UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

EMPIRICAL STUDY OF THE VISUAL REASONING CAPABILITIES OF THE NEURAL STATE MACHINE

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

GABRIEL ALEJANDRO CHAPERON BURGOS

PROFESOR GUÍA:
JORGE PÉREZ ROJAS

PROFESOR CO-GUÍA:
FELIPE BRAVO MÁRQUEZ

MIEMBROS DE LA COMISIÓN:
JOSÉ MANUEL SAAVEDRA RONDO
BENJAMÍN BUSTOS CÁRDENAS
FRANCKLIN RIVAS ECHEVERRÍA

SANTIAGO DE CHILE
2023

Estudio empírico de las capacidades de razonamiento visual de la Neural State Machine

El área de aprendizaje profundo es un área dentro de las ciencias de la computación, la estadística y la matemática donde los practicantes diseñan redes neuronales profundas para lograr imitar habilidades que son inherentemente humanas. En esta área se usan *tareas* con el fin evaluar la capacidad de un modelo para llevar a cabo una habilidad humana, como reconocimiento de objetos, clasificación de texto o reconocimiento de voz.

A finales del 2019 una nueva arquitectura llamada Neural State Machine (NSM) fue propuesta para la tarea de respuesta de preguntas visuales, donde se espera que un modelo pueda responder preguntas que están basadas en una imagen. La arquitectura se inspira fuertemente en máquinas de estado tradicionales de teoría de autómatas, y funciona recorriendo un camino por los objetos de la imagen de forma iterativa hasta encontrar la respuesta a la pregunta.

En este trabajo estudiamos de forma empírica las limitaciones de esta nueva arquitectura. De teoría de autómatas sabemos que la falta de memoria en las máquinas de estado tradicionales limita el tipo de entradas que pueden procesar. Considerando esta observación y el diseño de la NSM basado en máquinas de estado, nosotros conjeturamos que la arquitectura va a ser incapaz de procesar algunos tipos de preguntas basadas en imágenes.

Para probar nuestra hipótesis usamos una metodología experimental. Primero definimos categorías de preguntas donde pensamos que la NSM tendrá problemas. Estas preguntas vienen de esfuerzos previos en la literatura de establecer puntos de referencia para sistemas multimodales de texto y visión. Luego evaluamos la arquitectura y comparamos los resultados con resultados base donde la NSM alcanza un desempeño prácticamente perfecto.

Nuestros hallazgos muestran que la NSM efectivamente tiene problemas para responder las preguntas propuestas. La disminución en el desempeño varía en cada caso, llegando en ocasiones a niveles aleatorios. Nuestros resultados sugieren que para tener una solución exhaustiva para la tarea de respuestas de preguntas basadas en imágenes es necesario ir más allá de una red neuronal que representa una máquina finita de estados.

Empirical study of the visual reasoning capabilities of the Neural State Machine

The field of deep learning is a subfield of computer science, statistics and mathematics where practitioners try to build deep neural networks that mimic, to some extent, abilities inherent to human beings. In this field, *tasks* are used to evaluate the ability of a model to perform specific human skills, like object recognition, text classification or speech recognition.

In late 2019, a new architecture called Neural State Machine (NSM) was proposed for the task of visual question answering, where a model has to answer a question based on an image. The network is heavily inspired by traditional state machines from automata theory, and works by iteratively following a path on the image trying to find the answer to the question.

In this work we empirically study the limitations of this new architecture. From automata theory we know that traditional state machine's lack of memory limits the kind of inputs they can process. Considering this observation and the networks inspiration on state machines we hypothesize the network will be unable to process certain kinds of image-based questions.

We prove our hypothesis using an experimental approach. First we define a number of question categories where we think the NSM will struggle. These questions come from previous efforts in the literature to establish benchmarks for multimodal visual-text systems. Next we evaluate our architecture and compare the results to a baseline in which the NSM performs almost perfectly.

Our findings show the NSM indeed struggles in the proposed proposed questions, with varying degrees of decrease in performance, reaching in some cases random performance. Our results suggests that, in order to have a comprehensive solution for the question answering problem, one would need to go beyond a neural network representation of a finite state-machine.

Ojalá esto sirva de algo.

Agradecimientos

Gracias a mi mamá[1], mi papá[1], mi meme, la bego y el jorge.

[1]Contribuciones iguales.

Table of Content

List of Tables

List of Figures

Chapter 1

Introduction

One of the many abilities of us human beings is that we are able to answer simple questions based on an image. This task is commonly known as Visual Question Answering (VQA) [3], and it is used to evaluate the visually-grounded reasoning capabilities of neural systems. Examples of such simple image-based questions can be seen in Figure 1.1.

Some of the available datasets for the task make use of what is know as a *scene graph*, a graph that captures the objects and relations between them, where objects are nodes and relations are represented using edges. Objects and relations can have properties describing them in more detail, like color, size, orientation, etc. Models may use these scene graphs to reason over a higher-level representation of the visual data. An example of the process of extracting a scene graph from an image can be sen in Figure 1.2.

In this work we evaluate different cognitive capabilities of an architecture presented in late 2019 that was specifically designed to process these scene graphs and answer questions about the images they come from. The network is called Neural State Machine (NSM) and its design is inspired by traditional finite state machines. The model is described in depth in Chapter 3, but for now it suffices to know that the network interprets the scene graph as

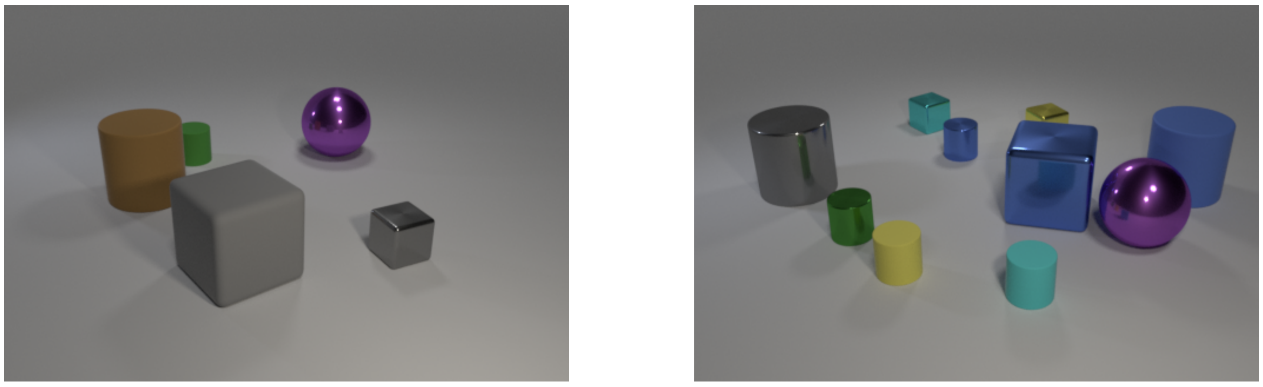

(a) What is the color of the sphere? (b) How many small cylinders are there?

Figure 1.1: Simple image-based questions that require abilities like counting and identifying colors. These abilities are learned very early in human development.

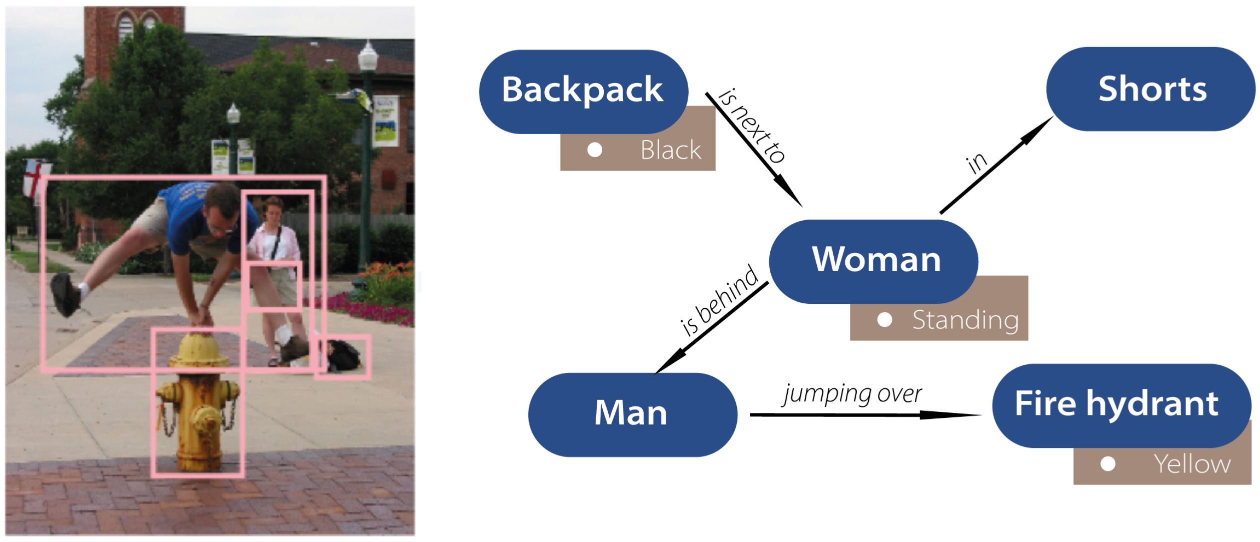


Figure 1.2: An example of an image and its corresponding scene graph. The graph captures many of the images actors, their properties and the relations between them. Image taken from Visual Genome [20].

a state machine and traverses it using the question as a guiding signal. After reaching the end of the path, the final node (or state) the machine is in is used to answer the question. A simplified schematic of the process can be seen in Figure 1.3.

This architecture hasn't been the subject of study of any work until this date, and so the aim of this thesis is to shed preliminary light into its capabilities and limitations.

> **Research Aim** The aim of this work is to study the limitations of a newly proposed architecture, the Neural State Machine.

The architecture was proposed as a general architecture for Visual Question Answering, as one of the datasets it was evaluated on was the original VQA dataset [3]. Also, as we will see in Chapter 3, the architecture makes use of two key architectures, an LSTM that has been proven to be Turing complete by Siegelmann et al. [31], and a Feed Forward Network, which has been proven to be an universal approximator by Hornik et al. [14]. This precedent points in the direction of the NSM having similar, general properties.

However, from automata theory we know that state machines are limited in the kind of inputs they can accept. Most notably, they lack any form of memory, so simple tasks like identifying a palindrome are impossible for a state machine, since it cannot remember the previous symbols it has seen. To understand the true nature of the architecture, we follow a experimental approach, laid out in the following research objectives.

Research Objectives

1. The first step of this work is to implement the architecture. The code for the architecture wasn't open sourced in the original paper, so this is a necessary first step.

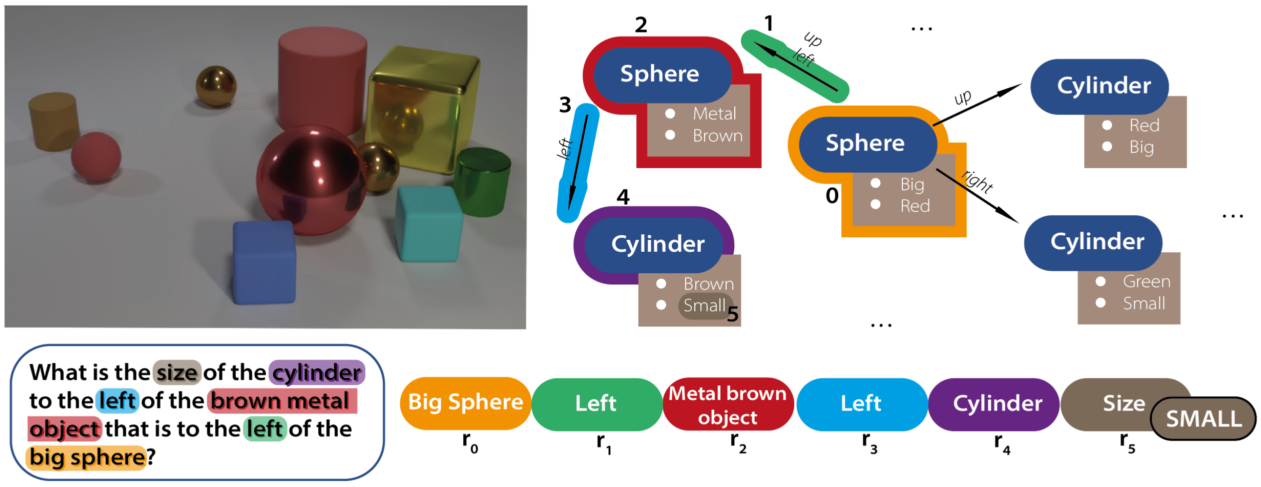


Figure 1.3: An overview of the NSM and how it processes both the graph and the question to find an answer. The architecture uses the question as a guiding signal to traverse the scene graph and reach a final node which contains the answer to the question.

2. To validate our implementation of the architecture, we will define a question category suitable for the NSM and evaluate it only in this subset.
3. Next we will design questions we hypothesize the NSM cannot answer. These questions should be motivated by the limitations from traditional automata.
4. Finally we will evaluate the NSM using the proposed question categories to identify its weaknesses.

Given the seemingly contradictory nature of the NSM, in this work we want to answer the following two main research questions.

Research Questions

1. Does the performance of the NSM decrease when asked questions that intuitively require memory to solve?
2. In what types of questions does the network's performance decrease the most?

Since the NSM is strongly inspired by finite state machines, we theorize there are many question categories that don't fit with its design and therefore it will struggle answering them. Formally our hypothesis is as follows.

Hypothesis There are natural questions over graph data that the NSM cannot answer. These include questions that require the ability to count, that require comparisons between objects and questions where logical conjunction must be performed.

All these categories intuitively require memory to be answered. A simple such question category to picture this idea are comparison questions. Something like *Is object A the same* *color as object B?* requires to focus on two objects at the same time, which would require *remembering* object A before focusing on object B. This simply doesn't fit in the framework of following a path to reach a single node.

Contributions The contributions of this work are the following.

1. And open source implementation of the Neural State Machine. To the best of our knowledge this is the only working implementation[1] of the architecture.
2. A set of experiments and results showing the pitfalls of the architecture.

The implementations and replication code for the experiments can be found at `https://github.com/gchaperon/neural-state-machine`.

The rest of this work is structured as follows. Chapter 2 presents an overview of the related work, to have a better understanding where in the field of Machine Learning this work is located. As mentioned, Chapter 3 presents the architecture in detail, defining it formally and drawing parallels with traditional machines. Chapter 4 explains our methodology, the data used and the question categories. Following, Chapter 5 presents the results obtained and in Chapter 6 we discuss the results, highlighting how they relate to our hypothesis. Finally, Chapter 7 summarizes our findings and gives directions for future research.

[1]Another attempt at implementing the network is `https://github.com/ceyzaguirre4/NSM`, but as the author states, it is not ready to be used.

Chapter 2

Related Work

In this chapter we review three main aspects of the related work. First we discuss the most relevant datasets for the VQA task. We next describe different models that have been proposed for the VQA task. Finally we review the different architectures proposed in the literature that simulate traditional machines. The first two sections attempt to give the reader a general overview of the VQA task, while the third presents two models that are closest to the spirit of the NSM, as a machine-inspired neural network.

2.1 VQA - Datasets

The VQA task together with its first training dataset was released by Antol et al. in 2015 [3] . This dataset contains 330K images from the MSCOCO dataset [23], and over 760K free-form questions that were produced using crowdsourcing. The main characteristic of this dataset is that both the questions and the answers are open-ended, in contrast with other datasets at the time that used closed vocabularies as possible answers [10, 26, 32, 5].

Another well known dataset for this task is CLEVR [18]. In this dataset images were generated automatically using Blender (a 3D modeling toolkit) [8]. The associated questions are generated using templates that are filled in with the information used to create the images, like relative position of objects, the colors of these objects or their shapes. The images contain three different types of objects (spheres, cylinders, and cubes), and can appear in two sizes, four colors, and two textures. The compiled dataset contains 100k images and 853k unique questions. Since the dataset is automatically generated, it allows for tight control over the contents of the images and questions, and so this dataset is specifically designed to evaluate different kinds of reasoning abilities, like spatial relationship understanding, counting, and comparisons.

Krishna et al. [20] present Visual Genome, a dataset that contains images from MSCOCO and questions about them. The main difference with previous datasets is that every image is tagged with a lot more information. Each image has bounding boxes for objects and their labels, together with their attributes (color, size, etc), and also there are labels for the relationships between these objects (behind, holding, jumping over, etc). All this information is used to create a scene graph from the image which can be used to support the learning process for a model, be it for pure image analysis or for the VQA task. The dataset contains 100k images with over 1.6M questions, all of which were crowdsourced from online workers.

The GQA dataset proposed by Hudson et al. [17] is based on Visual Genome and uses the graphs that capture the information of each image to generate questions in an automated way. This is accomplished by using a finite set of templates where the information from the graphs is filled in. This technique allows them to use real world images and context, while also having granular control over the questions and the cognitive skills necessary to answer them. This dataset contains 100k images and over 22M questions, although they also provide a smaller question split where the different categories have been balanced, which contains around 1.7M questions. Most of the literature report their results on the latter.

2.2 VQA - Methods

Models for solving the VQA task can be classified into two main categories. The first category corresponds to models that have strong inductive biases and that were specifically designed for the VQA task, taking into account the reasoning skills necessary to be successful. These models are generally highly interpretable and work very well on auto-generated datasets, but they tend to struggle generalizing to real world scenarios. An example of a model in this category is the *Neural Module Network* (NMN) proposed by Andreas et al. [2]. It consists of a finite set of specific modules specialized for different types of reasoning, which are composed depending on the structure of the question. Standard NLP tools are used to process the question (like POS tags and dependency parsing) and determine what specific combination of module must be instantiated. Many other architectures have been proposed that follow the same idea of the NMN [15, 19, 27], but with relatively minor changes to the types of modules, the query processing or the way the different modules communicate.

The second category corresponds to models that seek to learn dense representations of each image-question pair and use this vector to find the answer. An early example of this idea is the work from Shih et al. [30]. They obtain a vector representation from the question via parsing heuristics, and a series of object regions using an region proposal network (a standard procedure in the field of computer vision). Then, they compute the vector similarity between the question and each of the regions and use the highest-scoring pair to predict the answer. This gives the model the ability to focus on regions of the image that are relevant to the question. Lu et al. [25] expand on this idea, and introduce the concept of co-attention. In this setup the model learns to focus on the image using the question and vice versa, use the image to focus on the relevant words from the question.

After the advent of the Transformer architecture [33], an architecture that has seen major success in the NLP world, a couple of models based on this architecture have been proposed [24, 22, 21, 7]. They use *pretraining* techniques, where the model is first trained in large amounts of unlabeled data to develop a general understanding of the world. Then this model is *finetuned* in a specific task. The knowledge from the pretraining stage is used as a starting point and the model learns specific information for the task of interest.

2.3 Machine-inspired Architectures

The idea of simulating classical machines using neural architectures is not new. Two examples of this idea are the Neural Turing Machine [12] and the Differentiable Neural Computer [13]. These two examples have an external memory where they can store and retrieve values.

A Neural Turing Machine (NTM) architecture contains two basic components: a neural network controller and a memory bank. Like most neural networks, the controller interacts with the external world via input and output vectors. Unlike a standard network, it also interacts with a memory matrix. This memory is essentially a list of vectors that the controller can access using selective read and write operations (also called read and write heads). These operations focus on certain locations of the memory using two main addressing mechanisms: focus by content and focus by location. This means that when the read or write heads receive a vector from the controller, they can choose to operate over the most similar vector in memory or operate over a vector chosen based on the previous memory location.

The Differentiable Neural Computer (DNC) is in many ways the successor to the NTM. The main idea of how it works is the same, meaning that it has a controller and heads for writing and reading, but there are two main differences. The first is that the write head can also erase content from memory, not just write to it. The second difference relates to the addressing mechanisms, where the DNC has three of them. The first one is very similar to the NTM, where the memory is accessed based on the similarity to the key vector from the controller. The second records transitions between consecutively written locations, allowing the network to operate on a sequence of memory locations previously visited. The third form of attention allocates memory for writing, which allows unused locations to be delivered to the write head.

Chapter 3

The architecture in depth

The Neural State Machine is a deep learning architecture designed to jointly process graph and text data by simulating a finite state machine. The architecture was proposed for the task of VQA, where the graph comes from the image and the text corresponds to a question being asked about the image. For the description of the network we will use this task to guide our explanation, but it should be noted that the architecture is designed to process graphs.

The NSM proceeds in two stages, *modeling* and *inference*. In the modeling stage the question is converted into a sequence of *instructions* that have to be executed to answer it. In the inference stage the graph is treated as a state machine and the instructions are used to guide the reasoning reaching a final state, which is then used to answer the question.

3.1 Preliminaries

We begin this section by recalling the definition of a traditional state machine. A finite-state machine (also known as finite-state automaton) is an abstract machine that works by reading symbols from an input sequence and, according to a set of defined transition rules, changes its internal state.

Formally, a finite state machine is defined as a five-tuple $(\Sigma, S, s_0, \delta, F)$, where:

- Σ is the input alphabet. The symbols read from the input belong to this set.
- S a finite set of the different states the machine can be at any given moment.
- $s_0 \in S$ the initial state. The states the machine is in before reading any symbol from the input.
- $\delta : S \times \Sigma \to S$ a transition function. A function that takes the current state of the machine and an element from the input, and outputs the next state.
- $F \subseteq S$ a set of final states.

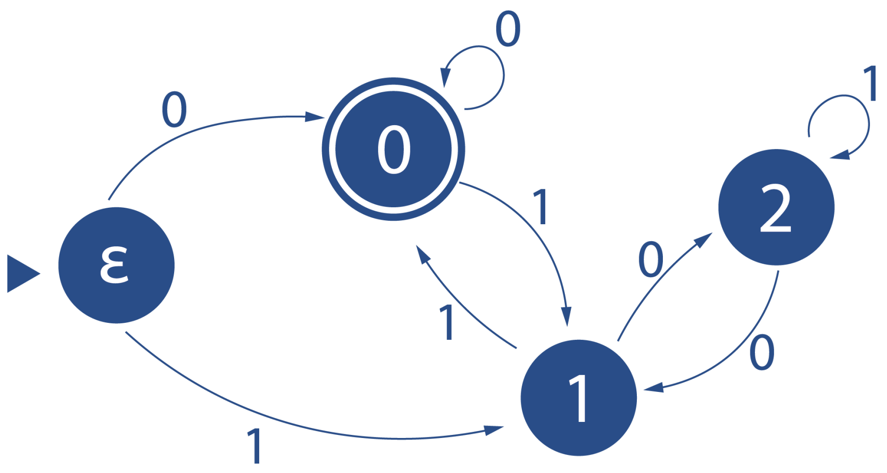


Figure 3.1: A simple example of a finite state machine with four states. The machine has states $S = \{\varepsilon, 0, 1, 2\}$, the input alphabet is $\Sigma = \{0, 1\}$, the initial state is $s_0 = \varepsilon$, there is just one final state $F = \{0\}$ and the transition function is depicted by the edges and their labels in the graph.

The machine receives an input, a sequence of characters from the alphabet Σ, reads one character at a time and changes state according to the transition function δ. After reading the last character of the input the machine checks if the state it is in belongs to F and, if it does, it is said the machine *accepts* the input.

A common way of representing state machine is as a graph, where states are represented as the nodes of the graph and the valid transitions between states are captured by the edges of the graph. Figure 3.1 shows and example state machine that accepts binary strings that are multiples of three.

3.2 Formal definition

Having the previous definition of a traditional state machine as context, understanding the NSM becomes much simpler. The main ideas to have in mind in the following (more technical) sections is that the machine state is no longer a single state but rather a probability distribution over the states, and the transition function is not a fixed function but a parameterized function, a neural network that will *learn* how to shift the current probability distribution to the next state.

Similarly to the classic state machine, the NSM is defined as a 6-tuple $(C, S, E, \{r_i\}_{i=0}^{N}, p_0, \delta)$ where

- C is the architecture's concept vocabulary, or *alphabet*, to continue with the traditional machine analogy. It consists of a set of *concepts* that include objects (dog, cat, etc), properties (green, tall, etc) and relationships (on top of, talking to, etc). These concepts are represented as vectors.

- S the set of states of the NSM.
- E the set of directed edges connecting the states. These edges represent valid transitions of the model. Contrary to a classical state machine, where the graph edges are derived from the transition function, in the NSM's case these edges are explicit in the model and the transition function δ redistributes probability along these edges.
- r_i a sequence of instruction that are processed by the transition function δ. These instructions guide the probability shifting process.
- $\delta_{S,E} : p_i \times r_i \rightarrow p_{i+1}$ the transition function; a neural network that at each step takes the current probability distribution p_i and a new instruction r_i and computes the new distribution p_{i+1}. This neural module *learns* to shift the distribution during training.

In the following sections each of these NSM components is described in detail.

3.3 Concept Vocabulary

The NSM operates using a discrete set of concepts, called the *concept vocabulary*, that is used to represent the information in both the image's scene graph and the guiding instructions. Concepts are grouped in $L + 1$ sets, where L sets correspond to object properties, i.e. $C_0 = \{red, blue, ...\}$, $C_1 = \{cube, sphere, ...\}$, etc. and the remaining set groups the possible relations $C_L = \{on\ top, in\ front, ...\}$. Therefore, $C = \bigcup_{i=0}^{L} C_i$. Furthermore, the NSM defines an extra set D of $L + 1$ embeddings of the categories themselves, that is, if $C_0 = \{red, blue, ...\}$ then $D_0 = color$. Following the previous example, D would look like $D = \{color, shape, ..., relation\}$.

3.4 States and transitions

Each graph consists of a set of objects found in the image that represent the states S of state machine, where each state $s \in S$ has L properties $\{s^j\}_{j=0}^{L-1}$, $s^j \in C_j$. Similarly, the graph has a set of edges $(s, s') \in E$ that relate two objects. Each of the edges as a single attribute describing the relation (on top, below, etc).

The data we use for this work already contains ground truth scene graphs for each image, meaning that we start directly with a scene graph and we don't have to do any processing to obtain a scene graph from an image. Indeed, in the original NSM work [16] the authors had to perform a prepossessing step to extract a scene graph from an image. In the literature this technique is called *scene graph generation* [34] [6].

3.5 Instructions

The next step is to obtain the sequence of instructions $\{r_i\}_{i=0}^{N}$ from the question. This process has three steps: embedding, projection and decoding.

First each word of the question is embedded. For this work, since we want to remove as much ambiguity as possible, we use one-hot encoding. Another viable solution would be to use off-the-shelf precomputed embeddings, like they did in the original paper using GloVe [29].

Next, the NSM projects each of the embedded tokens to the concept vocabulary. To accomplish this, the model takes each token and applies a learned function that maps the token to the closest concept in the concept vocabulary C or, if there is no similar concept, leaves the original token as is. The operation starts by computing a similarity score between the input token and the concept vocabulary as follows:

$$P_i = softmax\left(w_i^\top \mathbf{W} C^*\right) \tag{3.1}$$

where C^* is the concept vocabulary C concatenated with a learned default vector c', and $\mathbf{W}$ are parameters, initialized to the identity matrix. C^* and c' are designed to give the architecture enough freedom to leave some words untouched. Using this score the network performs a soft-selection of the closest concept in the vocabulary as follows:

$$v_i = P_i\left(c'\right) w_i + \sum_{c \in C} P_i\left(c\right) c \tag{3.2}$$

If there is no concept close enough, the score $P_i\left(c'\right)$ dominates the weighted sum and w_i is left mostly unchanged.

Summarizing, if a token is sufficiently close to a concept, then the token gets mapped. Otherwise the token remains as its original vector. In this process all distances are computed according to the embedding space.

After translating the word sequence $\{w_i\}_{i=1}^{P}$ to the normalized sequence $\{v_i\}_{i=1}^{P}$, the network proceeds to the decoding step. In this step, the normalized vectors are used to obtain $N+1$ reasoning instructions $\{r_i\}_{i=0}^{N}$. The normalized sequence is passed through an LSTM network to obtain a final hidden state q that represents the sequence. The vector q is repeated $N+1$ times and is passed to a simple recurrent neural network to obtain a $N+1$ hidden states $\{h_i\}_{i=0}^{N}$. It is these hidden states that are finally used to produce the instructions r_i in the following manner. Let $V^{P\times d} = \{v_i\}_{i=1}^{P}$ be the matrix of vectors v_i stacked:

$$r_i = softmax\left(h_i V^\top\right) V \tag{3.3}$$

Conceptually, each hidden vector h_i is used to perform a soft selection of one of the question words, using the attention mechanism [4]. A probability distribution over the question words is computed based on the similarity with the hidden vector h_i and then all question words are averaged using the similarity score as weight. This makes it so that the most similar words dominate the weighted sum.

3.6 Transition function - State machine simulation

With the components described in previous sections, the states S, the edges E, the sequence of instructions $\{r_i\}_{i=0}^{N}$ and the concept vocabulary C and D, we can finally understand how the NSM simulates an automaton. Conceptually the transition function δ is in charge of taking a probability distribution p_i and, guided by the instruction r_i, computes a new distribution p_{i+1}. Starting from a uniform distribution p_0, this process is repeated N times reaching the final distribution p_N and instruction r_N. Finally, using p_N and r_N, the whole graph is aggregated into a single vector, which is used to predict the answer to the question.

The first step in this process is identifying to which concept category the instruction r_i is referring to. The instruction could have selected a relation within the question or a specific property of an object, so the first step is to identify this concept category. For this, the NSM computes the following vector R_i for the instruction being processed.

$$\begin{aligned} R_i &= softmax\left(r_i \circ D\right) \\ r_i' &= R_i\left(L\right) \end{aligned} \tag{3.4}$$

In this equation $\circ$ is the dot product, and let's recall that $r_i \in \mathbb{R}$, $D \in \mathbb{R}^{(L+1)\times h}$, so therefore $R_i \in \mathbb{R}^{L+1}$ and $r_i' \in [0, 1]$.

After deciding to which category the instruction r_i relates to, this information is used to select one of the properties of each node and edge. This is then used to compute a score for each node and relation in the graph in the following way.

$$\begin{aligned} \gamma_y\left(s\right) &= \sigma\left(\sum_{j=0}^{L-1} R_i\left(j\right)\left(r_i \circ \mathbf{W}_j s^j\right)\right) \\ \gamma_i\left(e\right) &= \sigma\left(r_i \circ \mathbf{W}_L e'\right) \end{aligned} \tag{3.5}$$

Where $\{\mathbf{W}_j\}_{j=0}^{L-1}$ and $\mathbf{W}_L$ are learned parameters and σ is a non-linearity. Conceptually, this step computes a relevance score for each node and edge with respect to the instruction by performing a weighted average of its attributes using the vector R_i.

Having the scores for each node and edges allows the NSM to accomplish the main goal of this step, to compute the next probability distribution over the nodes. For this, the NSM redistributes probability based on each node's individual score and also the score of its neighbours. How much the individual score adds to the new distribution compared to the score of its neighbours is decided by the value $r_i' = R_i\left(L\right)$ which, as described previously, represents if the instruction refers to a relation or not. The final expression for p_{i+1} is as follows

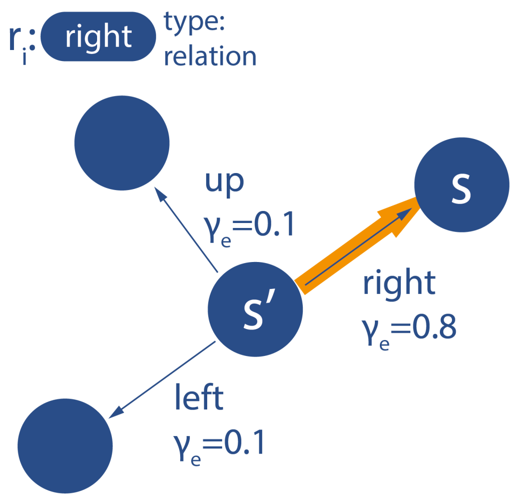


Figure 3.2: A state transition where probability gets redistributed through an edge.

$$
\begin{aligned}
p_{i+1}^{s} &= softmax_{s \in S} \left(\mathbf{W}_s \cdot \gamma_i \left(s \right) \right) \\
p_{i+1}^{r} &= softmax_{s \in S} \left(\mathbf{W}_r \cdot \sum_{(s',s) \in E} p \left(s' \right) \cdot \gamma_i \left(\left(s, s' \right) \right) \right) \\
p_{i+1} &= r_i' \cdot p_{i+1}^{r} + \left(1 - r_i' \right) \cdot p_{i+1}^{s}
\end{aligned}
\tag{3.6}
$$

Figure 3.2 shows a simplified schematic of a state transition. In the figure, the instruction is closest to the relation *to the right* and therefore most of the probability score of node s' is transferred to node s through the connecting edge labeled as *to the right*.

This redistribution process is repeated N times to reach the final state r_N. In this stage, the graph's state is aggregated using instruction r_N into a single vector m, which can then be used in a simple feed forward network to predict an answer to the question. In this particular work, the final m vector is concatenated with the question vector q and passed through a 2-layer feed forward network. The precise equation to obtain vector m is as follows

$$
m = \sum_{s \in S} p_N \left(s \right) \sum_{j=0}^{L-1} R_N \left(j \right) s^j \tag{3.7}
$$

Chapter 4

Research methodology

Let's first refresh our hypothesis.

> There are natural questions over graph data that the NSM cannot answer. These include questions that require the ability to count, that require comparisons between objects and questions where logical conjunction must be performed.

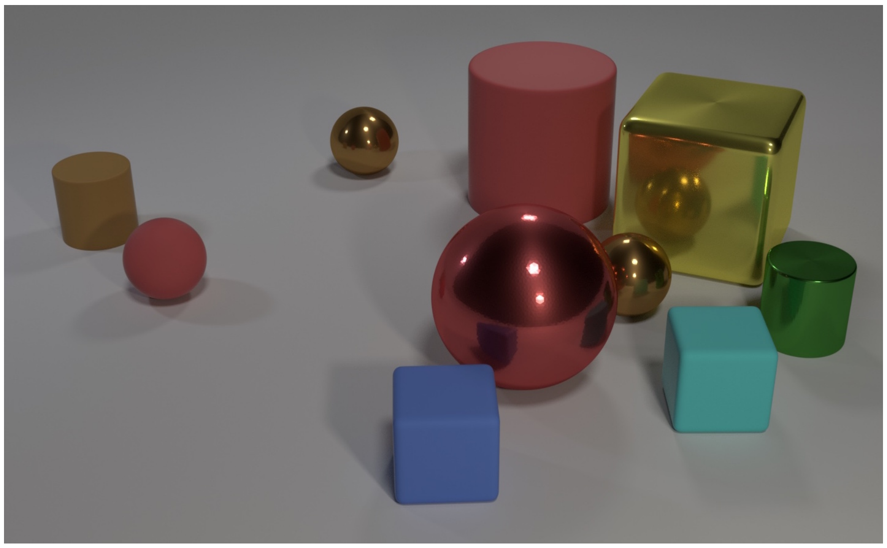
We will prove this hypothesis using an experimental method. First we will establish a baseline using questions that we know from previous works the NSM can answer successfully [16]. With this baseline at hand we will show that there is a set of questions where the architecture shows a significant decrease in performance.

This chapter is structured in four parts; (1) first we present the rationale behind our dataset selection, (2) then we describe in detail the dataset used, (3) following we define the question sets, grouped by abilities required to answer them, that will be used to perform experiments and (4) we finalize with the details of input representation for the architecture.

4.1 Dataset Selection

For our objective of studying the graph-traversing limitations of the NSM, we require data with very specific properties. Most notably, since the architecture is designed to process graphs instead of raw images, the original paper relied on off-the-shelf scene graph generation architectures to circumvent this limitation. In our setup, this preliminary step adds unnecessary noise to the input of the NSM, and therefore we require a dataset that provides ground-truth labels for the scene graphs. This immediately rules out most of the well-known VQA datasets, like COCO [23] and the original VQA [3] dataset (and its various successors [35, 11, 1]).

Another important property is the ability to control the types of questions contained in the dataset, and hopefully have enough questions of each category so that we can train and evaluate the architecture using only each subset of questions. This lends itself nicely



Q: Are there an equal number of large things and metal spheres?
Q: What size is the cylinder that is left of the brown metal thing that is left of the big sphere?

Figure 4.1: A sample image and questions from CLEVR. Figure taken from Johnson, et al. [18].

for a *synthetic* dataset, where questions are generated by filling predefined templates. Two notable datasets fall into this category, CLEVR [18] and GQA [17]. Both match our first criteria of having scene graph ground truth labels, but we ultimately decided for CLEVR because GQA didn't provide the source code to generate new questions. This is an important restriction because many of the question categories we want to study are underrepresented in the datasets, and so to perform comparable and meaningful experiments we need to generate more questions to reach a minimum number for each category.

4.2 CLEVR Dataset

The CLEVR dataset [18] is a synthetically generated evaluation dataset specifically designed to test the visual reasoning capabilities of VQA systems. The dataset is multimodal, meaning it contains both images and questions about these images. Following we explain how these images and questions were generated and describe some of its properties. An example image-question pair can be seen in Figure 4.1.

4.2.1 Images

The dataset contains around 100k images synthetically rendered using Blender [8], a 3D modeling software. Each image contains a variable number of objects placed over a smooth surface. Objects have four kinds of properties; color, shape, size and material. Figure 4.2 shows the whole range of attributes available in CLEVR.

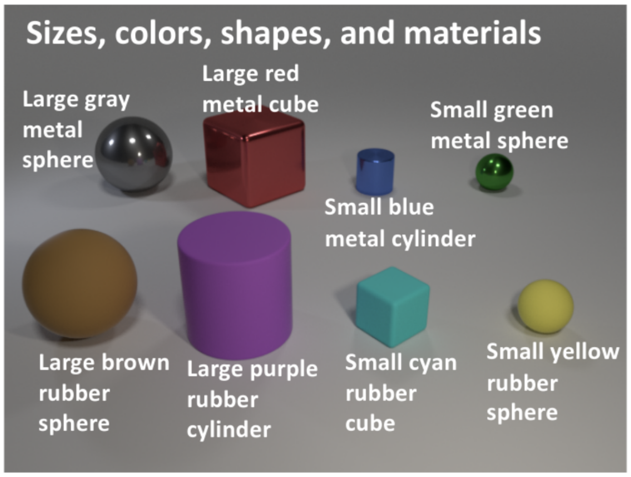
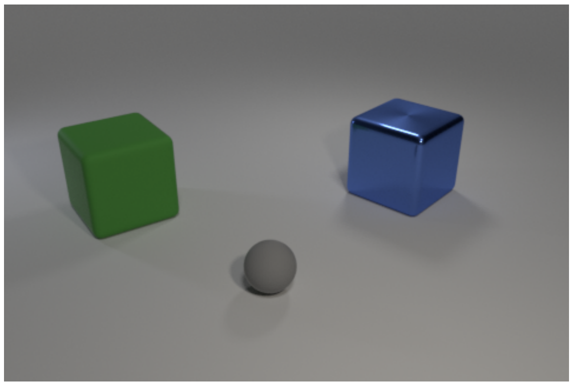

Sizes, colors, shapes, and materials
Large gray metal sphere
Large red metal cube
Small green metal sphere
Small blue metal cylinder
Large brown rubber sphere
Large purple rubber cylinder
Small cyan rubber cube
Small yellow rubber sphere

Figure 4.2: Landscape of object attributes available in CLEVR.



```
{
  "objects": [
    {
      "color": "blue", "size": "large",
      "shape": "cube", "material": "metal",
    },
    {
      "color": "gray", "size": "small",
      "shape": "sphere", "material": "rubber",
    },
    {
      "color": "green", "size": "large",
      "shape": "cube", "material": "rubber",
    },
  ],
  "relationships": {
    "right": [[], [0, 2], [0]],
    "behind": [[], [0], [0, 1]],
    "front": [[1, 2], [2], []],
    "left": [[1, 2], [], [1]],
  },
}
```

Figure 4.3: An example image and its associated scene graph.

Additionally, since the dataset was generated synthetically, each image comes with a ground truth *scene graph*, that contains all the information present in the image, including spatial relations between objects. Figure 4.3 contains an example scene graph in JSON format. Notice how each object has four attributes and there is a `relationships` field that captures all the available spatial relationships between objects.

4.2.2 Questions

CLEVR contains around one million questions synthetically generated using a custom template engine. The dataset contains 90 questions families, each with 2 to 4 templates. Using the scene graph described previously, which contains all the information in an image, the template engine takes in a generic template containing placeholders for objects and fills them

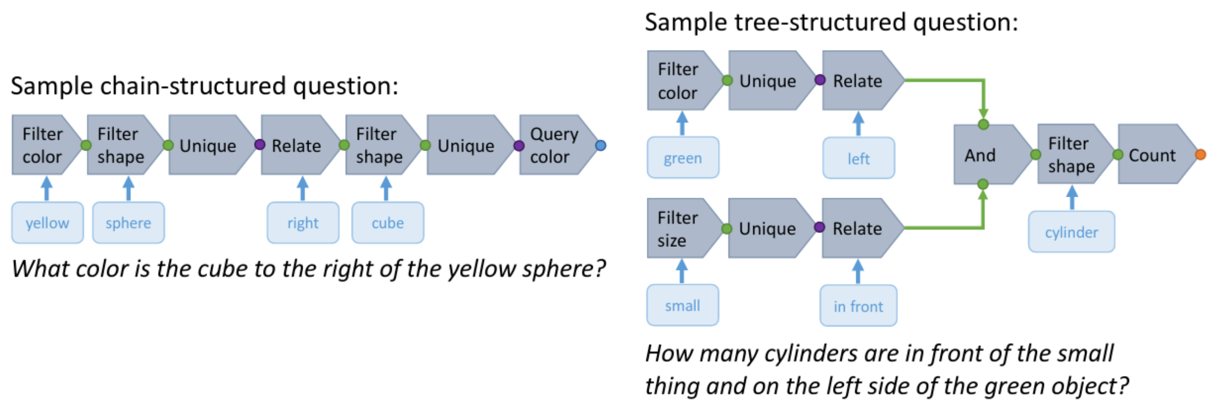

Sample chain-structured question:
Filter color
Filter shape
Unique
Relate
Filter shape
Unique
Query color
yellow
sphere
right
cube
What color is the cube to the right of the yellow sphere?
Sample tree-structured question:
Filter color
Unique
Relate
green
left
Filter size
Unique
Relate
small
in front
And
Filter shape
Count
cylinder
How many cylinders are in front of the small thing and on the left side of the green object?

Figure 4.4: Example questions and their functional programs.

in using the scene graph data. The following is an example template and a question that could be produced from this template.

> What color is the <OBJ> that is <REL> the <OBJ>?
> What color is the **cube** that is **to the right of** the **yellow sphere**?

Each question template *defines* a family of questions that share the exact same structure, but are filled in with different values depending on each image. The dataset also defines a so called *functional program* that captures the logical operations that have to be performed to answer each question. This language includes basic operations like counting, filtering and comparing attributes. Example questions and their functional program can be found in Figure 4.4.

4.3 Question families chosen for this work

The CLEVR dataset contains 90 question families, but not all of them are useful for our goals. In this section we precisely describe the question families we chose to perform experiments on, and the rationale behind these decisions. Each subsection also contains example questions in each category together with their accompanying functional program.

4.3.1 Questions with jumps

The first category contains questions with “jumps”, meaning they require to pay attention to different objects in the scene and "jump" between them following relations. After following this relation chain, the question ends by querying one of the properties of the final object. Below is an example question and its functional program.

> What is the color of the cube to the right of the sphere that is above the cylinder?

```
query_color(
  uniq(
    filter_shape(
      cube,
      relate(
        right,
        uniq(
          filter_shape(
            sphere,
            relate(
              above,
              uniq(
                filter_shape(
                  cylinder,
                  scene()
)))))))))
```

We use this first category as a baseline and, as mentioned earlier, our experiments will focus on showing how the following categories compare to it.

4.3.2 Counting questions

This category includes questions where their final operation requires counting objects. In the dataset there are two kinds of questions that require counting; questions where counting is an *intermediate* operation and question where counting is the *final* operation. Here are two examples, respectively.

Are there the same number of blue objects and red objects?

```
equal_integer(
  count(
    filter_color(
      red,
      scene()
  )),
  count(
    filter_color(
      blue,
      scene()
  ))
)
```

And

How many objects are there to the right of the blue cube?

```
count(
  filter_color(
    red,
    relate(
      right,
      uniq(
        filter_shape(
          cube,
          filter_color(
            blue,
            scene()
))))))
```

For the latter example the question ends with a counting operation, whereas in the former example the counting operation occurs before a final comparison. In a different section of this work we test if the NSM is able to perform comparisons between objects, so in order to isolate the reasoning capabilities we test with each question category, we choose only questions that end in a counting operation.

As a special case of this category we define a small subcategory that checks if the model is able to answer questions about object *existence*, i.e. being able to count zero or more objects. An example question below.

Are there any red cubes to the right of the small cylinder?

```
exist(
  filter_shape(
    cube,
    filter_color(
      red,
      relate(
        right,
        uniq(
          filter_shape(
          cylinder,
          filter_size(
            small,
            scene()
)))))))
```

4.3.3 Questions with conjunctions

The next question category includes questions that require performing a logical `and` operation. This involves comparing two "branches" in a question, and reasoning about both at the same time. Below an example.

What is the color of the object that is to the right of the big cube and to the left of the grey sphere?

```
query_color(
  uniq(
    intersect(
      relate(
        right,
        uniq(
          filter_shape(
            cube,
            filter_size(
              big,
              scene()))))),
      relate(
        left,
        uniq(
          filter_shape(
            sphere,
            filter_color(
              grey,
              scene()))))
)))
```

As one can appreciate, the question has two *branches*, one that identifies objects to the right of the big cube and the other focuses on objects to the left of the grey sphere. The final question queries a property of an object that satisfies *both* of these spatial conditions.

The logical `and` operation is an intermediate operation and, in this case, the question ends by querying a property of an object. However, there are other terminal operation applicable to conjunction questions, like counting. Since we test the counting capabilities of the model in a different experiment, for this section we chose only questions that require conjunction and finish by querying a property of an object identified by the conjunction.

4.3.4 Questions with implicit relations

The category of questions with implicit relations evaluates if the model is capable of focusing on an object that shares a property with another object. This relation of *sharing a property* is not explicitly represented in the scene graph of the image, so we call this category *implicit* relation. An example question below.

What is the size of the other object that is the same color as the metal cylinder?

```
query_size(
  uniq(
    same_color(
      uniq(
        filter_shape(
          cylinder,
          filter_material(
            metal,
            scene()
))))))
```

In this kind of questions the model needs to perform a jump between objects just like in section 4.3.1, but without using a relation from the graph. This makes this category of questions considerably harder.

Similar to section 4.3.3, the intermediate `same_color` operation could have ended in a counting operation, e.g. *How many object are the same size as the metal cylinder?*. Again, in this section we focus only on questions that end by performing a querying operation, a reportedly simple operation for the NSM, in order to isolate the challenge of using an implicit relation.

4.3.5 Questions with comparisons

The final category includes questions that require comparing a property between two objects. In this sense, it is similar to the conjunction question where there are two *branches* and some operation is applied to both of them. An example comparison question follows.

> Is the size of the cube that is to the left of the yellow sphere the same as the rubber cylinder?

```
equal_size(
  query_size(
    uniq(
      filter_shape(
        cube,
        relate(
          left,
          uniq(
            filter_color(
              yellow,
              filter_shape(
                sphere,
                scene()))))))),
  query_size(
    uniq(
      filter_material(
        rubber,
        filter_shape(
          cylinder,
          scene()))))
)
```

All the questions in this category have binary yes/no answers. Although the answer space is simple, these questions require some form of memory to remember the property of one of the object while the property of the other object is being computed.

4.4 Input representation

While the image representation (as a scene graph) was presented in the listing of Figure 4.3, the way the question is passed to the network undergoes some transformations from the raw representation as text presented in previous sections. Namely, we ignore the *text* representation of each category since the same question can be asked in different ways. For example

> What is the size of the object to the right of the blue cube?
> There is an object to the right of the blue cube. What is its size?

As one can see, this is the same question and contains the exact same information, but the position of the attributes and the question itself vary. To avoid this variability we use the functional program associated with this question category. For this example it would be

```
query_size(
  uniq(
    relate(
      right,
      uniq(
        filter_color(
          blue,
          filter_shape(
            cube,
            scene()
))))))
```

Furthermore, since this representation includes parenthesis and commas, and the input to the NSM is a sequence of tokens, we choose to represent the functional program using prefix notation (also known as polish notation), which is the final representation that we input to the network.

```
query_size uniq relate right uniq
    filter_color blue filter shape cube scene
```

The arity of each functions in the CLEVR world is fixed, so no parenthesis are required to parse this program. This provides a compact and unambiguous representation of the question that isolates the reasoning capabilities of the model from the ability to deal with noise in the input.

Chapter 5

Experiments

The network was implemented using the PyTorch framework [28] and the training process was done using the PyTorch Lightning framework [9]. The network was trained using the Adam optimizer and for the most part we used the hyperparameter ranges recommended in the original paper, i.e. learn rates $[5 \cdot 10^{-5}, 10^{-4}, 5 \cdot 10{-}5, 10^{-3}]$, batch sizes $[32, 64, 128]$ and ELU non-linearity σ.

We use an RNN hidden size of 100, since in early experiments we notice that larger values don't impact the network's performance. The input size used was 45, because this was the number of unique tokens in the questions for CLEVR and we used one-hot encoding. We tried a number of computation steps in the range of [0..6] but found that with 4 computation steps the network reached the best performance and after that there were no improvements in any of the question groups we evaluated. Since the model was evaluated in a synthetic dataset we don't observe overfitting, so we don't use any kind of regularization. The network was trained until train loss stopped improving. All experiments took between 6 and 12 hours to complete, and the hardware used varied between either an RTX 2080 and RTX 3090.

Each experiment was performed using a different number of training examples, because not always the original categories of CLEVR aligned with the ones used in this work, so for some categories there were too few questions. To fix this, we used the official CLEVR generation code to sample questions specifically belonging to the categories we've described in the previous chapter. We aimed for around 100k to 200k questions for each category.

For each experiment the train and test sets are balanced with respect to the possible answers. For example, in the counting questions, if the number of possible counts found in pictures is six, then each count has the same amount of questions. Considering this, the metric used for all experiment is the test accuracy.

5.1 Questions with jumps

Our baseline is the result of the architecture trained and evaluated in questions with "jumps". Table 5.1 shows accuracy results per number of jumps in the question and per each final query

Table 5.1: Test results per number of jumps and final query type of the NSM, trained using questions that only include jumps and end by querying a property.

nhop \ query_*	size	color	material	shape
0	1.00	1.00	1.00	1.00
1	0.99	0.99	0.99	0.99
2	0.99	0.99	0.99	0.99
3	0.99	0.99	0.98	0.99

Table 5.2: Results of the NSM when trained on questions that have up to two jumps, but tested in a test dataset that has 3-jump questions only.

nhop \ query_*	size	color	material	shape
3	0.95	0.95	0.94	0.95

operation.

The train set had around 120k questions and the test set had around 27k questions. A model that chooses randomly an answer for each query category would have $\frac{1}{2} = 0.5$ accuracy for the property size, $\frac{1}{8} = 0.125$ accuracy for color, $\frac{1}{2} = 0.5$ accuracy for material and $\frac{1}{3} = 0.33$ accuracy for shape.

We also tested the ability of the NSM to generalize to questions that require more jumps than what the architecture saw in the training set. To this end, we trained the NSM with questions having only zero, one or two jumps, and evaluate with questions that require three jumps. Table 5.2 shows the results obtained using this setup.

5.2 Counting questions

The first category of questions beyond the baseline is what we arbitrarily consider the simplest category, which are the counting questions. Within this category we further separate into questions of *existence* and actual counting.

5.2.1 Existence

Table 5.3 shows the results of training the NSM using questions of the simplest counting case, where the answer is only whether an object exists or not. That is, it is a special counting question asking if the count is greater than 0.

For this experiment the train set had around 100k questions and the test set had 10k questions. Since these questions are *yes* or *no* questions, the accuracy of a random model would be 0.5.

Table 5.3: Simplest counting case. Results of the NSM answering questions of whether an objects exists in a picture or not.

nhops	0	1	2	3
accuracy	0.99	0.61	0.64	0.66

Table 5.4: General counting case. Questions were balanced based on answers. Results are grouped by number of jumps to reach the final query and object count.

nhops \counts	0	1	2	3	4	5
0	0.75	0.38	0.41	0.19	0.31	0.70
1	0.42	0.31	0.21	0.21	0.26	0.39
2	0.49	0.37	0.24	0.22	0.32	0.32
3	0.47	0.36	0.21	0.25	0.29	0.36

5.2.2 Counting

Following the simpler case, Table 5.4 shows the results of training and evaluating with counting questions, grouped by number of jumps to reach the final query and number of objects matching the query.

The train set had around 70k questions and the test set around 10k questions. For this category, extra care was taken to balance the dataset by answer to avoid biases. Each possible count had the same amount of training and testing questions, so that the network couldn't guess the most common object count and achieve a high accuracy. Given this, a model that chooses randomly has an accuracy of $\frac{1}{6} = 0.166$.

5.3 Conjunction

Table 5.5 presents the results of the question category that involves a logic `and` and ends by querying an object's property. Results are grouped by queried property.

For this experiment the train set had around 140k questions and the test set had around 15k. Similar to the jumping questions, the accuracies of a random model are 0.5, 0.12, 0.5 and 0.33 for properties size, color, material, and shape, respectively.

Table 5.5: Conjunction accuracy results, grouped by property.

query_*	size	color	material	shape
acc	0.80	0.72	0.80	0.76

Table 5.6: Implicit relation results grouped by property used to jump and final queried property. The training set didn't contain any questions in the diagonal since those would be trivial to answer.

same_* \ query_*	size	color	material	shape
size	-	0.75	0.91	0.86
color	0.73	-	0.75	0.65
material	0.92	0.73	-	0.88
shape	0.86	0.59	0.87	-

Table 5.7: Results of comparison questions grouped by queried property and number of jumps included in the question.

nhops \equal_*	size	color	material	shape
0	0.85	0.71	0.82	0.57
1	0.62	0.50	0.60	0.51
2	0.54	0.52	0.55	0.48

5.4 Implicit relations

Table 5.6 shows the results of the experiments with questions having implicit relations. Results are grouped as follows: each row represents the property used to relate two objects and columns represent the final property that was queried after the jump. Since jumping using a property and then querying that same property (e.g. *What is the color of the object of the same color as the small ball?*) would be trivial to answer, the diagonal is left empty.

For this experiment we used a train set of 210k questions and a test set of 15k questions. The random accuracy for each property are the same as the previous section.

5.5 Comparison

Table 5.7 show the results of the comparison questions grouped by final queried property and number of jumps involved in the selection of the objects to be compared. Since for comparison questions two branches must be resolved and then the final nodes of each branch must be compared according to the queried property, jumps could be involved in determining those final nodes of each branch.

For this experiment we used 350k train questions and 15k test questions. These questions are binary yes/no questions, so random accuracy is 0.5 for all subcategories.

Chapter 6

Discussion

6.1 Baseline

Our first step in the experimentation process was to verify the claimed capabilities of the NSM in previous work [16]. Table 5.1 shows a strong indication that the architecture is able to answer questions with jumps, since it achieves virtually perfect accuracy in all subcategories, regardless of the number of jumps required to solve the question. To further validate the baseline, we test the network on questions that require more jumps than what it has seen in the training process. For this we train the network on questions that require at most two jumps, but evaluate using questions that require three jumps. As we can see in Table 5.2, the network is indeed able to generalize, and we could say it has learned the general *algorithm* for performing steps, rather than an *ad-hoc* method to answer the specific question types it was trained on.

One thing to note about this experiment is that the training was done using 4 processing steps. Any number below proved to lower the accuracy of the network, which is inline with intuition since each processing step focuses on an object, and there are four objects involved in a three-step question. Slightly more processing steps didn't yield any improvement, and many more (say, ten), decreased the network's performance. All of this is inline with previous work, validates our implementation for this thesis and serves as a baseline for comparison when evaluating the network in the proposed categories.

6.2 Hypothesis categories

Existence Among the question categories we propose in our hypothesis, we start with counting questions since it is the simplest to conceptualize. Table 5.3 shows the simplest case of counting, where the only answers are either zero or more than zero. Immediately we can see a decrease in performance of the NSM compared to jumping questions. The only subcategory where the NSM performs comparably to our baseline is when the questions doesn't even require hops to be answered, e.g. *Is there a blue object?*. However, when any number of jumps are required to identify the queried object, the network's performance drops abruptly. A random baseline would have a 0.5 accuracy, so the results for multiple hops is closer to random performance than to the optimal of jumping questions.

Counting When we add difficulty by requiring a specific number as answer, we see no outstanding performance in Table 5.4. A random baseline for this table would have 0.16 accuracy on all subcategories, and we see that most cells are below 0.4 accuracy. We observe also a considerable drop in performance when requiring one or more jumps, and in every row the highest accuracy is achieved when answering for zero objects. This could be explained because the "lack" of objects in the image can be encoded in a similar way as in the existence questions, but encoding a specific number in the final vector that aggregates the graph proves to be challenging. The number of objects in the image was fixed by the available images in CLEVR, so we couldn't experiment with higher objects counts, but we speculate the higher the number of possible counts the lower the network's performance is to be expected.

An interesting phenomenon occurs in this category, where consistently the performance drops for one, two, three and four objects, but ramps up for five objects. If we focus on the first row of Table 5.4, where this phenomenon is more noticeable, we see that when no jumps are involved the network reaches a remarkable accuracy when counting both zero and five objects in the scene. From Table 5.3 we see that the network is indeed capable of distinguishing between *no* objects and *some* objects. In the case of counting we could think that the network can encode the information of *many* objects, for which it ends up predicting 5 most of the time, but on the contrary, is unable to make a difference when encoding 3 or 4 in the final vector m of Eq 3.7.

The overall decrease in performance could be explained by the final aggregation operation in Eq 3.7. In Eq 3.5 we see that the network is indeed capable of giving a score to multiple nodes, and so if a question asks to count *red cubes*, the network will indeed be able to highlight *all* red cubes in the graph. However, we think this information is lost in Eq 3.7. Since $\sum p_N = 1$, regardless of the number of objects in the scene, it is difficult for the architecture to encode a specific number in vector m.

Conjunction Next up we observe Table 5.5. Again we see a decrease in performance compared to jump questions, although not so steep compared to the counting questions. Accuracy of a random model would be less than 0.5, so the results for this category are actually closer to the questions with jumps. Interestingly, there is a correlation between the results per queried property and the respective random performance, since there are just two options for size and material, but three for shape and eight for color.

There is no definitive answer on how to explain the accuracy of this experiment, but we can provide some speculative possible explanations. First, one thing to notice is that the final operation of selecting a property of an object matches the design of the network, whereas the counting operation is simply not supported by the original design. The network would have to rely on a more complex representation of the information in the final aggregating vector to encode a count instead of an object property. Another idea is that the specific conjunction questions created for CLEVR don't have jumps within each branch. For example, consider the question

> *What is the color of the sphere left of the cylinder and below the cube?*

and compare to

> *What is the color of the sphere that is both left to the cylinder and below the red cube that is to the left of the blue cube?*

In the first case the final node of both branches is reached by directly identifying an object, but in the second case, one of the branches requires following a relationship to reach the final node of the branch. This might increase the complexity of the questions, but it is not explored in the CLEVR dataset.

The simplicity of the branches could allow the network to reach the final node in only two steps. Using the first question as example, in the first step both the cylinder *and* the cube could be attended to. Then, the next instruction would have to encode both relationships, *left* and *below*, so that in the second equation of Eq 3.6 a node that satisfies both relationships receives the most probability weight. This is indeed possibly if we look at Eq 3.3, where an instruction is free to attend to as many tokens from the input question as needed. Conceptually, this would equate to computing both question branches *in parallel*.

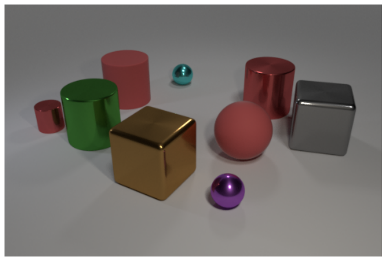
The previous mechanism relies on the assumption that the instruction in Eq 3.3 is able to attend to an arbitrary number of tokens in the input question. This is possible in theory, but in practice it is extremely hard for a softmax layer to reliably highlight multiple positions in the input vector. It is for this reason that we believe the network is far from a perfect performance on this question category. The instructions would have to be *overloaded* with information, which we believe could hinder the network's operation.

Implicit relations The category of implicit relations follows the trend of having lower results than the questions where the jumps are explicit in the graph. However, among the unfavorable categories the model performs remarkably well. Most notably, we can see in Table 5.6 the questions that requires a jump by the same size and the last query is about the object's material, the network achieves 0.91 accuracy. The inverse also holds, i.e. when the jump is done by same material and the final query is about size. Let's remember that both properties size and material have only two values each, so the variability in those subcategories is small.

Similar to conjunction questions (see Table 5.5), we see that the network performs worse in questions that involve color, since it is the property that has the highest number of different values.

The high performance of the network can be explained by a faulty implementation of the algorithm to create this category of questions in the CLEVR dataset. Although the authors took the necessary precautions to avoid having questions with answers that are self contained, meaning it is not even necessary to look at the image to answer them (e.g. *What is the color*

There is a small purple thing that is the same material as the big gray object; what shape is it?



What material is the other thing that is the same size as the gray object?

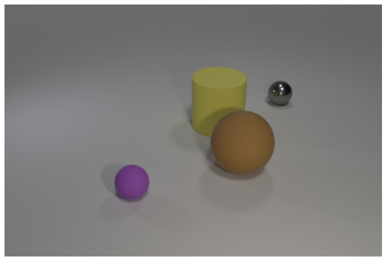


Figure 6.1: The first figure shows a question in the category of implicit relations where the relation is actually not needed to identify the queried object. In this case, there is only one purple thing, and so it is not necessary to associate its material to the big grey object. Compare to the second question, where it is mandatory to first identify the grey object (the small grey ball) and then, using its size (small) find the other small object in the image.

of the objects that is the same color as the blue cube?), they didn't verify that the implicit relations had to be followed in order to identify the final objects. This is best understood with examples. Figure 6.1 shows an example of this phenomenon, where sometimes the final queried object can be identified without attending to anything else, just because there is not enough variety of objects in the image. The first question of Figure 6.1 expects an agent to identify a "small purple thing" using the material of "the big gray object". Since there is only one small purple object, knowing its material is superfluous. When this phenomenon occurs, the network is only required to focus on an object and select one of its property, an operation we have already establish the NSM can do. Manual exploration of the test set shows that around 50% of the questions in this category suffer from this problem, which explains the high performance of the model.

The poor performance of the network on the rest of the questions (like the second question of Figure 6.1) might be explained by the distribution update equations in Eq 3.6. From the second equation we see that *the only* mechanism the network has to carry any information from step i to step $i+1$ is via the relationship component p^r_{i+1}. p^s_{i+1} is computed using only the current instruction and the properties of nodes, and the previous probability weight of each node is not considered. With this in mind, for a question like *What material is the other thing that is the same size as the gray object?*, even though the network could focus on *the grey object*, its probability weight could not be transferred to any other object because the next instruction could not include a spacial relation, as there are none in the question.

Comparisons The category of comparison questions shows a behaviour similar to the existence questions. As we can see in Table 5.7, the performance of the network is relatively good when there are no jumps involved in the question, but when some hops are required to reach the final node of each branch, the performance worsens to random levels (let's remember all comparison questions are binary yes/no questions).

To attempt explaining the relatively high performance when there are no hops invFeedbackolved, we reuse the argument used in the discussion for conjunction questions. Albeit difficult in practice, Eq 3.3 indeed gives the network enough freedom to attend to multiple objects in a single instruction. Let's use as example the question *Are the cube and cylinder the same color?*. The network could attend to both the cube and cylinder, an the final aggregated vector m of equation Eq 3.7 would contain both object's colors. If these colors are the same, then only one color would be present in m, but if the colors are different, then more that one color would appear in m. This could allow the network to correctly answer the question.

Where we think the network fails for this case is again in Eq 3.6, and its inability to conditionally update the state based on the previous state. Even assuming the network could compute branches in parallel, as speculated for questions with conjunctions, the presence of a relationship in a instructions forces *all* nodes to transfer its probability to neighboring nodes. Let's consider a question where only one branch includes a relationship, like *Is the cube the same color as the sphere above the cylinder?*. Assuming the first instruction correctly pushes the NSM to focus on both the cube an the sphere, when trying to transition only one branch using the relation *above*, all nodes will transfer its probability via this relation, since the second equation from Eq 3.6 applies to all nodes. This would make the network shift the high probability mass the cube had, effectively "forgetting" about the relevance of the cube. This would make it impossible for the network to answer the question, after having lost the initial focus on the cube.

Chapter 7

Conclusions

Let's once again remember our hypothesis

> There are natural questions over graph data that the NSM cannot answer. These include questions that require the ability to count, that require comparisons between objects and questions where logical conjunction must be performed.

In the previous chapter we've seen the NSM has virtually random performance on counting and comparison questions. The NSM's performance also decreases in questions that require conjunctions (logical `and`) and that feature implicit relations (see subsection 4.3.4 for a description). In these last two categories the decrease in performance is not so steep, but it is clearly lower compared to an almost perfect score in jump-query questions, for which the network was designed. These question categories are found in the CLEVR dataset which was designed to test multiple reasoning capabilities of VQA systems, and so it is sensible to use them to evaluate the NSM. These findings show strong evidence in favor of our hypothesis.

Summarizing, in this work we've shown how the NSM is unable to answer question types that involve counting, performing comparisons and logical conjunctions. Our results show that although the network doesn't degenerate to random levels on most cases, indeed its performance is well below compared to its intended target questions.

Our proposed question categories were strongly influenced by our notions of traditional state machines, and how their pitfall is their lack of memory. Hence, our 'questions required (in our opinion) the use of some sort of "memory" to be answered, either be it storing a running number for counting, or remembering the state of a previous computation, as in the comparison questions. Understanding the *cause* of the low performance of the NSM is challenging as we would've had to perform a more thorough analysis of the network's weights and their evolution over the training process, which quickly becomes intractable.

The insight obtained in this work helps lay ground for future work. A straightforward improvement, inspired from automata theory, could be adding a simple LIFO memory module to the network, simulating a push-down automaton. Previous work on machine-like architectures [12, 13] have shown good results when allowing the network to interact with a memory module, but to the best of our knowledge, this technique hasn't been applied to answering questions based on graphs (or images). This extension wasn't explored in this work because of time constraints.

Finally, this work provides the community with an open source implementation of the network and the training details to replicate this work. The original authors didn't open-source the code for the network nor their experiments, so this would be the only open source implementation. Hopefully this can encourage more people to experiment with the network and further this research line.

Bibliography

[1] Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. Don't just assume; look and answer: Overcoming priors for visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4971–4980, 2018.

[2] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016.

[3] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *International Conference on Computer Vision (ICCV)*, 2015.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Rob Miller, Aubrey Tatarowicz, Brandyn Allen White, Samuel White, and Tom Yeh. Vizwiz: nearly real-time answers to visual questions. In *International Cross-Disciplinary Conference on Web Accessibility*, 2010.

[6] Tianshui Chen, Weihao Yu, Riquan Chen, and Liang Lin. Knowledge-embedded routing network for scene graph generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6163–6171, 2019.

[7] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Learning universal image-text representations. *arXiv preprint arXiv:1909.11740*, 2019.

[8] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[9] William Falcon and Kyunghyun Cho. A framework for contrastive self-supervised learning and designing a new approach. *arXiv preprint arXiv:2009.00104*, 2020.

[10] Donald Geman, Stuart Geman, Neil Hallonquist, and Laurent Younes. Visual turing test for computer vision systems. *Proceedings of the National Academy of Sciences of the United States of America*, 112, 03 2015.

[11] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6904–6913, 2017.

[12] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[13] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, Oct 2016.

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[15] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. 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, 2017.

[16] Drew Hudson and Christopher D Manning. Learning by abstraction: The neural state machine. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[17] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6700–6709, 2019.

[18] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997, 2017.

[19] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2989–2998, 2017.

[20] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International journal of computer vision*, 123(1):32–73, 2017.

[21] Gen Li, Nan Duan, Yuejian Fang, Ming Gong, Daxin Jiang, and Ming Zhou. Unicoder-vl: A universal encoder for vision and language by cross-modal pre-training. In *AAAI*, pages 11336–11344, 2020.

[22] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[24] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, pages 13–23, 2019.

[25] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *Advances in neural information processing systems*, pages 289–297, 2016.

[26] Mateusz Malinowski and Mario Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. *Advances in neural information processing systems*, 27, 2014.

[27] David Mascharka, Philip Tran, Ryan Soklaski, and Arjun Majumdar. 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, 2018.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[29] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[30] Kevin J Shih, Saurabh Singh, and Derek Hoiem. Where to look: Focus regions for visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4613–4621, 2016.

[31] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

[32] Kewei Tu, Meng Meng, Mun Wai Lee, Tae Eun Choe, and Song-Chun Zhu. Joint video and text parsing for understanding events and answering queries. *IEEE MultiMedia*, 21(2):42–70, 2014.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[34] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 670–685, 2018.

[35] Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Yin and yang: Balancing and answering binary visual questions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5014–5022, 2016.

[36] Guangming Zhu, Liang Zhang, Youliang Jiang, Yixuan Dang, Haoran Hou, Peiyi Shen, Mingtao Feng, Xia Zhao, Qiguang Miao, Syed Afaq Ali Shah, et al. Scene graph generation: A comprehensive survey. *arXiv preprint arXiv:2201.00443*, 2022.