	0.1	filter gender	262	0.01	choose flavor	9	0.0003
verify length	3816	0.1	verify width	255	0.01	choose opaqnss	8	0.0002
verify weather	3729	0.1	filter width	252	0.01	choose taller	4	0.0001
filter activity	3603	0.1	verify face expression	242	0.01	verify room	4	0.0001
choose location	3125	0.1	filter opaqnss	221	0.01	choose shorter	4	0.0001
choose material	3121	0.1	choose cleanliness	220	0.01	filter brightness	3	0.0001
filter shape	3097	0.1	choose shape	216	0.01	verify brightness	3	0.0001
filter age	2687	0.1	filter fatness	212	0.01	filter texture	3	0.0001
filter height	2610	0.1	choose sportActivity	195	0.01	choose hardness	2	0.0001
verify shape	2592	0.1	filter race	175	0.01	choose smaller	2	0.0001
same	2272	0.1	verify fatness	153	0.005	choose larger	1	0.00003
verify height	2271	0.1	verify opaqnss	150	0.004	choose brightness	1	0.00003
same material	2117	0.1	choose pattern	138	0.004	choose lower	1	0.00003
filter length	2054	0.1	choose state	137	0.004	choose realism	1	0.00003
choose size	2030	0.1	verify hardness	133	0.004	choose higher	1	0.00003
verify pose	1936	0.1	verify depth	131	0.004			
filter face expression	1563	0.05	filter hardness	121	0.004			

Table 2.2: GQA dataset function counts and percentage of frequencies in descending order.

2.4. DATASETS FOR COMPOSITIONAL VISUAL REASONING

Some functions exhibit a very low level of granularity, making them highly specific with very few occurrences, such as `choose realism` and `choose higher` that only appear one time in the programs. Instead of removing these infrequent functions, a pre-processing step is applied to group them into higher granularity functions. For instance, `choose` functions can be aggregated into broader categories like `choose relation` and `choose attributes`. Additional details regarding this function preprocessing can be found in Section 3.3.1.

Model	validation	testdev
BAN [Kim et al., 2018]	61.5	55.2
CTI [Do et al., 2019]	61.7	54.9
MCAN [Yu et al., 2019]	-	57.4
MMN [Chen et al., 2021]	-	60.4
NMS [Hudson and Manning, 2019a]	-	63.2
HAN [Kim et al., 2020]	-	69.5
LXMERT [Tan and Bansal, 2019]	59.8	60.0
OSCAR [Li et al., 2020]	-	61.6
CFR [Nguyen et al., 2022]	73.6	72.1

Table 2.3: Accuracy results on the GQA dataset validation and testdev splits from [Nguyen et al., 2022].

GQA is the testbed for several visual reasoning architectures, including the ones mentioned above as well as others like graph neural networks-based architectures [Kim et al., 2020]. In general, we observe that monolithic approach tend to have better performances than NMN-based approaches. Yet, they lack the step-by-step explicit reasoning offered by NMNs. Table 2.3 showcases the performance of state-of-the-art models on the GQA dataset [Hudson and Manning, 2019b].

For additional details about the dataset, we refer the reader to the GQA website.

Chapter 3

Multi-modal Neural Module networks

Contents

3.1	Introduction	30
3.2	Multi-modal representations	31
3.3	Neural modules	34
3.3.1	Pre-processing	35
3.3.2	Attention modules	38
3.3.3	Boolean modules	39
3.3.4	Answer modules definition	41
3.4	Program examples	42
3.5	Generator	48
3.5.1	From questions in natural language to functional programs	48
3.5.2	Arguments prediction	49
3.5.3	Generator optimisation	50
3.5.4	Program inference	50
3.5.5	Experimental validation	52
3.6	Executor	52
3.6.1	Module initialisation	52
3.6.2	Weight sharing	53
3.6.3	Modular network instantiation	54
3.6.4	Reasoning process	54
3.6.5	Answer prediction	55
3.7	Evaluations	55
3.7.1	Experimental settings	56
3.7.2	Unimodal vs crossmodal representations	56
3.8	Conclusion	58

3.1 Introduction

Visual reasoning models face the challenge of effectively reasoning about complex scenes in a transparent manner. While recent state-of-the-art models [Tan and Bansal, 2019, Lu et al., 2019b, Li et al., 2019b] leverage attention-based mechanisms and some even incorporate multi-step attention-based reasoning [Hudson and Manning, 2018, Perez et al., 2018], they often lack the modular aspect that characterizes human-like reasoning [Fodor, 1983, Clune et al., 2013].

To address this limitation, we focus on Multi-modal Neural Module Networks (Multi-modal NMN), combining two essential components. The first component involves the utilization of multi-modal representations, which allows for the integration of language and vision representations to capture their relationship using pretrained cross-modal encoders. The second component involves the implementation of neural module networks, which enable modular reasoning on the cross-modal representations and decomposes complex tasks into interpretable and reusable components.

As mentioned in the previous section, our model takes an image, a question, and a program triplet as input and predicts an answer. We extract aligned language and vision features for the image and question using a cross-modal transformer [Tan and Bansal, 2019]. The program, represented as a sequence of modules, is used to build an NMN, which is then executed on the image to answer the question.

Throughout this chapter we describe the core architecture of our model (Figure 3.1). We begin in Section 3.2 by explaining the process of extracting multi-modal representations and how we adapt them to our modular approach. We also detail the pre-processing steps applied to the functional programs from the GQA dataset [Hudson and Manning, 2019b] in Section 3.3.1, which involves consolidating infrequent functions into a condensed dictionary. In Section 3.3 we define the functional dictionary of our modular approach. To translate the question to a program, we use a generator described in Section 3.5. Then, in Section 3.6 we explain how the executed program operates on the image to produce the final answer. Finally, we proceed to evaluation experiments to compare the impact of multi-modal vs. unimodal representations on our NMN architecture in Section 3.7.

3.2. MULTI-MODAL REPRESENTATIONS

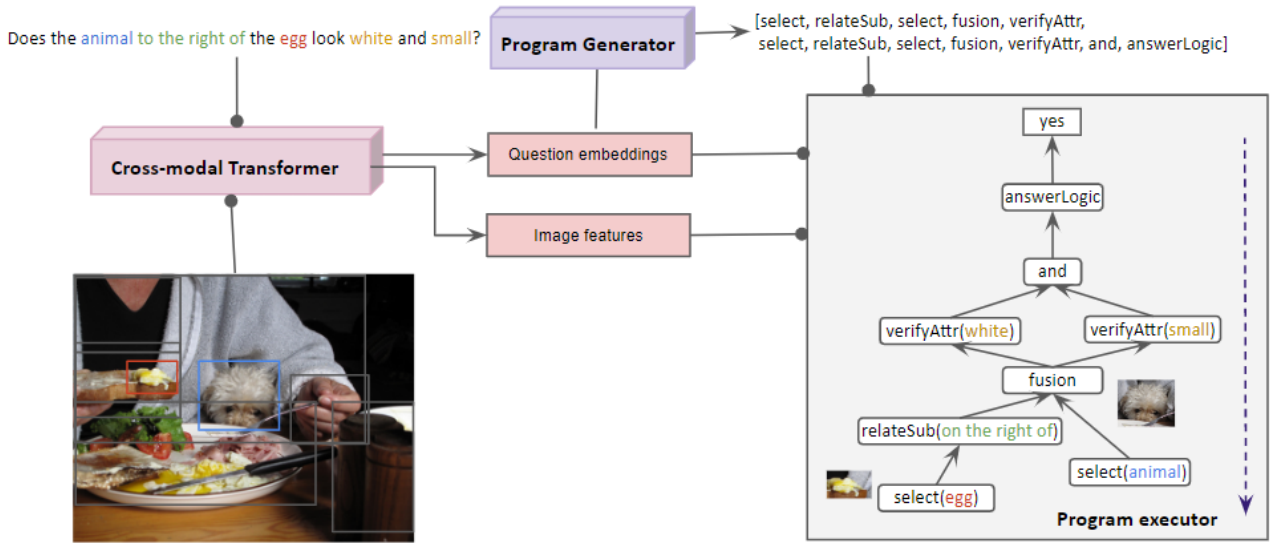


Figure 3.1: The proposed modular VQA framework: The (question, image) pair is used by a transformer model to generate aligned cross-modal embeddings for words and objects (3.2). These are used by a Program Generator (3.5) module to produce a program (represented as a sequence of sub-task modules), which will be then applied by the Program Executor (3.6) module to the image to answer the question.

3.2 Multi-modal representations

Compositional visual reasoning aims to perform logical and geometrical inferences involving several related objects in a complex scene. Under the modular VQA paradigm, we handle three types of inputs: Question (Q), Image (I), and Program (P). In this section, we focus on the encoding of the questions and the images.

We want to incorporate a compositional aspect into the representation of questions and images. As opposed to holistic representations where the question is condensed into a single hidden vector, typically achieved by aggregating word embeddings from models like word2vec [Mikolov et al., 2013] or fasttext [Bojanowski et al., 2016], or by employing more advanced techniques such as recurrent neural networks like GRU [Cho et al., 2014] or LSTM [Sutskever et al., 2014]. Simultaneously, the image representation typically involves using CNN spatial feature maps, like those extracted by popular models such as VGG [Simonyan and Zisserman, 2015] or ResNet [He et al., 2016]. In our approach, we represent the question using its word embeddings and the image using its object descriptors. The utilization of image region-based features known as “bottom-up features”, have shown to significantly boost VQA performance, as demonstrated in previous studies [Anderson et al., 2018].

3.2. MULTI-MODAL REPRESENTATIONS

To obtain region-based representations for the images, we employ the features provided by Faster R-CNN [Ren et al., 2015b], an object detection model designed to recognize objects from specific classes and determine their locations by bounding box detection. The pipeline of this region proposal-based framework can be summarized as follows: 1) a region proposal network (RPN) generates region proposals identified by their bounding box coordinates, and 2) an object detection network, namely Fast-RCNN [Girshick, 2015], uses the region proposals to label the objects. These two components are based on a shared convolutional neural network such as [He et al., 2016, Simonyan and Zisserman, 2015, Zeiler and Fergus, 2014]. It is worth noting that the provided features in the GQA dataset are based on the Faster-RCNN with ResNet-101 version.

The question word embeddings can be categorized into two major groups, as illustrated in the “embeddings taxonomy” figure 3.2 sourced from [Wang et al., 2021]. The first category comprises context-independent representations. In this category, each word possesses a unique learned vector representation that remains consistent regardless of the specific textual context in which the word is used during feature extraction. Notable examples of this category include word2vec [Mikolov et al., 2013], GloVe [Pennington et al., 2014], and FastText [Bojanowski et al., 2016]. The second category encompasses context-dependent representations. Here, the representation of a word is influenced by the textual context that surrounds it during feature extraction. This category includes embeddings based on Recurrent Neural Networks (RNN) [Sutskever et al., 2014], such as CoVe [McCann et al., 2017] and ELMo [Sarzynska-Wawer et al., 2021], as well as Transformer-based embeddings like BERT [Devlin et al., 2019], ALBERT [Lan et al., 2020] and GPT4 [OpenAI, 2023].

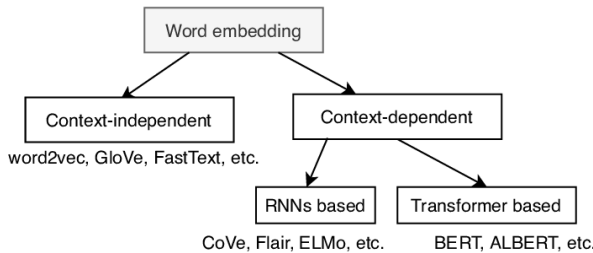


Figure 3.2: A-taxonomy-of-word-embeddings, from [Wang et al., 2021]

The VQA task is inherently multi-modal, necessitating an understanding of both textual and visual information. It involves the comprehension of language semantics, grounding of concepts within images, and the consideration of cross-modal interactions. This task requires an understanding of the

3.2. MULTI-MODAL REPRESENTATIONS

relationships within each individual modality as well as an alignment between different modalities.

Recent research has addressed the challenge of Vision and Language Pre-training (VLP) with the goal of introducing a cross-modal foundation for encoding both textual and visual information. This approach builds upon the previously described unimodal encoding techniques for language and vision, incorporating an element that merges cross-modal information. State-of-the-art VLP models have yielded substantial improvements across various tasks, including image captioning [Zhou et al., 2020], Image-Text-Matching (ITM) [Yang et al., 2022], and Visual Question Answering (VQA). For instance, in the VQA domain, models like ViLBERT [Lu et al., 2019b], VisualBERT [?], and LXMERT [Tan and Bansal, 2019] achieved good performance on widely recognized VQA datasets, such as VQA2.0 [Goyal et al., 2017] and GQA [Hudson and Manning, 2019c].

Capitalizing on the advancements and performance of publicly accessible pre-trained VLP models, we employ a transfer learning strategy that leverages such models as the backbone for encoding both text and image data. This approach offers the dual benefits of reducing computational cost and eliminating the need to train a feature extractor from scratch. Moreover, it allows us to benefit from the knowledge and information embedded within these pre-trained representations, enabling us to concentrate our efforts on the downstream task of modular reasoning.

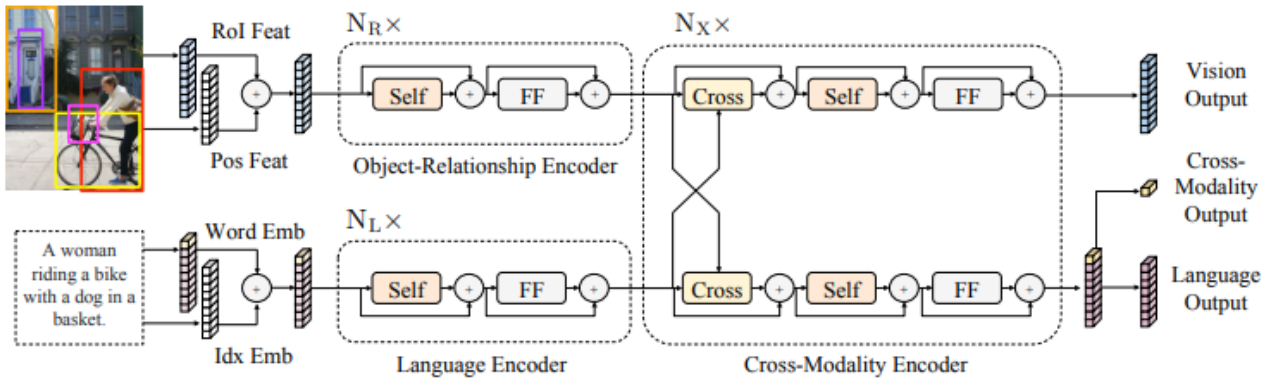


Figure 3.3: LXMERT architecture [Tan and Bansal, 2019], self: self-attention, FF: feed forward, cross: cross-attention.

In this study, we utilize LXMERT [Tan and Bansal, 2019] as our framework for extracting cross-modal language and vision representations (Figure 3.3). LXMERT is a transformer model pretrained on various unimodal and multi-modal tasks, including masked cross-modality predictions, masked object predictions, cross-modality matching, and VQA. LXMERT has demonstrated good performance

3.3. NEURAL MODULES

across various tasks and the fine-tuned model on GQA is publicly available. For the representation of images, LXMERT takes advantage of Faster-RCNN [Ren et al., 2015b] object regions, effectively capturing the visual information. Furthermore, word embeddings are learned in a manner similar to BERT [Devlin et al., 2019], providing a contextual representation of textual data. It is worth mentioning that we only use the cross-modality encoder representations and discard the answer classification component. More precisely, we freeze LXMERT weights and we pass the image I through the object-relationship encoder and the question Q through the language encoder. Then, the Cross-modality Encoder aligns the representations using co-attention mechanism to finally output the object bounding box features v_j of each object o_j in the image I and the embedding h_i of each word q_i in the question Q . The LXMERT encoders operate with a hidden dimension of 768 and the Faster-RCNN object detection system consistently identifies 36 objects. The language output is a matrix L of size $(max_len \times 768)$ where max_len is the maximum question length in the dataset. Each row in this matrix corresponds to the word embeddings of a particular word in the question. The vision output is a matrix V of size (36×768) each line represents the features of an object region identified in the image.

3.3 Neural modules

Our NMN approach tackles complex reasoning tasks by decomposing them into simpler sub-tasks, inspired by visual reasoning skills like object detection, attribute identification, object relation recognition, comparison, logical inference, etc.

Modules are designed to be functional and generic, each module has dependencies d_m to get information from the previous one(s) and arguments a_m to condition its behavior.

To allow the flow of information and enhance effective collaboration among modules in the reasoning chain, we utilize dependencies. This enables the output of a given module to serve as input to the next modules in the chain. Modules can exhibit different dependency patterns based on their position in the reasoning chain. Modules at the beginning of the chain have no dependencies, while unary modules rely on a single dependency and binary modules have two dependencies. These dependency patterns allow the modules to interact and build upon each other’s output, fostering a coherent and dynamic reasoning process. The management of the modules’ dependencies is addressed in Section 3.6.

3.3. NEURAL MODULES

To make the modules generic and have them adapt their behavior based on the specific context, we use a generic type modules as described in Section 2.2.2.2. These modules employ textual arguments that determine their specific behavior. These textual arguments act as “queries”, similar to how attention mechanisms use “queries” to determine which information to focus on.

We developed a library of modules tailored to address specific sub-tasks. These modules are designed to be intuitive and interpretable, using simple building blocks like dot products and MLPs. They are categorized into three groups based on their output type: attention, boolean, and answer modules. In the following subsections, we discuss the preprocessing of the GQA dataset modules and provide detailed explanations for each of the modules types.

3.3.1 Pre-processing

The purpose of pre-processing the GQA dataset programs is to create a more manageable and concise catalog. As mentioned in Section 2.4, the original dataset has 136 functions (Table 2.2) that lack generality, with some modules being exceptionally rare for the model to be able to learn their functionalities. To address this, we define a shorter list of 29 modules by grouping similar modules under more generic ones, the resulting modules functions and their counts are in Table 3.1. Each resulting module possesses its own distinct canonical structure and definition, implying that we can only merge functions that correspond to the same canonical structure in terms of dependencies and arguments.

For attention modules (details in Section 3.3.2), we start by clustering them based on their functional types: `select`, `filter`, and `relate`. Subsequently, we proceed with their preprocessing. We maintain the `select` module in its original format. However, the `filter` module has several specific instances that are attribute related such as `filter color`, `filter size`, `filter material`, `filter width`, and more. We group all these `filter` instantiations under the `filter attribute` module. Moreover, the `filter` modules related to spatial positions, such as `filter vposition` and `filter hposition`, are collectively categorized as `filterPos`. Lastly, `filter` modules that involve negation in their arguments, like `filter(Not red)`, are combined into the `filterNot` module.

The `Relate` modules needed refinement to distinguish the direction of the relations and grasp the semantic nuances between the subjects (the doer of an action) and the objects (the entity receiving the action). In scenarios where the subject and the relation are given, the `RelateObj` needs to focus

Module	count	frequency
select	1776383	37.3
fusion	433842	9.1
relateSub	413964	8.7
queryName	388118	8.2
answerLogic	378648	8.0
exist	288378	6.1
filterAttr	203294	4.3
relateObj	148103	3.1
queryAttr	115167	2.4
verifyAttr	106027	2.2
or	77120	1.6
verifyRelObj	56434	1.2
and	53005	1.1
queryPos	52834	1.1
chooseRel	42939	0.9
filterPos	38792	0.8
chooseAttr	38624	0.8
choosePos	35643	0.7
filterNot	27863	0.6
verifyPos	23886	0.5
chooseName	15450	0.3
same	12229	0.3
different	10091	0.2
verifyRelSub	8022	0.2
common	5670	0.1
relateAttr	3012	0.1
sameAll	2272	0.05
compare	1969	0.04
differentAll	1434	0.03

Table 3.1: Module functions counts and frequencies after preprocessing.

on the object of the relation, a question example is “What is the girl wearing?”. Conversely, when provided with the object and the relation, the `RelateSub` must focus on the subject (“Who is wearing a dress?”). This separation of concerns has also been explored in prior research, as demonstrated in works such as [Li et al., 2019a, Chen et al., 2021]. This essential information is extracted from the unprocessed modules arguments. Additionally, another category of undirected relations is identified by the `RelateAttr` modules, which inquire about objects sharing common attributes like color or material with a given object. For instance, “What is the name of the object having the same color as the cat?”.

3.3. NEURAL MODULES

A new module function, labeled **fusion**, was developed to address a particular type of questions where a function is needed to merge two object inputs. For instance, consider the question, “What is the name of the vehicle that is made of the same material as the lock?”. In this scenario, the NMN must classify the object’s name, which is a vehicle and shares the same material as the lock.

For the boolean modules described in Section 3.3.3, we retain the **and**, **or** and **exist** modules in their original format. However, we differentiate the **same** and **different** modules based on the number of input objects they process. As such, the **same** and **different** modules now specifically handle comparisons involving two input objects, while **sameAll** and **differentAll** modules have been introduced to accommodate comparisons between multiple input objects.

Furthermore, when it comes to the **verify** modules, we followed a similar preprocessing strategy as employed for the **relate** modules. Attribute-related **verify** modules are collectively grouped under **verifyAttr**, spatial position **verify** modules are consolidated under **verifyPos**, and **verify** modules dealing with object relations are categorized into **verifyRelObj** and **verifyRelSubj** modules. This categorization is based on the direction of the relation being examined.

For answer modules (detailed in Section 3.3.4), we keep the **common** module as it is. Within the **choose** modules category, we organize them based on the type of argument they address: ‘attribute’, ‘position’, ‘name’, and ‘relation’. Specifically, **choose hposition** and **choose vposition** modules are grouped under **choosePos**. For modules related to attributes, such as **choose location**, **choose material**, and **choose size**, they fall under the **chooseAttr** category. **Choose** modules associated with object categories are consolidated within the **chooseName** module. Each of these modules taking a single object as input. The **choose** modules that involve two objects and query about the relation type between these two objects are gathered under **chooseRel**. Additionally, we introduce a distinct **choose** module labeled **compare**. This module is designed for the purpose of comparing two input objects based on a specified attribute, encompassing functionalities such as **chooseHealthier** or **chooseOlder**. In the case of **query** modules, we’ve made it more structured. Attribute-related **query** modules are now grouped under **queryAttr**, object category **query** modules are unified under **QueryName**, and spatial position **Query** modules are consolidated into **QueryPos**. To project the outputs from the boolean modules into the answer vocabulary, we’ve introduced a new module called **answerLogic**.

This grouping approach helps to simplify the programs and reduces redundancy by consolidating functions with similar behaviors. Additionally, it aids in handling rare modules more efficiently by

3.3. NEURAL MODULES

bringing them together under a common module. The result is a streamlined set of modules that are easier to interpret and work with for a better reasoning and understanding of the data.

Indeed, the final modules list is inspired by previous works such as [Li et al., 2019a] and [?]. While these previous works serve as valuable references and inspiration, the module list in our study also incorporates some distinct design choices. It is worth emphasizing that there is no unique or definitive way to define the module dictionary.

3.3.2 Attention modules

Name	Dependencies	Definition
Select	–	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{Y} = r(\mathbf{WV}),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{Y} \odot \mathbf{x}))$
FilterAttr	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{Y} = r(\mathbf{WV}), \mathbf{z} = \sigma(\mathbf{W}(\mathbf{Y} \odot \mathbf{x})),$ $\mathbf{o} = \min(\mathbf{a}, \mathbf{z})$
FilterNot	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{Y} = r(\mathbf{WV}), \mathbf{z} = \sigma(\mathbf{W}(\mathbf{Y} \odot \mathbf{x})),$ $\mathbf{o} = \min(\mathbf{a}, 1 - \mathbf{z})$
FilterPos	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{Y} = r(\mathbf{WV}), \mathbf{z} = S(\mathbf{W}(\mathbf{Y} \odot \mathbf{x})),$ $\mathbf{o} = \min(\mathbf{a}, \mathbf{z})$
RelateSub	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{Va})), \mathbf{Z} = r(\mathbf{WV}),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{Z}))$
RelateObj	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{Va})), \mathbf{Z} = r(\mathbf{WV}),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{Z}))$
RelateAttr	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{Va})), \mathbf{Z} = r(\mathbf{WV}),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{Z}))$
Fusion	[a ₁ , a ₂]	$\mathbf{o} = \min(\mathbf{a}_1, \mathbf{a}_2)$

Table 3.2: Attention module definitions. σ : Activation function, r : RELU, \mathbf{W} : weight matrix, \mathbf{a} : attention vector (36×1), \mathbf{V} : visual features (768×36), \mathbf{t} : text features (768×1), \mathbf{o} : attention vector (36×1) \odot : Hadamard product, \min : element-wise minimum.

Attention modules highlight relevant object regions from the image and output an attention vector \mathbf{o} of size 1×36 over the image regions. The modules definitions are given in the Table 3.2. Attentions are probabilities representing the relevance of each bounding box in the image. For example, the **Select** module, implements an object detection function $select : txt \mapsto bbox$. It identifies the most relevant object from the image based on a textual argument representing the object name. The module assesses the likelihood of each object bounding box $bbox$ in the image I being the visual representation of the textual concept txt . The probability is represented as $p(bbox|txt, I, \theta)$, where θ denotes the model’s parameters. The **Filter** modules essentially singles out and attend to the objects that match

3.3. NEURAL MODULES

the given textual criteria. The textual arguments can encompass attributes such as colors, sizes, or spatial positions such as left or right.

The `RelateSub` and `RelateObj` modules facilitate reasoning about the subject-object relationships within the given context. They encapsulate two relation types: actions (wearing, eating, etc.) and spatial relations (to the left of, to the right of, next to, on, etc.). The `relateAttr` module directs attention to objects that share a specified attribute with the input object. This attribute is determined by the argument, such as “same material” or “same color”.

The `fusion` module merges two input attentions into a single attention output. This merging process is implemented using a *min* function, which retains only the regions that are highly relevant based on the information from both attention inputs.

The activation function (σ in Table 3.2) of the output layer W for each attention module can be a Sigmoid function when dealing with Binary cross-entropy or a Softmax function for multi-class cross-entropy.

3.3.3 Boolean modules

Boolean modules (Table 3.3) produce boolean-valued outcomes represented by a single bit. The output can take on values like yes/true/one with a probability of p and no/false/zero with a probability of $1 - p$. The `And` and `Or` modules are designed to handle boolean operations such as logical conjunction (AND) and logical disjunction (OR), and utilize probabilities to reason about the occurrence of the two input dependencies. These modules operate in a way that closely resembles probability theory applied to events in boolean reasoning, for example given two independent events A and B , $p(A \text{ and } B) = p(A) \times p(B)$ and $p(A \text{ or } B) = p(A) + p(B)$. For example, “Are there both snow *and* grass in the photo?”.

The `Verify` modules are responsible for asserting a specific property about the inputs. For instance, the `VerifyAttr` module verifies whether the input object possesses a particular attribute or not. For example, “*Is* the planter that is to the left of the fence *made of clay*?”.

The `Same` and `Different` modules make comparisons and determine if specific properties are common between two objects. The `Same` module determines whether the two given objects share a similar property (same color or same material). For example, “Do the shirt and the jar have the *same*

3.3. NEURAL MODULES

Name	Dependencies	Definition
And	$[\mathbf{b}_1, \mathbf{b}_2]$	$\mathbf{o} = \mathbf{b}_1 \times \mathbf{b}_2$
Or	$[\mathbf{b}_1, \mathbf{b}_2]$	$\mathbf{o} = \mathbf{b}_1 + \mathbf{b}_2 - \mathbf{b}_1 \times \mathbf{b}_2$
Same	$[\mathbf{a}_1, \mathbf{a}_2]$	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_1)), \mathbf{z} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_2)),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
SameAll	$[\mathbf{a}]$	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
Different	$[\mathbf{a}_1, \mathbf{a}_2]$	$\mathbf{o} = 1 - \text{same}(\mathbf{a}_1, \mathbf{a}_2)$
DifferentAll	$[\mathbf{a}]$	$\mathbf{o} = 1 - \text{same}(\mathbf{a})$
Exist	$[\mathbf{a}]$	$\mathbf{o} = \sigma(\mathbf{W}([\mathbf{a} \parallel \max(\mathbf{a}) \parallel \min(\mathbf{a}) \parallel \text{mean}(\mathbf{a})]))$
VerifyRelSub	$[\mathbf{a}_1, \mathbf{a}_2]$	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_1)), \mathbf{z} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_2)),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
VerifyRelObj	$[\mathbf{a}_1, \mathbf{a}_2]$	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_1)), \mathbf{z} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_2)),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
VerifyAttr	$[\mathbf{a}]$	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
VerifyPos	$[\mathbf{a}]$	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$

Table 3.3: Boolean module definitions. σ : Sigmoid, r : RELU, \mathbf{W} : weight matrix, \mathbf{a} : attention vector (36×1), \mathbf{b} : boolean scalar, \mathbf{V} : visual features (768×36), \mathbf{t} : text features (768×1), \odot : Hadamard product, $[a \parallel b]$: concatenation, \min : element-wise minimum.

color?”. while the **Different** module, as the name suggests, does the opposite, it assesses whether the two objects have different property (different color, different material).

3.3.4 Answer modules definition

Name	Dependencies	Definition
ChooseName	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
ChooseAttr	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
Compare	[a ₁ , a ₂]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_1)), \mathbf{z} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_2)),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
ChoosePos	[a]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
ChooseRel	[a ₁ , a ₂]	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_1)), \mathbf{z} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_2)),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
Common	[a ₁ , a ₂]	$\mathbf{x} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_1)), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a}_2)),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
QueryName	[a]	$\mathbf{x} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x}))$
QueryAttr	[a]	$\mathbf{x} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x}))$
QueryPos	[a]	$\mathbf{x} = r(\mathbf{W}(\mathbf{V} \mathbf{a})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x}))$
AnswerLogic	[b]	$\mathbf{o}_{\text{yes}} = \mathbf{b}, \mathbf{o}_{\text{no}} = \mathbf{1} - \mathbf{b}$

Table 3.4: Answer module definitions. S : softmax, r : RELU, \mathbf{W} : weight matrix, \mathbf{a} : attention vector (36×1), \mathbf{b} : boolean scalar, \mathbf{V} : visual features (768×36), \mathbf{t} : text features (768×1), \odot : Hadamard product.

The last type of modules is the answer modules (Table 3.4). They are learned to maximize the likelihood of the correct answer given the question, program and image triplet. The `choose` modules predict an answer by choosing between two terms given in the textual argument. For instance, the `chooseRel` module classifies the relation between two given objects using a textual argument representing two relations to choose from, as in “Is the cooked pizza to the left or to the right of the knife?”, the textual argument is an aggregation of the two terms “to the left | to the right”.

The `query` modules classify the name, attribute or position of the input attention vector. For example, the `queryName` module classifies the category of the object given by the input dependency, as in “What is the laptop on?”.

Lastly, the `AnswerLogic` module does not have trainable parameters and its purpose is to map a boolean input b to the answer vocabulary domain. The “yes” class is assigned the b value and the “no”

3.4. PROGRAM EXAMPLES

class is assigned the $1 - b$ value.

3.4 Program examples


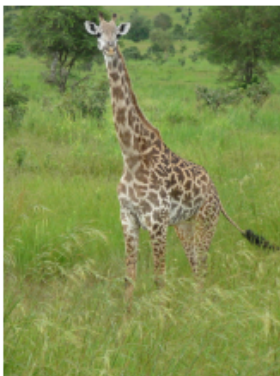


To demonstrate the practical application of the modules in real-world scenarios, we provide a table containing examples of natural language questions, their corresponding programs, answers and related images.

Question & Program prototype & Answer	Image
<p>What is the fence made of?</p> <pre>[select(fence), queryAttr(material)]</pre> <p>wood</p>	
<p>Which kind of furniture is green?</p> <pre>[select(furniture), filterAttr(green), queryName()]</pre> <p>desk</p>	
<p>What is the color of the plate on the right?</p> <pre>[select(plate), filterPos(right), queryAttr(color)]</pre> <p>white</p>	
<p>Is it outdoors or indoors?</p> <pre>[select(scene), chooseAttr(outdoors indoors)]</pre> <p>outdoors</p>	



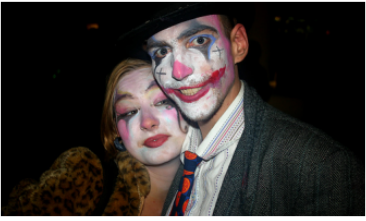

3.4. PROGRAM EXAMPLES

<p>Is the person on the left or on the right? [select(person), choosePos(left right)] left</p>	
<p>What do the field and the palm have in common? [select(field)', select(palm tree)', common()] color</p>	
<p>Is the color of the gloves different than the color of the leaves? [select(leaves), select(gloves), different(color), answerLogic()] yes</p>	
<p>Are there both cats and whales in this photo? [select(cat), exist(), select(whale), exist(), and(), answerLogic()] no</p>	

3.4. PROGRAM EXAMPLES

<p>What device is the water behind of?</p> <pre>[select(water), relateObj(behind), select(device), fusion(), queryName(name)]</pre> <p>cell phone</p>	 A hand holding a black mobile phone in front of a lake. The phone is held up, and the background shows a calm body of water with trees on the far shore under a clear sky.
<p>What is the size of the giraffe?</p> <pre>[select(giraffe), queryAttr(size)]</pre> <p>large</p>	 A tall giraffe standing in a grassy field. The giraffe is facing left, and its long neck and spotted pattern are clearly visible against the green grass and some trees in the background.
<p>Is the small table made of metal?</p> <pre>[select(table), filterAttr(small), verifyAttr(metal), answerLogic()]</pre> <p>no</p>	 A group of children sitting around a small table in a classroom. The children are engaged in an activity, and a teacher or adult is standing nearby. The room is filled with educational materials and colorful decorations.
<p>Which side is the house on?</p> <pre>[select(house), queryPos(hposition)]</pre> <p>left</p>	 A skateboarder performing a trick on a street. The skateboarder is in mid-air, and the street is lined with houses. The house on the left side of the frame is the focus of the query.

3.4. PROGRAM EXAMPLES

<p>Is the laptop to the right or to the left of the small papers? [select(papers), filterAttr(small), select(laptop), chooseRel(to the left of to the right of)] left</p>	
<p>Is the wheel chair in the bottom part of the image? [select(wheelchair), verifyPos(bottom), answerLogic()] no</p>	
<p>What type of clothing is not white, the shirt or the hat? [select(clothing), filterNot(white), chooseName(shirt hat)] hat</p>	
<p>Are both the seat and the life preserver the same color? [select(life preserver), select(seat), same(color), answerLogic()] no</p>	

3.4. PROGRAM EXAMPLES

<p>Are the people the same gender?</p> <pre>[select(person), sameAll(gender), answerLogic()]</pre> <p>no</p>	
<p>Does the mat have a different color than the flower?</p> <pre>[select(flower), select(mat), different(color), 'answerLogic()]</pre> <p>yes</p>	
<p>Do these animals have different types?</p> <pre>[select(animal), differentAll(type), answerLogic()]</pre> <p>yes</p>	
<p>What is the name of the vehicle that is made of the same material as the lock?</p> <pre>[select(lock), relateAttr(same material), select(vehicle), fusion(), queryName(name)]</pre> <p>carriage</p>	

3.4. PROGRAM EXAMPLES

<p>Is the man to the left of a spectator?</p> <pre>[select(spectator), select(man), verifyRelSub(to the left of), answerLogic()]</pre> <p>no</p>	
<p>Is the woman that is to the left of the bus wearing shorts?</p> <pre>[select(bus), relateSub(to the left of), select(woman), fusion(), select(shorts), verifyRelObj(wearing), answerLogic()]</pre> <p>yes</p>	
<p>Which is younger, the lady or the baby?</p> <pre>[select(baby), select(lady), compare(younger)]</pre> <p>baby</p>	
<p>Are there either stop signs or fire hydrants that are not white?</p> <pre>[select(stop sign), filterNot(white), exist(?), select(fire hydrant), filterNot(white), exist(?), or(), answerLogic()]</pre> <p>no</p>	


<p>Are the grapes on top of the table and the fruits inside the box both red?</p> <pre>[select(table), relateSub(on top of), select(grapes), fusion(), verifyAttr(red), select(box), relateSub(inside), select(fruit), fusion(), verifyAttr(red), and(), answerLogic()]</pre> <p>yes</p>	
--	---

Table 3.5: Program examples

3.5 Generator

As detailed in the related-work Chapter 2, the NMN framework consists of two primary components: the generator and the executor. In this section, our focus lies on the generator, which translates the input question into a corresponding program representation. In subsection 3.5.1 we delve into the intricate process of generating modules, providing an explanation of how the program’s sequence of modules is constructed based on the input question and image. The next subsection 3.5.2 sheds light on the textual arguments generation process and we elucidate how the model predicts and incorporates textual arguments into the generated program sequence.

3.5.1 From questions in natural language to functional programs

The generator takes as input the question features w_i (and the image) and outputs an autoregressive sequence of instructions to follow $P = [m_1, \dots, m_n]$ from the modules dictionary 3.1. This task necessitates a sequence-to-sequence generation model, such as recurrent neural networks (GRU [Cho et al., 2014], LSTM [Sutskever et al., 2014]) or transformers [Devlin et al., 2019]. To adhere to the architecture employed by LXMERT [Tan and Bansal, 2019], we choose the transformer network in a decoder form. The problem at hand can be viewed as a language translation task, akin to translating a sequence from French to English. However, in this context we perform translation from the natural language domain to the modules tokens domain. The goal is to convert a question expressed in natural language into a corresponding sequence of modules, which can then be executed to infer an answer. The output is a sequence, therefore the decoder form of a transformer is employed.

3.5. GENERATOR

The sequence derived from the final layer of the LXMERT cross-modality encoder serves as the “memory” input to the transformer decoder. These representations encompass the features of each word from the question, aligned with their corresponding object features extracted from the image.

The loss function is given by Equation 3.1, with L_m being the average of the cross-entropy losses of the generated modules and the target expected modules. In this equation, T represents the number of tokens in the program, reflecting the length of the generated module sequence that is being compared to the target modules.

$$L_m = \frac{1}{T} \sum_{t=1}^T L_{CE}(\hat{m}_t, m_t^*) \quad (3.1)$$

3.5.2 Arguments prediction

In order to enhance the precision and relevance of the generated program sequence, and enable the model to incorporate pertinent information from the input question into the textual arguments, we establish a functional representation of the modules’ arguments in relation to the input question.

For modules that require a textual argument, we represent these textual arguments as a function of the input question. The arguments used by the modules in the program are directly derived from the words present in their related questions.

To capture the textual arguments for each module m , the generator g outputs the argument a as a function of the words w_1, w_2, \dots, w_n in the input question Q :

$$a = g(Q) = g(w_1, w_2, \dots, w_n) \quad (3.2)$$

In a technical sense, the predicted argument features are computed using a weighted sum of the encoder output values. The weight for each value is determined by the transformer attention weights generated by the encoder-decoder attention layer.

The preliminary experiments revealed that the attention mechanism may not consistently exhibit the expected behavior, as commonly observed in language translation tasks. In typical translation scenarios, the encoder-decoder attention maps tend to demonstrate a strong and relevant correlation between the English words and their corresponding French words.

To fine-tune the textual argument predictions, we supervise the attention mechanism to attend on

3.5. GENERATOR

the desired question words. The CE metric is used to quantify the similarity between the module’s attention distribution and the target question words. Concretely, for each module m_t requiring a textual argument, we extract the corresponding argument from the dataset. Subsequently, we create a one-hot vector a_k^* where the position of the argument in the question is marked as 1 and the rest of the vector elements are 0s. We then compute the cross-entropy between this one-hot vector a_k^* and the output attention weights vector \hat{a}_k of the respective module token as formulated by the equation 3.3.

$$L_a = \frac{1}{K} \sum_{k=1}^K L_{CE}(\hat{a}_k, a_k^*) \quad (3.3)$$

3.5.3 Generator optimisation

The objective function aims to 1) maximize the likelihood of generating the correct program given the input question, and 2) maximize the likelihood of attending on the target word from the question when predicting a module. The final generation loss, denoted as L_G is represented by the equation:

$$L_G = \alpha \cdot L_m + \beta \cdot L_a \quad (3.4)$$

Here, α and β are weighting terms that balance the importance of the respective objectives.

During training, we have the question and program pairs and we train the decoder in an autoregressive manner. This training process makes use of the PyTorch Transformer decoder layer implementation and is automatically parallelized for time efficiency.

3.5.4 Program inference

During inference, the Transformer decoder operates in an open loop without accessing the ground-truth program, so it is essential to ensure that the generated program adheres to the structural constraints and produces plausible programs. Inspired by [Hu et al., 2017], we define a set of affine inequality constraints for each module m of the form $S_t \cdot X_m \geq b_m$ to control the decoding process, where S_t represents the current decoding state, X_m is a vector and b_m is a scalar representing the constraints applied of the module m . This allows to mask the modules that do not respect their constraints. For instance, a module that requires a certain type dependency can only be predicted if and only if a module, which provides that type of dependency, was predicted before it.

3.5. GENERATOR

To control the generation process and ensure that the generated modules adhere to consistency constraints, we employ a state vector S_t that keeps track of the program characteristics for each generation step t . This state vector is represented as: $S_t = (|att| \ |bool| \ |ans| \ \Upsilon)$, where $|att|$ and $|bool|$ are the number of unconsumed attention and Boolean outputs, while $|ans|$ specifies whether the answer module is present in the generated program. Υ represents the number of remaining steps in the generation process, defined as $\Upsilon = T - t$, where T is the maximum allowed program length.

We’ve established four constraints to control the decoding process:

- For predicting a module, it’s essential to ensure that there are more available outputs in S_{t-1} than the predicted module can consume.
- The prediction of an answer module is permissible only when there are no unconsumed attention or boolean outputs remaining in S_{t-1} .
- Predicting an attention or boolean module is only feasible if no answer module has already been predicted in S_{t-1} .
- Adhere to the remaining time constraint T by refraining from predicting modules whose outputs cannot be fully processed within the remaining Υ time.

We create a state-transition matrix P to update the generation state at each time step. P has a shape of $(N \times 4)$ where N is the number of modules in the dictionary \mathcal{M} . Each row in the matrix P corresponds to a specific module in the dictionary, and the elements in that row denote the expected impact on the program characteristics when the corresponding module is predicted. For example, the row of index 1 in the matrix P , which corresponds to the **Select** module, has the value of $P_{1,\cdot} = (1 \ 0 \ 0 \ -1)$. This indicates that when the **Select** module is predicted, the value of $|att|$ will be increased by one, as the module generates an attention vector without consuming one as a dependency. However, there will be no change in the values of $|bool|$ and $|ans|$, as the **Select** module does not consume any Boolean or answer dependencies, nor does it output any Boolean or answer values. Additionally, the number of remaining steps Υ will be decreased by 1.

The transition operation leading to the new state is represented by the equation: $S_t = S_{t-1} + P_m$, where m is the index of the module chosen at step t .

3.5.5 Experimental validation

To assess the performance of the program generation task, we track the decreasing value of the training loss function and calculate accuracy. We consider as accurate the generated programs that exactly match their ground-truth programs in a token-wise-manner. The training performance for program prediction attains an accuracy of 74%. It's worth noting that the generator task is relatively easy in comparison to the executor task. Consequently, similar to the approach outlined in [Askarian et al., 2021], the experiments in this research primarily center around the executor training by using the ground truth programs as inputs to the executor.

3.6 Executor

As described in the previous section, a functional program consists of a sequence of modules and their textual arguments. It represents the reasoning structure to employ in order to answer a question related to an image. The executor is responsible of instantiating the NMN and executing it on the related image in a sequential manner to answer the question. It is also responsible of managing the module dependencies.

3.6.1 Module initialisation

To implement the modules based on the modules' list \mathcal{M} and their definitions from the previous section, we utilize PyTorch. The \mathbf{W} matrices representing the Linear layers are initialized using the He initialization method [He et al., 2015]. These layers are followed by a RELU activation function for introducing non-linearity into the model. In the case of modules sharing their weights (Weights sharing is detailed in Section 3.6.2), we follow a specific initialization process to ensure weight sharing among them. The shared layer is initialized in one of the modules, and then the rest of the modules that share this layer receive it as a dependency during their initialization. We give more details on the weight sharing in the following subsection 3.6.2. To efficiently manage and retrieve the modules when constructing the NMNs, we store them in a dictionary, with the module names as the keys.

Below is an implementation example of the `select` module in Python:

3.6. EXECUTOR

Algorithm 1 Python-style pseudocode for select module.

```
# dim_txt: hidden dimension size of the input argument
# dim_vis: hidden dimension size of the bounding box
# dim: hidden dimension size of the module
# vis: Image features (36 x dim_vis)
# txt: textual argument (1 x dim_txt)

class Select(nn.Module):

    def __init__(self, dim_txt=768, dim_vis=768, dim=768, nb_obj=36):
        super().__init__()
        self.linear_txt = nn.Linear(dim_txt, dim)
        self.linear_vis = nn.Linear(dim_vis, dim)
        self.linear_out = nn.Linear(dim, 1)
        nn.init.kaiming_normal_(self.linear_txt.weight)
        nn.init.kaiming_normal_(self.linear_vis.weight)
        nn.init.kaiming_normal_(self.linear_out.weight)
        self.sigmoid = nn.Sigmoid()

    def forward(self, txt, vis):
        txt = F.relu(self.linear_txt(txt))
        vis = F.relu(self.linear_vis(vis))
        eltwise_mul = torch.mul(txt, vis)
        output = self.linear_out(eltwise_mul).transpose(0, 1)
        return self.sigmoid(output)
```

3.6.2 Weight sharing

Despite the modules being shallow with few parameters, their cumulative number of parameters significantly increases when they are chained together. To remedy this, we use a weight sharing technique to reduce the total number of model parameters. This also allows the shared layers to have a better learned behavior and to be updated based on a larger number of training examples. The overall principle is that a sharing is made only between some of the textual and visual layers, each module having a distinct output layer to guarantee its fine-tuning to the module’s sub-task.

The decision regarding parameter sharing among layers in different modules is determined through an assessment of module similarities. This assessment involves the analysis of functional and architectural properties of the modules. The former is derived from the module reasoning sub-task and the latter is derived from the module layer architectures. The strategy is exemplified in the following by conducting a comparison of several modules, wherein their inherent similarities and differences are explained. The modules include several types of non-linear layers: textual layers, visual layers and output layers. Textual layers take word embeddings as input, while visual layers take bounding box features as input. Both types of layers produce hidden vectors, which are then combined using element-wise multiplication (Hadamard product). The resulting vector is then passed to the output layer, which maps the hidden vector to the module’s output space.

Specifically, the `Select` module and the `FilterAttr` module are two examples of such modules.

3.6. EXECUTOR

The `Select` module detects a relevant bounding box given the name of an object, while the `FilterAttr` detects a relevant bounding box given an attribute. Functionally, they both solve a perceptual problem but have different textual argument semantics. Architecturally, they both have the same layer structure: a textual layer, a visual layer, and an output layer. We decide to share the visual layer between these two modules but use different textual layers to respect the semantic differences between their textual arguments.

However, `FilterAttr` can share its textual layer with other modules having an attribute as a textual argument, like `VerifyAttr` or `FilterNot`.

The `Same` and `Different` boolean modules assess whether or not two selected objects share the same characteristic (provided by the textual argument). The probability p of two objects being similar is one minus the probability of them being different. Therefore, they share the same layers including the output layer and we use the function $p(\text{Different}) = 1 - p(\text{Same})$ to differentiate them.

The object relations modules such as `RelateSub` and `RelateObj` have similar functionalities and neural structures. They share their visual layers to get a common scene representation and they share the textual layer due to the semantic similarity of their arguments (a relation).

3.6.3 Modular network instantiation

In the neural module instantiation, the program is utilized to make function calls to the modules and execute them over the image representation. This process takes place in the executor’s forward function, where it iterates through the program module prototypes. The executor is also responsible for converting the program structure from its language format to the input format required by the NMN. To do so, module arguments are converted from words to embeddings. The program executor is responsible of managing module dependencies by using a memory buffer to save the outputs that serve as inputs for the next modules. The memory buffer is particularly valuable in scenarios involving two-branch programs. For example, an NMN containing the `and` module requires a memory buffer to store the output of the first branch, before executing the modules of the second branch to produce the second output and later merge them in the computation of the conjunction.

3.6.4 Reasoning process

Algorithm 2 presents the execution loop of a program in an abstract manner.

3.7. EVALUATIONS

Algorithm 2 Python-style pseudocode for Program execution.

```
# program: the program
# img: image representation
# arg: argument word
# emb: embedding mapping the words to their representations
# executor: dictionary mapping the modules parameters
# output: recurrent output, initialized with zeros

for (module_name, arg) in program: # iterate over the program modules
    if arg: txt = emb(arg)
    module = executor[module_name]

    if module == 'select': # marks the begging of the reasoning branch
        memory_buffer = output
        output = module.forward(txt, img)

    elif module.nb_dependencies == 1:
        output = module.forward(txt, output, img)

    elif module.nb_dependencies == 2:
        output = module.forward(txt, memory_buffer, output, img)

answer = output.argmax()
```

3.6.5 Answer prediction

At the end of the reasoning process, the final module provides the answer to the question, it outputs a probability distribution over the 1842 answer classes. There are two types of questions, binary (yes/no) and open questions. For binary questions, the `answerLogic` module is used, while open questions are managed by one of the other answer modules, which are comprehensively listed in Table 3.4. The predicted answer, denoted as \hat{y} , is determined as the answer y with the highest output probability within the answer set A , as shown by the equation:

$$\hat{y} = \arg \max_{y \in A} p(a|Q, P, I, \theta) \quad (3.5)$$

3.7 Evaluations

The experiments outlined in this section are devoted to assessing the influence of multi-modal representations of vision and language on our NMN architecture, we compare unimodal vs multimodal representations. This preliminary investigation lays the foundation for subsequent chapters, where we directly employ multi-modal representations for our experiments.

The executor is optimized using the cross-entropy loss, which involves comparing the predicted answer with the ground-truth answer for each question, program and answer triplet. We use the `testdev_all` set of the GQA dataset.

$$L_e = \frac{1}{N} \sum_{n=1}^N L_{CE}(\hat{y}_n, y_n^*) \quad (3.6)$$

3.7.1 Experimental settings

Unimodal representations. For unimodal word embeddings we use both non-contextual and contextual off-the-shelf pretrained embeddings. For non-contextual embeddings, we opt for FastText [Bojanowski et al., 2016] pre-trained word embeddings, that showcase good performance on several downstream tasks, and for its ability to compute embeddings for words that did not appear in the training data. The contextual embeddings, which represent the embedding of a given word with their context by attending on the surrounding words in the question and mixed with their positional information, offer a richer representation for our arguments. To achieve this, we employ Transformer-based BERT pretrained embeddings [Devlin et al., 2019]. This approach allows us to assess the impact of context-aware embeddings on the argument representation. For the unimodal image region representations we use those provided with the GQA dataset [Hudson and Manning, 2019b] that are extracted by using the bottom-up Faster-RCNN model.

Cross-modal representations. For cross-modal representations, we encode both the question and the image with LXMERT as detailed in section 3.2.

3.7.2 Unimodal vs crossmodal representations

In Table 3.6 we show the impact of different question words and image regions encodings on the performance using two different training setups **setup1** and **setup2** trained following the best training strategies from the next Chapter 4. These setups will be detailed in the following chapter and only serve here as training strategies to be able to measure the impact of using uni-modal representations against aligned cross-modal representations on the executor performance.

- **Setup1:** NMN trained using Teacher guidance with soft matching from Section 4.4.
- **Setup2:** NMN trained using Teacher guidance with hard matching from Section 4.4.

The different evaluated representations in Table 3.6 are:

3.7. EVALUATIONS

- **FastTextV**: Employ unimodal non-contextual fastText embeddings [Bojanowski et al., 2016] along with Faster-RCNN bounding box features already provided by the GQA dataset [Hudson and Manning, 2019b].
- **BertV**: Use unimodal contextual language and vision representations, where contextual text embeddings are extracted by the BERT model [Devlin et al., 2019] and Faster-RCNN bounding boxes features are provided by the GQA dataset [Hudson and Manning, 2019b].
- **LXV**: Employ the cross-modal representations encoded by the LXMERT model [Tan and Bansal, 2019].

Embedding-training	Accuracy
FastTextV-setup1	0.495
BertV-setup1	0.506
LXV-setup1	0.630
FastTextV-setup2	0.511
BertV-setup2	0.485
LXV-setup2	0.632

Table 3.6: Language and vision representation comparison on `testdev-all`.

Cross-modal aligned features provided by LXMERT (denoted as **LXV**) have shown a significant increase in accuracy, with a +12.4% improvement when training with **setup1** and a +12.1% improvement when training with **setup2**. This validates our intuition that leveraging cross-modal features pretrained on diverse tasks and large datasets can greatly benefit NMNs. By incorporating these features, the modular reasoning process is performed with a better understanding of word embeddings and bounding box features, leading to enhanced performance and more accurate predictions. In contrast to related works [Andreas et al., 2016a, Andreas et al., 2016b, Li et al., 2019a], which infer answers by combining the NMN output with the LSTM hidden representation of the question to capitalize on linguistic priors regarding questions and answers, our approach refrains from using this shortcut. Instead, we rely on the cross-modal input representation to address the claimed NMN’s limitation in effectively capturing prior knowledge about the language modality.

FastText and **BERT** unimodal representations lead to comparable results, with **BERT** surpassing **fastText** in the **setup1** experiments while the opposite is observed when training with **setup2**. A more detailed comparison, explaining the differences between setup1 and setup2 and the implications they have on the model’s training, will be performed in Chapter 4 Section 4.4.

3.8 Conclusion

This chapter has provided a foundation for understanding Multi-modal Neural Module Networks (NMN), shedding light on their operational principles, capabilities, and design choices. NMN is an effective architecture in tackling complex multi-modal tasks by decomposing them into easier sub-tasks, each efficiently handled by a dedicated module. The promising potential of NMN makes the visual reasoning more *transparent*. The initial inquiry concerned multi-modal representations and their importance in capturing information from language and vision and aligning them. Experimental validation was conducted to assess the effectiveness of these representations in handling complex questions. The core of the chapter revolved around the neural modules that compose the reasoning process. We delved into the module definitions and their functionalities, emphasizing their roles in enabling modular reasoning. Additionally, the concept of weight sharing was introduced as a technique to reduce model complexity. The generator component was presented to convert natural language questions into functional programs. The process of predicting arguments and optimizing the generator further demonstrated the flexibility and adaptability of the architecture. Moreover, the executor's role in instantiating the NMN based on the program and managing module dependencies was elucidated. Finally, the chapter provided insights into the reasoning process and the final answer classification.

Chapter 4

Guided-Training of neural module networks

Contents

4.1	Introduction	59
4.2	Input guidance	60
4.3	Output feedback	62
4.3.1	Attention loss	62
4.3.2	Boolean loss	63
4.4	Intermediate targets coding	63
4.4.1	Attention targets	63
4.4.2	Boolean targets	66
4.5	Experiments	66
4.5.1	Evaluated methods	66
4.5.2	Results analysis	67
4.5.3	Implementation details	69
4.5.4	Modules training evolution	70
4.5.5	Qualitative analysis of the modular approach	71
4.6	Conclusion	73

4.1 Introduction

Neural module networks (NMNs) offer the advantage of conducting explicit and task-specific reasoning processes based on a given question. However, their training methodology primarily optimizes the various reasoning sub-tasks that constitute the question based solely on the final answer accuracy. Nevertheless, both empirical experiments and prior research reveal that the minimization of the final answer error alone might not consistently ensure that the modules accurately fulfill

4.2. INPUT GUIDANCE

their intended sub-tasks. Consequently, the need arises for supplementary guidance to optimize the intermediate modules, ensuring that they execute the reasoning sub-tasks without resorting to “shortcuts” or otherwise compromising the integrity of the reasoning process.

This chapter delves into teacher guidance, which addresses the previously mentioned issues to enhance the performance of NMNs. It commences by elucidating “Input guidance” in Section 4.2, encompassing its definition and the concept of “Decaying Teacher Forcing”. Subsequently, it introduces “Output feedback” in Section 4.3, concentrating on attention and boolean losses. The chapter explores implementation details, delving into the effects of teacher guidance. Finally, we conclude it by presenting insights gathered from the conducted experiments.

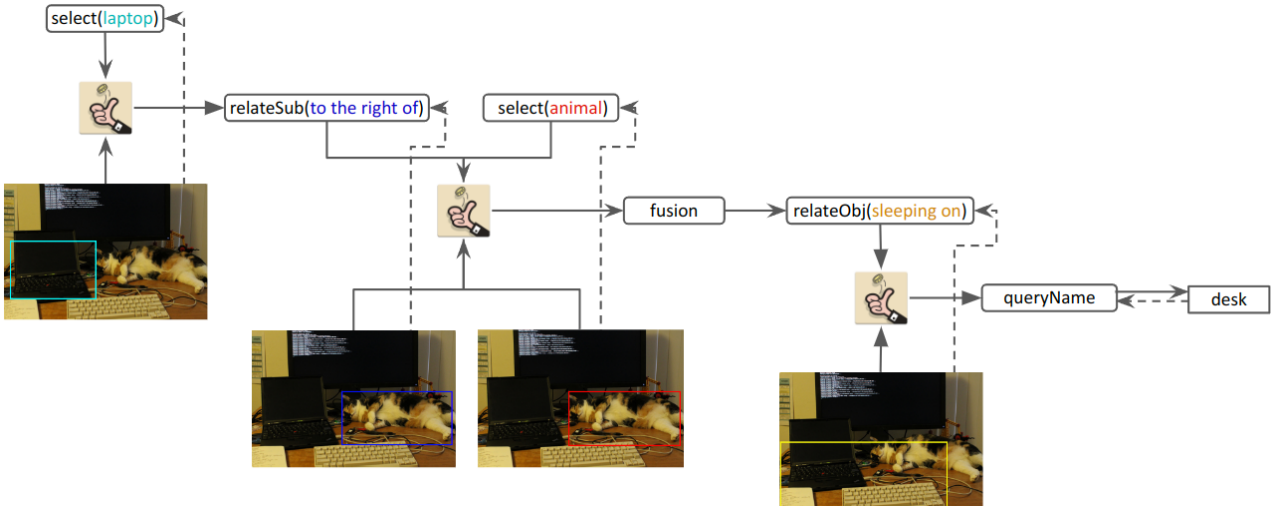


Figure 4.1: The teacher guidance for the program execution process related to the question ‘On what is the animal to the right of the laptop sleeping?’. Plain arrows represent input guidance 4.2, dotted arrows represent the output feedback 4.3 and bounding boxes represent the GT intermediate targets.

4.2 Input guidance

The challenge of training NMNs collaboratively lies in the fact that the inputs of subsequent modules depend on the outputs of previous ones, which can propagate prediction errors. For instance, imagine the program depicted in Figure 4.1. If the initial module `select(laptop)` produces an inaccurate probability distribution, failing to correctly attend to the intended “laptop” object, this error can extend to subsequent modules, consequently, the module `relateSub(to the right of)` may attend to the right side of the incorrect object, leading to inconsistent reasoning. The same issue

4.2. INPUT GUIDANCE

applies to other intermediate and final modules in the chain.

Teacher forcing (TF) [Williams and Zipser, 1989] can address this issue by providing intermediate modules with the correct ground-truth (GT) inputs instead of their previous modules’ outputs. This helps the intermediate modules to learn their reasoning sub-task in a correct setup.

We adopt a decaying teacher forcing technique, influenced by the concept of scheduled sampling [Bengio et al., 2015]. This approach facilitates a gradual shift in the training of modules, transitioning from individual optimization to a more cohesive joint training process. In fact, when a module is provided with its golden (GT) input and expected output, it is independently optimized to perform its specific sub-task. And when modules are *jointly trained* in a sequential manner, they learn to adapt their behaviors to work together and engage in explicit reasoning without taking shortcuts. This collaborative approach enables the modules to develop a better understanding of the question composition and enables them to perform compound reasoning operations.

Notably, we provide the modules with input guidance through decaying teacher forcing (TF). As shown in Fig. 4.1, at each reasoning step t the executor randomly decides whether to use the predicted output \hat{o}_{t-1} or the ground-truth target o_{t-1}^* from the previous module m_{t-1} as its input. This decision is made by flipping a coin, where o_{t-1}^* is chosen with a probability of ϵ_e and \hat{o}_{t-1} with a probability of $1 - \epsilon_e$. The coin-flipping process for input selection occurs at each reasoning step t during training, allowing the model to train on various sub-programs. The probability ϵ_e of selecting o_{t-1}^* depends on the epoch number e . As training progresses and the epoch number increases, ϵ_e decreases, giving more preference to the module’s predictions over the ground-truth intermediate targets.

From a back-propagation perspective, during the early stages of training with decaying teacher forcing the gradients are computed based on the losses of individual modules when processing correct inputs. As a result, the backward gradient flow of the loss is interrupted at the first ground-truth input. However, in the case of collaborative module interactions without TF, the full back-propagation can be computed. The intermediate outputs are preserved in continuous form throughout the program execution, enabling the flow of backward gradients between modules. Errors and updates can be back-propagated through the entire network, facilitating effective learning and enhancing the overall performance of the NMN.

4.3 Output feedback

The other facet of teacher guidance is intermediate output feedback. Instead of solely optimizing intermediate modules based on the final answer classification loss, this approach is reformulated as a multi-task (MT) objective. Here, individual module performances are considered, providing feedback based on their outputs errors compared to the ground truth intermediate targets (4.4). This feedback mechanism aids in regulating module weights and the learning of their corresponding sub-tasks.

In this setup, the overall loss consists of a weighted sum $L = \alpha L_{att} + \beta L_{bool} + \gamma L_{answer}$ of individual losses for the attention modules, Boolean modules and answer modules, with α , β and γ scaling factors.

In the following, we provide details on the attention loss and Boolean loss.

4.3.1 Attention loss

The motivation behind the implementation of attention loss is to compel the attention modules to focus attention on the designated target object. For instance, consider the module prototype `select(laptop)` as depicted in Figure 4.1. The aim is to ensure that the generated attention output distribution has its maximum value closely aligned with the intended target, thereby producing an attention vector that accurately identifies the “laptop” object. Similarly, for the `relateSub(to the right of)` module, the goal is to enforce its output to correspond to the target, effectively identifying the object positioned to the right of the laptop.

To achieve this, during the forward pass of the reasoning process, we store the intermediate outputs in a designated memory buffer. This memory buffer is indexed to allow mapping of each saved intermediate output to its corresponding module function name. At the end of the forward pass, we access this memory buffer and calculate the Cross Entropy (CE) loss between its content and the intermediate targets (details are given in Section 4.4). The computation of loss averages is module-specific, thereby enabling the penalization of each module based on the average of its individual errors and accounting for variations in their batch occurrences to prevent overemphasis on frequent modules at the expense of infrequent ones. The attention loss L_{att} is the accumulation of losses from all attention modules. It is calculated as the sum of the individual module losses, normalized by the number of occurrences of each module in the batch. Mathematically, this is represented as

4.4. INTERMEDIATE TARGETS CODING

$L_{att} = \sum_{m \in \mathcal{M}} \frac{L_m}{|m|}$, where L_m signifies the cumulative losses computed for each instance of module m , and $|m|$ represents the count of occurrences of module m within the batch.

4.3.2 Boolean loss

In a similar way to the attention loss, we also integrate intermediate losses for Boolean modules. These modules generate a singular class output that represents the probability of the given premise being True. Our objective is to drive the probability of true premises output close to 1, while pushing the probabilities of incorrect premises towards 0. For example, the `VerifyAttr(red)` module validates whether the object attended by the input dependency possesses the attribute argument “red”. Specifically, when the attended object is indeed red, our goal is to enhance the probability of the module’s output converging towards 1. Conversely, if the attended object is not red, we want to push the output towards 0. The L_{bool} follows the same loss computation schema as described for the attention loss in the previous subsection.

4.4 Intermediate targets coding

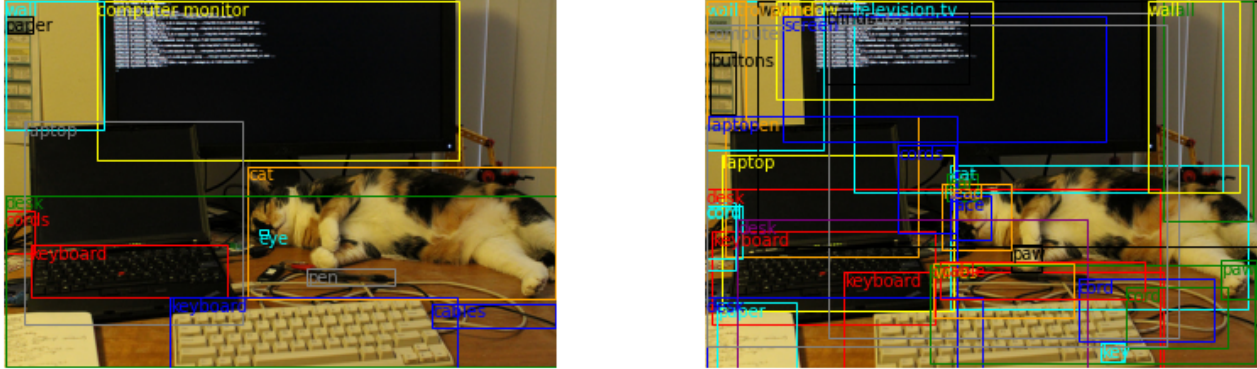
Intermediate targets are the ground truth targets of the intermediate modules outputs o_t^* that serve to compute the intermediate modules losses. To construct the intermediate targets, we rely on the GQA pre-processed programs to get relevant data and adapt their format to the modules’ structures. In the following sub-sections we detail the coding process of the attention targets and the Boolean targets.

4.4.1 Attention targets

For attention modules, the target bounding boxes are provided by the image graph. However, these objects differ from those extracted by the image feature extractor, as illustrated in Figure 4.2. In Figure 4.2a, the ground-truth bounding boxes framing various objects in the image are more precise and accurate since they were manually labeled from the Visual Genome dataset [Krishna et al., 2017]. On the other hand, the bounding boxes obtained from the feature extractor are automatically generated and they are more numerous and overlapping. In fact, the Faster R-CNN model [Ren et al., 2015b], employed in our feature extractor LXMERT [Tan and Bansal, 2019], requires a fixed hyper-parameter to determine the number of output bounding boxes, which is set to 36. Subsequently, we delve into

4.4. INTERMEDIATE TARGETS CODING

the process of establishing correspondences between the bounding boxes in the image graph and those generated by Faster R-CNN.



(a) Boxes from the GQA image graph

(b) Boxes from LXMERT

Figure 4.2: Comparison between the ground-truth bounding boxes and the LXMERT bounding boxes.

To provide a clearer insight into this process, we present the data pertaining to the NMN of Figure 4.1. The original program structure can be observed in Figure 4.3, where each row represents an execution step along with its associated information. The `inter_id` column indicates the assigned target object IDs. We cross-reference these IDs with the data from the image graph in Figure 4.4 to extract the corresponding object coordinates using the `object_id` column.

	operation	dependencies	argument	inter_id
step				
0	select	[]	laptop	738229
1	relateSub	[0]	to the right of	738227
2	select	[]	animal	738227
3	fusion	[1, 2]		738227
4	relateObj	[3]	sleeping on	738228
5	queryName	[4]	name	None

Figure 4.3: Raw program structure.

This process of identifying intermediate object targets and retrieving their coordinates from the graph enables us to perform the necessary computations to determine the nearest bounding box from the LXMERT bounding boxes. To establish correspondences between the bounding boxes in the image graph and those obtained from Faster-RCNN, we calculate the intersection over union (IoU) factor. Specifically, we compute the IoU between the ground-truth bounding box from the image graph and the 36 Faster-RCNN bounding boxes.

4.4. INTERMEDIATE TARGETS CODING

object_id	name	h	w	y	x
738233	corcs	37	27	191	0
738232	cables	22	112	274	387
738231	computer monitor	145	328	0	83
738227	cat	120	279	150	220
738234	eye	5	8	207	230
738238	keyboard	47	179	221	23
738228	desk	156	500	176	0
738229	laptop	184	198	109	17
738244	paper	16	25	14	0
738245	wall	117	90	0	0
738242	pen	16	80	242	273
738230	keyboard	65	260	268	149

Figure 4.4: Object identifiers, names and coordinates from the image graph data.

As shown in Figure 4.2b, the bounding boxes extracted by LXMERT exhibit overlaps, often causing an object to be covered by multiple boxes. For instance, the laptop object is encompassed by both a yellow and a blue bounding box (labelled “laptop” in the top-left corner). The respective Intersection over Union (IoU) values with the laptop box from the graph are 0.7324 and 0.7027. This situation raises a question: should we exclusively regard the bounding box with the highest IoU (Top1) as the target, or should we choose the nearest bounding boxes based on the ranking of IoU measures? To address this query, we implement and assess both approaches that we call hard matching and respectively soft matching. It’s worth noting that not all modules have available ground-truth targets that can be extracted. This situation arises, for instance, when dealing with questions that inquire about the existence of a particular object in the image, and that object is, in fact, non-existent.

Hard matching. In the hard matching approach, a ground-truth bounding box b_g is matched with the bounding box o_i^* from the feature extractor that has the highest Intersection over Union (IoU) factor. The produced target vector is a one-hot-like vector. In this case, the Softmax function is applied to the output of the attention modules, and the resulting probability vector with a sum of 1 is compared to the target bounding box through Cross Entropy (CE) loss.

Soft matching. The soft mapping matches b_g with all o_i^* that have an IoU value above a threshold. This approach yields multiple target boxes and the target vector can have multiple ones, akin to a multi-label classification task. We utilize the Sigmoid function on the output attention vector, each element being transformed into a probability. Subsequently, a comparison is performed between this

4.5. EXPERIMENTS

vector and the multi-label vector, employing Binary Cross Entropy (BCE) loss.

4.4.2 Boolean targets

For Boolean modules, we rely on the provided answer to infer the module’s targets and generate the intermediate Boolean targets. For example, given the question ‘Is the color of the gloves different than the color of the leaves?’ and its program `[select(leaves), select(gloves), different(color), answerLogic()]`, the Boolean module `different(color)` target is inferred from the answer, if the answer is ‘yes’ then the module target is $o_i^* = 1$ and if the answer is ‘no’ then $o_i^* = 0$.

For questions involving two reasoning branches, where the program incorporates an `and` or `or` module, we deduce the intermediate Boolean module targets based on logical Truth tables for AND and OR operations. While it is true that AND and OR operations are logically irreversible, as their single outputs cannot uniquely identify their dual inputs, we only build intermediate targets for the scenarios where reversibility is feasible. In the case of AND, when the output is True, it indicates that both inputs are necessarily True. Conversely, for OR, if the output is False, it implies that both inputs are necessarily False. Ambiguous combinations are disregarded, no intermediate targets are constructed for such cases and these modules don’t have a specific intermediate loss.

4.5 Experiments

As presented in the previous sections, we propose two teacher-guidance techniques to improve NMNs for VQA. First, we use “input guidance” to prevent the propagation of wrong predictions and “output feedback” to tweak the modules’ weights in order to perform their assigned sub-task. In this section, we present the different training variants and analyze their impact on the NMN performance. We also resume the experiments from Chapter 3 for a more extensive comparison of the usage of multi-modal representations against uni-modal representations.

4.5.1 Evaluated methods

We design several experiments to evaluate our hypotheses and employ the following notations to describe the various experimental setups:

- **TF**: Apply decaying teacher forcing to guide the inputs of the modules. Details in Sec. 4.2.

4.5. EXPERIMENTS

- **MT**: Apply multi-task losses to guide the expected outputs of the modules. Details in Sec. 4.3.

For both **TF** and **MT** training strategies, we experiment with the two different matching techniques described in Sec. 4.4.1:

- **Hard**: Employ the hard matching technique.
- **Soft**: Use the soft matching technique.

4.5.2 Results analysis

Model	accuracy
LXV-TF-hard	0.548
LXV-MT-hard	0.598
LXV-TF-MT-hard	0.630
LXV-TF-soft	0.536
LXV-MT-soft	0.563
LXV-TF-MT-soft	0.632
FasttextV-TF-MT-hard	0.495
BertV-TF-MT-hard	0.506
BertV-TF-MT-soft	0.485
FasttextV-TF-MT-soft	0.511

Table 4.1: Performance of various training methods and encodings on the `testdev-all` set.

Training methods. We aim to enable modular reasoning for visual question answering on the GQA dataset. We evaluate the effectiveness of our approach by measuring the answer accuracy of several models (described in Sec. 4.5.1), and report the results in Table. 4.1. Overall, our findings demonstrate that using a combination of input guidance (denoted as **TF**) and output feedback (**MT**) achieves the highest accuracy, with a score of 63.2%.

When comparing **LXV-TF** (decaying teacher forcing) with **LXV-MT** (multi-task loss), we observe that the multi-task loss alone achieves higher accuracy than using decaying teacher forcing alone. This can be attributed to the fact that when using TF alone, the final loss L is solely determined by the answer modules loss L_{answer} and during early training stages, the application of TF limits the backpropagation process, preventing it from reaching the first modules of the programs. As a result, the impact of L_{answer} on the first modules of the program is limited.

4.5. EXPERIMENTS

Interestingly, the combination of multi-task loss and decaying teacher forcing exhibits complementary effects, leveraging the strengths of both techniques to enhance training dynamics and overall performance.

To assess the effectiveness of the decaying teacher forcing guidance, we compare **LXV-MT** against **LXV-TF-MT**. The **TF** guidance has led to accuracy improvements with both **soft** and **hard** matching settings for NMN. Decaying teacher forcing can be viewed as a form of curriculum learning, where the model trains on programs of increasing length and complexity. During training, we observed a faster increase in accuracy for the models using **TF** compared to those without TF, as the answer modules receive ground-truth inputs in the early stages. As training progresses, the training performance continues to improve until it reaches a peak, after which it slightly degrades due to the reduced use of TF and the modules adjusting to collaborative functioning. Nonetheless, as training continues, the testing performance surpasses that of the models without TF.

When combining the **MT** loss with **LXV-TF**, modules are optimized based on their intermediate outputs losses and they can benefit from the additional guidance provided by the back-propagation of L_{att} and L_{bool} . We reach the best performances outlined by **LXV-TF-MT-soft** and **LXV-TF-MT-hard**. The increase in accuracy ranges from +8.2% in the **hard** matching setting to +9.6% in the **soft** matching setting.

Input encodings. Next, we measure the impact of different input representations on the performance as detailed in Chapter 3, Section 3.7. For unimodal embeddings we encode the question with **fastText** word embeddings or the **BERT** language model, and the image with Faster-RCNN features. For cross-modal representations, we encode the question and the image with LXMERT, denoted as **LXV**. The experiments are conducted using the best training strategies based on the previous comparative analysis, we employ the TF guidance and the MT loss for all the experiments.

When comparing **fastText** and **BERT**, empirical observations indicate that **BERT** tends to achieve better performance when utilizing hard matching, which involves a focused and selective attention mechanism. Conversely, **fastText** demonstrates improved performance with the soft matching mechanism, enabling a multi-label approach. The choice between these matching mechanisms relies on the inputs of the models and the training strategy, as each model may demonstrate superior performance in different scenarios.

4.5. EXPERIMENTS

LXMERT’s cross-modal aligned features, referred to as LXV, have yielded substantial improvements in accuracy compared to uni-modal representations. Integrating these features empowers the modular reasoning process with a deeper comprehension of word embeddings and bounding box characteristics, resulting in superior performance and more precise predictions.

4.5.3 Implementation details

In our experiments, we utilize the base architecture described in Chapter 3 and enhance it by incorporating our teacher-guidance techniques, which include both input guidance and output feedback. Below, we provide the implementation details for each of these teacher-guidance methods. We implement the code and conduct experiments using PyTorch on Nvidia GPUs.

In the context of input guidance of the decaying teacher forcing, the probability of employing teacher forcing is calculated as $\epsilon_e = \epsilon_{e-1} - \eta$. We set the initial teacher forcing probability at $\epsilon_1 = 1$ and reduce it by η every epoch. In our initial experiments, we selected a decaying factor of $\eta = 0.1$. Consequently, the teacher forcing guidance was gradually phased out over just 10 epochs, after which the modules were allowed to train independently in an open-loop manner. However, we observed that this decaying factor was relatively high, and the modules required more substantial teacher forcing. Subsequently, we adjusted the factor to $\eta = 0.05$, extending the decaying teacher forcing period to 20 epochs. This modification allowed for a more gradual transition, providing the modules with increased teacher guidance over a more extended training period.

Regarding the output feedback mechanism, we determine distinct weights for each type of loss when calculating the multi-task loss. Our selection of these weights is based on the observed scales of each loss type: we assign 0.1 to the answer loss, 0.7 to the attention loss, and 0.2 to the boolean loss. To calculate the losses for various module functions and compute their average loss, we make use of a masking mechanism to calculate the averages per module function.

Optimization. As for the optimizer used in training, we utilize the Stochastic Gradient Descent (SGD) optimizer and conduct experiments with two different learning rates: 0.1 and 0.03. The batch size employed is 1024.

4.5.4 Modules training evolution

In this subsection, we provide an analysis of the training loss evolution observed during the training process of our model. These loss curves serve as invaluable tools for understanding the learning dynamics of the different modules, especially when integrating the output feedback and the input guidance as part of our teacher-guidance strategies. The input guidance is used for 20 epochs and the output feedback is preserved for all the duration of training. The visualizations, as depicted in Figure 4.5, categorize the modules into three types: attention modules, Boolean modules, and answer modules. Each type is represented separately to allow for a closer examination of their training behaviors.

Starting with the attention modules (Figure 4.5a), we note some interesting patterns. The `relateAttr` module exhibits a positive trend with decreasing loss over time, indicating effective learning. However, other attention modules show unique behaviors. For instance, `select`, `relateSub`, and `relateObj` initially demonstrate loss reduction when teacher forcing is at its peak. Still, they encounter an increase in error as teacher forcing diminishes, showcased by the subsequent dip in performance. This is explained by the fact that when reducing teacher forcing probability, the modules start propagating their outputs to subsequent modules that are inherently less accurate than ground truth targets. Consequently, the subsequent modules receive less accurate inputs, which leads to an increase in errors during the training process. The modules `filterAttr`, `filterNot`, and `filterPos` present a different challenge. After approximately 40 epochs, their losses plateau and show a slight increase as teacher forcing wanes.

Turning to the Boolean modules (Figure 4.5b), they predominantly demonstrate a decreasing loss trend during training, converging toward minimal errors. Modules like `same`, `sameAll`, `different`, and `differentAll` particularly benefit from weight-sharing techniques. Nevertheless, `and` and `or` modules appear to have higher losses compared to the rest. Despite having no trainable parameters themselves, these modules accumulate the errors from the previous reasoning branches, which can result in higher losses. The fact that the `exist` module has a relatively high loss can be attributed to the inherent difficulty of its task.

Lastly, the answer modules (Figure 4.5c) have a more diverse range of behaviors. Some modules, like `chooseName`, `queryName`, and `queryAttr`, display steady loss reduction throughout training.

4.5. EXPERIMENTS

In contrast, `queryPos`, `chooseRel`, and `choosePos` show quicker loss reduction initially, eventually reaching a plateau. The `compare` module presents the most substantial error, primarily due to the relatively low number of `compare` questions in the training data, as indicated in the dataset section 2.4. The use of the Binary Cross-Entropy loss with logits further accentuates these errors. In contrast, `AnswerLogic` modules exhibit comparatively lower errors, reflecting the simpler nature of their task—choosing between “yes” or “no” as answers, compared to the open-ended questions addressed by other answer modules.

In summary, these training curves provide valuable insights into the learning behaviors and challenges associated with different modules in the model.

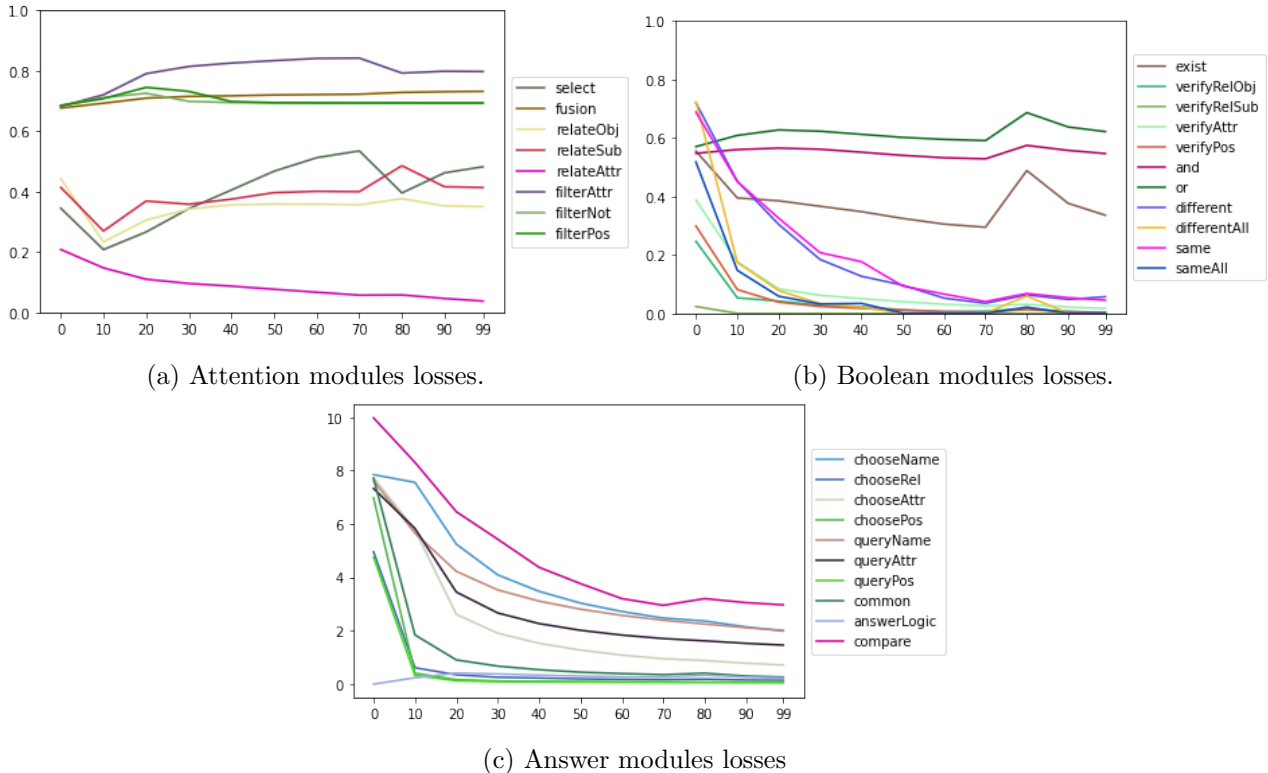


Figure 4.5: Modules’ training loss curves plotted every 10 epochs.

4.5.5 Qualitative analysis of the modular approach

In Figure 4.6, we provide a visual representation of the reasoning process for three distinct questions, showcasing instances where the NMN exhibits the expected behavior. We highlight the bounding boxes with the highest attention values from the attention output vector. For boolean modules, we

4.5. EXPERIMENTS

display the output probability and finally the predicted answer.

The first example, labeled as “Question 1”, queries ‘What color are the tennis shoes, white or red?’ This question falls under the category of attribute choice assessment. It involves a sequence of operations within the NMN. Initially, the `select(shoes)` module localizes the shoes object. Subsequently, the `filterAttr(tennis)` module is employed to narrow the attention specifically to tennis shoes and retaining focus on the previously identified object. The resulting attention vector is then fed into the `chooseAttr(white | red)` module. This module utilizes the provided input arguments to classify the answer and predict the correct response, which, in this case, is ‘white’.

In “Question 2”, categorized as an object existence assessment, the task is to identify white skateboards within the given image, phrased as ‘Do you see any skateboards that are white?’. The process unfolds as follows: initially, the first step `select(skateboard)` effectively singles out the skateboard as the primary object of interest. Subsequently, the second module `filterAttr(white)` shifts the focus of attention towards white objects. However, since the skateboard is not white, the attention pivots towards the white building. The `exist` module then evaluates whether there is an object with a notably high attention value. Based on this evaluation, it generates a low probability score of 0.3, from which the answer module `answerLogic` predicts the answer ‘no’.

In “Question 3”, which is a query-type question asking ‘What is the item of furniture to the right of the television called?’, the program execution unfolds as follows. Initially, the `select(television)` module identifies the television object within the image. Subsequently, the focus is directed towards objects positioned to the right of the television, a task performed by the `relateSub(to the right of)` module. Among these objects, the `select(furniture)` module singles out the furniture in question. The outputs from these two modules, representing the objects to the right of the television and the selected piece of furniture, are merged by the `fusion` module to ensure sustained attention on them. Finally, the `queryName` module undertakes the classification of the name of the detected object and predicts it, with the result being ‘couch’ in this particular instance.

These examples demonstrate the explainability of our approach and the ability to trace the model’s decision-making process.

In Figure 4.7, we provide examples where the NMN fails to predict the correct answer. We have highlighted the bounding box with the highest attention value from the intermediate attention vectors

4.6. CONCLUSION


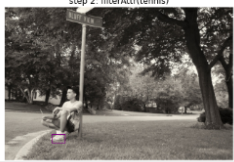


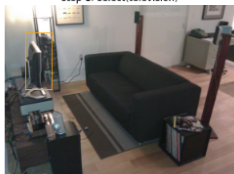
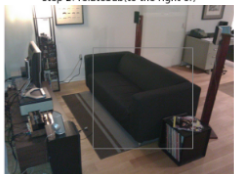


Question 1: What color are the tennis shoes, white or red?				
		step 3: chooseAttr answer = white		
Question 2: Do you see any skateboards that are white ?				
		step3: exist prob(output) = 0.329	step 4: answerLogic answer = no	
Question 3: What is the item of furniture to the right of the television called?				
				step 4: queryName answer = couch

Figure 4.6: Visualization of the reasoning process.

in each case. For the first question, which inquires about whether the hat is dry or not, the select module correctly identifies the hat object. However, the verifyAttr module assigns a low probability value, leading to an incorrect answer. In the second example, a common error occurs where the model predicts an answer that is semantically close to the correct answer, such as “brown” instead of “orange”. In the third example, the question about the type of vehicle in front of the flag. The model successfully detects both the flag and the vehicle but fails to classify the correct type of vehicle, which is a “van”. It’s important to note that other incorrect predictions in the testdev-all set can also be attributed to failures in the intermediate states of the reasoning process, ultimately resulting in incorrect answers.

4.6 Conclusion

In this chapter, we presented a teacher-guidance training strategy for NMNs, which has demonstrated several key contributions. We proposed two techniques to better supervise the program modules performance. The first technique, known as input guidance, involves supplying ground truth inputs to intermediate modules instead of their previous module’s outputs. This approach helps prevent error propagation during the early stages of training. The second technique, intermediate output feedback,

4.6. CONCLUSION








Question 1: Is the hat dry?				
step 1: select(hat) 	step 2: verifyAttr prob(output) = 0.032	step 3: answerLogic answer = no GT = yes		
Question 2: Which color is the hydrant on the left?				
step 1: select(hydrant) 	step 2: filterPos(left) 	step 3: queryAttr answer = brown GT = orange		
Question 3: Which kind of vehicle is in front of the flag?				
step 1: select(flag) 	step 2: relateSub(in front of) 	step 3: select(vehicle) 	step 4: fusion 	step 4: queryName answer = Truck GT = van

Figure 4.7: Visualization of the reasoning process for some incorrect answers.

calculates individual module losses and optimizes them to perform their specific sub-tasks.

Our approach enhances generalization and promotes a transparent reasoning process, as evidenced by the experimental results on the GQA dataset. By harnessing our proposed approach, the neural modules acquire the capability to learn their reasoning sub-tasks both independently and in an end-to-end manner. This not only enhances training efficiency but also increases the interpretability of the system, allowing for a better understanding of the underlying reasoning processes. In addition to the aforementioned contributions, our work paves the way to a better understanding of NMNs for the task of visual reasoning.

Chapter 5

Curriculum learning for neural module networks

Contents

5.1	Introduction	76
5.2	Curriculum Learning setup	76
5.2.1	Difficulty criterion	76
5.2.2	Scheduler	78
5.2.3	Sampling function	78
5.2.4	Performance evaluator	79
5.3	Experiments	79
5.3.1	Evaluated methods	80
5.3.2	Results analysis	81
5.3.3	Modules performances	84
5.3.4	Implementation details	85
5.3.5	Additional experiments	86
5.4	Conclusion	87

5.1 Introduction

To optimize the training of Neural Module Networks (NMNs) beyond teacher guidance, we develop a dynamic approach known as Curriculum Learning (CL). This training strategy revisits the learning process by thoughtfully organizing the training examples before feeding them to the NMN. Given the inherently modular architecture of NMNs, it may seem obvious that commencing with the acquisition of shorter programs before gradually advancing to more complex ones is a reasonable approach. Curriculum Learning (CL) [Elman, 1993, Soviany et al., 2022, Wang et al., 2022b] consists in learning the easier parts of the task first, before tackling it entirely. In this chapter, we study this training strategy and elaborate several difficulty criteria based on which we order the training examples. In Section 5.2, we build on previous works using CL as mentioned in the related work Section 2.3.3 and define the CL strategy we adopted in the context of NMNs. Then, in Section 5.3, we detail the evaluated approaches and analyse the results. Ultimately, we draw insightful conclusions regarding the integration of CL principles within the modular framework. This Chapter presents our VISAPP conference paper [Aissa et al., 2023a].

5.2 Curriculum Learning setup

Our objective is to investigate Curriculum Learning (CL) techniques for Visual Question Answering (VQA) and identify a CL method that not only reduces training expenses but also optimizes data utilization through efficient sampling and repetition techniques. Typically, a CL method comprises several components, including a difficulty criterion, a scheduler and a sampling function. In the subsequent subsections, we provide insights into each of these elements.

5.2.1 Difficulty criterion

The difficulty criterion allows to characterize the samples: training starts with the “easiest” samples, then progressively moves toward more “difficult” samples. Different criteria were adopted in the literature, including program length, answer hierarchy, and question loss. For instance, question loss was employed with some success as a difficulty criterion in [Sachan and Xing, 2016] for QA and in [Askarian et al., 2021] for VQA about synthetic CLEVR [Johnson et al., 2017a] images. However, computing question loss requires a first training iteration over all the training data. Our difficulty

5.2. CURRICULUM LEARNING SETUP

criteria is based on the premise that reasoning about a *single* object and its properties is simpler than examining the relations between *several* objects or comparing their attributes. The number of different objects in the question should then be a good indication of the complexity of reasoning and thus a relevant *a priori* difficulty criterion for CL. In Figure 5.1 we showcase four examples, each for a different difficulty level.


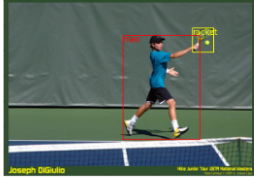

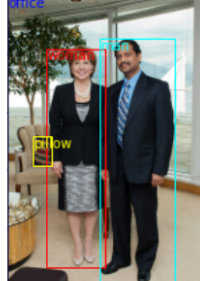
<p>Question: Which color do you think the train car is?</p> <p>Program: <code>select(train car), queryAttr(color).</code></p> <p>Difficulty level: 1</p> 	<p>Question: Is the racket to the right or to the left of the person in the middle of the picture?</p> <p>Program: <code>select(person), filterPos(middle), select(racket), choosePos(to the left or to the right).</code></p> <p>Difficulty level: 2</p> 	<p>Question: What do you think is the piece of furniture to the right of the white animal that is lying on the dining table?</p> <p>Program: <code>select(dining table), relateSub(lying on), select(animal), fusion, filterAttr(white), relateSub(to the right of), select(furniture), fusion, queryName.</code></p> <p>Difficulty level: 3</p> 
 <p>Question: Do both the man in the office and the woman to the right of the pillow look happy?</p> <p>Program: <code>select(office), relateSub(in), select(man), fusion, verifyAttr(happy), select(pillow), relateSub(to the right of), select(women), fusion, verifyAttr(happy), and, answerLogic.</code></p> <p>Difficulty level: 4</p>		

Figure 5.1: GQA dataset samples with different difficulty levels.

Program length is another potentially relevant criterion that takes into account the flow of gradient in the NMN structure. Shorter programs necessitate fewer gradient computation steps, while longer programs require more extensive gradient computations. This criterion is closely tied to the number of objects within a question: questions with more objects generally result in longer programs. However, program length is not solely contingent on the number of objects. Even questions involving a single object can entail a high number of functions applied to that object. For instance, consider the question “Is there a rabbit that is white and black in the picture?” which corresponds to the following program: `[select(rabbit), VerifyAttr(black), Exist(), select(rabbit), VerifyAttr(white), Exist(), and(), answerLogic()]`. Despite addressing a single object, this reasoning chain consists of 8 modules.

Given these considerations, multiple criteria can be combined to establish the increasing difficulty

5.2. CURRICULUM LEARNING SETUP

of training samples in CL. As a primary criterion, we sort samples based on the number of objects and group them according to the number of concepts within the question. A secondary criterion refines this ordering within each group: for each specific number of objects, we start with shorter programs, progress to medium-length ones, and conclude with longer programs. This approach enables a structured and progressively challenging learning process. Next we delve into the specifics of the scheduler.

5.2.2 Scheduler

The role of the scheduler in Curriculum Learning (CL) is to determine the timing for updating the curriculum, indicating when to transition from one difficulty level to the next. Various scheduling strategies can be employed, some of which rely on heuristics derived from factors such as loss values, training time, or the number of examples seen.

In our approach, we have chosen a systematic and predetermined sample size for each difficulty level, offering a straightforward controlled solution. Throughout CL, we maintain a consistent sample size of 1 million examples per CL iteration. The scheduler can also be tasked with repeating a training setup if specific transition criteria are not met.

5.2.3 Sampling function

The sampling function plays an important role in shaping the selection of training examples across various difficulty levels by assigning specific weights to individual examples. The effective approach we propose is to ensure a balanced distribution of occurrence probabilities among different types of answer modules. This also helps reduce the answer modules classification bias. Another criterion, which draws inspiration from boosting techniques, involves giving preference to programs (examples) that have resulted in higher errors during training. This is achieved by computing the average loss for each module across all previously encountered examples and assigning greater weight to programs composed of modules with higher loss values. It is important to note, for the number of training examples, that we employ a sampling approach *with replacement*. Consequently, the count of *distinct* examples processed by the executor is lower than the specified sample size of 1M.

Employing a CL strategy, where training examples are organized by their level of difficulty, introduces a potential challenge known as “catastrophic forgetting”. This phenomenon, documented

5.3. EXPERIMENTS

in the literature [ROBINS, 1995, Greco et al., 2019], entails the loss or interference with previously acquired knowledge when new information is learned. To prevent the risk of catastrophic forgetting, we augment the current training sample (reflecting the current difficulty level) with a random selection of samples from previous levels. This approach ensures that the model continues to learn from a diverse range of examples, preserving its ability to handle both simple and complex tasks. To provide further detail, beginning with the second Curriculum Learning (CL) iteration we incorporate a sampling mechanism where 20% of the training sample is drawn from the examples encountered in the preceding iterations.

5.2.4 Performance evaluator

The performance evaluator is a component or mechanism that assesses the model’s performance on the training examples or curriculum at a given difficulty level. Several evaluators are employed to compare CL and standard learning. Since our focus is on reducing the *cost* of training, we measure the total number of example presentations during training as the computational cost (Comp. cost). We also mention the maximal number of *different* examples seen during training (*# examples*), as different methods can make use of larger or smaller portions of the training set. While we focus on the cost of training, we also want to reach an accuracy that is close to the one obtained by standard learning, so we also report the accuracy of the predicted answers.

5.3 Experiments

To expand the pool of examples available for CL, we opted to utilize the unbalanced version of the GQA dataset [Hudson and Manning, 2019b]. While the balanced version of the dataset maintains a uniform distribution of answers, it contains a relatively limited number of examples, approximately one million. By using the unbalanced dataset, we gain access to a larger and more diverse set of examples, which provides a richer training environment for our CL experiments. This choice allows us to explore a wider range of question types, difficulties, and answer distributions, ultimately enhancing the robustness and adaptability of our CL strategy for the VQA task. In the following section, we will provide a comprehensive overview of the various methods we have evaluated for CL.

5.3.1 Evaluated methods

When describing the different performed experiments, we use the following notations, which correspond to different algorithmic choices:

- **Unbalanced:** We train on all the examples from the unbalanced GQA train split, we use the traditional random batch training strategy and the model sees all the data examples in every epoch.
- **Balanced:** We train on the balanced version of the GQA dataset. At every epoch, the model is trained on all the balanced dataset training examples.
- **Random:** Instead of training on all the dataset examples, only 1M random selection of examples from the unbalanced dataset are presented to the model at every iteration.
- **CL:** The model is trained using Curriculum Learning and the sampling is driven by the number of objects in the programs. At every CL iteration the model sees 1M examples filtered from the unbalanced dataset by the curriculum sampler. The training needs 4 CL iterations to be complete, where each iteration has an increased difficulty given by the number of objects of its programs (ranging from 1 to 4).
- **Length (L):** The curriculum sampler filters the programs by their lengths for each number of objects, a CL-iteration is defined by a number of objects and a program length (short or medium or long).
- **Weights (W):** We use several sampling weights for the filtered programs:
 - **W.a:** To make the answer modules distribution of the resulting sample more uniform we use the ‘answer module’ weighting; this balances the answer modules occurrences in the result sample so that the model equally sees all the defined answer modules.
 - **W.b:** The ‘modules loss’ weighting indicates that an example’s weight is proportional to the sum of the average losses of the modules composing its program, to focus the model on harder examples.
 - **Uniform:** indicates that the sampling is uniform, so the sampling with replacement results in a sample that is uniformly distributed over all the dataset.

5.3. EXPERIMENTS

- **Pretrain (P)**: The model’s parameters are initialized from a model trained using the **Random** variant described above.
- **Repeat (R)**: We repeat the same CL-iteration twice.

5.3.2 Results analysis

Model	CL configuration			Iterations	Number of examples (\leq)	Accuracy
	weighting	pretraining	iterations/level			
CL+W.a	answer	–	1	4	4 M	0.642
CL+W.b	losses	–	1	4	4 M	0.635
CL+W.a+P	answer	2 iterations	1	[2] + 3	5 M	0.670
CL+W.a+P+R	answer	2 iterations	2	[2] + 5	7 M	0.681

Table 5.1: Results on `testdev-all` for several CL strategies.

This section presents an analysis of the performance and the cost of our modular VQA framework with multiple CL training strategies, followed by a comparison with models not using CL to show the effectiveness of our proposed training approach.

Comparison of CL methods. We start by a comparative analysis of the proposed CL strategies as described in Sec. 5.3.1. Table 5.1 reports the performance of our model based on the different CL configurations. The goal of CL is to make the training more effective and to achieve the highest accuracy while training for fewer iterations. Therefore, for each model are shown the number of iterations and training examples required to reach the highest accuracy.

From the results, it is clear that the ‘answer module’ weighting is the most effective weighting function. One can see this as a balancing of the answer modules presence over the training sample. The CL+W.a model (using the ‘answer module’ weighting) achieves higher accuracy results than the CL+W.b model (with the ‘loss’ weighting), both reaching their top respective accuracies after 4 training iterations only. The ‘answer module’ weighting also yields better accuracy than the ‘uniform’ weighting after the same number of training iterations. This is shown by comparing CL+L (‘uniform’ weighting by default) and CL+L+W.a in Table 5.2. Moreover, the accuracy of CL+L+W.a continues to increase after the 11th iteration to achieve its top at iteration 12. The superior performance of the ‘answer module’ weighting function in two different comparable settings makes us select this weighting for the rest of the experiments.

5.3. EXPERIMENTS

Model	Computation cost	# examples	Accuracy
CL+L	11	11 M	0.650
CL+L+W.a	12	12 M	0.655

Table 5.2: Results on `testdev-all` with program length as a refinement for the CL difficulty measure. Computation cost is the number of seen examples per iteration times the number of iterations.

The refinement of the CL difficulty (or hardness) measure using the number of question objects (Length-CL difficulty measure) increases the CL+W.a top accuracy by 1%, see the CL+L+W.a line in Table 5.2. However, this improvement has a significant cost, as CL+L+W.a requires 12 training iterations (12M examples) unlike CL+W.a which only needs 4 iterations (4M examples). This reinforces the idea that with a more refined difficulty measure the model has more time to adjust to difficult examples, and its accuracy gradually increases to achieve a better top accuracy in a CL setting. But training on 12M examples is expensive since the overall dataset size has 14M examples. We thus decided to explore different options to obtain comparable results at a lower cost.

A promising finding was that pretraining the models for a few iterations with randomly sampled 1M examples each leads to an accuracy increase of over 1.5%, as shown by the CL+W.a+P model which was pretrained for only 2 iterations. This “warms up” the model to the modular aspect of our VQA framework, allowing it to be more general and effective before starting the CL.

An interesting finding was that the model reached peak accuracy before iterating over the full CL configuration. The accuracy drop resulting after the 4th iteration may be explained by model overfitting on the questions with 4 objects. Indeed, in the GQA dataset these questions have a substantially unbalanced answer distribution.

A further finding is that repeating the same CL-iteration twice (as in CL+W.a+P+R) improves the top accuracy results by 1.1%, while only moderately increasing the number of iterations. This can be explained by the fact that doubling the number of training iterations helps the model better understand the structure of problem without augmenting the training data size. As detailed in Sec. 5.3.1, when sampling with replacement we obtain a number of distinct examples that is slightly lower than the sample size, therefore the reported number of examples (# examples) is an upper bound of the number of examples actually employed.

As a general conclusion, we consider the CL+W.a+P+R model as the best modular VQA model that scores the best accuracy of 68.1% after 7 training iterations using less than 7M distinct examples,

5.3. EXPERIMENTS

i.e. less than half of the training data.

Impact of CL. We perform several experiments to assess the impact of the CL on our compositional visual reasoning framework. We do this by training our model without CL (Unbalanced, Balanced, and Random configurations described in Sec. 5.3.1), then comparing the accuracy performance and the experiment cost in terms of both computation cost and number of different training examples. In Table 5.3 we report the accuracy and cost results of the conducted experiments and compare them to the performance of our best CL model **CL+W.a+P+R**.

Model	Computation cost	# examples	Accuracy
Unbalanced	9×14 M	14 M	0.702
Balanced	50×1.4 M	1.4 M	0.678
Random	12×1 M	≤ 12 M	0.694
CL+W.a+P+R	7×1 M	< 7 M	0.681

Table 5.3: Comparison of our CL model (CL+W.a+P+R) with no-CL models (Unbalanced, Balanced, and Random) on the `testdev-all` set.

The Unbalanced model (trained on the entire unbalanced training set of 14M) achieves the highest accuracy value of 70.2%. This model also has the highest training cost among the evaluated models.

The Balanced model, trained on the balanced dataset for a large number of epochs, achieves lower results than the Unbalanced model. This is partly due to the fact that the balancing reduces not only the number of questions in the dataset, but also the diversity of the programs. Also, to the use of the unbalanced `testdev-all` for evaluation.

By comparing our best CL model (**CL+W.a+P+R**) to the models trained without CL (no-CL), we find very significant gains in terms of computational cost, *e.g.* an 18-fold reduction compared to the top contender, the model trained on the Unbalanced dataset. The price to pay—a drop of only 2% in accuracy—appears reasonable. The Random model, trained on randomly sampled 12M examples, performs almost as well as the Unbalanced model, an expected result since both models use a similar amount of distinct training examples (12M vs. 14M). The Unbalanced model requires an almost 9 times more expensive training than Random, but the improvement in accuracy (70.2 % vs. 69.4%) hardly justifies it. However, the proposed CL model has an almost 2 times lower computational cost than Random, confirming the superiority of curriculum learning in this type of application.

5.3.3 Modules performances

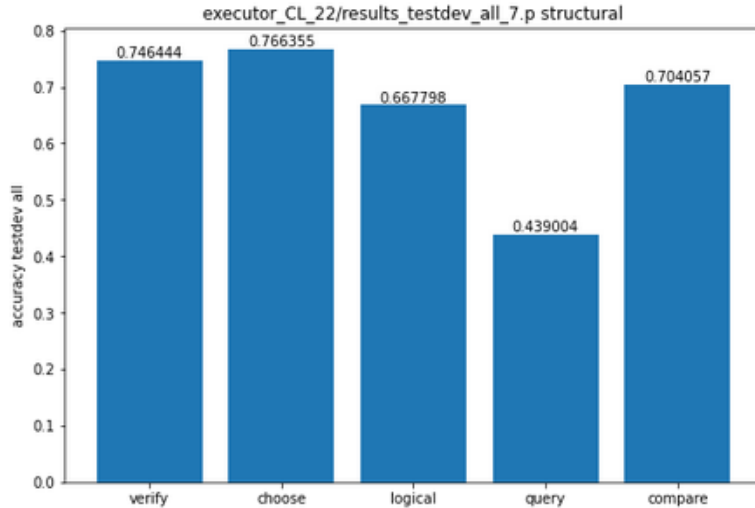


Figure 5.2: Accuracy histograms for the structural types of the questions.

To gain deeper insights into our model’s performance in relation to the structural and semantic types of questions, we have generated in Figures 5.2 and 5.3 accuracy plots for our best Curriculum Learning (CL) trained model for each question type.

As described in Section 2.4, the structural type is determined by the final answer module used in the question program. Notably, questions falling into the “choose”, “verify”, and “compare” structural types exhibit higher accuracies when compared to the overall average accuracy across all questions. Among these, “choose” questions, which require selecting between two alternatives, exhibit notably high accuracy levels. This is primarily attributed to the fact that the alternatives presented in the choose module’s textual argument directly correspond to the two classes from which the answer must be chosen. Consequently, this indirectly reduces the number of potential answers and, as a result, lowers the complexity of these questions in comparison to other structural types.

The logical questions involve the questions requiring logical inferences using dual reasoning branches inputted to AND or OR operations, these questions have an accuracy of 66.7%.

Conversely, the “query” questions exhibit a lower accuracy of 43.9%. These types of questions involve open-ended inquiries related to object classification, attribute classification, and relationship querying, which tend to be more intricate and challenging, resulting in comparatively lower accuracy levels.

5.3. EXPERIMENTS

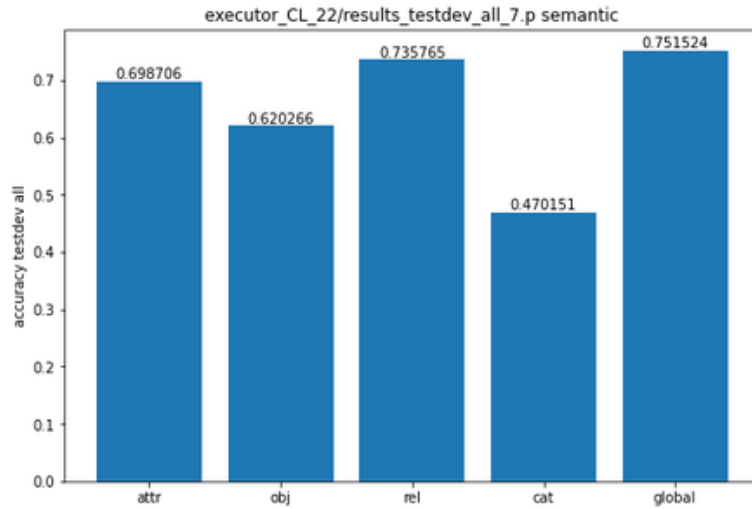


Figure 5.3: Accuracy histograms for the semantic types of the questions.

Transitioning to an examination of the model’s performance in relation to question semantic types, which pertain to the primary subject of inquiry in the question:

Global semantic type questions, inquiring about weather or location, exhibit the highest performance, achieving an accuracy rate of 75.15%.

Questions that seek information about the subject or object of a described relationship also achieve a high performance level of 73.5%. It’s worth noting that this is the most prevalent semantic type in the dataset, encompassing 52% of the questions.

Semantic type questions related to object categorization within a class, exhibit the lowest performance in our evaluation. For instance, questions like ‘What kind of fruit is on the table?’ entail not only localizing the fruit’s bounding box in the image but also classifying it to a more specific category within a larger class. Despite receiving a higher level of granularity for an object class as indicated in the question, the model encounters difficulties when attempting to identify the finer details or lower granularity of the object class.

5.3.4 Implementation details

The CL experiments use the output feedback mechanism from Section 4.3 and we determine distinct weights for each type of loss when calculating the multi-task loss. Our selection of these weights is based on the observed scales of each loss type: we assign 0.1 to the answer loss, 0.7 to

5.3. EXPERIMENTS

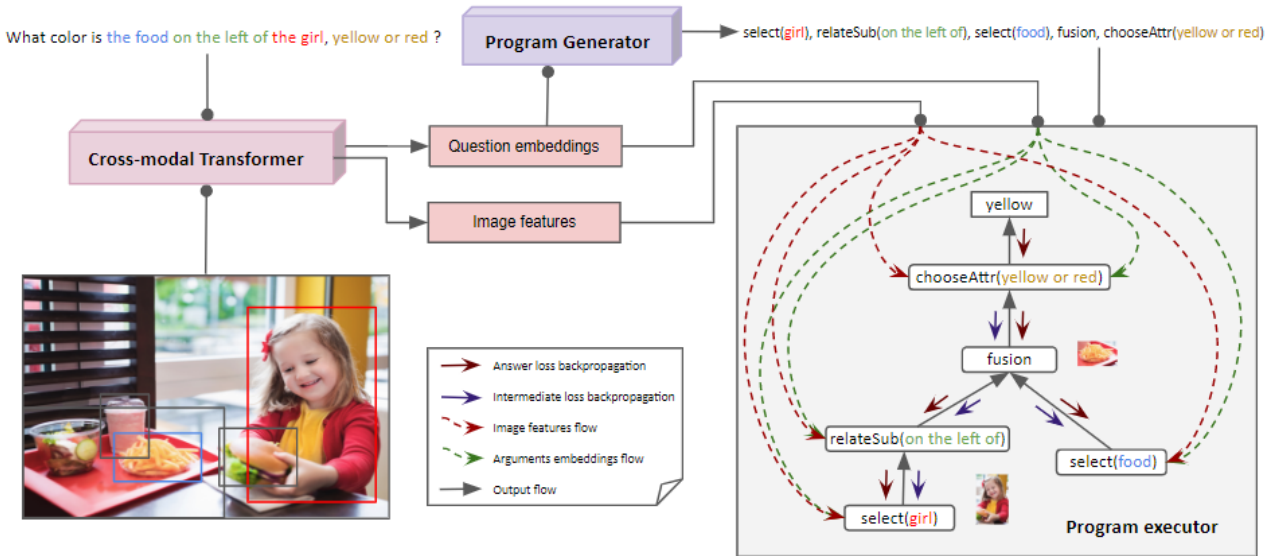


Figure 5.4: The program (represented as a sequence of sub-task modules) is applied by the Program Executor module to the image to answer the question. The proposed work focuses on improving the Program Executor by using several curriculum learning (CL) strategies.

the attention loss, and 0.2 to the boolean loss. As for the optimizer used in training, we utilize the Stochastic Gradient Descent (SGD) optimizer and conduct experiments with the same learning rate of 0.1. The batch size employed is 1024.

5.3.5 Additional experiments

In addition to the weighting mechanisms discussed in Section 5.3.1, we conducted further primary experiments that were not explicitly mentioned in the research papers or the preceding sections. Specifically, we explored alternative weighting mechanisms when sampling the training examples for each difficulty level. One approach was to balance the examples within the samples based on the occurrences of all program modules, not only the answer module in the program as done using the weighting “answer module frequency”. Another approach involved proportional weighting based on the number of parameters associated with the program modules. Programs featuring modules with a greater number of parameters were assigned higher weights for selection. Although that the experiments we conducted were not exhaustive in comparison to the other weighting techniques, these alternative weighting mechanisms yielded lower validation results when compared to the “answer module frequency” and the “modules loss” weighting mechanisms.

5.4 Conclusion

In this chapter we presented several Curriculum Learning (CL) strategies within a Neural Module Network (NMN) framework for Visual Question Answering (VQA). Our model employs an NMN architecture composed of multiple neural modules, each capable of performing a reasoning sub-task. We compare several CL strategies for training our NMN architecture to solve a VQA task. Our model is evaluated on the GQA dataset [Hudson and Manning, 2019b] and shows very interesting results in terms of computational cost reduction concerning the required training examples and training iterations.

To drive the CL strategy, we introduced a difficulty measure based on the number of objects in the question. Additionally, we examined a difficulty refinement technique based on the program length, which, in this context, proved to be less effective. Furthermore, we assessed several weighting techniques to control the sampling of the programs within the same difficulty group with the observation that a balance between module samples based on the frequency of answer modules appeared to yield favorable outcomes in CL. By applying the appropriate CL strategy we achieve close accuracy results by training on a judiciously sampled 50% of the training data, compared to an NMN model trained without CL on the entire training set.

Chapter 6

Conclusion and Perspectives

To conclude this PhD manuscript, we will provide a comprehensive summary of the work conducted and the key findings obtained throughout this research. Furthermore, we will delve into the potential perspectives.

Motivated by the compositional aspect of reasoning and the multimodal nature of inputs and their interactions in a visual question answering (VQA) setup, we proposed a Neural Module Network (NMN) architecture that takes as input a question, an image and a functional program representing the step-by-step reasoning process to follow in order to infer an answer.

We also consider compositional representations of both the image and the question, where the image is represented by its objects and the question by its words. Exploiting the prowess of Transformer models in learning cross-modal representations, we use a vision and language pre-trained model as our features extractors. Specifically, we employ LXMERT [Tan and Bansal, 2019], a model pre-trained on a large-scale dataset encompassing a variety of unimodal and multi-modal vision and language tasks to learn vision and language alignments. LXMERT is also fine-tuned on the GQA dataset, the dataset we use in this research. Through experimental comparisons between uni-modal and multi-modal representations, we have demonstrated the superior efficacy of multi-modal representations and their positive impact on our NMN architecture [Aissa et al., 2023b].

We design and develop our module taxonomy, where each module specializes in learning a distinct reasoning sub-task. These modules fall into three categories based on their functional roles: 1) Attention Modules: these modules have the objective of highlighting pertinent regions within the image. They learn tasks such as object detection, attribute detection, and relationship detection. 2) Boolean

CONCLUSION AND RESEARCH PERSPECTIVES

Modules: this category encompasses modules that make logical inferences. They perform tasks like conjunction, disjunction, or verification to assess the veracity of a given property. 3) Answer Modules: they classify the answer to the question.

For each question and image, a Neural Module Network (NMN) is constructed. This construction adheres to a predefined functional program, which is executed sequentially on the image to predict an answer. This process involves the cooperation and information exchange between modules through dependencies, where the output of one module serves as the input for the subsequent module in the sequence. To mitigate the inherent complexity of the tasks, we employ a weight-sharing technique that considers the functional and structural characteristics of the diverse modules.

While NMNs provide a more transparent and interpretable reasoning process compared to monolithic architectures that rely on single, all-encompassing blocks, it does present notable training challenges. Consequently, we introduce a series of training techniques aimed at boosting NMN’s performance and reducing training costs.

First, we use teacher guidance techniques to supervise the modules. Aside from training the modules along the reasoning chain to collectively yield precise answers and optimising them based on the answer error, we introduce an innovative teacher-guidance method that specifically targets the enhancement of intermediary modules within the reasoning chain. This approach ensures that these modules perform their designated reasoning sub-tasks, avoiding shortcuts that might compromise the overall reasoning process.

Two distinct teacher-guidance techniques are introduced in our work. One of these techniques is inspired by the widely-used teacher-forcing method, commonly employed in sequence-to-sequence models. Here, we provide the ground-truth input to subsequent modules rather than relying on the predicted output from the previous module. This prevents error propagation, especially during the early training stages. As training progresses, we gradually reduce the usage of teacher forcing. This approach allows the modules to learn their specific tasks autonomously and adapt to collaborative reasoning with other modules over time. Another teacher-guidance technique we employ draws inspiration from prior research and resembles multi-task training. In this approach, our loss function takes the form of a weighted sum of the losses incurred by individual modules. A thorough comparative analysis is provided in our publication [Aissa et al., 2023b] elucidating the advantages and efficacy of our teacher-guidance approach in the context of NMNs.

CONCLUSION AND RESEARCH PERSPECTIVES

Another notable contribution centers on the design of a Curriculum Learning (CL) strategy designed to reorder the training examples in a progressively challenging manner. In essence, the model commences its training with simpler examples and gradually proceeds to more difficult ones. This approach is rooted in the proven effectiveness of CL in leveraging available data resources more effectively. To this end, we formulate a CL training strategy tailored for NMNs that employs the number of concepts contained in a question as a difficulty metric. In other words, questions with more concepts are considered more difficult. Furthermore, we conduct experiments exploring various weighting techniques aimed at refining the ordering of examples within each difficulty level. This helps to balance the occurrences of answer modules. Our comprehensive experimental results affirm that the judicious application of CL strategies can significantly reduce the training overhead for NMNs, as detailed in our research paper [Aissa et al., 2023a].

From a broader point of view, it would be interesting to see if a curriculum learning strategy could benefit from the compositionality of other tasks, such as compositional action recognition in videos. Breaking down complex actions into simpler sub-actions and gradually introducing more challenging combinations could potentially enhance the training of machine learning models for action recognition problems in visual scenes. This exploration finds particular relevance in the context of benchmarks like the one proposed by [Luo et al., 2021], which provides a structured hierarchy and compositional breakdown of complex human activities. By leveraging CL, models could potentially follow a trajectory from comprehending basic sub-actions to gradually tackling more intricate action compositions. Such an approach has the potential to advance the recognition and parsing of complex activities in video data, with applications like surveillance systems, video-games or human-robot interaction.

Recently, due to the rapid advancement of Large Language Models (LLMs) such as ChatGPT [OpenAI, 2023], BLOOM [Scao et al., 2022] or [Almazrouei et al., 2023], new techniques were proposed aimed at addressing the task of VQA by harnessing the capabilities of these LLMs. These emerging approaches exploit the power of language and employ guided prompting for multi-step reasoning in complex VQA tasks. Similar to NMNs, a class of these techniques adopts a code generator and executor framework, which introduces a novel dimension to VQA. Here, the code generator translates questions into Python-like modular programs through prompting an LLM. Subsequently, the generated code is executed on the image, facilitated by a Python interpreter and a set of API calls. This approach has been explored in recent research works such as ViperGPT, VisProg, and CodeVQA

CONCLUSION AND RESEARCH PERSPECTIVES

[Surís et al., 2023, Gupta and Kembhavi, 2023, Subramanian et al., 2023], offering promising avenues for enhancing VQA capabilities. As described in this work, the planning and executing paradigm offers a more explainable and traceable decision-making process and can be extended to other domains, including video games, where an agent has to learn how to navigate complex objectives and missions within a game environment.

Bibliography

- [Agrawal et al., 2016] Agrawal, A., Batra, D., and Parikh, D. (2016). Analyzing the behavior of visual question answering models. *arXiv preprint arXiv:1606.07356*.
- [Aissa et al., 2023a] Aissa, W., Ferecatu, M., and Crucianu, M. (2023a). Curriculum learning for compositional visual reasoning. In Radeva, P., Farinella, G. M., and Bouatouch, K., editors, *Proceedings of VISIGRAPP 2023, Volume 5: VISAPP*, pages 888–897. SCITEPRESS.
- [Aissa et al., 2023b] Aissa, W., Ferecatu, M., and Crucianu, M. (2023b). Multimodal representations for teacher-guided compositional visual reasoning. In Blanc-Talon, J., Delmas, P., Philips, W., Popescu, D., and Scheunders, P., editors, *Advanced Concepts for Intelligent Vision Systems, 21st International Conference (ACIVS 2023)*. Springer International Publishing.
- [Almazrouei et al., 2023] Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., Cojocaru, R., Debbah, M., Goffinet, E., Heslow, D., Launay, J., Malartic, Q., Noune, B., Pannier, B., and Penedo, G. (2023). Falcon-40B: an open large language model with state-of-the-art performance.
- [Anderson et al., 2018] Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., and Zhang, L. (2018). Bottom-up and top-down attention for image captioning and visual question answering. In *CVPR*.
- [Andreas et al., 2016a] Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016a). Learning to compose neural networks for question answering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1545–1554, San Diego, California. Association for Computational Linguistics.

BIBLIOGRAPHY

- [Andreas et al., 2016b] Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016b). Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48.
- [Antol et al., 2015] Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433.
- [Arinaldi et al., 2018] Arinaldi, A., Pradana, J. A., and Gurusanga, A. A. (2018). Detection and classification of vehicles for traffic video analytics. *Procedia computer science*, 144:259–268.
- [Askarian et al., 2021] Askarian, N., Abbasnejad, E., Zukerman, I., Buntine, W., and Haffari, G. (2021). Curriculum learning effectively improves low data VQA. In Rahimi, A., Lane, W., and Zuccon, G., editors, *Australasian Language Technology Association Workshop (ALTA) 2021*, pages 22–33. ACL.
- [Bahdanau et al., 2015] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- [Bengio et al., 2015] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*.
- [Bojanowski et al., 2016] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *Transactions of ACL*, 5.
- [Cadene et al., 2019] Cadene, R., Dancette, C., Cord, M., Parikh, D., et al. (2019). Rubi: Reducing unimodal biases for visual question answering. *Advances in neural information processing systems*, 32.
- [Chen et al., 2021] Chen, W., Gan, Z., Li, L., Cheng, Y., Wang, W. Y., and Liu, J. (2021). Meta module network for compositional visual reasoning. In *WACV*, pages 655–664.

BIBLIOGRAPHY

- [Chen et al., 2020] Chen, Y.-C., Li, L., Yu, L., El Kholy, A., Ahmed, F., Gan, Z., Cheng, Y., and Liu, J. (2020). Uniter: Universal image-text representation learning. In *European conference on computer vision*, pages 104–120. Springer.
- [Cho et al., 2014] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- [Clune et al., 2013] Clune, J., Mouret, J.-B., and Lipson, H. (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society b: Biological sciences*, 280(1755):20122863.
- [De Marneffe and Manning, 2008] De Marneffe, M.-C. and Manning, C. D. (2008). The stanford typed dependencies representation. In *Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation*, pages 1–8.
- [Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- [Do et al., 2019] Do, T., Do, T.-T., Tran, H., Tjiputra, E., and Tran, Q. D. (2019). Compact trilinear interaction for visual question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 392–401.
- [Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Housby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*.
- [Elman, 1993] Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71–99.
- [Fodor, 1983] Fodor, J. A. (1983). *The modularity of mind*. MIT press.

BIBLIOGRAPHY

- [Girshick, 2015] Girshick, R. B. (2015). Fast R-CNN. *CoRR*, abs/1504.08083.
- [Goodfellow et al., 2016] Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA. <http://www.deeplearningbook.org>.
- [Goyal et al., 2017] Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D. (2017). Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *CVPR*.
- [Greco et al., 2019] Greco, C., Plank, B., Fernández, R., and Bernardi, R. (2019). Psycholinguistics meets continual learning: Measuring catastrophic forgetting in visual question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3601–3605, Florence, Italy. Association for Computational Linguistics.
- [Gupta and Kembhavi, 2023] Gupta, T. and Kembhavi, A. (2023). Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.
- [He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision (ICCV 2015)*, 1502.
- [He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- [Hu et al., 2017] Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. (2017). Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 804–813.
- [Hudson and Manning, 2019a] Hudson, D. and Manning, C. D. (2019a). Learning by abstraction: The neural state machine. *Advances in Neural Information Processing Systems*, 32.
- [Hudson and Manning, 2018] Hudson, D. A. and Manning, C. D. (2018). Compositional attention networks for machine reasoning.

BIBLIOGRAPHY

- [Hudson and Manning, 2019b] Hudson, D. A. and Manning, C. D. (2019b). Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709.
- [Hudson and Manning, 2019c] Hudson, D. A. and Manning, C. D. (2019c). GQA: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*.
- [Johnson et al., 2017a] Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. (2017a). CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, pages 1988–1997.
- [Johnson et al., 2017b] Johnson, J., Hariharan, B., Van Der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., and Girshick, R. (2017b). Inferring and executing programs for visual reasoning. In *ICCV*, pages 3008–3017.
- [Kafle and Kanan, 2017a] Kafle, K. and Kanan, C. (2017a). An analysis of visual question answering algorithms. In *ICCV*.
- [Kafle and Kanan, 2017b] Kafle, K. and Kanan, C. (2017b). Visual question answering: Datasets, algorithms, and future challenges. *Computer Vision and Image Understanding*, 163:3–20.
- [Kervadec et al., 2021] Kervadec, C., Antipov, G., Baccouche, M., and Wolf, C. (2021). Roses are red, violets are blue... but should VQA expect them to? In *CVPR*, pages 2776–2785.
- [Kim et al., 2020] Kim, E.-S., Kang, W. Y., On, K.-W., Heo, Y.-J., and Zhang, B.-T. (2020). Hypergraph attention networks for multimodal learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14581–14590.
- [Kim et al., 2018] Kim, J.-H., Jun, J., and Zhang, B.-T. (2018). Bilinear attention networks. *Advances in neural information processing systems*, 31.
- [Kim et al., 2016] Kim, J.-H., Lee, S.-W., Kwak, D., Heo, M.-O., Kim, J., Ha, J.-W., and Zhang, B.-T. (2016). Multimodal residual learning for visual qa. *Advances in neural information processing systems*, 29.

BIBLIOGRAPHY

- [Kim et al., 2021] Kim, W., Son, B., and Kim, I. (2021). Vilt: Vision-and-language transformer without convolution or region supervision. In *International Conference on Machine Learning*, pages 5583–5594. PMLR.
- [Kirillov et al., 2023] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., and Girshick, R. (2023). Segment anything. *arXiv:2304.02643*.
- [Klein and Manning, 2003] Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*, pages 423–430.
- [Krishna et al., 2017] Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., Bernstein, M. S., and Fei-Fei, L. (2017). Visual genome: Connecting language and vision using crowdsourced dense image annotations. *Int. J. Comput. Vision*, 123(1):32–73.
- [Kumar et al., 2010] Kumar, M., Packer, B., and Koller, D. (2010). Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23.
- [Lamb et al., 2016] Lamb, A. M., ALIAS PARTH GOYAL, A. G., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.
- [Lan et al., 2020] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- [Lechat, 2021] Lechat, A. (2021). *Apprentissage incrémental semi-supervisé pour les applications de vision artificielle*. Theses, Normandie Université.
- [Li et al., 2019a] Li, G., Wang, X., and Zhu, W. (2019a). Perceptual visual reasoning with knowledge propagation. In *ACM MM, MM '19*, page 530–538, New York, NY, USA. ACM.

BIBLIOGRAPHY

- [Li et al., 2021] Li, J., Selvaraju, R., Gotmare, A., Joty, S., Xiong, C., and Hoi, S. C. H. (2021). Align before fuse: Vision and language representation learning with momentum distillation. *Advances in neural information processing systems*, 34:9694–9705.
- [Li et al., 2019b] Li, L. H., Yatskar, M., Yin, D., Hsieh, C.-J., and Chang, K.-W. (2019b). Visualbert: A simple and performant baseline for vision and language. In *Arxiv*.
- [Li et al., 2020] Li, X., Yin, X., Li, C., Zhang, P., Hu, X., Zhang, L., Wang, L., Hu, H., Dong, L., Wei, F., et al. (2020). Oscar: Object-semantics aligned pre-training for vision-language tasks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXX 16*, pages 121–137. Springer.
- [Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13*, pages 740–755. Springer.
- [Liu et al., 2018] Liu, C., He, S., Liu, K., and Zhao, J. (2018). Curriculum learning for natural answer generation. In *IJCAI*, page 4223–4229. AAAI Press.
- [Lu et al., 2019a] Lu, J., Batra, D., Parikh, D., and Lee, S. (2019a). Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d’Alché Buc, F., Fox, E. B., and Garnett, R., editors, *NeurIPS*, pages 13–23.
- [Lu et al., 2019b] Lu, J., Batra, D., Parikh, D., and Lee, S. (2019b). ViLBERT: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., and Garnett, R., editors, *NeurIPS*, volume 32. Curran Associates, Inc.
- [Luo et al., 2021] Luo, Z., Xie, W., Kapoor, S., Liang, Y., Cooper, M., Niebles, J. C., Adeli, E., and Li, F.-F. (2021). Moma: Multi-object multi-actor activity parsing. *Advances in neural information processing systems*, 34:17939–17955.
- [Malinowski and Fritz, 2014] Malinowski, M. and Fritz, M. (2014). A multi-world approach to question answering about real-world scenes based on uncertain input. *Advances in neural information processing systems*, 27.

BIBLIOGRAPHY

- [Mascharka et al., 2018] Mascharka, D., Tran, P., Soklaski, R., and Majumdar, A. (2018). Transparency by design: Closing the gap between performance and interpretability in visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4942–4950.
- [McCann et al., 2017] McCann, B., Bradbury, J., Xiong, C., and Socher, R. (2017). Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30.
- [Mihaylova and Martins, 2019] Mihaylova, T. and Martins, A. F. T. (2019). Scheduled sampling for transformers. In *Proceedings of ACL: Student Research Workshop*, pages 351–356, Florence, Italy. Association for Computational Linguistics.
- [Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013.
- [Nguyen et al., 2022] Nguyen, B. X., Do, T., Tran, H., Tjiputra, E., Tran, Q. D., and Nguyen, A. (2022). Coarse-to-fine reasoning for visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4558–4566.
- [OpenAI, 2023] OpenAI (2023). Gpt-4 technical report.
- [Palanisamy, 2020] Palanisamy, P. (2020). Multi-agent connected autonomous driving using deep reinforcement learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.
- [Partee et al., 1984] Partee, B. et al. (1984). Compositionality. *Varieties of formal semantics*, 3:281–311.
- [Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.
- [Perez et al., 2018] Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

BIBLIOGRAPHY

- [Pfungst and Hans, 1965] Pfungst, O. and Hans, C. (1965). A contribution to experimental animal and human psychology. *Clever Hans (the Horse of Mr. Von Osten)*.
- [Radford et al., 2021] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.
- [Redmon and Farhadi, 2016] Redmon, J. and Farhadi, A. (2016). Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*.
- [Ren et al., 2015a] Ren, M., Kiros, R., and Zemel, R. (2015a). Exploring models and data for image question answering. *Advances in neural information processing systems*, 28.
- [Ren et al., 2021] Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Gupta, B. B., Chen, X., and Wang, X. (2021). A survey of deep active learning. *ACM computing surveys (CSUR)*, 54(9):1–40.
- [Ren et al., 2015b] Ren, S., He, K., Girshick, R., and Sun, J. (2015b). Faster R-CNN: Towards real-time object detection with region proposal networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *NeurIPS*, volume 28. Curran Associates, Inc.
- [ROBINS, 1995] ROBINS, A. (1995). Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146.
- [Sachan and Xing, 2016] Sachan, M. and Xing, E. (2016). Easy questions first? A case study on curriculum learning for question answering. In *Proc. 54th Annual Meeting of the ACL*, pages 453–463, Berlin, Germany. ACL.
- [Sarzynska-Wawer et al., 2021] Sarzynska-Wawer, J., Wawer, A., Pawlak, A., Szymanowska, J., Stefaniak, I., Jarkiewicz, M., and Okruszek, L. (2021). Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135.
- [Scao et al., 2022] Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. (2022). Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

BIBLIOGRAPHY

- [Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. pages 1–14. Computational and Biological Learning Society.
- [Singh et al., 2018] Singh, T., Khan, S. S., and Chadokar, S. (2018). A review on automatic parking space occupancy detection. In *2018 International Conference on Advanced Computation and Telecommunication (ICACAT)*, pages 1–5.
- [Soviany et al., 2022] Soviany, P., Ionescu, R. T., Rota, P., and Sebe, N. (2022). Curriculum learning: A survey. *Int. J. Comput. Vis.*, 130(6):1526–1565.
- [Subramanian et al., 2023] Subramanian, S., Narasimhan, M., Khangaonkar, K., Yang, K., Nagrani, A., Schmid, C., Zeng, A., Darrell, T., and Klein, D. (2023). Modular visual question answering via code generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 747–761, Toronto, Canada. Association for Computational Linguistics.
- [Surís et al., 2023] Surís, D., Menon, S., and Vondrick, C. (2023). Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*.
- [Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.
- [Tan and Bansal, 2019] Tan, H. and Bansal, M. (2019). LXMERT: Learning cross-modality encoder representations from transformers. In *Proceedings of EMNLP-IJCNLP*, pages 5100–5111, Hong Kong, China. Association for Computational Linguistics.
- [Tomar et al., 2022] Tomar, A., Kumar, S., and Pant, B. (2022). Crowd analysis in video surveillance: A review. In *2022 International Conference on Decision Aid Sciences and Applications (DASA)*, pages 162–168.
- [Tsai et al., 2018] Tsai, C.-C., Tseng, C.-K., Tang, H.-C., and Guo, J.-I. (2018). Vehicle detection and classification based on deep neural network for intelligent transportation applications. In *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1605–1608.

BIBLIOGRAPHY

- [Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *NeurIPS*, volume 30. Curran Associates, Inc.
- [Wang et al., 2021] Wang, C., Nulty, P., and Lillis, D. (2021). A comparative study on word embeddings in deep learning for text classification. In *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, NLPPIR '20, page 37–46, New York, NY, USA. Association for Computing Machinery.
- [Wang et al., 2022a] Wang, J., Ye, T., Gu, Z., and Chen, J. (2022a). Ltp: Lane-based trajectory prediction for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17134–17142.
- [Wang et al., 2022b] Wang, X., Chen, Y., and Zhu, W. (2022b). A survey on curriculum learning. *TPAMI*, 44(9):4555–4576.
- [Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280.
- [Wu et al., 2019] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R. (2019). Detectron2. <https://github.com/facebookresearch/detectron2>.
- [Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR.
- [Yang et al., 2022] Yang, J., Duan, J., Tran, S., Xu, Y., Chanda, S., Chen, L., Zeng, B., Chilimbi, T., and Huang, J. (2022). Vision-language pre-training with triple contrastive learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15671–15680.
- [Yang et al., 2016] Yang, Z., He, X., Gao, J., Deng, L., and Smola, A. (2016). Stacked attention networks for image question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 21–29.

BIBLIOGRAPHY

- [Yu et al., 2019] Yu, Z., Yu, J., Cui, Y., Tao, D., and Tian, Q. (2019). Deep modular co-attention networks for visual question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6281–6290.
- [Zabłocki et al., 2014] Zabłocki, M., Gościewska, K., Frejlichowski, D., and Hofman, R. (2014). Intelligent video surveillance systems for public spaces – a survey. *Journal of Theoretical and Applied Computer Science*, 8:13–27.
- [Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part I 13*, pages 818–833. Springer.
- [Zhang et al., 2021] Zhang, P., Li, X., Hu, X., Yang, J., Zhang, L., Wang, L., Choi, Y., and Gao, J. (2021). Vinvl: Revisiting visual representations in vision-language models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5579–5588.
- [Zhou et al., 2020] Zhou, L., Palangi, H., Zhang, L., Hu, H., Corso, J., and Gao, J. (2020). Unified vision-language pre-training for image captioning and vqa. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13041–13049.

Synthèse de la thèse en français

Introduction

L'intelligence artificielle (IA) vise à rendre les machines plus intelligentes, en leur permettant d'apprendre, de raisonner et d'effectuer des actions. L'objectif ultime est de permettre aux systèmes d'IA d'interagir de manière semblable à la perception et à la cognition humaine. Cependant, cette quête a rencontré des défis, en particulier dans les tâches intuitives pour les humains, mais difficiles à décrire formellement, comme la reconnaissance de la parole ou la reconnaissance faciale. Cette frontière de l'IA a été fortement influencée par l'avènement des réseaux neuronaux connexionnistes, qui excellent dans la reconnaissance de formes et l'apprentissage [Goodfellow et al., 2016].

Le raisonnement visuel dans le domaine de l'IA implique que les machines interprètent des données visuelles de manière similaire aux humains. Cela inclut des tâches telles que la compréhension de scènes complexes, la déduction de relations entre objets et la réponse à des questions basées sur des images. La Réponse aux Questions Visuelles (QRV) est une application pertinente du raisonnement visuel, combinant la vision par ordinateur et le traitement du langage naturel pour permettre aux systèmes de comprendre des scènes visuelles complexes et de répondre à des questions à leur sujet. Cependant, comprendre le raisonnement complexe de ces systèmes peut être difficile, comme illustré par l'exemple de Clever Hans [Pfungst and Hans, 1965], un cheval qui semblait répondre à des questions arithmétiques en réagissant en réalité à des indices subtils de la part des observateurs humains.

Dans cette thèse, notre objectif est de permettre aux modèles de réseaux de neurones de réaliser un raisonnement transparent sur le monde visuel en décomposant les tâches de raisonnement complexes en une séquences d'étapes constituant un programme, semblables aux processus modulaires que les humains utilisent [Clune et al., 2013, Fodor, 1983]. En QRV compositionnelle, les composants se réfèrent à des éléments tels que des objets, des attributs et des relations présents dans une question

associée à une image. En suivant les principes de la sémantique compositionnelle [Partee et al., 1984], nous cherchons à comprendre comment ces composants sont combinés pour raisonner sur le monde visuel et répondre de manière explicite à des questions liées aux images. Diverses techniques modulaires ont été proposées, l’une des plus importantes et qui est au cœur de notre travail repose sur les réseaux de modules neuronaux (NMN) [Andreas et al., 2016b, Johnson et al., 2017b, Li et al., 2019a]. Dans les NMNs, le processus de raisonnement est divisé en modules plus petits spécialisés, et ces modules sont entraînés conjointement. Ils travaillent en collaboration de manière séquentielle, où les sorties des modules sont utilisées comme entrées pour les modules suivants, afin de résoudre des tâches complexes. Chaque module est conçu pour résoudre une sous-tâche spécifique, et les modules peuvent être combinés pour aborder diverses questions et problèmes. Par exemple, pour répondre à la question “De quelle couleur est le fruit du côté droit, rouge ou vert ?” concernant l’image présentée dans la Figure 6.1, nous avons besoin de trois modules différents: un pour se focaliser sur le côté droit de l’image, un autre pour reconnaître le fruit, et un module supplémentaire pour décider entre le rouge et le vert en tant que couleur du fruit.

Dans nos travaux, nous utilisons la base de données GQA [Hudson and Manning, 2019b], largement adoptée pour l’entraînement et l’évaluation de modèles de raisonnement visuel compositionnel. GQA présente des questions complexes concernant des images réelles. En plus des triplets question-image-réponse, cette base de données contient également les programmes représentant la séquence de fonctions à exécuter pour répondre à la question concernant l’image, ainsi que les graphes des images provenant de Visual Genome [Krishna et al., 2017]. Cette base de données comprend plusieurs ensembles de données pour l’apprentissage, la validation et le test. Elle est disponible en deux versions : l’une non équilibrée comprenant 18 millions d’exemples et l’autre équilibrée avec plus d’un million d’exemples.

Représentations multimodales et réseaux de modules neuronaux

Les modèles de raisonnement visuel sont confrontés au défi de raisonner efficacement sur des scènes complexes de manière transparente. Bien que les modèles multimodaux récents [Tan and Bansal, 2019, Lu et al., 2019b, ?] utilisent des mécanismes basés sur l’attention, et certains intègrent même un raisonnement basé sur l’attention en plusieurs étapes [Hudson and Manning, 2018, Perez et al., 2018, Yang et al., 2016], ils manquent souvent d’aspect modulaire qui permet un raisonnement plus explicite.



Figure 6.1: Image extraite de la base de données GQA [Hudson and Manning, 2019b].

Pour remédier à cette limitation, nous nous concentrons sur l'introduction de représentations multimodales dans les réseaux de modules neuronaux. Les représentations multimodales intègrent des représentations linguistiques et visuelles pour capturer leurs relations à l'aide d'encodeurs pré-entraînés multimodaux [Tan and Bansal, 2019]. Les NMN permettent un raisonnement modulaire sur les représentations multimodales, décomposant ainsi des tâches complexes en modules interprétables et réutilisables. De plus, l'utilisation des représentations multimodales dans le cadre des NMNs permet d'initialiser le processus de raisonnement en fournissant des indices a priori sur les informations encapsulées dans les concepts et les objets concernés par les problèmes de raisonnement.

Notre modèle prend en entrée une image, une question et un programme et prédit une réponse. Nous extrayons les encodages textuelles et visuelles alignées pour l'image et la question en utilisant un Transformer multimodal pré-entraînés tel que LXMERT [Tan and Bansal, 2019]. Le générateur traduit la question en une séquence de modules appelée programme, ensuite l'exécuteur utilise le programme pour construire un NMN, qui est exécuté sur l'image pour répondre à la question.

LXMERT. Pour extraire des représentations multimodales nous employons LXMERT qui est un modèle de Transformer pré-entraîné sur diverses tâches uni-modales et multimodales, y compris les prédictions croisées masquées, les prédictions masquées d'objets, la mise en correspondance entre les modalités et la QRV. Ce modèle a démontré de bonnes performances sur diverses tâches, et le modèle affiné sur GQA est publiquement disponible. Pour les représentations visuelles, LXMERT exploite les régions d'intérêt de Faster-RCNN [Ren et al., 2015b], capturant efficacement les objets présents dans les

images. De plus, les plongements lexicaux sont appris de manière similaire à BERT [Devlin et al., 2019], fournissant une représentation contextuelle des données textuelles. Comme montré dans la Figure 6.2, le modèle est composé de trois blocs d’encodeurs de type Transformer [Devlin et al., 2019], le premier apprend les représentations visuelles, le deuxième apprend les représentation textuelles et le dernier aligne les deux modalités. Les deux encodeurs uni-modaux appliquent le mécanisme de l’auto-attention et l’encodeur multimodal applique le mécanisme de l’attention croisée. Il convient de mentionner que nous figeons les paramètres du modèle, nous utilisons uniquement les représentations en sortie de l’encodeur multimodal et nous abandonnons le composant de classification des réponses.

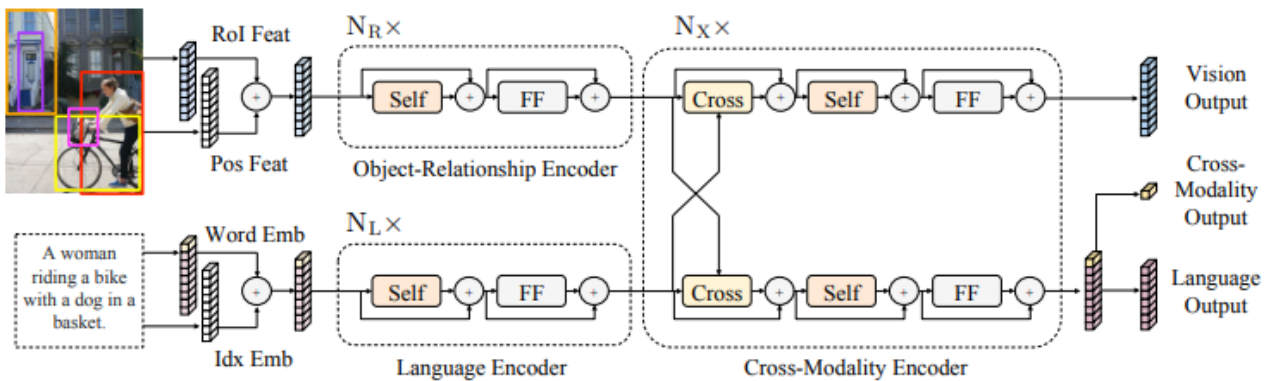


Figure 6.2: Architecture du modèle LXMERT [Tan and Bansal, 2019].

NMN. Les NMNs nécessitent en premier lieu l’élaboration et la définition d’un catalogue de modules représentant les différents modules spécifiques aux différentes sous-tâches de raisonnement. Pour ce faire, nous pré-traitem les fonctions des programmes proposées par la base de données GQA et nous nous inspirons des travaux antérieurs pour la définition des modules. Le pré-traitement nous a permis de réduire le nombre de fonctions à 29 au lieu des 136 disponibles dans [Hudson and Manning, 2019b]. Les modules se regroupent en trois catégories en fonction de la nature de leurs sorties. Les modules d’attention sont responsables des sous-tâches perceptuelles telles que la localisation d’objets (`select`), d’attributs (`filterAttr`) ou de relations (`relateSub`). Ils produisent un vecteur d’attention en sortie, où chaque valeur indique la pertinence de chaque objet de l’image par rapport à la sous-tâche concernée. Ensuite, les modules booléens effectuent des opérations logiques telles que les conjonctions (`and`) ou les disjonctions (`or`), ainsi que des vérifications (`verifyAttr`), et ils produisent en sortie un scalaire indiquant la probabilité que la sortie soit ‘vraie’. Enfin, les modules de réponse classifient la réponse finale à la question (`queryName`). Dans les définitions et implémentations des modules, nous veillons à

ce que les modules aient une structure peu profonde qui se base sur des multiplications matricielles et des couches de neurones non linéaires. Le Tableau 6.1 montre la définition de certains modules de notre NMN. Les définitions de tous les modules sont fournies dans les tableaux 3.2, 3.3 et 3.4 du manuscrit.

Noms	Dépendances	Soties	Definition
Select	–	attention	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{Y} = r(\mathbf{W} \mathbf{V}), \mathbf{o} = \sigma(\mathbf{W}(\mathbf{Y} \odot \mathbf{x}))$
RelateSub	[\mathbf{a}]	attention	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{Y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})), \mathbf{Z} = \sigma(\mathbf{W}(\mathbf{Y} \odot \mathbf{x}))$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{Z}))$
VerifyAttr	[\mathbf{a}]	booléen	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})), \mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
And	[$\mathbf{b}_1, \mathbf{b}_2$]	booléen	$\mathbf{o} = \mathbf{b}_1 \times \mathbf{b}_2$
ChooseAttr	[\mathbf{a}]	réponse	$\mathbf{x} = r(\mathbf{W} \mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V} \mathbf{a})), \mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
QueryName	[\mathbf{a}]	réponse	$\mathbf{x} = r(\mathbf{W}(\mathbf{V} \mathbf{a})), \mathbf{o} = \sigma(\mathbf{W} \mathbf{x})$

Table 6.1: Définitions de certains modules. σ : fonction d’activation non-linéaire, r : RELU, \mathbf{W} : matrice de poids, \mathbf{a} : vecteur d’attention (36×1), \mathbf{b} : scalaire booléen (1×1), \mathbf{V} : représentations visuelles (768×36), \mathbf{t} : représentations textuelles (768×1), \odot : produit matriciel de Hadamard.

Bien que les modules individuels aient une structure peu profonde, leur composition résulte dans une structure globale profonde, surtout lorsque de nombreux modules sont impliqués dans un programme. Par conséquent, pour réduire le nombre total de paramètres de notre NMN, nous partageons les poids entre les couches non linéaires de ces modules. Cette approche se base sur la compatibilité structurelle et fonctionnelle des modules. Elle a l’avantage de réduire le nombre de paramètres à optimiser, tout en permettant une meilleure optimisation des poids partagés.

Le cadre des NMNs se base sur un générateur-exécuteur. Le générateur est un modèle séquence-à-séquence qui traduit la question en un programme. Les travaux précédents [Andreas et al., 2016b, Hu et al., 2017, Li et al., 2019a, Chen et al., 2021] employaient des parseurs linguistiques [Klein and Manning, 2003], des réseaux de neurones récurrents ou des Transformers. Nous utilisons un décodeur de type Transformer pour apprendre une projection de la question vers l’espace des fonctions modulaires. L’exécuteur se base sur le programme pour instancier un NMN et l’exécute sur l’image afin de prédire une réponse à la question posée.

Expériences et résultats. Pour évaluer comment les représentations multimodales affectent les performances des NMNs, nous les comparons aux représentations uni-modales. Pour les représentations uni-modales des questions, nous utilisons les plongements de mots de FastText [Bojanowski et al., 2016] et les plongements contextuels de mots de BERT [Devlin et al., 2019]. Pour les représentation uni-modales

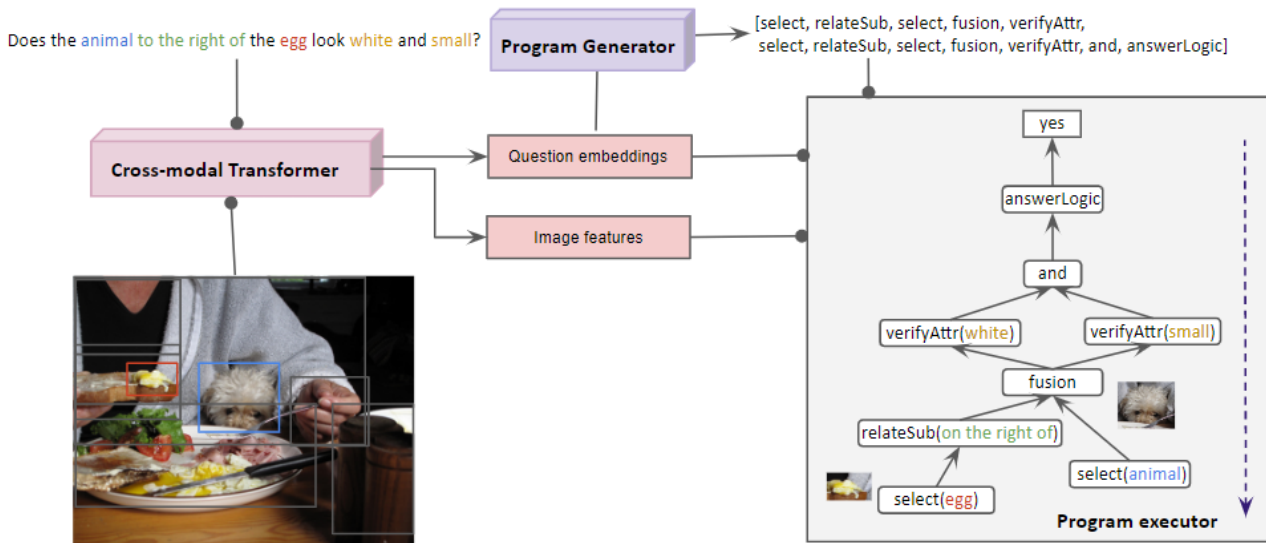


Figure 6.3: Le cadre QRV modulaire proposé : La paire (question, image) est utilisée par un modèle de Transformer pour générer des représentations alignés pour les mots et les objets. Celles-ci sont utilisées par un générateur pour produire un programme (une séquence de modules de sous-tâches), qui sera ensuite appliqué par un exécuteur de programmes sur l'image pour répondre à la question.

des images, nous employons les boîtes englobantes fournies par un Faster-RCNN [Ren et al., 2015b]. Nous comparons ces représentations dans deux stratégies d'apprentissage différentes des NMNs et nous constatons que les représentations multimodales apportent un gain de précision de 12%.

Apprentissage guidé pour les NMNs

Optimiser les NMNs en se basant uniquement sur la précision des réponses prédites ne garantit pas que les modules intermédiaires de la chaîne de raisonnement respectent leurs sous-tâches spécifiques. Pour remédier à cela, nous utilisons une stratégie d'apprentissage guidée afin de mieux superviser l'apprentissage des modules intermédiaires. 1) Premièrement, nous nous inspirons du mécanisme de l'enseignement forcé (teacher forcing) [Williams and Zipser, 1989, Lamb et al., 2016] et l'échantillonnage planifié [Bengio et al., 2015], des méthodes appliquées pour les réseaux de neurones récurrents. Le principe consiste à utiliser les vérités terrain en entrée des modules intermédiaires au lieu d'utiliser les sorties prédites par les modules précédents. Cette approche évite la propagation des erreurs effectuées par les modules situés en amont de chaîne de raisonnement vers les modules en aval. Pour mettre en œuvre cette idée, nous adaptons un échantillonnage planifié dégressif permettant une transition progressive vers un entraînement collaboratif des modules. 2) Deuxièmement, nous

introduisons une fonction d'erreur multi-tâches qui facilite l'optimisation des paramètres des modules en fonction de leurs erreurs relatives. Cette fonction d'erreur multi-tâches et une moyenne pondérée des erreurs des différents types de modules (attention, booléen et réponse).

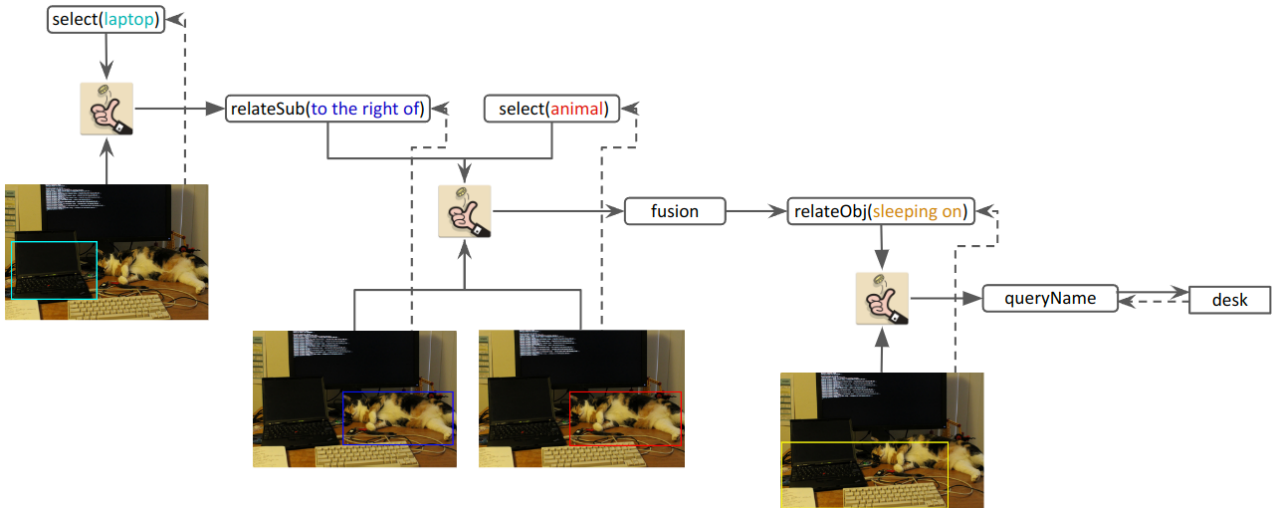


Figure 6.4: L'apprentissage guidé pour l'exécution du programme relatif à la question ' Sur quoi l'animal à droite de l'ordinateur portable est en train de dormir? '. Les flèches continues représentent l'échantillonnage planifié (voir Section 4.2), les flèches discontinues représentent l'erreur multi-tâches (voir Section 4.3), et les boîtes englobantes représentent les cibles intermédiaires.

Pour construire les cibles des modules intermédiaires, nous faisons un traitement des données de GQA [Hudson and Manning, 2019b] pour implémenter les correspondances entre les vérités terrain disponibles dans les graphes des images et les boîtes englobantes générées par notre extracteur de régions d'intérêt Faster-RCNN [Ren et al., 2015b]. Nous utilisons deux techniques de correspondances. La première, appelée "attention stricte", cible la boîte englobante ayant la plus grande intersection sur union avec l'objet de vérité terrain. La deuxième technique, nommée "attention douce", inclut comme cibles toutes les boîtes englobantes ayant une valeur d'intersection sur union avec l'objet de vérité terrain supérieure à un seuil défini. Pour l'attention stricte, nous utilisons l'entropie croisée multiclasse, tandis que pour l'attention douce, nous employons l'entropie croisée binaire, qui est semblable à un problème d'optimisation multi-labels. Les travaux menés dans cette section ont abouti à la publication d'un article scientifique [Aissa et al., 2023b].

Expériences et résultats. Nous mettons en place plusieurs expériences afin de comparer les effets des deux stratégies d'apprentissage guidée mentionnées précédemment. L'ensemble de données équilibré de

GQA est utilisé pour l’entraînement de nos modèles proposés. Nous constatons que la fonction d’erreur multi-tâches donne de meilleures performances que l’échantillonnage planifié. Cette observation s’explique par le fait que l’utilisation exclusive de l’échantillonnage planifié limite la rétro-propagation du gradient de l’erreur aux modules en amont de la chaîne de raisonnement, en particulier lors des premières étapes de l’apprentissage lorsque la probabilité d’échantillonnage planifié est élevée. En combinant la fonction d’erreur multi-tâches avec l’échantillonnage planifié, nous observons un effet complémentaire qui améliore les performances des NMNs. Cette combinaison donne les meilleures performances, avec une précision atteignant les 63%. Nous faisons aussi une analyse qualitative des résultats obtenues en affichant les sorties des différents modules de la chaîne de raisonnement pour des exemples correctement classifiés dans le Tableau 4.6 et des exemples mal classifiés dans le Tableau 4.7. En résumé, les modules intermédiaires parviennent dans la plupart des cas à bien identifier la boîte englobante concernée par le module perceptuel et à suivre correctement le schéma de raisonnement indiqué par le programme. Les erreurs les plus courantes sont dues à la difficulté intrinsèque des questions posées et à la diversité des objets et de leurs propriétés. Par exemple, le NMN peut se tromper sur la couleur d’un objet, même s’il l’a correctement identifié dans l’image. De plus, il peut également se tromper sur la catégorie de l’objet recherché dans l’image, en prédisant par exemple la classe “camion” au lieu de la classe “utilitaire”.

Apprentissage curriculum pour les NMNs

L’apprentissage curriculum [Elman, 1993, Soviany et al., 2022, Wang et al., 2022b] est une stratégie d’entraînement qui réorganise les exemples des données d’apprentissage dans un ordre de difficulté croissante au lieu de les organiser de manière aléatoire. Cette approche permet aux modèles de réseaux de neurones d’apprendre à résoudre des problèmes faciles avant de progressivement s’attaquer à des problèmes de plus en plus difficiles. En conséquence, cela peut réduire le nombre de données d’apprentissage nécessaires pour atteindre de bonnes performances. Cette stratégie a été rarement appliquées aux NMNs. À titre d’exemple, [Askarian et al., 2021] emploie cette stratégie pour des NMN mais dans le cadre de données synthétiques provenant de la base de données CLEVR [Johnson et al., 2017a]. Les résultats obtenus sont prometteurs. Cependant, la nature synthétique des images présente un niveau de complexité moindre, avec moins de classes et de modules, par rapport aux images réelles de GQA [Hudson and Manning, 2019b]. Par conséquent, nous développons une stratégie d’apprentissage

curriculum spécifiquement adaptée à notre NMN pour des données du monde réel.

Nous partons de l'hypothèse selon laquelle le nombre d'objets mentionnés dans la question est un indicateur de sa difficulté. Ainsi, plus le nombre d'objets concernés par la question est élevé, plus la question est considérée comme difficile. En conséquence, nous catégorisons les questions en fonction du nombre de concepts qu'elles contiennent, allant de 1 à 4 concepts, ce qui nous permet de créer quatre groupes d'exemples. De plus, pour affiner l'ordre des exemples au sein d'un même groupe, nous utilisons une autre mesure de difficulté basée sur la longueur de la question, c'est-à-dire le nombre total de mots dans la question.

L'apprentissage curriculum nécessite également une planification pour décider du moment où passer d'un niveau de difficulté à un autre. Dans notre cas, nous avons choisi une taille d'échantillon prédéterminée pour chaque niveau de difficulté, offrant ainsi une solution contrôlée simple. Le passage d'un niveau de difficulté à un autre s'effectue après que le modèle ait vu 1 million d'exemples. De plus, nous essayons aussi la répétition de l'entraînement à chaque niveau de difficulté afin de donner aux modules plus de temps pour optimiser leurs paramètres à ces niveaux de difficultés.

Ensuite, nous définissons une fonctions d'échantillonnage pour sélectionner les 1M d'exemples utilisés pour l'apprentissage à chaque niveau de difficulté. L'échantillonnage est effectué avec remplacement et la fonctions d'échantillonnage attribue des probabilités de sélection à chaque exemple dans le groupe de difficulté correspondant au niveau en cours. Deux fonctions sont établies: la première a pour but d'uniformiser la présence des modules de type réponse dans l'échantillon final, tandis que la deuxième attribue des probabilités de sélection proportionnelles à la moyenne des erreurs d'apprentissage des modules qui composent le programme de chaque exemple. Ainsi, plus l'erreur des modules du programme est élevée, plus l'exemple a de chances d'être sélectionné.

L'apprentissage des réseaux de neurones sur un ensemble de données ordonnées comporte le risque d'oubli catastrophique [ROBINS, 1995, Greco et al., 2019]. Afin d'atténuer ce risque, nous sélectionnons de manière aléatoire 20% des 1 million d'exemples de chaque niveau de difficulté parmi les itérations précédentes. Ainsi, le modèle continue à être optimisé en fonction de quelques exemples du passé ayant une difficulté inférieure à celle du niveau actuel.

Les contributions mentionnées font le centre de notre article scientifique [Aissa et al., 2023a].

Expériences et résultats. Pour évaluer l'impact de l'apprentissage curriculum sur notre NMN,

nous utilisons l'ensemble non-équilibré de GQA pour accéder à un plus grand nombre d'exemples, nous permettant d'avoir 1 million d'exemples pour chaque niveau de difficulté. Nous comparons les différentes stratégies d'apprentissage curriculum entre elles et les comparons aux modèles entraînés avec un ordre d'exemples complètement aléatoire. Nous observons que le choix de la bonne stratégie d'apprentissage curriculum permet d'accélérer le processus d'apprentissage et de réduire le nombre de données nécessaires pour atteindre une précision comparable à un modèle entraîné sur deux fois plus de données sans apprentissage curriculum. La meilleure stratégie consiste à utiliser la mesure de difficulté basée sur le nombre de concepts dans la question, avec une répétition de l'apprentissage de chaque échantillon deux fois, en plus de pré-entraîner le modèle pendant deux itérations sur un million d'exemples tirés aléatoirement à partir de toute la base d'apprentissage et de remédier à l'oubli catastrophique en intégrant une petite partie des exemples des itérations précédentes. Cette stratégie nous permet d'atteindre une précision des réponses de 68%.

Conclusion

Cette thèse se focalise sur le raisonnement visuel compositionnel. Pour obtenir des réponses à des questions visuelles complexes, nous les décomposons en plusieurs sous-tâches génériques, abordées en plusieurs étapes de manière explicite. Nous avons introduit une architecture de réseau de modules neuronaux (NMN) dynamique qui rassemble plusieurs modules, chacun se concentrant sur une sous-tâche de raisonnement particulière, telle que la détection d'objets, d'attributs ou de relations. Pour représenter les questions et les images, nous utilisons une approche compositionnelle, en encodant les mots de la question et les objets de l'image. De plus, nous tirons parti d'un encodeur multimodal pour aligner les représentations textuelles et visuelles, ce qui s'est avéré avantageux pour notre NMN.

Afin d'améliorer les performances de notre NMN, nous avons mis au point plusieurs stratégies d'apprentissage. L'une de ces stratégies est l'apprentissage guidé [Aissa et al., 2023b], qui renforce la supervision des modules intermédiaires de la chaîne de raisonnement pour optimiser leur apprentissage. Cela les rend plus performants dans l'accomplissement de leurs sous-tâches spécifiques et favorise leur collaboration pour produire des réponses correctes. Nous avons également adopté une stratégie d'apprentissage en curriculum [Aissa et al., 2023a] qui emploie le nombre de concepts de la question comme mesure de difficulté afin de permettre une utilisation plus efficace des données disponibles et réduire les coûts d'apprentissage.

