Universität der Bundeswehr München

A Neural-Symbolic Framework for Mental Simulation

Michael Kissner

Vollständiger Abdruck der von der Fakultät für Informatik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Gutachter:

1. Prof. Dr.-Ing. Helmut Mayer
2. Prof. Dr. rer. nat. Martin Werner

Die Dissertation wurde am 21.02.2020 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Informatik angenommen. Die mündliche Prüfung fand am 30.07.2020 statt.

Abstract

We present a neural-symbolic framework for observing the environment and continuously learning visual semantics and intuitive physics to reproduce them in an interactive simulation. The framework consists of five parts, a neural-symbolic hybrid network based on capsules for inverse graphics, an episodic memory to store observations, an interaction network for intuitive physics, a meta-learning agent that continuously improves the framework and a querying language that acts as the framework's interface for simulation. By means of lifelong meta-learning, the capsule network is expanded and trained continuously, in order to better adapt to its environment with each iteration. This enables it to learn new semantics using a few-shot approach and with minimal input from an oracle over its lifetime. From what it learned through observation, the part for intuitive physics infers all the required physical properties of the objects in a scene, enabling predictions. Finally, a custom query language ties all parts together, which allows to perform various mental simulation tasks, such as navigation, sorting and simulation of a game environment, with which we illustrate the potential of our novel approach.

Zusammenfassung

Wir präsentieren einen neuro-symbolischen Rahmen, für die Beobachtung der Umgebung und kontinuierliches lernen von visueller Semantik und intuitiver Physik, um diese dann in einer interaktiven Simulation wiederzugeben. Der vorgestellte Rahmen besteht aus fünf Teilen, einem neuro-symbolischen hybriden Netzwerk basierend auf Capsules für inverse Grafik, einem episodischen Gedächtnis zum Speichern von Beobachtungen, einem Interaktionsnetz für intuitive Physik, einem Meta-Lernagenten zur kontinuierlichen Verbesserung des Rahmens und einer Abfragesprache, die als Schnittstelle des Rahmens für Simulationen fungiert. Durch lebenslanges Meta-Lernen wird das Capsule Netzwerk ständig erweitert und trainiert, sodass es sich mit jedem Schritt besser an seine Umgebung anpasst. Dies ermöglicht das Lernen von neuer Semantik anhand von wenigen Beispielen und minimaler Eingabe von einem Orakel über die gesamte Lebenszeit. Aus dem aus Beobachtungen Gelernten leitet der Teil für intuitive Physik alle benötigten physischen Eigenschaften der Objekte in einer Szene ab und ermöglicht somit Vorhersagen. Schließlich werden alle Teile durch eine angepasste Abfragesprache zusammengeführt, womit verschiedene mentale Simulationen ausgeführt werden können, wie Navigation, Sortierung und die Simulation einer Computerspielumgebung, anhand derer wir das Leistungsvermögen unseres neuen Ansatzes demonstrieren.

Danksagung

Besonders möchte ich meinem Doktorvater Herrn Univ.-Prof. Dr. Helmut Mayer für sein entgegengebrachtes Vertrauen und die Möglichkeit der Promotion danken. Durch seine Betreuung und Unterstützung wurde mir ein sehr interessantes Forschungsthema ermöglicht. Aus unseren Gesprächen habe ich immer viel gelernt und neue Blickwinkel entdeckt für meine Arbeit.

Ebenfalls herzlich danken möchte ich Herrn Univ.-Prof. Dr. Martin Werner für seine Bereitschaft das Zweitgutachten zu übernehmen. Sein tiefer Einblick in das Thema hat mir stets geholfen frühzeitig mögliche Probleme zu identifizieren und eine Lösung zu finden.

Natürlich möchte ich auch allen derzeitigen und ehemaligen Angehörigen der Professur für Visual Computing und Professur für Geoinformatik danken für die interessanten Diskussionen und deren Hilfe während meiner Promotion.

Mein besonderer Dank gilt meiner Partnerin, meinen Eltern, meiner Schwester und meinen Freunden auf dessen Unterstützung und Verständnis ich während dieser Zeit immer zählen konnte.

Contents

Chapter 1

Introduction

In order for a robot or an intelligent agent to plan its next move, it needs to be able to simulate the environment and possible outcomes to make informed decisions. This process is referred to as mental simulation and resembles a game engine in many ways, as it allows the agent to explore and interact like a player would in a game. And just as in gaming, for very demanding tasks the simulation environment needs to accurately reflect reality, be it trough physical accuracy or semantic content, so that the agent is able to perform adequate planning.

Our motivation and one of the open problems in deep learning is to find a way to learn to perform mental simulations based on observations, by constructing an inverse game engine [Battaglia et al., 2013, Ullman et al., 2017] and trying to understand the world in a generative way. The ability to do so is not only interesting from an artificial intelligence (AI) point of view, but also from a cognitive science perspective, as it is believed that this might resemble and, thus, help to understand the way humans are able to perform mental simulations.

A mental simulation engine needs to perform three tasks: First, it has to be able to extract, store and reproduce all the required semantic information from its environment. Second, it needs the ability to modify and visualize properties of its observation. And third, it must be able to perform intuitive physics. While executing each of these tasks in isolation has proven to be possible with adequate results, merging them has been quite problematic. This is due to the fact that each of the tasks operates on and produces different or insufficient representations of information not suitable for the others. E.g., modifying observations and intuitive physics work most reliably on a symbolic representation with attributes, preferably in a graph or tree structure, which current visual extraction processes can only partially deliver.

Ideally, a unified structure for visual information should be chosen on which all aspects of a mental simulation pipeline can operate reliably. Modern game engines, such as Godot by the Godot Engine Team [2020], give a hint at what might be a good candidate for this structure: a scene-graph.

Scene-graphs consist of a tree-structure, where the root node represents the scene as a whole and each hierarchy of descendants the parts or parts-of-parts of the parent node, i.e., objects that appear in the scene. For a game engine to be able to render this tree, each node is equipped with attributes that describe the object it represents. This graph is easy to manipulate and apply intuitive physics on. The key we have identified in creating a mental simulation engine is, thus, defining a deep learning algorithm that is capable of producing such a scene-graph from an image.

1.1 Problem Statement

In this thesis, we approach the creation of a mental simulation framework capable of interacting with a planning agent. We subdivide this goal and identify four major problem domains that need to be solved, so that it is in fact usable by any type of planning agent. We provide a solution that addresses all of the following problems:

1. **Unified Semantic Information:** A suitable vision algorithm must be devised that is capable of producing a scene-graph usable by the other systems in the framework.
2. **Querying and Simulating:** A simulation pipeline must be defined that the agent may interface with and query.
3. **Explainability:** The framework must be able to present explainable results, so that an agent is capable of comprehending why specific events occur in a simulation.
4. **Lifelong Meta-Learning:** The framework must be able to continuously learn and adapt to new environments.

We note that these points are tightly linked. The ability to simulate and query the result of a simulation requires a semantically rich data structure. As a rich data structure needs to be produced, the algorithm generating this structure should intuitively become more capable in explaining its decisions. And as the algorithm becomes more capable to explain, the path to tackle meta-learning should become more clear.

Through the intertwined nature of the problems, they represent the minimal set of issues we need to contend with in order to define an effective mental simulation framework. Thus, when approaching the definition of such a framework, we must constantly evaluate the interoperability of the domains to ensure that the final solution is coherent.

1.2 Approach and Thesis Outline

After giving a short introduction to the mathematical concepts we require throughout the thesis and a brief overview of related work, we follow the path outlined by the problem statement. Due to the broad nature of the subject, we focus mainly on toy examples throughout the thesis.

In Chapter 4 we begin by describing a generative grammar capable of producing a parse-tree that closely resembles a classical scene-graph found in game engines. This grammar is used as a guide to design an inverse-graphics pipeline and the attempt to invert it. The result is comparable to a capsule network [Hinton et al., 2011, Sabour et al., 2017] and we make use of the same terminology, but modify the inner workings of the individual capsules in order to generate a visual parse-tree from an image. The resulting parse-trees are stored as observations in an episodic memory.

The final structure of our neural-symbolic capsule network is highly modular and allows for a symbolic interpretation for each of its parameters. In Chapter 5 we use this explainability to devise a meta-learning algorithm that continuously improves the network with each new scene it encounters in a few-shot approach. This meta-learning process modifies existing capsules, adds new ones and re-routes the flow of information inside the network nearly autonomously, requiring only occasional feedback from an oracle.

With all observations stored in memory we are able to perform simple tracking and introduce an intuitive physics pipeline based on interaction networks [Battaglia et al., 2016] in Chapter 6 that is capable of predicting future events from past observations. We find that by closely analyzing all the semantics in memory we are able to deduce useful physical properties, increasing the versatility of intuitive physics by including complex physical interactions. With this additional information, the original parse-tree generated by the capsule network becomes a full-fledged scene-graph in the game engine sense.

Capsule network, episodic memory, meta-learning pipeline and intuitive physics are tied together to form a mental simulation pipeline. However, in this raw state it is still not usable by a planning agent. In Chapter 7 we, therefore, devise a custom querying language that makes the process of mental simulation more explicit by providing a streamlined interface and demonstrate some of many possible use-cases. Our resulting framework and all its components, named VividNet, are depicted in Figure 1.1.

In Chapter 8 we discuss the advantages and shortcomings of our approach. We focus on the comparison to artificial neural networks and highlight how our approach addresses some of the issues found in the classical approach, such as the lack of explainability and the binding problem. Finally, a conclusion referring to our problem statement and an outlook are given.

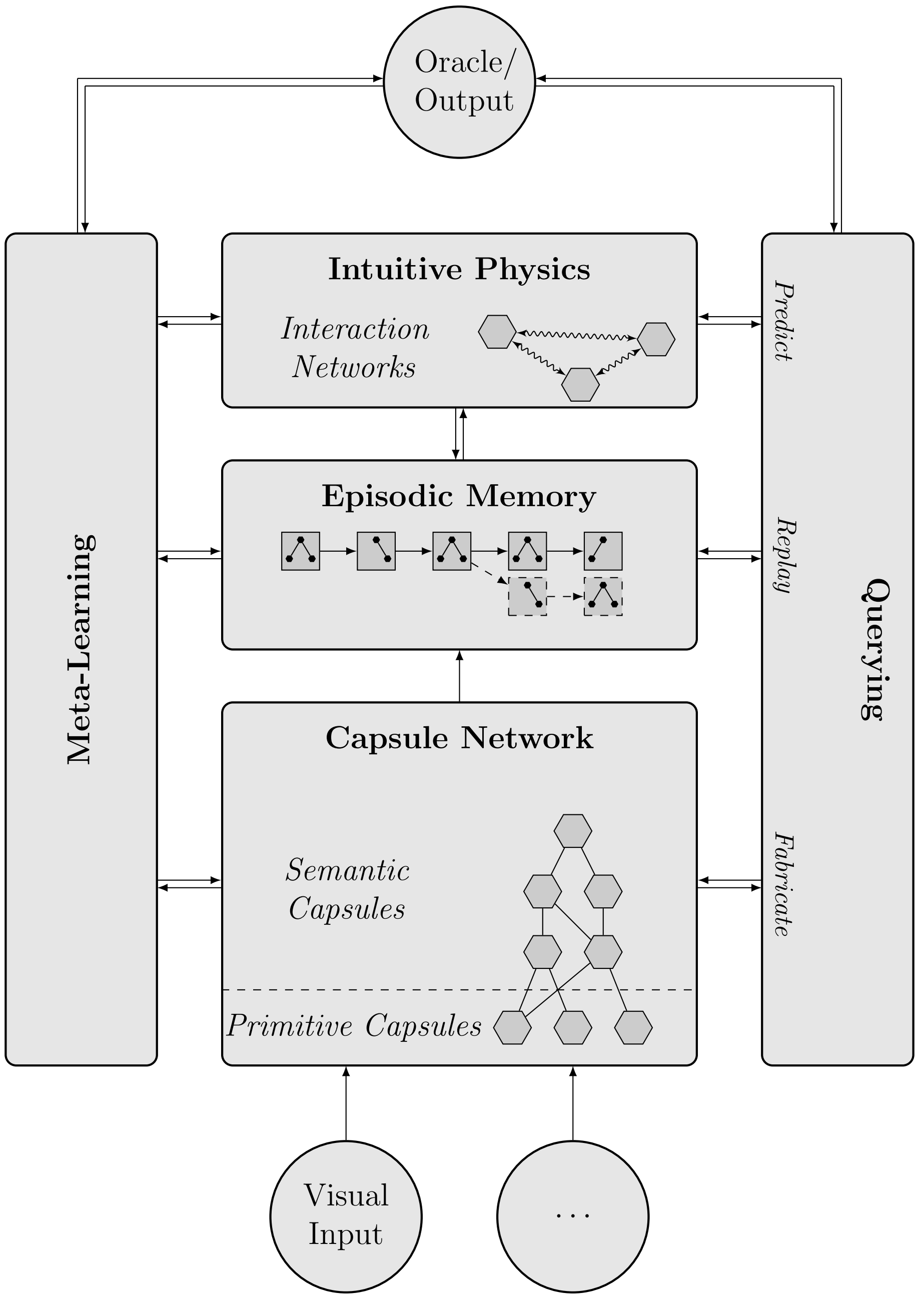


Figure 1.1: The final VividNet framework. Chapter 4 introduces the capsule network and episodic memory, Chapter 5 discusses the meta-learning pipeline, Chapter 6 adds intuitive physics and Chapter 7 covers the querying process.

Chapter 2

Mathematical Preliminaries

For our discussion of the meta-learning pipeline, we borrow the language used primarily in two areas of mathematics: topology and group theory. We use topology for a deeper understanding of the structure of an object's configuration space, consisting of its attributes. The attributes themselves also play an important role in our analysis, for which we employ group theory to study their behavior under transformations. In this chapter we present these concepts and the notation we use.

2.1 Quotient Spaces

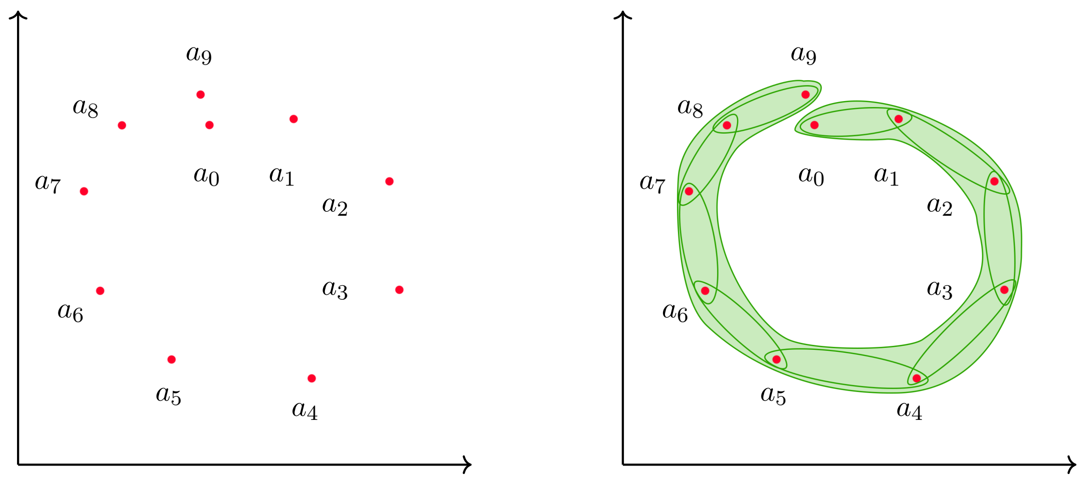
To introduce quotient topologies, we need the notion of a topological space. A topological space $(X, \mathcal{T}_X)$ is a set X and a collection $\mathcal{T}_X$ of subsets of X that satisfy the following propositions [Nakahara, 2003]:

1. The empty set $\emptyset$ and X are in the collection $\mathcal{T}_X$ ($\emptyset, X \in \mathcal{T}_X$).
2. The finite or infinite union of subsets found in $\mathcal{T}_X$ also belongs to $\mathcal{T}_X$ (any number of $U_i \in \mathcal{T}_X$ satisfy $\bigcup_i U_i \in \mathcal{T}_X$).
3. The intersection of finite subsets found in $\mathcal{T}_X$ also belongs to $\mathcal{T}_X$ (finite $U_i \in \mathcal{T}_X$ satisfy $\bigcap_i U_i \in \mathcal{T}_X$).

The collection $\mathcal{T}_X$ defines the topology and we refer to the individual subsets U_i as open sets.

As an example, consider gathering individual data points a_i for training purposes. These points form a point cloud in some $\mathbb{R}^n$ and we treat them as the set X. In most machine learning algorithms, such as manifold learning, one of the goals is to find the neighborhood relations between the individual points, which are vital in performing meaningful inter- and extrapolations. These neighborhood relations can be used to define a topology in the sense of $\mathcal{T}_X$ (cf. Figure 2.1).

The situation in Figure 2.1 is quite ambiguous from a machine learning perspective. It suggests that the data forms an open circle, but it might as well be a closed circle. While the data X remains the same, the assumed topology defined by $\mathcal{T}_X$ for an open or a closed circle are very different and provide a conflicting interpretation.



Point Cloud X

Topology of $(X, \mathcal{T}_X)$ (only selected elements of $\mathcal{T}_X$ are shown)

Figure 2.1: The topology of the dataset influences its interpretation. Here, one of many possible topologies for the data (left) is shown (right) in which a_9 can only be reached from a_0 by traversing through all other points first, instead of having a direct connection.

We can, however, construct a closed circle out of an open circle by gluing together its ends. The result of gluing together parts of a topological space is referred to as a quotient space. Such a quotient space is created by identifying points in X with each other, essentially regarding them as equal. In our machine learning example of Figure 2.1, this would mean that we assume that the end-points of the open circle, a_0 and a_9, actually describe the same point and, thus, close the loop.

Points in X are identified using an equivalence relation denoted by $a \sim b$. However, $a \sim b$ does not restrict us to only identify single points with one another as in our example. We may also have an entire equivalence class

$$[a] = \{x \in X | x \sim a\} \quad . \tag{2.1}$$

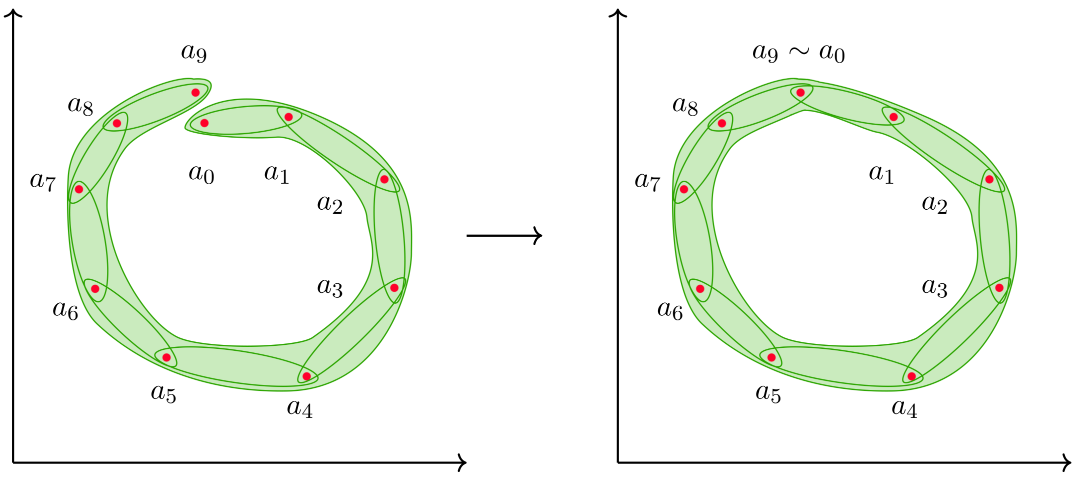
Using these equivalence classes, we introduce the quotient space $Y = X/\sim$, which is a rigorous way of defining our gluing process:

$$Y = \{[x] | x \in X\} \quad . \tag{2.2}$$

Further, the canonical quotient map $q\colon X \to X/\sim$ also induces a topology on Y

$$\mathcal{T}_Y = \{U \subseteq Y | q^{-1}(U) \in \mathcal{T}_X\} \quad , \tag{2.3}$$

making $(Y, \mathcal{T}_Y)$ a topological space. Figure 2.2 shows the concrete example of identifying the data points a_0 and a_9 to form the quotient space $Y = X/\{a_0, a_9\}$.

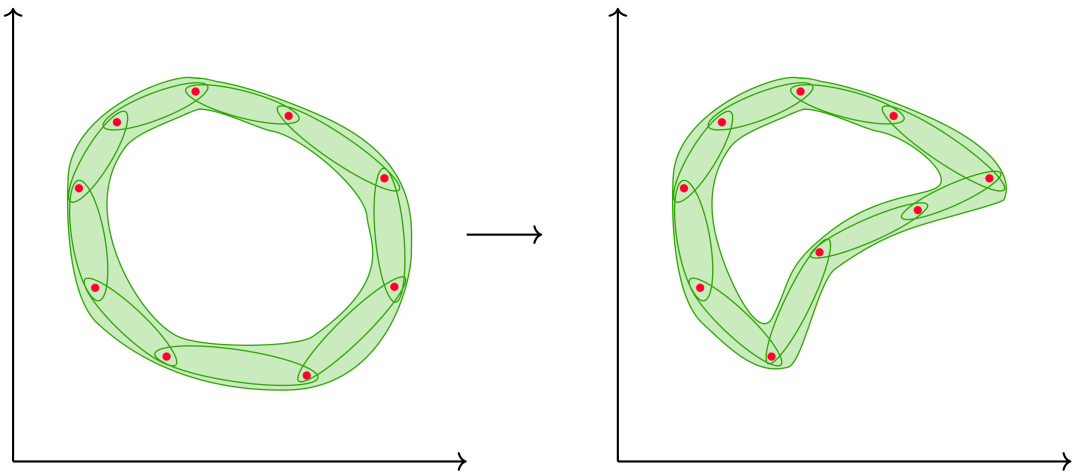


Topology of $(X, \mathcal{T}_X)$ (only selected elements of $\mathcal{T}_X$ are shown)

Topology of $(Y, \mathcal{T}_Y)$ (only selected elements of $\mathcal{T}_Y$ are shown)

Figure 2.2: By identifying the points a_0 and a_9 of $(X, \mathcal{T}_X)$, the loop is closed to form a circle.

Next, we discuss the concept of homeomorphism. From a purely topological standpoint, the value of an individual element of Y does not matter. In our example, this is equivalent to saying that their position in $\mathbb{R}^n$ is of no topological consequence. We can continuously move around the points without changing the topology and get a set $\hat{Y}$, such as in Figure 2.3.



Topology of $(Y, \mathcal{T}_Y)$ (only selected elements of $\mathcal{T}_Y$ are shown)

Topology of $(\hat{Y}, \mathcal{T}_{\hat{Y}})$ (only selected elements of $\mathcal{T}_{\hat{Y}}$ are shown)

Figure 2.3: $(Y, \mathcal{T}_Y)$ and $(\hat{Y}, \mathcal{T}_{\hat{Y}})$ are said to be homeomorphic. Both are also homeomorphic to the unit circle S^1, but not to the interval $[0, 1]$.

The two topological spaces $(Y, \mathcal{T}_Y)$ and $(\hat{Y}, \mathcal{T}_{\hat{Y}})$ are said to homeomorphic, as there exists a function $f\colon Y \to \hat{Y}$ which satisfies:

1. f is bijective.
2. f is continuous.
3. f^{-1} is continuous.

We refer to such a function f as a homeomorphism and use $Y \simeq \hat{Y}$ to denote that two spaces are homeomorphic.

For machine learning purposes, however, the value of the individual elements of Y does matter. We consider a special type of topological space referred to as a metric space (M, d) [Burago et al., 2001]. Here, M is a set and $d\colon M \times M \to \mathbb{R}$ a metric or distance function on M. This metric satisfies the following conditions:

1. $d(x, y) = d(y, x)$.
2. $d(x, x) = 0$ and $d(x, y) > 0$ for $x \neq y$.
3. $d(x, y) + d(y, z) \geq d(x, z)$.

We see that this metric space (M, d) induces a natural topology determined by d and the open discs

$$U_\epsilon(M) = \{y \in M | d(x, y) < \epsilon\} \quad , \tag{2.4}$$

are referred to as the metric topology.

This induced topology also allows us to form quotient metric spaces $M/\sim$ on which we define a quotient semi-metric d_R. We refer to this function as a semi-metric, as it allows for two points x and y with $x \neq y$ to have a vanishing distance $d_R(x, y) = 0$, violating condition 2 of a regular metric.

To construct d_R, we again consider Figure 2.2. If we were to measure the distance between a_1 and a_8 on our quotient metric space, it seems obvious that we should follow the sequence (a_1, a_0, a_9, a_8) and sum up the individual distances $d(a_1, a_0)+d(a_0, a_9)+d(a_9, a_8)$. However, our original metric does not take into account that a_0 and a_9 now describe the same point in the quotient space and $d(a_0, a_9) \neq 0$. Instead, to ensure that the distance between a_0 and a_9 is not counted, we perform a check if there is a shorter path by taking into account all points that are equivalent at each step in the sequence. In our case, we find $a_0 \sim a_9$ and skip this hop in the sequence. To formalize this, we define the quotient semi-metric to be

$$d_R([x], [y]) := \inf\{\sum_{j=1}^{k} d(x_j, y_j)\} \quad , \tag{2.5}$$

where we take the infimum over all possible sequences $(x_1, \cdots, x_k)$ and $(y_1, \cdots, y_k)$, with $x_1 \in [x]$, $y_k \in [y]$ and $y_j \sim x_{j+1}$. This ensures that $d_R([x], [y]) \leq d(x, y)$.

Generally, a metric space doesn't have a well defined dimension and we employ methods out of fractal geometry to make an estimate, such as the Hausdorff dimension [Hausdorff, 1918] or the box-counting dimension [Falconer, 1990]. We use a definition similar to the box-counting dimension for our metric space (M, d). We assume M is a subset of some $\mathbb{R}^n$

and consider a space $\hat{M}$, for which there exists a continuous, surjective map $\psi\colon M \to \hat{M}$, such that for every point $p \in M$ we have

$$\|p - \psi(p)\| < \rho \quad , \tag{2.6}$$

where $\|\cdot\|$ measures the Euclidean distance in $\mathbb{R}^n$. We are essentially allowing points in $\hat{M}$ to rearrange themselves by a small margin. Finally, our contracted box-counting dimension (CBC) is defined as

$$\dim_{CBC} M := \min_{\psi} \lim_{\epsilon \to 0} \frac{\log N_{\psi}(\epsilon)}{\log 1/\epsilon} \quad , \tag{2.7}$$

where $N_{\psi}(\epsilon)$ is the number of ϵ^n-sized boxes on an ϵ-spaced grid in $\mathbb{R}^n$ that cover $\hat{M}$. Through this construction $\hat{M}$ is a contraction of M and removes small numerical errors that accidentally generate unnecessary dimensions in our dataset. For practical purposes, we take ϵ to be a small, yet computable value and continue to the limit until the point the dimension starts to converge.

2.2 Groups and Representations

To understand how transformations of the input affects the output of a function, group and representation theory is a helpful tool. By a group [Willwacher, 2014] we mean a set G together with a map (referred to as multiplication) of the form

$$\circ\colon G \times G \to G \tag{2.8}$$

which satisfies:

1. For $g_1, g_2, g_3 \in G$ we have $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$. (*Associativity*)
2. There exists an identity element $\mathbf{1}$, such that for any $g \in G$ holds $\mathbf{1} \circ g = g \circ \mathbf{1} = g$. (*Identity*)
3. For all $g \in G$ there exists an inverse element $g^{-1} \in G$, such that $g \circ g^{-1} = g^{-1} \circ g = \mathbf{1}$. (*Inverse*)

One example for a group are invertible linear maps between vector spaces. In the case of $\mathbb{R}^n \to \mathbb{R}^n$, we have invertible $n \times n$ matrices. We refer to this group as $GL(\mathbb{R}^n)$ or $GL(n)$ in short and as $GL(V)$ for a general vector space V. When considering this group in detail, we see that there exist multiple subsets of $n \times n$ invertible matrices, which themselves form a group under the conditions above. One such example is the group of rotations $SO(n)$ and we refer to a group that satisfies this condition as a subgroup and write $SO(n) \subset GL(n)$.

We also define a product of two groups, $G \times H$, by creating a set of pairs (g, h) and defining the product as $(g_1, h_1)(g_2, h_2) = (g_1 g_2, h_1 h_2)$. In this case, we explicitly see that G and H are subgroups of $G \times H$.

While a group element of G transforming other elements of G through multiplication is interesting in itself (for example, matrix-matrix multiplication), we are interested in how it behaves when applied to another space (for example, matrix-vector multiplication). We use the term action to describe how a group element acts on such a space X, i.e., the left group action is a function $\varphi\colon G \times X \to X$ that satisfies:

1. For all $x \in X$, we have $\varphi(\mathbf{1}, x) = x$. (*Identity*)
2. For all $g, h \in G$ and $x \in X$, we have $\varphi(gh, x) = \varphi(g, \varphi(h, x))$. (*Compatibility*)

With an equivalent set of axioms as the left group action, the right group action $\varphi\colon X \times G \to X$ is defined. With both of these group actions in place, we simplify the φ notation and write $g \cdot x$ or $x \cdot g$ to imply an element g acting on x from the left or the right.

Next, we introduce homomorphisms between groups, not to be confused with homeomorphisms between topologies. A homomorphism is a map $f\colon G \to H$ that preserves the operations of the structures, i.e., $f(g_1 g_2) = f(g_1) f(g_2)$.

This allows us to define a representation of a group G on a vector space V as a homomorphism $\rho\colon G \to GL(V)$. Group representations are useful if the group G is an abstract group and we want to study its behavior on a more concrete environment, such as $\mathbb{R}^n$. As an example, if we take the group of 2-dimensional (2D) rotations $SO(2)$, we may represent it as a set of 3×3 matrices that define a rotation around some specific axis in 3-dimensional (3D) space using $\rho\colon SO(2) \to GL(3)$, essentially making it a 3D rotation.

We now study how these group actions can help us to better understand data in a machine learning context following the terminology set out by Higgins et al. [2018]. The idea of disentanglement in the context of deep learning is to find attributes or latent variables which act independently of the others. As an example, the *position* of a table is disentangled from its *color*, but *color* is often tightly entangled with other material properties, such as *roughness*.

To find entanglement in the set of all attributes A, we study the effect that the change of a subset A_1 has on the remainder A_2, where $A = A_1 \oplus A_2$ and by $\oplus$ we mean the direct sum of two attribute sets $\{\alpha_1, \alpha_2\} \oplus \{\alpha_3, \alpha_4\} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. Consider a group $G = G_1 \times G_2$ that acts on A by a group action $\cdot\colon G \times A \to A$ and on each A_i independently as $\cdot_i\colon G_i \times A_i \to A_i$, such that

$$(\mu_1, \mu_2) \cdot (\vec{\alpha}_1, \vec{\alpha}_2) = (\mu_1 \cdot_1 \vec{\alpha}_1, \mu_2 \cdot_2 \vec{\alpha}_2) \tag{2.9}$$

for all $\mu_1 \in G_1$ and $\mu_2 \in G_2$. In this case, we refer to A_1 and A_2 as being disentangled with respect to G.

Closely related to the idea of entanglement is equivariance. Consider a map $f\colon X \to Y$. We say that f is equivariant with respect to some group G if

$$f(\rho_X(\mu) \cdot x) = \rho_Y(\mu) \cdot f(x) \tag{2.10}$$

is true for all $\mu \in G$ and all $x \in X$ and where $\rho_X : G \to GL(X)$ and $\rho_Y : G \to GL(Y)$ are representations of G on X and Y. Thus, a transformation of the input results in an equivariant transformation of the output.

As an example, consider a function f that measures the average brightness of an image. Here, transforming an input image uniformly so that it has a lighter tone will result in a higher brightness output. With respect to this group of transformations, f is equivariant. Next, we consider a group of transformations that lightens one half of the image and darkens the other by the same amount. Under such a group, the brightness output remains unchanged and f would be invariant to such transformations.

Chapter 3

State of Research

The idea that a computer could understand an image, simulate what it has seen and finally visualize the result, similar to the process of vision-based mental simulation in humans, has inspired many researchers. One of the pioneers in this field has been Lawrence G. Roberts, who was able to extract basic geometric properties from simple shapes, apply transformations and re-render them in a new pose [Roberts, 1963]. In the following, we give a brief overview of recent research focusing on efforts that utilize deep learning [Goodfellow et al., 2016].

3.1 Inverse Graphics and Visual Question Answering

For fully differentiable convolutional neural networks (CNN) [LeCun et al., 1989, Gu et al., 2018] it isn't entirely clear what attributes of an object get encoded in its latent representation, how entangled they are and where exactly in the network they are located. There has been considerable progress in making these models more interpretable [Simonyan et al., 2014, Mahendran and Vedaldi, 2015, Ribeiro et al., 2016, Lipton, 2018, Montavon et al., 2018, Zhang et al., 2018] and it is generally believed that an object's representation is present in the later layers, which can be visualized by inverting the neural network [Dosovitskiy and Brox, 2016]. However, it has been shown that CNNs have a bias towards detecting textures instead of shapes [Geirhos et al., 2019] and are often fooled by the same object in a different pose [Alcorn et al., 2019], indicating that the representation might be quite different to that of a human and possibly lacking some important attributes for our understanding.

During the quest for a more understandable representation for objects, there has been considerable effort to detect the poses of objects in a scene, such as finding their general orientation and bounding box [Song and Xiao, 2016] or their rotational pose [Yang et al., 2015, Mahendran et al., 2017, Wang et al., 2018b]. All these approaches explicitly search for attributes. Using predictability minimization [Schmidhuber, 1992], generative adversarial networks (GANs) [Niemitalo, 2010, Goodfellow et al., 2014] or variational auto-encoders [Kingma and Welling, 2014], these pose attributes often emerge naturally in the latent variables, but require some disentanglement [Kulkarni et al., 2015, Chen et al., 2016, Higgins et al., 2017, Nguyen-Phuoc et al., 2019] in order to be usable.

While the pose is interesting, its meaning only becomes clear when the actual shape of an object is known. General image segmentation results are often not sufficient to describe a 3D shape. Classically [Szeliski, 2010], a shape prior is chosen and then fit to the image [Chan and Zhu, 2005, Calakli and Taubin, 2011, Dame et al., 2013, Häne et al., 2017]. Newer methods of shape fitting employ deep learning to aid in the fitting [Izadinia et al., 2017]. They require a set of 3D models that resemble the shape of real objects as well as possible, such as ShapeNet [Chang et al., 2015].

Apart from fitting individual shapes, also the outcome of an entire procedural modeling pipeline may be fitted. This process is far more complicated, as it not only involves fitting a single shape, but also requires an understanding of the relations between parts. However, a deeper understanding of object-part relations and the overall semantics does allow for more complex tasks, such as reconstructing hidden objects based on perceived symmetries [Bokeloh et al., 2010]. Also in inverse procedural modeling, deep learning is employed [Felzenszwalb et al., 2010, Tulsiani et al., 2017, Zou et al., 2017, Liu et al., 2018, Romaszko et al., 2018, Gafni et al., 2019]. In this regard, a method related to our approach are stacked capsule autoencoders [Kosiorek et al., 2019], which encode the object-part relation by assigning a different capsule to each.

A widely held belief is that one understands something better if one has the ability to generate it. In computer vision specifically, it is assumed that it should be easier to understand the interpretation of an image, if it can be generated using a grammar, a tree structure, a shape program, etc., which produces an image based on a series of commands. The goal is to infer these programs directly from the image using different machine learning methods [Teboul et al., 2010, Stava et al., 2014, Wu et al., 2017b, Liu et al., 2019a, Tian et al., 2019].

A structured scene representation is ideal to answer questions regarding the scene [Yi et al., 2018, Mao et al., 2019], such as about composition or appearance, referred to as visual question answering (VQA). On the other hand, it has also been shown, that such in-depth scene understanding is not necessarily required for VQA. It can be performed by a fully differentiable approach [Hudson and Manning, 2018], using a mixture of convolutional and recurrent networks.

Further, as an inverse to the idea of VQA, GeNeVa [El-Nouby et al., 2019], a proposed GAN, is capable of drawing images based on text, implying that it has some hidden understanding of the scene. Such an understanding of the scene also allows the user to manipulate and re-render it [Reed et al., 2015, Yao et al., 2018].

3.2 Intuitive Physics

The previously discussed methods capable of describing a scene can be well suited for physical predictions, but also require a move towards video input instead of a static image. The most obvious way to perform physical simulations would be a hard-coded engine, such as PhysX [Nvidia Corporation, 2018]. These engines can calculate mechanical interactions accurately (billiard balls colliding) and even include advanced electro-magnetic effects (a concentrated beam of light igniting paper). Yet, this would require full knowledge of the physical properties, for which it is nearly impossible to infer them from an image alone, such as weight or temperature. However, a human is still able to perform crude predictions from images, without knowing much else in a process often referred to as intuitive physics. This insight motivates the use of deep learning in order to replicate this skill.

We differentiate between two different approaches to intuitive physics: The first is to learn and predict physical effects directly based on image data [Lerer et al., 2016, Ehrhardt et al., 2017, 2018, Iten et al., 2018]. This allows the model to be learned without any prior assumptions and has the added benefit that it can consider the entire image for clues about physical effects, including those that might have been missed by a more knowledge-based approach. However, such end-to-end systems have the problem of trying to solve two tasks, vision and physics, at once, each of which is difficult on its own.

The second approach to intuitive physics is the one presented here, viewing vision and physics as two distinct domains, akin to how simulation [Battaglia et al., 2013] and game engines work [Ullman et al., 2017]. This allows for more interpretability and stability [Chang et al., 2017], but requires prior extraction of all important information from the image into an adequate representation. One such approach are interaction networks by Battaglia et al. [2016], which model physical effects on pairwise interactions of objects. We introduce this approach in more detail in Section 6.1. However, besides interaction networks there other models for physical scene understanding [Chang et al., 2017, Wu et al., 2017b,a, Steenkiste et al., 2018, Kipf et al., 2018].

Interaction networks can be connected to the scene understanding methods discussed above or directly to a CNN [Watters et al., 2017] to learn from video data. There are also interesting extensions, such as automatic relation detection in scenes [Raposo et al., 2017], integration of unsupervised methods [Zheng et al., 2018], integrating graph networks [Sanchez-Gonzalez et al., 2018], optimization of the flow of information to and from the network [Hamrick et al., 2017] and, finally, utilization in planning systems [Pascanu et al., 2017], as we will do.

3.3 Neural Network Architectures and Meta-Learning

Designing neural networks is not a straightforward task and requires experience and trial and error when doing it manually. This was recognized early in the research and various genetic algorithms were developed [Miller et al., 1989, Kitano, 1990] to automate the search for the design of a neural network. This process of neural architecture search (NAS) [Elsken et al., 2019] is better understood now, with widely available libraries, such as Auto-Keras [Jin et al., 2018], and also utilizes modern methods of reinforcement learning [Baker et al., 2017].

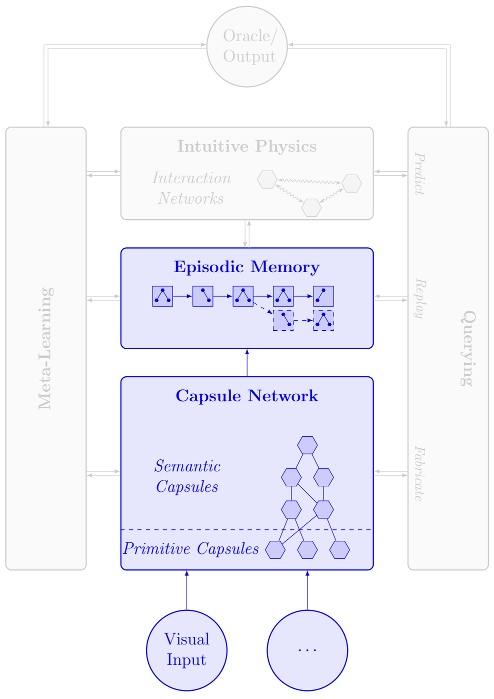
NAS generally produces a static neural network that is trained once and then used, but is not meant to learn anything new thereafter. For safety critical applications [Amodei et al., 2016] having a static is often a requirement, as the architecture was found with a specific dataset and purpose and it is possible that further training might lead to unexpected behavior [McCloskey and J.Cohen, 1989, Ratcliff, 1990]. In many cases transfer learning [Tan et al., 2018] is employed to take a pre-trained model and apply it to a new problem, but such an approach has a limited versatility [Yosinski et al., 2014]. To make a neural network more dynamic, lifelong meta-learning is used to let it learn and reconfigure itself even after it is already in use [Chen and Liu, 2018, Yoon et al., 2018, Parisi et al., 2019].

Lifelong meta-learning has the obvious need for more data, which is often limited. One possible remedy is the augmentation by synthetic data for which various 3D datasets have been proposed [Chang et al., 2015, Gaidon et al., 2016, Mueller et al., 2018, Johnson et al., 2017], each addressing a special use-case. Alternatively, full simulation environments [Shah et al., 2017, Johnson-Roberson et al., 2017] based on computer games can be used. Models pre-trained on such synthetic datasets have been shown to generalize well and reduce the amount of training needed for real environments [Sun and Saenko, 2014, Peng et al., 2015, Tobin et al., 2017, Tremblay et al., 2018], making it an ideal candidate for training set augmentation.

Chapter 4

Neural-Symbolic Capsules

In neuroscience, it is argued that neurons in the cerebral cortex should not be seen as the fundamental computational unit of the brain and one should instead consider groups of about 100 neurons organized as cortical minicolumns [Buxhoeveden and Casanova, 2002]. This shifts the analysis of cognitive functions in the brain to an abstraction layer above that of neurons, where the minicolumns form the nodes of a network-of-networks.



For artificial neural networks the neuron is still seen as the fundamental computational unit and introducing further abstraction layers between the neuron and the final network has mainly been pursued in the field of neural-symbolic reasoning [Garcez et al., 2009, Besold et al., 2017]. Here, we propose such an abstraction layer for vision by introducing special containers for compact machine learning algorithms, such as neural networks, which form the fundamental building blocks of our network-of-networks, loosely based on the idea of capsules proposed by Hinton et al. [2011].

In this chapter, we explore the inner workings of our neural-symbolic capsules and how they interact as part of a capsule network[1]. We begin with a brief review of the classical capsule network, which inspired our neural-symbolic approach and its name, but differs strongly in its actual inner workings, as the original capsule does not act as a container for a neural network. We next introduce a generative grammar, initially assuming that it is hand-designed to procedurally generate images based on a set of attributes and show how the generative process for a single symbol in this grammar can be inverted to form a neural-symbolic capsule. While this is in reverse order of how the final capsule

[1]The results in this and the following chapter were previously published at the German Conference on Pattern Recognition [Kissner and Mayer, 2019b].

network works, where we invert a capsule network to find its generative grammar, it does help motivate both our naming scheme and the design process. Next, we explore how neural-symbolic capsules are connected in a network-of-networks to form a capsule network and discuss its properties. Finally, we show how our capsule network is related to the generative grammar we used as a design guide and acts as its dual representation.

4.1 Classical Capsules

The classical capsule network is an extension of artificial neurons outputting vectors instead of a single scalar. This requires a different internal mechanism, known as the routing-by-agreement protocol [Sabour et al., 2017], to produce the desired output. The overall idea is to multiply the inputs by a weight matrix instead of a weight vector and then use the result to predict what the individual inputs should have been, in order to iteratively improve the output vector.

The output vector can be interpreted as an attribute vector and each capsule can be seen to represent some visual object. Further, as the capsules are equivariant, the information loss between the layers is reduced in comparison to, for example, pooling in CNNs [Krizhevsky et al., 2012, LeCun et al., 1998].

Recently, additional extensions for capsules have been proposed, such as using matrices internally [Hinton et al., 2018], employing 3D input [Zhao et al., 2019], improving equivariance [Lenssen et al., 2018] and even a proposal for hardware acceleration has been made [Marchisio et al., 2019].

The neural-symbolic capsules presented here act more as a container [Hinton, 2014] and diverge somewhat from their original construction, but the overall idea of routing-by-agreement, interpreting the output vectors as attributes and the capsules as objects remains the same.

4.2 Attributed Generative Grammar

Our generative grammar is inspired by [Leyton, 2001, Martinovic and Gool, 2013, Martinovic, 2015]. It procedurally generates an image based on interpretable input information. We require that our grammar is context-free, non-recursive and has only a finite number of symbols in order to avoid infinite productions. In the following, all the individual components of the grammar are explained in detail.

$$G = (S, V, \Sigma, R, A, D) \tag{4.1}$$

S:	the axiom
V:	non-terminal symbols
Σ:	terminal symbols
R:	set of production rules r
A:	set of attributes α
D:	set of decoders g for rule r

The Axiom: Each generative grammar produces a specific type of image described using a symbol, i.e., the axiom. This symbol is a compound noun which sufficiently represents the image's content the grammar produces. For example, to generate an image of a street scene, we construct a grammar with [*street-scene*] as its axiom.

Non-Terminal Symbols: Continuing with the [*street-scene*], in order to draw this symbol, we use a set of objects that represent the parts of the axiom symbol. These parts also describe nouns and we refer to them as non-terminal symbols. For example, a street may contain a [*house*] symbol. Generally, the axiom consists of multiple parts, which we describe by concatenating the individual symbols such as [*house*][*house*][*post-office*]. We also consider the axiom itself as a non-terminal symbol.

Each part may also be made up of parts. We refer to the process of splitting an object into its parts as a production. As all these objects represent real-world entities, this splitting process must be finite.

Terminal Symbols: We refer to the most primitive parts reached during production as the terminal symbols. For our purpose of image generation, the terminal symbols represent renderable primitives, such as a [*square*] or an [*edge*]. As will become apparent later, our goal is a low number of terminal symbols. I.e., we aim to find a small set of renderable primitives from which all non-terminal symbols can be constructed.

Production Rules: For each non-terminal symbol in our grammar, a set of parts is produced as a mixture of non-terminal or terminal symbols. The set of produced symbols is determined by a rule r. We write

$$r : \Omega \to \lambda_1 \cdots \lambda_n \quad \text{where } \Omega \in V, \lambda \in \bigcup_{i \in \mathbb{N} \setminus \{0\}} (V \cup \Sigma) \tag{4.2}$$

to indicate that a non-terminal symbol Ω produces the symbols λ_1 to λ_n. Our [*street-scene*] grammar would, thus, have a rule r_1 that produces [*house*][*house*][*post-office*] from [*street-scene*]. We may continue this process, splitting the [*house*] into its floors, each floor into rooms and furniture. Finally we would split the furniture using a rule such as r_i :[*table*]→[*square*][*triangle*] for the 2D case. Here, we chose [*square*] and [*triangle*] as our graphical primitives, i.e., terminal symbols. By defining a similar rule for the [*post-office*] symbol, we are able to produce all terminal symbols for the entire [*street-scene*] axiom. We refer to the final tree-structure as the parse-tree and an example parse-tree of a [*house*] is shown in Figure 4.1.

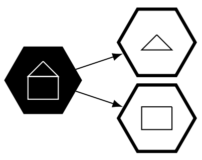


Figure 4.1: Producing terminal symbols from a [*house*] symbol. The black hexagon represents a non-terminal symbol and the white hexagons the terminal symbols.

Attributes: As rendering a [*street-scene*] using the same houses over and over again would be rather pointless, we attach attributes to each symbol to allow for variety. By $\vec{\alpha}_j = (\alpha_j^1, \cdots, \alpha_j^n)$ we denote the vector of attributes for a symbol λ_j and restrict the range of the attribute vector to $[0,1]^n$. These attributes describe the visual appearance of a symbol and are equivalent to attaching *adjectives*, *preposition* and *verbs* to the corresponding noun. As we did with symbols, we introduce the notation $\{\cdots\}$ for the semantic representation of a specific attribute $\alpha_j^{\cdots}$.

The lexical class of each attribute becomes clear by looking at the specific example of an [*office-chair*]. Assume this chair is made out of some metal and is able to corrode. We measure how corroded the chair is on a scale of 0 to 1 and encode this in the *adjective* attribute {*corroded*}. The [*office-chair*] also has some {*position*} attribute, which allows us to set it in relation to other objects in a scene using *prepositions*, such as "on" or "near". Finally, the chair is able to swivel around its base over time, which is encoded in the {*swivel*} *verb* attribute, that defines a relative rotation between the seat and base. Note that {*swivel*} is different from {*rotation*}, as the former describes an internal rotation between two parts, whereas the latter describes the rotation of the chair as a whole.

Decoders: The attributes of a rule $r : \Omega \rightarrow \lambda_j$ are produced by means of decoders g. These decoders take the attributes $\vec{\alpha}_\Omega$ of the left-hand side and generate the attributes $\vec{\alpha}_j$ of the right-hand side, using

$$\alpha_j^i = g_j^i(\vec{\alpha}_\Omega) \quad . \tag{4.3}$$

We also introduce a special type of drawing-decoder, which takes terminal symbols and their attributes and draws them on the screen, essentially rendering the grammar parse-tree. For simplicity, we refer to this drawing function as a decoder g as well, but note that it is not attached to any rule. Here the resulting attributes $\vec{\alpha}_j^i$ represent the pixel color values, as shown in Figure 4.2 for a [*house*] symbol.

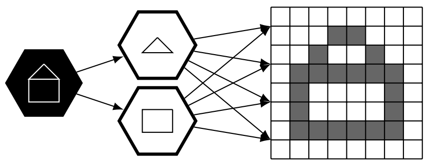


Figure 4.2: Generating an image from a [*house*] symbol.

4.2.1 Style, Pose, Configuration and Viewpoint

A distinct choice of rules and attributes will lead to different effects on the final image. Specifically, we can summarize these as differences in style, pose, configuration or viewpoint.

Style: The most straightforward design choice are the attributes, specifically those interpretable as adjectives, which determine the possible styles the objects in the final image can have. Only a [*house*] with a {*red*} attribute will allow for variety in its redness. Also more complex style-attributes are possibly, such as a grading of how {*rustic*} a [*house*] is.

Pose: Attributes interpretable as verbs have a different effect on the final image. Instead of styles, they describe an object's different poses. Varying a particular attribute in the range $[0, 1]$ is equivalent to an animation, such as walking.

Configuration: A rule produces a fixed set of parts for an object. However, an object should still be classified correctly, even if it has a different configuration of parts. A [*house*] is still a [*house*], even if it produces two [*floor*] symbols instead of one. We, thus, may have multiple rules with the same left-hand side in Equation 4.2 and must choose an appropriate one when producing an image.

Viewpoint: Apart from different configurations, rules with the same left-hand side may also describe different viewpoints. Particularly in 3D, an object viewed from a different angle will have different visible parts, i.e., seem to have a different configuration of parts. These rules are, thus, equivalent to the nodes of an aspect graph.

4.2.2 Rendering and Occlusion

Producing an image using the grammar based on a symbol and corresponding attributes is equivalent to rendering an image in computer graphics. Yet, it is not necessary to begin this process at the axiom. We are free to choose any symbol and begin from there, choosing appropriate rules along the way.

However, there can be some ambiguity in ordering the final renderable primitives according to their depth, when doing it independent of the starting point. To remedy this problem, we define the symbol order in a production rule in accordance with the depth or drawing order, also referred to as painter's algorithm. Thus, [*floor*][*chair*][*table*] would entail first rendering the [*floor*], then the [*chair*] and finally the [*table*].

4.2.3 Completeness

For our discussion it is important that our grammar is complete. By complete we mean that it is able to produce any possible image of a given size $n \times n$ given an adequate set

G (cf. Equation 4.1). We will prove this explicitly.

Consider non-terminal symbols $\Omega_{m,m}$ each with a production rule of the form $r_{\Omega_{m,m}}\colon \Omega \to \lambda_{0,0} \cdots \lambda_{3,3}$, where $\lambda_{i,j}$ are terminal symbols, essentially splitting the $n \times n$ target image into $m \times m$ smaller 3×3 patches. For any image we can craft a constant decoder $g : \cdot \to \mathbb{R}^{3\times 3}$ that simply outputs the appropriate 3×3 patch and all symbols together would output the original image. Essentially g defines a single point in $\mathbb{R}^{3\times 3}$. This can be expanded by letting each decoder g take in one attribute, i.e., defining a line $g : \mathbb{R} \to \mathbb{R}^{3\times 3}$ in 3×3-dimensional space instead of a point. As long as this line goes through the point that was used in the constant case, the symbols are able to detect and reproduce the original image, but also others described by points on the line.

We may continue this construction by adding further attributes and essentially reduce our proof to the following: If our terminal symbol $\lambda_{i,j}$ can produce all possible 3×3 images, then our grammar is complete. This is trivially true, as we can always choose our terminal symbols to do a lossless discrete cosine transform with a maximum of 3×3 discrete attributes, to which we will refer as $[DCT]$. In this extreme case, our grammar is essentially a lossless JPEG-like decoding algorithm.

While this proves completeness, we are, of course, mainly interested in semantically rich cases. We split the domain of the $[DCT]$ symbols to not include all those transforms that look like an edge resulting in two terminal symbols instead, [*DCT-without-edges*] and [*edge*]. Our previous analysis still holds and we may repeat this splitting process as often as we deem sensible without hurting completeness. At some point, the [*DCT-without-edges-patches-...*] symbols will only have a very small domain it is actually used for, only drawing noise-like sections of the image, and all the interesting features are drawn by other terminal symbols with a richer interpretation.

4.3 From Symbols to Capsules

In the following we present our neural-symbolic capsules, each of which acts as a container for a regression model and a routing-by-agreement protocol managing the flow of information from the capsule's input to its output. These capsules are interpreted as distinct visual objects, which not only output the object's attributes, but also an activation probability expressing how confident the capsule is in its detection and use these terms interchangeably.

To motivate our design, we invert the previously defined generative grammar and its produced parse-tree to form a capsule network inspired by [Towell and Shavlik, 1993, 1994]. Each symbol, rule, attribute and decoder of the grammar has a dual element in the capsule network:

Terminal Symbols $\to$ Primitive Capsules: A terminal symbol is a renderable graphical primitive. In the parse-tree, these symbols are connected directly to the drawing function, e.g., shown in Figure 4.2. We refer to its inverse as a **primitive capsule**. These capsules take pixel data as input to determine if the image consists of the graphical primitive it represents and what attributes it has.

Non-Terminal Symbols $\rightarrow$ Semantic Capsules: As for the terminal symbols, non-terminal symbols are inverted to form **semantic capsules**. These capsules are only connected to other capsules and have no link to direct pixel data. Instead, they take other capsule's attributes and confidence values as input to produce their own.

By Ω and λ we will be referring to both the symbol and its inverse, the capsule, depending on context. The same applies to their corresponding attributes $\vec{\alpha}$, which remain unchanged in both representations.

Rules $\rightarrow$ Routes: For a rule $r : \Omega \rightarrow \lambda_i$ a decoder g transforms $\vec{\alpha}_\Omega$ into $\vec{\alpha}_1, \cdots \vec{\alpha}_{|\lambda|}$. We refer to the inversion of the rule r as a **route** and the inversion of the decoder g as the **encoder**. While g generally is not exactly invertible, we introduce γ as the actual encoder and let it be our best approximation, by minimizing

$$||g(\gamma(\vec{\alpha})) - \vec{\alpha}|| \quad . \tag{4.4}$$

The overall inversion process is summarized in Figure 4.3.

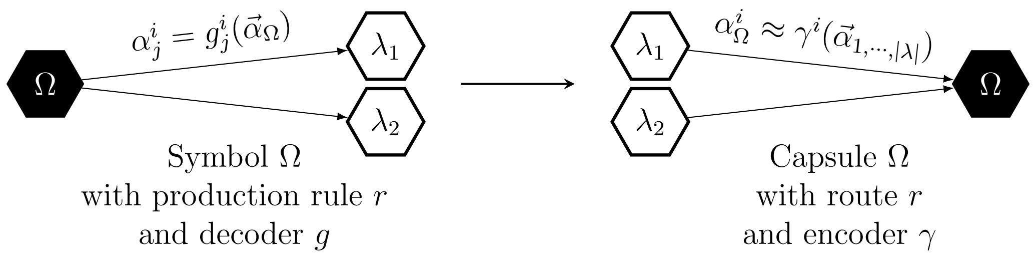


Figure 4.3: Inversion of a symbol of the grammar parse-tree results in a capsule. We illustrate both the symbol and the capsule using a hexagon to avoid confusion with neurons.

By using an individual capsule for each object, we avoid early on some of the entanglement that general object recognition methods face in deep learning [DiCarlo and Cox, 2007, Achille and Soatto, 2018].

4.3.1 Routing-by-Agreement

For the inversion it does not suffice to simply invert the individual elements to obtain a fully functional capsule network. While such a network can accurately process an image that represents the axiom of the underlying generative grammar, even with some generalization, its output will produce nonsense for anything else. We require an additional mechanism to measure the confidence each individual capsule has in detecting the correct object and an algorithm to calculate it. We use a modified routing-by-agreement protocol introduced in [Sabour et al., 2017] to find this activation probability p, which is equivalent to the confidence the capsule has.

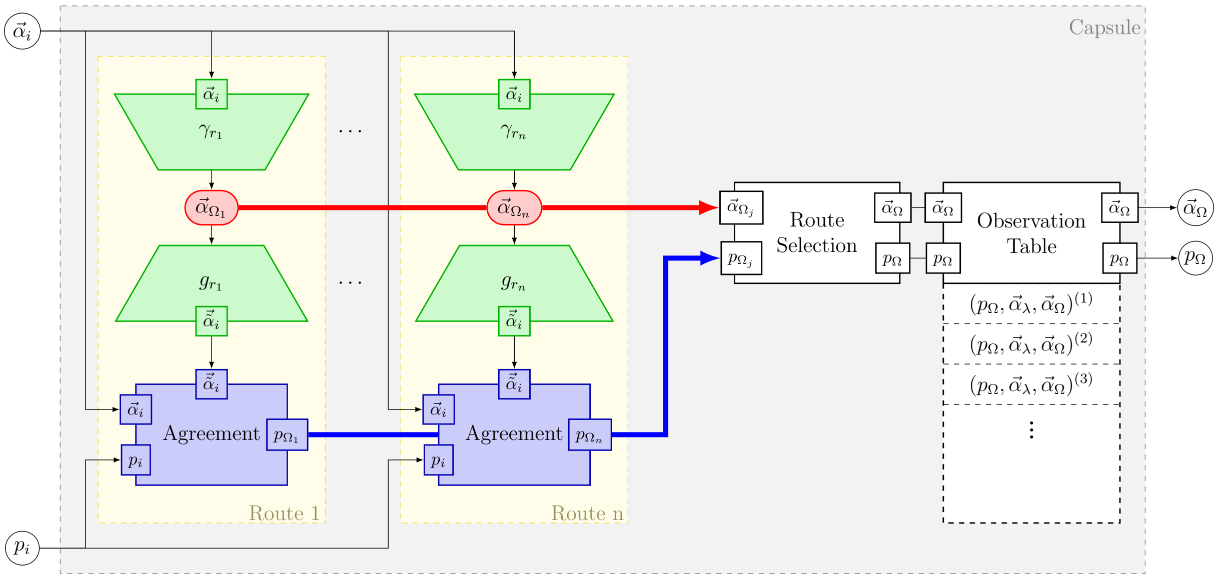

Capsule
$\vec{\alpha}_i$
p_i
γ_{r_1}
γ_{r_n}
$\vec{\alpha}_{\Omega_1}$
$\vec{\alpha}_{\Omega_n}$
g_{r_1}
g_{r_n}
$\vec{\tilde{\alpha}}_i$
Agreement
p_{Ω_1}
p_{Ω_n}
Route 1
Route n
$\vec{\alpha}_{\Omega_j}$
p_{Ω_j}
Route Selection
$\vec{\alpha}_\Omega$
p_Ω
Observation Table
$(p_\Omega, \vec{\alpha}_\lambda, \vec{\alpha}_\Omega)^{(1)}$
$(p_\Omega, \vec{\alpha}_\lambda, \vec{\alpha}_\Omega)^{(2)}$
$(p_\Omega, \vec{\alpha}_\lambda, \vec{\alpha}_\Omega)^{(3)}$

Figure 4.4: The inner structure of a capsule Ω with inputs $\vec{\alpha}_i, p_i$ and outputs $\vec{\alpha}_\Omega, p_\Omega$, representing our routing-by-agreement protocol (Equations 4.5 to 4.9). The outputs are stored in an observation table and individual routes are highlighted in yellow.

In addition, this routing-by-agreement protocol is able to determine which route is the most sensible, i.e., what production rule could have led to the configuration or viewpoint represented in the image. The overall idea behind the algorithm is simple: We take a symbol Ω and use the decoder γ to determine its attributes based on the input capsules or pixels. Using these attributes, we generate what the capsule expected the input should have been with encoder g and compare these results with the actual input. If both input and expected input agree, the capsule has a high activation probability p, otherwise p will be small. This is repeated for all routes of a capsule (i.e., rules with the same left-hand side) to find the route with highest activation, which is taken to be the correct one. In detail, our protocol works as follows (cf. Figure 4.4):

1. The output $\vec{\alpha}_{\Omega_r}$ for a route r is calculated using

$$\vec{\alpha}_{\Omega_r} = \gamma_r(\vec{\alpha}_{1,\cdots,|\lambda|}) \quad . \tag{4.5}$$

2. For each input $\vec{\alpha}_{j_r}$ for a route r, we estimate the expected input value $\vec{\tilde{\alpha}}_{j_r}$ as if $\vec{\alpha}_{j_r}$ were unknown, using the following equation:

$$\vec{\tilde{\alpha}}_j = g_{r,j}(\vec{\alpha}_{\Omega_r}) \quad . \tag{4.6}$$

3. The activation probability p_{Ω_r} of a route r is calculated as

$$p_{\Omega_r} = \frac{1}{|(\lambda)_r|} \sum_{(\lambda)_r} \frac{\|Z\left(\vec{\alpha}_i, \vec{\tilde{\alpha}}_i\right)\|_1}{|Z|} \cdot w\left(\frac{p_i}{\bar{p}_i} - 1\right) \quad , \tag{4.7}$$

where $(\lambda)_r$ denotes the set of all inputs that contribute to a route r, p_i the route's input capsule's activation probability, Z an agreement-function comparing the actual input to the expected input that outputs a vector of size $|Z|$, $\|\cdot\|_1$ the l_1-norm,

w some window function with $w(0) = 1$, $\sup\{w\} = 1$ and $\bar{p}_i$ the mean of all past probabilities the route has calculated using Equation 4.7 for that input.

4. Steps 1. - 3. are repeated for each $r \in R(\Omega)$.

5. Find the route that was most likely used (cf. Figure 4.5)

$$r_{\mathit{final}} = \max_r\{p_{\Omega_r}\} \tag{4.8}$$

and set the final output to

$$p_\Omega = p_{\Omega_{r_{\mathit{final}}}} \tag{4.9}$$
$$\vec{\alpha}_\Omega = \vec{\alpha}_{\Omega_{r_{\mathit{final}}}} \quad . \tag{4.10}$$

Note that $g(\gamma(\vec{\alpha})) = \vec{\tilde{\alpha}}$ is architecturally similar to an autoencoder [Hinton and Salakhutdinov, 2006], hence the names for g and γ. However, unlike an autoencoder, we have a known interpretation for the latent variables (attributes).

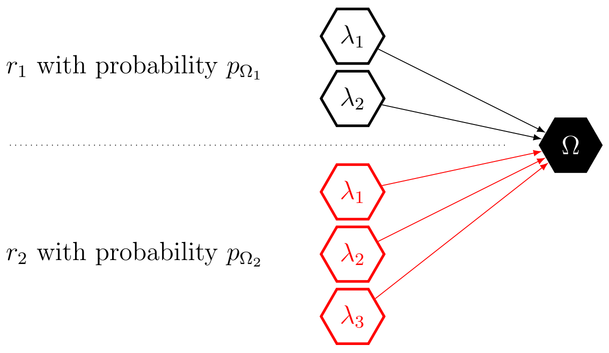

r_1 with probability p_{Ω_1}
λ_1
λ_2
Ω
λ_1
r_2 with probability p_{Ω_2}
λ_2
λ_3

Figure 4.5: Actual routes taken by routing-by-agreement. Here, $p_{\Omega_2} > p_{\Omega_1}$, thus, the output of r_2 is used for the capsule's attributes $\vec{\alpha}_\Omega$.

4.3.2 Agreement Function

In the following we discuss the agreement function found in Equation 4.7 in detail, which plays an integral role in both the capsule network as well as the meta-learning pipeline. Its main purpose is to give a measure of similarity between the input attributes of the encoder γ and the output attributes of the decoder g, but on a semantic level. In this sense it is closely related to the reconstruction loss used in auto-encoders.

Similarity of semantics implies the ability to be invariant to symmetries. A perfect square is indistinguishable from one that is rotated by $\frac{\pi}{2}$. In this case, a general L^2 loss would find a discrepancy numerically between attributes of two such squares, where there should be none semantically. Instead, we introduce a set $R_{\vec{\alpha}}$ of rotationally equivalent attributes $\vec{\alpha}$, which allows to propose an agreement function for semantic capsules:

$$Z\left(\vec{\alpha}_a, \vec{\alpha}_b\right) = \max\{w\left(\vec{\alpha}_b - \vec{\tilde{\alpha}}_a\right) \ : \ \vec{\tilde{\alpha}}_a \in R_{\vec{\alpha}_a}\} \quad , \tag{4.11}$$

where $w\colon \mathbb{R}^n \to \mathbb{R}^n$ describes a separate window function for each individual attribute in the vector.

The symmetries $R_{\vec{\alpha}}$ are found by explicitly by iterating over the possible n-fold rotational symmetries of $\frac{2\pi}{n}$. We choose some random set of attributes $\vec{\alpha}$, from which we produce two identical sets $\vec{\alpha}_a$ and $\vec{\alpha}_b$, but one with rotation $\vec{\alpha}_a^{\text{rot}} = 0$ and the other with $\vec{\alpha}_b^{\text{rot}} = \frac{2\pi}{n}$. The capsule is said to have a rotational symmetry, if

$$Z\left(g(\vec{\alpha}_a), g(\vec{\alpha}_b)\right) > \epsilon \quad , \tag{4.12}$$

holds true, where ϵ is some threshold and we use the agreement functions of the capsule's parts. In essence, we check if the attributes produce parts that are symmetric themselves by $\frac{2\pi}{n}$, which in turn check if their parts are symmetric all the way down the network until we reach the primitive capsules. In the case of our two squares, this would entail rendering them both and checking if the resulting images are similar.

For primitive capsules, finding an appropriate Z is dependent on the decoder g, as we are dealing with pixel data. However, as g is known for these capsules, we can incorporate the symmetries explicitly as well as invariance to semantically irrelevant lighting effects and noise.

With the primitive capsules' agreement function known, we recursively build up all the agreement functions of higher level capsules using Equation 4.12 for thresholding the equivalence class $[\alpha^i]$ of symmetries. This allows to generalize the agreement function of Equation 4.11 to all possible symmetries, for example a repetitive cloth texture with translational symmetries, by using

$$Z\left(\vec{\alpha}_a, \vec{\alpha}_b\right) = \max\{w\left(\vec{\alpha}_b - \vec{\hat{\alpha}}_a\right) \ : \ \vec{\hat{\alpha}}_a \in [\vec{\alpha}_a]\} \quad . \tag{4.13}$$

We further discuss finding equivalence classes in the following chapter.

4.3.3 Completeness

As for the grammar, we must show that our capsule network is complete as well. By completeness we mean that the neural-symbolic capsule network must have the ability to detect every meaningful feature in the image and be able to give semantic meaning to any set of such features.

We again show this explicitly by following the proof of our grammar's completeness in reverse. The discrete cosine transform is an invertible function, thus a $[DCT]$ primitive capsule is able to decode any feature found in the image into a set of coefficients. To add some meaning, we again split the $[DCT]$ capsule into a $[\textit{DCT-without-edges-patches-}\dots]$ primitive capsule and $[\textit{edge}][\textit{patch}]\dots$ primitive capsules. We, thus, construct feature detectors for edges, patches, etc., but also those we did not expect, such as noise, covering the entire spectrum of possibilities.

The feature detectors can be recombined as objects with a semantic meaning in a semantic capsule. Consider an encoder $\gamma\colon \mathbb{R}^m \to \mathbb{R}$, where m is the total number of attributes from the feature detectors, that projects the whole $\mathbb{R}^m$ onto a line that passes through a point p in $\mathbb{R}^m$. The point p is defined by the configuration of features that were extracted. A semantic capsule with such an encoder γ will detect this exact set of features as an object and even have a single attribute that allows it to generalize along the aforementioned line of configurations. We may always add a capsule, if we encounter a new object, enabling the capsule network to learn to detect every possible object and proving our requirement for completeness.

In the following chapter on meta-learning, we will explore in detail how this process can be performed in a more elegant way.

4.3.4 Observation Table

Our final capsule network differs from other feed-forward networks by lacking a specific output layer. Instead, we refer to the set of capsules that have the same distance from the input as a hierarchy level, to make the difference to layers clear. Every capsule in the network acts as a network output and describes important semantics, even if it would be considered a hidden node in the sense of traditional neural networks. We use the notion of hierarchy level merely for orientation when talking about a network.

We also extend our capsule design by another functionality. As the first level of primitive capsules uses pixels as input attributes and this set may have a smaller size than the actual input image, we must handle scanning the entire image in a sensible way, i.e., sliding filters. To this end, each capsule in the network is amended with a list of objects with valid attributes it has detected, the observation table (cf. Figure 4.4). As the sliding filter moves across the image, valid detections are added. An entry is admitted, if its activation probability p_Ω is larger than a threshold ρ. This observation table is not persistent and the objects are stored in the table only per image.

After detection by sliding filters, only the capsules in the first hierarchy level have entries in their observation tables. To perform a forward pass, we move up level-by-level and try out every permutation of input for the subsequent capsules, including setting them to 0-capsules with vanishing activation. Adding 0-capsules is equivalent to occlusion and, thus, to missing input. Every permutation that leads to a sufficiently large activation probability exceeding threshold ρ adds an entry to the observation table. Once every permutation was tested for each capsule in the current hierarchy level, the process continues until the last level. This avoids the need for duplicates of the same capsule in the capsule network.

We refer to the set of observations of a capsule as the set Λ_Ω. As these tables are reset at the beginning of each forward pass, we assume that all Λ_Ω of past observation tables are stored in more persistent memory as

$$\left((p_\lambda)^{(i)}, (\vec{\alpha}_\lambda)^{(i)}, (\vec{\alpha}_\Omega)^{(i)}\right)_r \quad . \tag{4.14}$$

This allows to calculate the mean value $\bar{p}_\Omega$ required by our routing-by-agreement protocol in Equation 4.7.

During a forward pass, the subset of activated capsules and routes, i.e., the entries in the observation tables, form one or multiple parse-trees. The topmost activated capsule's symbol, however, is not necessarily an axiom of the capsule network's inverse grammar (cf. Figure 4.6). This is especially true if multiple parse-trees with distinct root nodes can be formed from the observation tables, as a grammar only produces one. To differentiate them, we refer to these parse-trees as observed parse-trees Λ, which incorporate all capsule observations Λ_Ω. Each observed parse-tree also induces its own grammar with the root symbol being its axiom and we refer to these as the observed grammar and observed axiom.

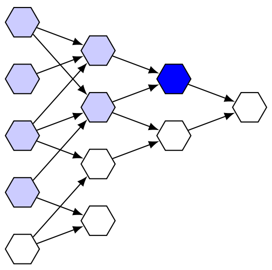


Figure 4.6: A capsule network with an observed parse-tree formed by the activated capsules (blue). The top most activated capsule (dark blue) acts as the observed axiom of the observed grammar.

4.3.5 Episodic Memory and Video

Verb attributes encode a sequence of poses on a time-frame of 0 to 1. Each pose in this sequence has to be seen in context of the previous, as a single pose may repeat itself in a sequence or in another verb attribute. In order to do this, we must extend the formulation of the capsules from detection based on single images to video data.

In order to make sense of video data, the capsules require the ability to track objects across frames, to be able to represent concepts such as continuity and permanence of objects. We expand the observation table Λ_Ω of a capsule Ω to include observations at various time-steps t, which we denote as $\Lambda_{\Omega,t}$, and store these past observations in Episodic Memory (cf. Figure 4.7), as part of the observed parse-tree Λ_t.

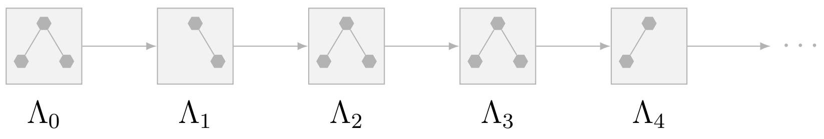


Figure 4.7: Schematic look at Episodic Memory. All past observed parse-trees Λ_t are chronologically ordered.

We denote the attributes of the ith observed object in Λ_Ω as $(\vec{\alpha}_\Omega)_t^{(i)}$, in accordance with Equation 4.14. If two observations, $(\vec{\alpha}_\Omega)_{t-1}^{(i)}$ at time $t-1$ and $(\vec{\alpha}_\Omega)_t^{(j)}$ at time t, describe the same object, i.e., it is tracked across the two frames, we denote this relation by (i, j). If an object is not tracked from the previous frame, or has entered the frame, we denote this as $(i, \cdot)$ and $(\cdot, j)$ respectively. All relations of a specific object class form the set $P_{\Omega,t}$.

For now, we assume that the camera moves smoothly and the time between frames Δt is small. This allows to describe an object's position and orientation in relation to the last frame based on small changes of the camera. We encode these changes in a camera rotation matrix $\mathbf{R}_t$ and a translation vector $\vec{x}_t$ in relation to the last frame. Both, the angles and the distance described by these quantities are small.

Further, we are able to predict the movement of objects based on their past locations with sufficient accuracy, by using a finite Taylor expansion of the form:

$$(\vec{\alpha}_\Omega)_t^{(i)} \approx (\vec{\alpha}_\Omega)_{t-1}^{(i)} + \frac{\delta(\vec{\alpha}_\Omega)_{t-1}^{(i)}}{\delta t}\Delta t + \frac{1}{2}\frac{\delta^2(\vec{\alpha}_\Omega)_{t-1}^{(i)}}{\delta t^2}\Delta t^2 + \cdots \quad . \tag{4.15}$$

However, as we use the camera as the frame of reference, we need to transform it from the previous frame of reference

$$(\vec{\alpha}_\Omega)_t^{(i)} \to \mathbf{R}_t \cdot (\vec{\alpha}_\Omega)_t^{(i)} + \vec{x}_t \quad , \tag{4.16}$$

where we let $\mathbf{R}_t$ and $\vec{x}_t$ only act on the position and rotation attributes.

Finally, we can formulate our tracking problem. By $P_{\Omega,t}$ we denoted the set of true relations between objects in frame $t-1$ and t. Assuming that $P_{\Omega,t}$ is unknown, let $\tilde{P}_{\Omega,t}^k \in \mathcal{P}_\Omega$ denote a set of unique relations, i.e., each index may only appear on the left-hand-side and right-hand-side once, including those of the type $(i,\cdot)$, $(\cdot,j)$ and let $\mathcal{P}_\Omega$ be the set of all such possible sets. Note that the true relations $P_{\Omega,t}$ are also in $\mathcal{P}_\Omega$, which we approximate using the first order expansion of Equation 4.15 combined with Equation 4.16:

$$P_{\Omega,t} = \min_{\tilde{P}_{\Omega,t}^k \in \mathcal{P}_\Omega, \mathbf{R}_t, \vec{x}_t} \left\{ \sum_{(i,j)\in \tilde{P}_{\Omega,t}^k} \left\| \vec{w} \cdot \left((\vec{\alpha}_\Omega)_{t-1}^{(i)} + \frac{\delta(\vec{\alpha}_\Omega)_{t-1}^{(i)}}{\delta t}\Delta t - \left[\mathbf{R}_t \cdot (\vec{\alpha}_\Omega)_t^{(j)} + \vec{x}_t \right] \right) \right\| \right\} \quad , \tag{4.17}$$

where $\vec{w}$ assigns a weight to each attribute in accordance with its importance for tracking and we again let $\mathbf{R}_t$ and $\vec{x}_t$ only act on the position and rotation attributes. For example, adjective attributes are less likely to change across frames and, thus, provide a good indicator for tracking, whereas verb attributes are prone to change.

By limiting ourselves to the first order expansion, we introduce an error of the order $(\Delta t)^2$. Thus, reducing the time-steps between frames does not only reduce the movement of the camera and the effect of Equation 4.16, but also the error of the prediction of the object's movement. Minimizing Equation 4.17 is, thus, dominated by checking all possible combinations and of complexity $\mathcal{O}(n!)$. However, each search is limited to a single object class with $n_\Omega = |\Lambda_\Omega|$ entries, considerably reducing the search space.

Particularly, we use the fact that each search is limited to n_Ω and begin tracking objects that are are in the scene, by sorting the classes $n_{\Omega_1} < n_{\Omega_2} < \cdots$. This allows to get an early estimate for the camera movement ($\mathbf{R}_t$ and $\vec{x}_t$), which we refine as we track more frequent objects to combat the problem of crowding. Once the complexity of $\mathcal{O}(n!)$ becomes an issue, we may use our refined knowledge of the camera movement and further reduce the complexity to $\mathcal{O}(n^2)$, by performing an initial prediction

$$(\tilde{\vec{\alpha}}_\Omega)_t = \mathbf{R}_t^{-1} \cdot \left[(\vec{\alpha}_\Omega)_{t-1}^{(i)} + \frac{\delta(\vec{\alpha}_\Omega)_{t-1}^{(i)}}{\delta t}\Delta t - \vec{x}_t \right] \tag{4.18}$$

and, assuming that the tracked object must be close, search individual relations directly using

$$\|(\vec{\alpha}_\Omega)_t^{(j)} - (\tilde{\vec{\alpha}}_\Omega)_t\| < \epsilon \cdot \Delta t^2 \quad . \tag{4.19}$$

Intuitive physics introduced later in chapter 6 allows additionally to incorporate physical predictions into Equation 4.17. This improves tracking objects that may be occluded for a short while and reappear, i.e., object permanence. With all this information available for each observation, each entry in episodic memory has a form similar to the example shown in Figure 4.8, promoting the parse-tree to a scene-graph.

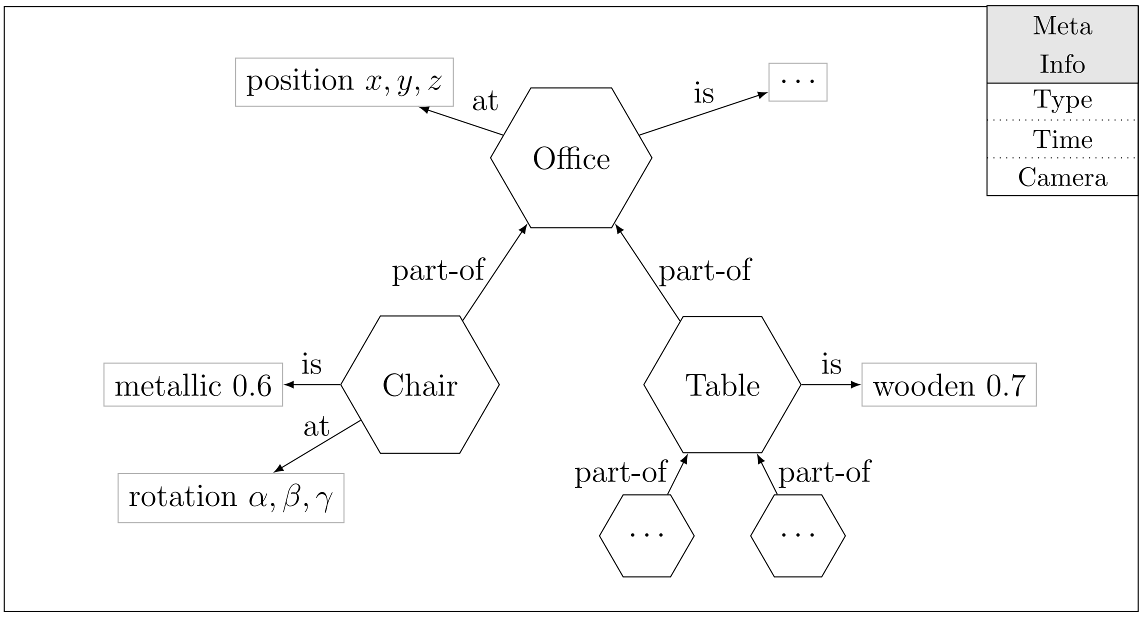


Figure 4.8: Example scene graph for an office scene extracted from capsule observation tables and meta-information.

4.3.6 Graphics Engine

The conversion of a grammar into a capsule network works both ways. We may always invert the network back into a generative grammar to produce an image and vice versa. In this sense, both representations are dual to each other. However, there can only be one grammar axiom, but as discussed previously, each capsule itself may be an observed axiom. We must, thus, choose one as the starting point, the compound noun that describes the scene we want to produce.

To produce the image, we follow the production rules, i.e., the inverted routes. A rule requires attributes, which must be provided alongside the chosen axiom and a decoder g, which we obtain directly from the route as shown in Figure 4.9. The starting axiom with these attributes is a compound noun and a collection of verbs and adjectives. It can also be formulated as a full description of the final image. As an example, a [*car*] with high values for {*rustic*} and {*red*} may be translated into a simple description, such as ”a rustic red car”.

Capsules, however, may have multiple routes, each expressing a different viewpoint, style or configuration. Thus, during production, a choice must be made between the valid rules to determine which to use. This can be either the original image's selection stored in memory or a new one. This choice alters the final image, but not its semantic content as laid out by the chosen axiom and its attributes. Which rules/routes are valid is explored in more detail in the next chapter.

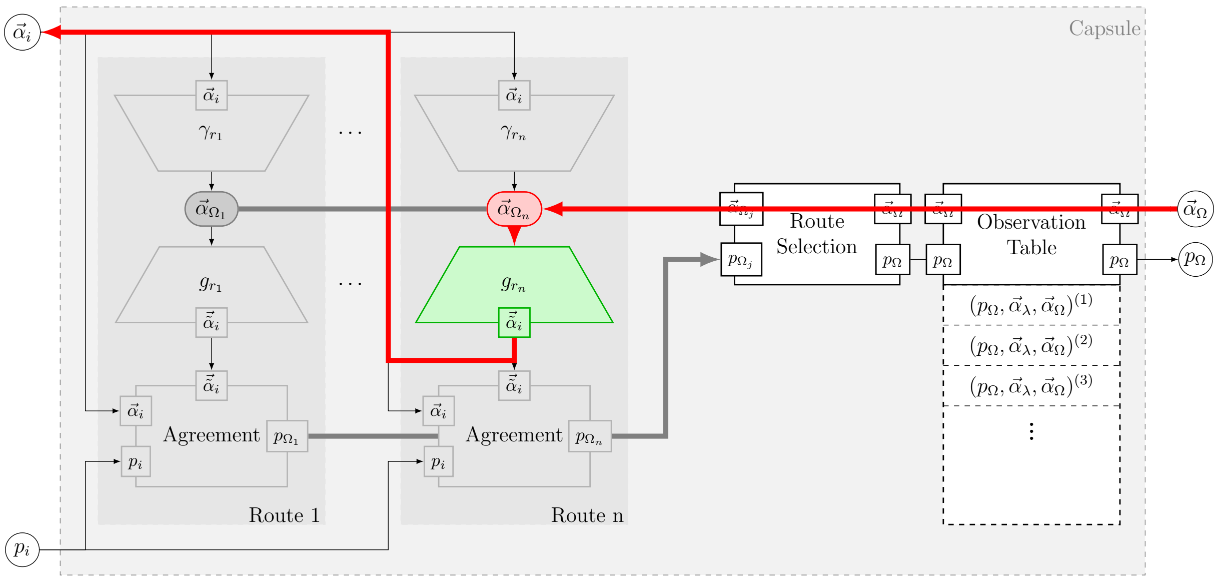

Capsule
$\vec{\alpha}_i$
γ_{r_1}
γ_{r_n}
$\vec{\alpha}_{\Omega_1}$
$\vec{\alpha}_{\Omega_n}$
g_{r_1}
g_{r_n}
$\tilde{\vec{\alpha}}_i$
Agreement
p_{Ω_1}
p_{Ω_n}
p_i
Route 1
Route n
Route Selection
$\vec{\alpha}_{\Omega_j}$
p_{Ω_j}
$\vec{\alpha}_\Omega$
p_Ω
Observation Table
$(p_\Omega, \vec{\alpha}_\lambda, \vec{\alpha}_\Omega)^{(1)}$
$(p_\Omega, \vec{\alpha}_\lambda, \vec{\alpha}_\Omega)^{(2)}$
$(p_\Omega, \vec{\alpha}_\lambda, \vec{\alpha}_\Omega)^{(3)}$

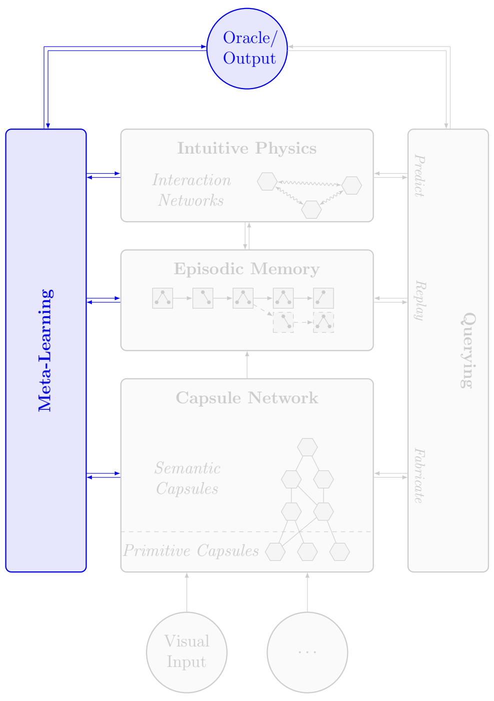
Figure 4.9: Reversing the direction of a capsule to generate instead to detect. The feed-backward process is highlighted in red.

Once the generation process of the grammar has reached the terminal symbols, the final image is rendered using the decoders of the primitive capsules. However, for rendering, depth ordering is required. This is provided by the mean activation probability $\bar{p}_\Omega$ stored for each symbol, which was determined according to the objects' visibility in feed-forward and now provides depth information for feed-backward operation.

Finally, by repeating this rendering process over multiple time-steps and smoothly varying the verb attributes of the axiom accordingly, video output is produced. By encoding a scene into a symbolic representation with the ability to render it later, we are essentially compressing the visual data for video playback.

Chapter 5

Training and Meta-Learning



Statically defining an entire neural-symbolic capsule network from scratch is not feasible, as it would require encoding a lot of information beforehand and would not take advantage of the benefits of learning. Instead, we propose a meta-learning algorithm that designs the architecture of the network automatically based on unknown image sequences it is exposed to and introduce an accompanying training algorithm to quickly learn from new environments over its entire lifespan.

The general idea and motivation for our lifelong meta-learning algorithm [Chen and Liu, 2018] is based on that our capsule network works as an inverted grammar and as such should also only have a single observed axiom that describes the scene. We postulate having a single observed axiom as the overall goal of the meta-learning pipeline and deduce further training rules, which we show to be sufficient to keep improving the capsule network indefinitely with each new scene it encounters.

5.1 Choosing and Training Primitive Capsules

Our capsule network follows the philosophical idea of innatism, i.e., it does not start out as a blank slate (or tabula rasa), but rather has a preexisting idea of the world and its structure. Specifically, we assume that all primitive capsules are predefined.

We will ignore the discrete cosine transform capsules introduced in our completeness proof for our analysis here and focus on those that extract specific features. Without a $[DCT]$ capsule, the initial set of primitive capsules limits what the network can actually analyze and learn. For example, if we only define $[\mathit{circle}]$ primitive capsules, the network will have a hard time to make sense of objects made up of squares (it would assume squares are made out of multiple 1×1-sized circles, which is highly inefficient). Ideally, we would define primitive capsules for the most basic set of primitives from which we are able to construct every kind of object. Such a set might include different types of surface-patches and edges, from which we then can infer more complex structures on subsequent hierarchy levels. Such an approach requires a lot of time to learn, as the network would need to understand, for example, that multiple patches bounded by edges form a surface, multiple surfaces define a square and multiple squares form a table.

Alternatively, we may begin with more complex primitives from which we can infer objects using fewer hierarchy levels. For example, if we directly begin with a $[\mathit{square}]$, the following level may already contain a semantic capsule describing a $[\mathit{table}]$ without the need for all the intermediary capsules. This is adequate for an environment full of squares, such as a tetris-like game. By adding a $[\mathit{triangle}]$ capsule, it would then be possible to detect further compound shapes, such as a simple $[\mathit{house}]$. Should the environment become so complex (general polygons, organic shapes, . . .) that the initial set of geometric shapes isn't sufficient, we can remedy this by either adding more primitive capsules or moving down one level of abstraction, promoting the previous primitives to semantic capsules and introducing new symbols, such as an $[\mathit{edge}]$. This is possible through the modularity of the network and apart from those modified, no other capsule needs to be retrained.

While we do not restrict the complexity of the primitive, it should be noted that there is a strong relation between the complexity of a renderable primitive and the design of γ and g. γ is only bounded by current regression learning methods and particularly n-dimensional pose estimation. For g, we rely on the current state of computer graphics. Here we have access to a near endless supply of 2D and 3D art assets [Quílez, 2017, Zhou et al., 2018] and physically-based rendering pipelines [Pharr et al., 2016]. Alternatively, we could also make use of previously trained CNNs, as they often have a latent representation for primitives in the early layers. However, it has proven difficult to assign a specific attribute to each such latent variable due to entanglement.

Once a primitive has been chosen for g to render, we are able to train γ. Let $\chi_{i,j} \sim U([0,1])$ be a uniform random variable for the jth attribute and $f(\cdot)$ describe the application of random backgrounds, occlusions and special effects to an image. We train γ using the synthetic dataset

$$\left((f(g(\chi_{i,j})))^{(i)}, (\chi_{i,j})^{(i)}\right) \quad . \tag{5.1}$$

To speed up the training, we may also use quantile functions Q_j for each attribute that better represent their actual distribution. However, this is neither required nor always possible and we set them to $Q_j(p) = p$ in this case. Using these quantile functions, γ is trained with the set

$$\left((f(g(Q_j(\chi_{i,j}))))^{(i)}, (Q_j(\chi_{i,j}))^{(i)}\right) \quad . \tag{5.2}$$

5.2 Training Semantic Capsules

Before we describe our meta-learning pipeline for semantic capsules, we explore some mathematical details. Each semantic capsule takes a configuration of its parts as input and outputs the corresponding object's attributes. Both, the input configuration of the parts as well as the object's attributes lie in a configuration space, also referred to as the latent space. However, as the exact structure of this space is unknown, our goal is to fit the capsule such that it best approximates this configuration space. This is not without ambiguity, but by exploring the possible topologies, we derive methods to perform the best approximation and introduce our meta-learning pipeline.

5.2.1 Configuration Spaces

The configuration space of the input attributes as well as the output attributes of a semantic capsule describe the plausible configurations of an object and this is often referred to as the manifold hypothesis. However, in our case, this name is misleading, as the underlying space is not always a manifold and rather a more general metric space. Here, we explore the topological structure of these spaces and the consequences for our meta-learning pipeline they entail.

We denote the configuration space of the inputs as A_λ and the configuration space of the outputs as A_Ω. They are embedded in some larger spaces $A_\lambda \subseteq M_\lambda$ and $A_\Omega \subseteq M_\Omega$, which describe all possible attributes, but may contain un-plausible configurations that lie outside of A_λ and A_Ω. Using this notation, the capsule's decoder is a map $g : M_\Omega \to M_\lambda$. The image of this decoder describes an approximated input configuration space $g(M_\Omega) = \tilde{A}_\lambda$, with $\tilde{A}_\lambda \subseteq M_\lambda$. Thus, ideally we would find that $\tilde{A}_\lambda = A_\lambda$ by minimizing Equation 4.4 and introduce a distance measure for attributes

$$d(\vec{\alpha}_{\lambda,k}) = ||g(\gamma(\vec{\alpha}_{\lambda,k})) - \vec{\alpha}_{\lambda,k}|| \quad , \tag{5.3}$$

of how far off we are of the correct configuration.

This also allows us to find an upper bound $d_{\tilde{A}}$ for the distance between the approximated configuration space $\tilde{A}_\lambda$ and any point in the true configuration space A_λ as

$$d_{\max} = \sup_{\vec{\alpha} \in A_\lambda} d(\vec{\alpha}) \quad . \tag{5.4}$$

5.2.2 Topology of M_Ω and M_λ

In the simplest case, there are two types of attributes, those that lie in the range of $[0, 1]$ (such as position) and those that lie in the range of $[0, 1]$ but are circular (such as rotation). The latter can be seen as 0 and 1 glued together to form $[0, 1]/\{0, 1\}$, which is homeomorphic to the unit circle S^1.

For this simple case, the spaces for all input and output attributes take on the following form:

$$M_\Omega = \prod_i [0,1] \times \prod_j S^1 \quad , \tag{5.5}$$

$$M_\lambda = \prod_\lambda \left[\prod_{i_\lambda} [0,1] \times \prod_{j_\lambda} S^1 \right] \quad . \tag{5.6}$$

Using this construction of M_Ω, we begin our analysis by looking at pairs of attributes $\alpha^i_\Omega, \alpha^j_\Omega$. The three possible combinations of $[0, 1]$ and S^1 result in either a closed surface, a cylinder section or a torus (cf. Figure 5.1).

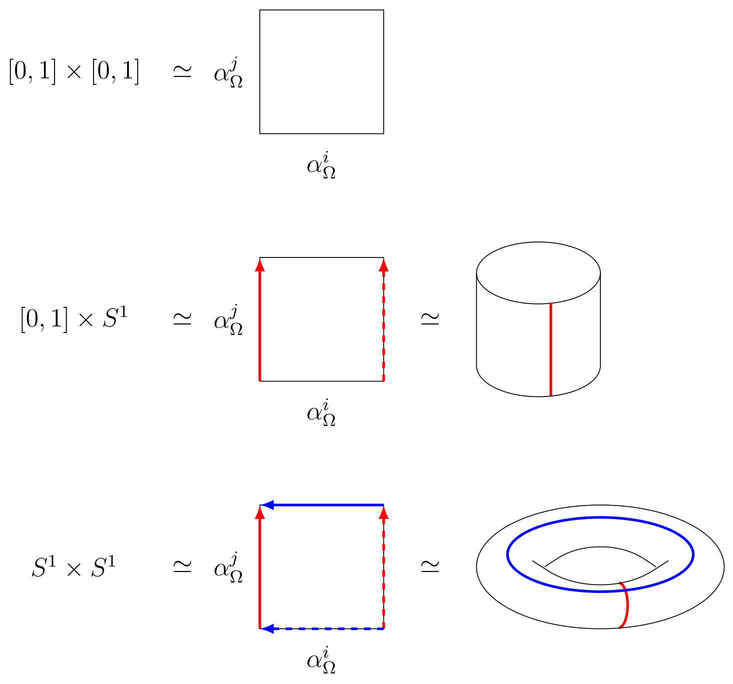


Figure 5.1: The three different combinations of $[0, 1]$ and S^1, a closed surface, a cylinder section and a torus. Filled lines are identified with the dashed line of the same color and their equivalent cut is highlighted on the quotient space. Arrows depict orientation.

To check the disentanglement of α^i_Ω and α^j_Ω with respect to a group G, we must show that there exists some $G = G_1 \times G_2$ under which they transform independently of each other. We take G to be translations in M_Ω and make the assumption that G_1 translates α^i_Ω and G_2 translates α^j_Ω independently. These translations can be performed explicitly on our gluing instructions in Figure 5.2, to check whether our assumed behavior of G_1 and G_2 is indeed correct and that then is disentanglement.

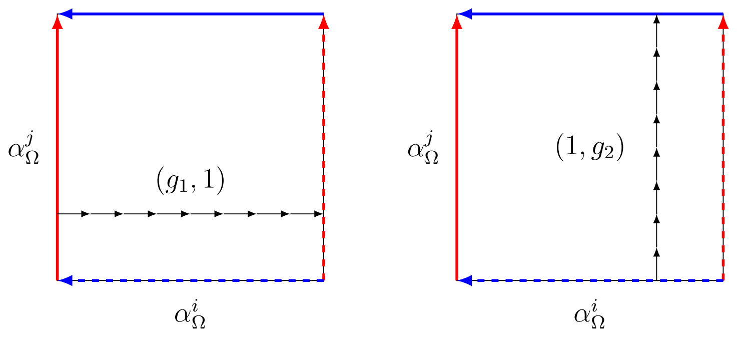


Figure 5.2: Visualization of disentanglement of translations, where G is decomposable as $G_1 \times G_2$. $(g_1, 1) \in G_1 \times G_2$ only acts on α^i_Ω (left side) and $(1, g_2) \in G_1 \times G_2$ only act on α^j_Ω (right).

Repeating this check for all attributes in M_Ω gives us a fully disentangled set of attributes. However, in general, M_Ω is not just a direct product of individual attributes. We continue our analysis by looking at pairs of attributes α^i_Ω and α^j_Ω with more complex gluing instructions of the general form $[0,1]^2/\sim$. Two such examples are given in Figure 5.3, the sphere and the torus with genus > 1.

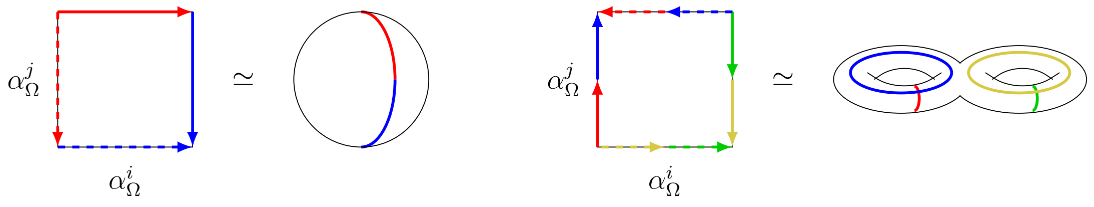


Figure 5.3: Gluing instructions on $\alpha^i_\Omega \times \alpha^j_\Omega = [0,1] \times [0,1]$ and their homeomorphic surfaces. The sphere and a torus with genus 2 are shown as examples. Filled lines are identified with the dashed line of the same color and their equivalent cut is highlighted on the resulting quotient space. Arrows depict orientation.

Our assumption that G can be decomposed into G_1 and G_2 breaks and we quickly find that these transformations no longer act independently on α^i_Ω and α^j_Ω, as shown in Figure 5.4 for the sphere.

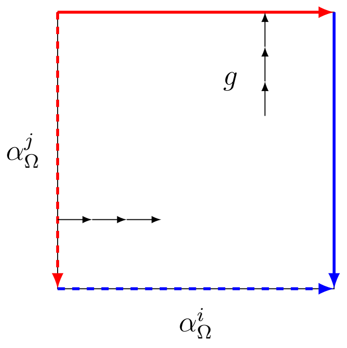


Figure 5.4: Visualization of entanglement, where G can't be decomposed and an element $g \in G$ may act on α^i_Ω, as well as α^j_Ω.

The failure to decompose G shows that α_Ω^i and α_Ω^j are indeed entangled. Using this explicit method, the analysis can be extended to all sets of attributes $[0,1]^l/\sim$. We write M_Ω in the more general form

$$M_\Omega = [0,1]^{n_1} \times \prod_{k=2}^{n} \left[\prod_{l=1}^{n_k} \left([0,1]^l/\sim_{k,l} \right) \right] \quad , \tag{5.7}$$

where the total number of attributes is $n = \sum_i n_i$. Now, if the equivalence relations are known exactly, we know which sets of attributes are entangled and which are not.

However, usually the exact topological structure of M_Ω as in Equation 5.7 is not known. Instead, we must probe it to find its structure. As done visually earlier, we examine curves mapped from $\tilde{A}_\lambda$ to M_Ω and, particularly, their behavior at the most likely identification regions on the boundary of M_Ω, i.e., the values 0 and 1 on each interval. Let $f : [0,1]^n \to M_\Omega$ describe the quotient map for M_Ω, which we are looking for. Here $[0,1]^n$ can be seen as the raw attribute vector $\vec{\alpha}_\Omega$ of the capsule, without any topological considerations. We choose some point on the boundary $\partial([0,1]^n)$ and create a curve with, ideally, constant velocity $c_p : [0,2] \to M_\Omega$, so that $f^{-1}(c_p(0)) \notin f(\partial([0,1]^n))$ and $f^{-1}(c_p(1)) = p$. This c_p describes a curve that crosses the region of identification $\partial([0,1]^n)$.

Next, we find the two points $p_1 = \lim_{t\to 0} f^{-1}(c_p(1-t))$ and $p_2 = \lim_{t\to 0} f^{-1}(c_p(1+t))$. Note that $p_1 \neq p_2$, but $p_1 = p$. The two points form an equivalence relation $p_1 \sim p_2$, since $g(p_1) = g(p_2)$. If we repeat this analysis for every point on $f(\partial([0,1]^n)) \setminus \partial M_\Omega$, we can reconstruct all the equivalence relations inherent in our quotient space.

However, we must identify p_2 in a practical setting, as f is unknown. We know c_p on the interval $[0,1]$. To find c_p on $(1,2]$, we push it forward onto M_λ as $\tilde{c}_p = g(c_p)$, where we know how the curve evolves on $\tilde{c}_p([0,1]) = g(c_p([0,1]))$. We then use the velocity and acceleration vector of the curve to advance it slightly forward on M_λ:

$$p_2 = \lim_{\Delta t \to 0} \gamma \left(\tilde{c}_p(1) + \frac{\partial \tilde{c}_p}{\partial t}(1) \cdot \Delta t + \frac{1}{2} \frac{\partial^2 \tilde{c}_p}{\partial t^2}(1) \cdot \Delta t^2 + \cdots \right) \quad . \tag{5.8}$$

This process is shown in Figure 5.5.

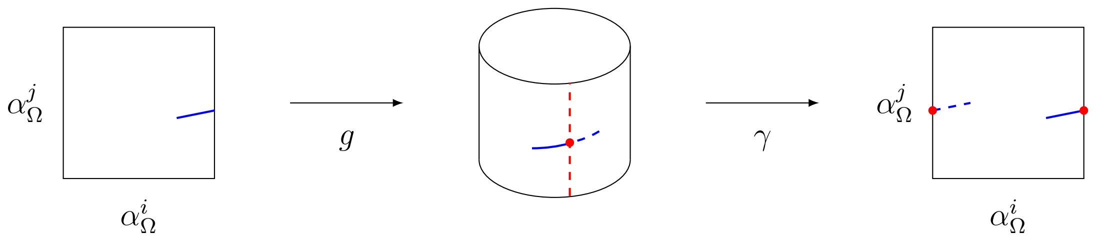


Figure 5.5: Curve crossing the boundary. $c_p([0,1])$ is depicted as a blue line and $c_p((1,2])$ as a dashed blue line. The red points describe p and the equivalent points p_1 and p_2.

The above examination gives us a better understanding of the equivalence relations, orientation and entanglement on the boundary of M_Ω and ideally this would be sufficient. However, this is not always the case. For example, consider again the rotation attribute of a 2D [*square*]. We saw that such a square is invariant to rotations of $\frac{\pi}{2}$, i.e., it loops back around before reaching the boundary. This means that it is perfectly possible to have equivalence relations inside the domain.

To find such equivalence relations for the interior of $(0, 1)$, we are inclined to follow the same procedure as before and construct and probe multiple geodesics. In practice, we reserve this expensive method using curves for completely unknown cases and perform simpler checks on those that are known, such as the progression of angles $\beta_n = \frac{\pi}{n}$, which we test explicitly for the rotation attribute as $(\pi, \frac{\pi}{2}, \frac{\pi}{3}, \cdots)$.

5.2.3 Covering the Input Configuration Space

With our construction of M_Ω in Equation 5.7 and the map $g : M_\Omega \to \tilde{A}_\lambda$, we have limited the possible configuration spaces $\tilde{A}_\lambda$ we are actually able to construct. Our goal is to find an $\tilde{A}_\lambda$ that resembles the true configuration space A_λ as closely as possible. However, we have two potential topological problems that need to be addressed:

1. $\tilde{A}_\lambda$ is always connected and A_λ might not be.
2. $\tilde{A}_\lambda$ and A_λ might not be homeomorphic.

Each of these differences between $\tilde{A}_\lambda$ and A_λ is solvable using one of these two methods:

1. We introduce multiple $\tilde{A}_{\lambda,i}$ and patch them together to cover all of A_λ. For our capsules this is straightforward, as it is nothing more than adding a new route r to the capsule, because each route r_i produces its own $\tilde{A}_{\lambda,r_i}$ using the route specific decoder g_{r_i}.
2. We add additional dimensions to M_Ω by expanding the attribute space, so that $\tilde{A}_\lambda$ itself becomes higher-dimensional and is able to overcome topological shortcomings.

These methods are illustrated in Figure 5.6. Note, that both methods can be used in any combination to solve the topological issues presented above.

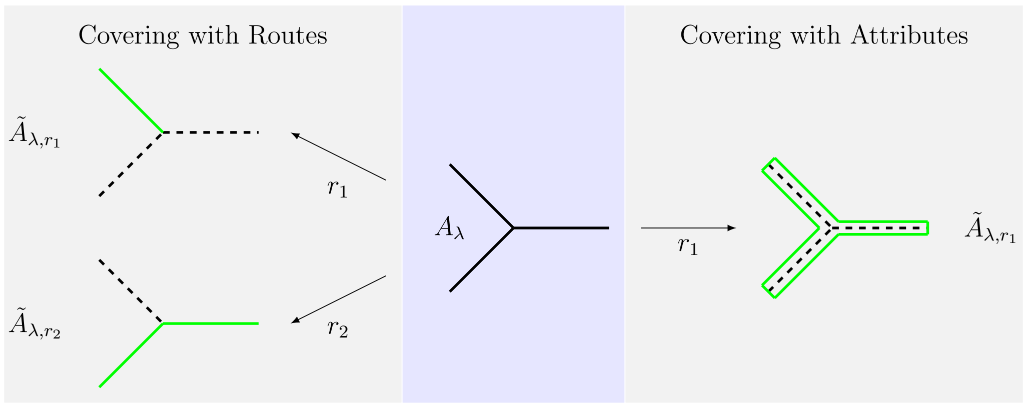

Covering with Routes
Covering with Attributes
$\tilde{A}_{\lambda,r_1}$
r_1
A_λ
r_1
$\tilde{A}_{\lambda,r_1}$
$\tilde{A}_{\lambda,r_2}$
r_2

Figure 5.6: Example of a problematic configuration space A_λ and options to cover it. Left-hand-side shows how A_λ is covered by two routes r_1 and r_2. Right-hand-side presents A_λ covered by a higher-dimensional space $\tilde{A}_\lambda$.

These covering methods, however, can lead to an overparameterization of the capsule and we might end up with redundant attributes. Let $\vec{\alpha}_\Omega|_{\alpha^k=[0,1]}$ denote an attribute vector with the k-th entry varied in the interval $[0,1]$. By redundant we mean an attribute α_Ω^k for which there exists another attribute α_Ω^l, such that $g(\vec{\alpha}_\Omega|_{\alpha^k=[0,1]})$ and $g(\vec{\alpha}_\Omega|_{\alpha^l=[0,1]})$ describe the same subset for any $\vec{\alpha}_\Omega$.

For example, a [*door*] capsule can have two verb attributes $\{close\}$ and $\{slam\}$. Both trace out the same curve in configuration space, describing the same movement, only at different speeds. Thus, they are numerically redundant attributes, yet encode semantically interesting information. This should motivate us to potentially keep some redundant attributes.

5.2.4 Fitting the Input Configuration Space

We assume for now that some decision was made in regard to a remedy to cover A_λ with $\tilde{A}_\lambda$. Yet, we are still faced with the problem that we only have a sparse idea of what A_λ actually looks like. Our goal is to fit $\tilde{A}_\lambda$ as well to the data and with as much generalization potential as possible as shown in Figure 5.6.

To perform the fitting, we employ machine learning techniques, such as deep learning for regression problems. These work perfectly well if we have a lot of data. However, we might be faced with a sparse situation as shown in Figure 5.7, leading to a lot of ambiguity. Fortunately, this is a common problem in the field of dimensionality reduction and manifold learning [Tenenbaum et al., 2000, Saul and Roweis, 2001, Maaten et al., 2009].

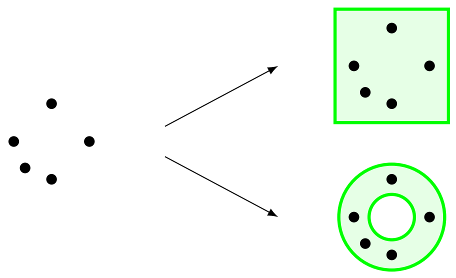


Figure 5.7: A dataset with unknown A_λ and two possible coverings with different topology.

To reduce ambiguity, we make use of the fact that we are dealing with high-level semantic information at this point, namely attributes with a known interpretation, instead of raw input data. Our approach is to assume some prior configuration space topology for $\tilde{A}_\lambda$ and as more data becomes available, let learning transform this space to slowly converge towards the true configuration space A_λ.

Defining such a prior configuration space is straightforward. We begin with the assumption that most attributes are disentangled, thus M_Ω has a similar flat topology as presented in Equation 5.5 or shown at the top of Figure 5.7. This allows us to treat groups of attributes of the same type individually. In accordance with this assumption, an augmented training set is created to encode all prior knowledge of $\tilde{A}_\lambda$ discussed here and both the encoder γ and decoder g are pre-trained with it.

For augmentation, our starting point is some observed input configuration, $\vec{\alpha}_{\lambda,0} \in A_\lambda$, which triggered the entire process. From this set of attributes, we must infer the capsule's own attributes $\vec{\alpha}_{\Omega,0} \in M_\Omega$. Generally, we are able to start from any point in M_Ω, but we choose the arithmetic mean of the individual attribute classes of the inputs $\vec{\alpha}_{\lambda,0}$, weighted by their size. We point out that the size is a purely visual measure and does not reflect the actual size of the object. This is done, as it is our best guess, but make exceptions for position, rotation and size. We propose the following equations to calculate the initial $\vec{\alpha}_{\Omega,0}$:

$$\vec{\alpha}_\Omega^{\,k} = \tilde{\gamma}^{\,k}(\vec{\alpha}_\lambda) = \frac{1}{\sum_\lambda \|\vec{\alpha}_\lambda^{\,size}\|} \sum_\lambda \vec{\alpha}_\lambda^{\,k} \cdot \|\vec{\alpha}_\lambda^{\,size}\| \quad . \tag{5.9}$$

$$\vec{\alpha}_\Omega^{\,rot} = \tilde{\gamma}^{\,rot}(\vec{\alpha}_\lambda) = \frac{1}{\sum_\lambda \|\vec{\alpha}_\lambda^{\,size}\|} \sum_\lambda \vec{\alpha}_\lambda^{\,rot} \cdot \|\vec{\alpha}_\lambda^{\,size}\| \quad . \tag{5.10}$$

$$\begin{aligned} \vec{\alpha}_\Omega^{\,size} = \tilde{\gamma}^{\,size}(\vec{\alpha}_\lambda) = &\max_{\lambda,i} \left(\mathbf{R}_\Omega^{-1} \cdot (\vec{\alpha}_\lambda^{\,pos} + \mathbf{R}_\lambda \cdot \vec{B}_{\lambda,i})\right) \\ &- \min_{\lambda,i} \left(\mathbf{R}_\Omega^{-1} \cdot (\vec{\alpha}_\lambda^{\,pos} + \mathbf{R}_\lambda \cdot \vec{B}_{\lambda,i})\right) \end{aligned} \tag{5.11}$$

$$\begin{aligned} \vec{\alpha}_\Omega^{\,pos} = \tilde{\gamma}^{\,pos}(\vec{\alpha}_\lambda) = \mathbf{R}_\Omega \cdot \frac{1}{2} \Bigg[&\max_{\lambda,i} \left(\mathbf{R}_\Omega^{-1} \cdot (\vec{\alpha}_\lambda^{\,pos} + \mathbf{R}_\lambda \cdot \vec{B}_{\lambda,i})\right) \\ &+ \min_{\lambda,i} \left(\mathbf{R}_\Omega^{-1} \cdot (\vec{\alpha}_\lambda^{\,pos} + \mathbf{R}_\lambda \cdot \vec{B}_{\lambda,i})\right) \Bigg] \quad , \end{aligned} \tag{5.12}$$

where $\tilde{\gamma}$ is our prior for γ, $\mathbf{R}_\lambda$ and $\mathbf{R}_\Omega$ indicate the Euler rotation matrix calculated from the rotation attributes $\vec{\alpha}_\lambda^{rot}$ and $\vec{\alpha}_\Omega^{rot}$ and $\vec{B}_{\lambda,i}$ the ith corner position vector of the bounding box of λ, i.e., pairwise permutations of $\vec{\alpha}_\lambda^{\,size}/2$ and $-\vec{\alpha}_\lambda^{\,size}/2$. Note, that the choice is arbitrary and we are free to use a different set of equations. These equations were chosen to have as few attribute collisions as possible without encoding too much knowledge a priori. A different choice, such as using the centroid of the rendered primitive instead of its bounding box, will lead to a different starting point in M_Ω, but will not impede our overall process. The chosen Equations 5.9-5.12 introduce some entanglement of the position and rotation attributes.

Now, let T_i indicate a random translation, rotation and scaling of the object that affects $\vec{\alpha}_\lambda^{pos}$, $\vec{\alpha}_\lambda^{rot}$ and $\vec{\alpha}_\lambda^{size}$ only. Further, let U_i indicate a random style transformation that sets all adjective attributes of the same type in $\vec{\alpha}_\lambda$ to the same random value in $[0, 1]$, if they are unused (i.e., for all observations in episodic memory Λ_Ω we have $\vec{\alpha}_\lambda^{j} < \epsilon$). Using these random transformations, the starting point $\vec{\alpha}_{\lambda,0}$ and assumed $\tilde{\gamma}$, we create the first augmented training set as

$$((\tilde{\gamma}(T_i \circ U_i \circ \vec{\alpha}_{\lambda,0}))^{(i)}, (T_i \circ U_i \circ \vec{\alpha}_{\lambda,0})^{(i)}) \quad . \tag{5.13}$$

In essence, we are encoding some of our information about how the world looks by augmentation with T_i and U_i. T_i contains our knowledge that an object is still the same type of object, even if it is moved, rotated or at a different size (if there are no reference objects). With U_i we actually encode our lack of knowledge about the possible styles of a new object. U_i assumes that an unused attribute could theoretically be of any value, in some sense mimicking style transfers. An [*apple*], for example, should have an adjective attribute {*metal*} of 0. Through augmenting it with U_i, we assume that a metallic apple is possible, as we have never seen it before ($\vec{\alpha}_\lambda^j < \epsilon$). However, as we don't know in what ratio the apple's [*stem*] and [*fruit*] are metallic, U_i sets them both to the same value and assumes that this is a metallic apple. We are free to introduce other augmentation strategies apart from T_i and U_i, that encode further prior knowledge of the world. Augmentation allows us to start training based on a single example in a one-shot manner.

Once more observations are made and there are more entries $\vec{\alpha}_{\lambda,k}$ in memory, we retrain γ using

$$((\tilde{\gamma}(T_i \circ U_i \circ \vec{\alpha}_{\lambda,k}))^{(i)}, (T_i \circ U_i \circ \vec{\alpha}_{\lambda,k})))^{(i,k)}) \quad . \tag{5.14}$$

As additional observations with different sets of attributes become available, less and less augmentation via U_i occurs and the resulting $\tilde{A}_\lambda$ begins to loose its disentangled topology induced by Equation 5.9 and converges towards a topology that covers its respective part of A_λ more closely. However, the augmentation by T_i remains unchanged and we, thus, do not allow that position, rotation and size get entangled with the other attributes, apart from the entanglement introduced in Equations 5.10-5.12.

Avoiding the augmentation we introduced here or introducing a different set, the meta-learning algorithm will still eventually cover the observed data with $\tilde{A}_\lambda$. However, this might take longer and more observations are needed. Our choices were made to increase the speed and allow for a better generalization through one-shot learning.

From our discussion thus far, we see that it is crucial to have a suitable distribution of points $\vec{\alpha}_{\lambda,i}$, from which the topological structure of A_λ can be inferred. In real environments, where there is no large amount of data with the same object in different styles, this is not always possible, thus, our approximated configuration space $\tilde{A}_\lambda$ represents a best guess until it has encountered a sufficient number of examples.

5.2.5 Verb Training

Thus far, our augmented training set for semantic capsules has ignored verb attributes. While we could simply allow U_i to augment verb attributes as it did the adjective attributes, we propose a better solution that can further increase the autonomy of the capsule network.

We assume that at some point in the life of a capsule it makes a new observation $\vec{\alpha}_{\lambda,l}$ and is prompted to train a verb attribute α^{new}. This indicates, that this observation is the continuation of a previous one, such as $\vec{\alpha}_{\lambda,j}$, but in a different pose. We'll refer to these as the original pose $(t = 0, \vec{\alpha}_{\lambda,j})$ and the new pose at $(t = 1, \vec{\alpha}_{\lambda,l})$, where $t \in [0, 1]$ indicates the change of pose over the entire animation. Now we have a set of $n = 2$ key-frames for the animation $\{(t = 0, \vec{\alpha}_{\lambda,j}), (t = 1, \vec{\alpha}_{\lambda,l})\}$. As new poses for α^{new} are observed, this list grows and is renormalized back to $t \in [0, 1]$. For a given verb attribute, we finally have a sequence of poses

$$\{(t_j = 0, \vec{\alpha}_{\lambda,j}), (t_{l_1} = 1/(n-1), \vec{\alpha}_{\lambda,l_1}), (t_{l_2} = 2/(n-1), \vec{\alpha}_{\lambda,l_2}), \cdots, (t_{l_{n-1}} = 1, \vec{\alpha}_{\lambda,l_{n-1}})\} \quad , \tag{5.15}$$

which are stored in the capsule's episodic memory.

At this point we introduce a special verb augmentation transformation W_i. If W_i acts on any observation $\vec{\alpha}_{\lambda,j}$ which is an original pose (i.e., $(t = 0, \vec{\alpha}_{\lambda,j})$), then W_i is just the identity and performs no transformation for that attribute. If W_i acts on any observation $\vec{\alpha}_{\lambda,l_i}$ which is a pose (i.e., $(t_i > 0, \vec{\alpha}_{\lambda,l_i})$ with $t > 0$), W_i takes the previous pose in the sequence $(t_{i-1}, \vec{\alpha}_{\lambda,l_{i-1}})$, chooses a random $t \in [t_{i-1}, t_i]$ and linearly interpolates between $\vec{\alpha}_{\lambda,l_{i-1}}$ and $\vec{\alpha}_{\lambda,l_i}$ to produce some new set of attributes $\vec{\alpha}_\lambda$ with $\alpha^{\text{new}} = t$. We thus have our new training regime

$$((\tilde{\gamma}(T_i \circ U_i \circ W_i \circ \vec{\alpha}_{\lambda,k}))^{(i)}, (T_i \circ U_i \circ W_i \circ \vec{\alpha}_{\lambda,k})))^{(i,k)}) \quad . \tag{5.16}$$

5.2.6 Expanding Memory

In the previous section we discussed briefly how to expand memory for verb attributes, by re-normalizing the sequence of poses. We perform a similar resizing of adjective attributes in memory, so that their range always lies in $[0, 1]$, as we have up to now always assumed that the entries in memory are perfectly normalized and correctly parameterized.

For this, we note that each time a new observation is made that triggered the meta-learning pipeline, some initial value is provided for the adjectives. However, this initial value, as will become clear in the following sections, is not necessarily correct, but rather a binary "*yes, it has this attribute*" $= 1$ or "*no, it does not have this attribute*" $= 0$.

The first time we are concerned with an observation $\vec{\alpha}_{\lambda,j}$ with an adjective attribute α^{adj}, where the meta-learning pipeline reported it needs to be trained ($\alpha^{\text{adj}} = 1$), we us it as a reference point and keep it at $\alpha^{\text{adj}}_{\lambda,j} = 1$. For scaling, we require a second reference point. For this, we can choose any previous observation, such as $\vec{\alpha}_{\lambda,l}$, with $\alpha^{\text{adj}}_{\lambda,l} = 0$.

We can measure the distance between these two reference points as $d_{\text{ref}} = \|\vec{\alpha}_{\lambda,j} - \vec{\alpha}_{\lambda,l}\|$. Now, each time a new observation $\vec{\alpha}_{\lambda,\text{new}}$ is added to memory, for which the meta-learning pipeline reports that α^{adj} needs to be trained, we measure its distance $d_{\text{new}} = \|\vec{\alpha}_{\lambda,\text{new}} - \vec{\alpha}_{\lambda,l}\|$ and set the adjective to $\alpha^{\text{adj}}_{\text{new}} = d_{\text{new}}/d_{\text{ref}}$. If $d_{\text{new}} > d_{\text{ref}}$, then the new attribute value $\vec{\alpha}_{\lambda,\text{new}}$ becomes the new reference value and all adjectives of this type in memory are rescaled back to $[0, 1]$ using $\alpha^{\text{adj}}_{\text{new}}$. Note that this only happens when the meta-learning pipeline reports that this specific attribute needs to be trained.

5.2.7 Equivariance

Our training regime introduces equivariance into γ and g for semantic capsules. This helps us to gain insight into how well the attributes parameterize the capsule. Underparameterization means that a capsule is not able to describe every possible configuration, style or pose of its parts (γ is not equivariant enough). This is a problem during feed-forward operation as objects might not get detected. Opposed to this, overparameterization means that some attributes have no effect on the capsule's parts (g is not equivariant enough). Here the situation is reversed, the capsule network is able to detect, but in a feed-backward operation, it is not able to correctly generate the described scene.

Checking equivariance explicitly is costly, as it needs to be performed with respect to some group and finding the correct representation can be difficult. However, we have different measures at our disposal to obtain a good picture of the situation. The overall performance of γ and g in the auto-encoder configuration $g \circ \gamma$ is an indication of how well the parameterization works. If the measure

$$Z\left(\tilde{A}_\lambda, (g \circ \gamma)(\tilde{A}_\lambda)\right) \tag{5.17}$$

over the complete $\tilde{A}_\lambda$ is low, it is most likely due to underparameterization. From this we devise another measure that can be used to check more specifically if over- or underparameterization is a likely problem:

- $\dim_{CBC} \tilde{A}_\lambda > \dim M_\Omega$ indicates underparameterization.
- $\dim_{CBC} \tilde{A}_\lambda \approx \dim M_\Omega$ indicates a suitable balance.
- $\dim_{CBC} \tilde{A}_\lambda < \dim M_\Omega$ indicates overparameterization.

Here $\dim_{CBC}$ means the contracted block-counting dimension (cf. Section 2.1) and dim the regular dimension. These are only indications, because extreme cases, such as space filling curves, may break this intuition. They are nonetheless useful in making an educated guess and we revisit these measures in our meta-learning pipeline.

5.3 Meta-Learning

In this section we describe the meta-learning agent and how it interacts and gives instructions to the training process discussed in the previous sections. It is far too difficult and too much effort to define the entire grammar with all rules and constraints from scratch to generate a complete capsule network. Instead, our approach is bottom-up and involves only the definition of the terminal symbols (primitive capsules). The meta-learning agent is then responsible for learning all non-terminal symbols (semantic capsules) and the rules (routes) connecting them. Apart from expanding the capsule network, the meta-learning agent is also in charge of the extension of the capsule's attributes. To make sense semantically of what it has learned, an oracle is questioned, the answer of which influences the training process. This is also referred to as active learning.

5.3.1 One Observed Axiom

Our generative grammar creates an image based on a single input symbol and its attributes. We showed that all images can be generated in such a way (cf. Section 4.2.3) and its inverse also holds true, i.e., each image can be described by a single symbol and its attributes (cf. Section 4.3.3). Thus, if there are multiple activated capsules with no common parent or a parent that did not activate, we interpret this as the observed grammar being incomplete due to the fact that the observed parse-trees don't have a common axiom. Figure 5.8 gives an example of such a situation and its remedy.

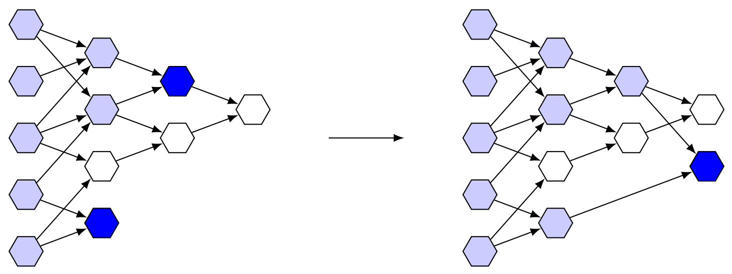


Figure 5.8: A capsule network with all activated capsules in blue (left). Here, the topmost activated capsules (dark blue) do not have a common parent capsule that activated, i.e., a single observed axiom. In this case, the meta-learning agent decided to add a common parent as the new axiom (right).

This does not mean that the capsule network may only have one such axiom overall, but rather, that there must be only one observed axiom for each observation. We will refer to the state of a missing observed axiom as an incomplete observed grammar. Including the case depicted in Figure 5.8, there are four causes based on which the agent detects an incomplete observed grammar (we include a verbal description for illustration):

A.1 A non-activated parent capsule is lacking a route.
"What existing symbol best describes these parts?"

A.2 A parent capsule is missing.
"What new symbol best describes these parts?"

B.1 An attribute is lacking training data.
"What existing attribute best describes this style or pose?"

B.2 An attribute is missing.
"What new attribute best describes this style or pose?"

Each of these causes can be remedied as indicated in the description, by adding a new route or capsule ((**A.1**), (**A.2**)) or by training an existing or adding a new attribute ((**B.1**), (**B.2**)). As discussed in detail in Sections 5.2.3 and 5.2.4, these remedies are equivalent to covering and fitting the configuration spaces. Our proposed meta-learning pipeline to rectify this problem consists of the following steps:

1. The meta-learning agent detects an incomplete observed grammar.
2. A weighted list of causes and supplementary information is [presented to an oracle as a question | used to make a decision].
3. The meta-learning agent acts on the decision and extends the capsule network by either:

(a) Adding a new route to a capsule with the same attribute setup as the previous routes.

(b) Adding a new capsule and connecting it, making sure it has the same set of attributes as its parts.

(c) Further training of a capsule's route with new training data for a specific attribute.

(d) Adding a new attribute to the capsule and all its ancestors.

Once the pipeline finishes, the capsule network may still be in a state of an incomplete observed grammar. To overcome this, the meta-learning process is repeated until there is a single observed axiom for the entire observed scene.

The decision of which remedy to choose is either performed by an oracle or, if available, past oracle decisions are used to automatically come to a conclusion. Thus, the decision is subjective. As an example, consider the situation presented in Figure 5.9: a capsule network with [*leg*], [*panel*] and [*chair*] capsules and route [*leg*][*leg*][*panel*] $\rightarrow$ [*chair*] makes a new observation triggering the meta-learning agent. It observed a scene of [*leg*][*leg*][*panel*], which did not activate the [*chair*] capsule. The oracle now must decide, if this is a [*chair*] of unknown style **(B.2)** or if it should be described by a different type of symbol, such as [*stool*] or [*table*] **(A.2)**. Also **(A.1)** and **(B.1)** may be possibilities that need to be considered.

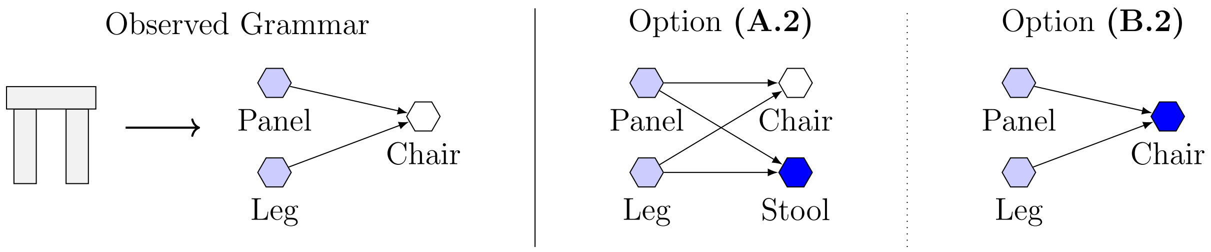


Figure 5.9: The observed grammar (left) is incomplete, because both the [*panel*] and [*leg*] capsule activated (light blue), but no common parent capsule activated (blue). Two possible remedies are presented, option **(A.2)** is to add a new parent capsule [*stool*] and option **(B.2)** is to add a new attribute.

5.3.2 The Decision Process

Each time the capsule network is confronted with an unknown situation (i.e., multiple top-most capsules that activated competing to be the observed axiom), it has to make a decision what training process it needs to start (**(A.1)** - **(B.2)**), by inquiring an oracle. In order to make a decision, the oracle is given specific features of the current situation. Assuming the oracle is a human, it is difficult to identify what these features are, as they contain meta-information about the observed state not encoded in the attributes alone. To capture as much information about these observations as possible, we propose the features in Table 5.1 with a short description of their interpretation.

#	Feature	Description
1	Top-most capsules share an unactivated Ω as parent.	The parts in the scene seem to belong to the same object Ω, but are in an unknown configuration.
2	Top-most capsules don't share an unactivated Ω as parent.	The parts in the scene seem to belong to different objects.
3	Parts tracked from previous scenes.	All parts were in the previous frame and should still belong to the same object.
4	$\Omega : Z(\vec{\alpha}, \vec{\tilde{\alpha}})$ indicates one attribute mismatch with no entry in memory $\alpha^i > \epsilon$.	It looks like the parts belong to object Ω, but with an unexpected use of the ith adjective.
5	$\Omega : Z(\vec{\alpha}, \vec{\tilde{\alpha}})$ indicates attribute mismatch for (position, rotation, size) only.	It looks like the parts belong to object Ω, but with an unexpected pose.
6	$\Omega : Z(\vec{\alpha}, \vec{\tilde{\alpha}})$ indicates attribute mismatch for more than half of all attributes.	It looks like that the parts belong to object Ω, but with very different attributes.
7	$\dim_{CBC} \tilde{A}_\lambda > \dim M_\Omega$ indicating under-parameterization in Ω.	The object Ω is not sufficiently described by its current set of attributes.
8	$\dim_{CBC} \tilde{A}_\lambda \approx \dim M_\Omega$ indicating a suitable balance in Ω.	The object Ω is suitably described by its current set of attributes.
9	$\dim_{CBC} \tilde{A}_\lambda < \dim M_\Omega$ indicating over-parameterization in Ω.	The object Ω has more attributes than needed.
	...	...

Table 5.1: Possible features that an oracle might base its decision on.

Using these features and the semantic interpretation of the attributes (adjectives, verbs and prepositions) and capsules (nouns), we are able to formulate actual questions for the oracle. Consider our previous example with two [*leg*] and one [*panel*] capsules activating, but not [*chair*]. Instead of presenting the oracle with this raw data, we may ask it based on Feature 5.1 *"Do these two [leg]s and one [panel] form a [chair]?"* and highlight the area in the image using the grammar. The answer then corresponds to one of the four training regimes:

A.1 A non-activated parent capsule is lacking a route.
"Yes, this is [chair]."

A.2 A parent capsule is missing.
"No, this is a [stool]."

B.1 An existing attribute is lacking training data.
"Yes, this is a {wooden} [chair]."

B.2 An attribute is missing.
"Yes, this is a {modern} [chair]."

To avoid inquiring an oracle each time, we can use these features and the past decisions of the oracle as a data set and train a classification model, such as the simple decision matrix shown in Table 5.2. By summing all the rows whose feature is true, we find the predicted decision (**(A.1)** - **(B.2)**) from the column with the highest value. Should the oracle make new decisions, the matrix is updated by adding 1 to the respective columns of all rows that were true.

Feature	A.1	A.2	B.1	B.2
Top-most capsules share an unactivated Ω as parent.	4	3	14	12
Top-most capsules don't share an unactivated Ω as parent.	5	19	1	0
Parts tracked from previous scenes.	14	1	17	12
$\Omega : Z(\vec{\alpha}, \vec{\tilde{\alpha}})$ indicates one attribute mismatch with no entry in memory $\alpha^i > \epsilon$.	1	0	12	2
$\Omega : Z(\vec{\alpha}, \vec{\tilde{\alpha}})$ indicates attribute mismatch for (position, rotation, size) only.	4	3	13	10
$\Omega : Z(\vec{\alpha}, \vec{\tilde{\alpha}})$ indicates attribute mismatch for more than half of all attributes.	12	14	4	4
$\dim_{CBC} \tilde{A}_\lambda > \dim M_\Omega$ indicating under-parameterization in Ω.	1	2	5	12
$\dim_{CBC} \tilde{A}_\lambda \approx \dim M_\Omega$ indicating a suitable balance in Ω.	12	5	10	6
$\dim_{CBC} \tilde{A}_\lambda < \dim M_\Omega$ indicating over-parameterization in Ω.	5	14	12	7
...				

Table 5.2: Example of a trained decision matrix with an excerpt of features derived from the observed parse-trees and what cause they indicate (number of past oracle decisions). Here Ω is the capsule with the highest p_Ω that did not activate.

The overall process is comparable to how an infant learns about the world, by first inquiring its parent and slowly learning the subjective way they classify and describe new objects.

5.4 Implementation and Results

For an example, we start with a capsule network that consists of three different primitive capsules, [*square*], [*triangle*] and [*circle*]. Each of them is implemented using signed distance fields [Hart et al., 1989, Hart, 1993, Quílez, 2017] as their renderer g. For the encoder γ we initially employed using neural architecture search, however realized that an AlexNet-like CNN [Krizhevsky et al., 2012] for regression works well enough as encoder γ.

The semantic capsules are implemented using a 4-layer deep dense neural network for regression with tanh activation functions as their encoder γ and decoder g, where the width is dependent on the number of attributes. During training, we employ dropout [Hinton et al., 2012] on individual input capsules to simulate occlusions. The full implementation is available as a Github repository at `https://github.com/Kayzaks/VividNet`.

To test our meta-learning implementation, we use a toy example based on an environment visually similar to Asteroids (Atari 2600). By presenting the capsule network with a scene, the meta-learning pipeline is immediately triggered, as a lot of [*square*], [*triangle*] and [*circle*] symbols are detected, but without a common parent as their observed axiom.

The oracle is queried to resolve these issues. However, depending on the subjective answer the oracle gives, the capsule network ends up with different configurations. In Figure 5.10 we present two such networks reached in one-shot. Note that the information content of each is slightly different, yet they are both able to correctly identify all objects in the scene as well as their relations.

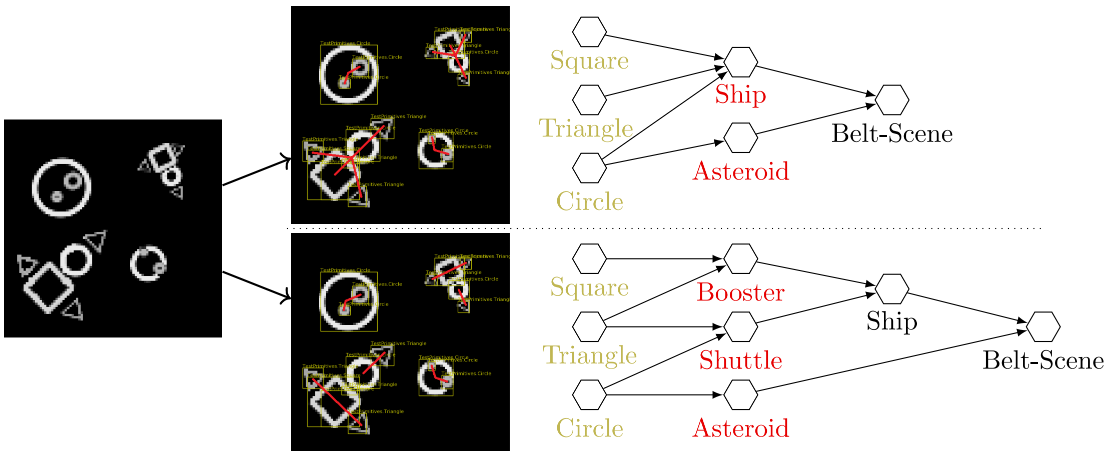
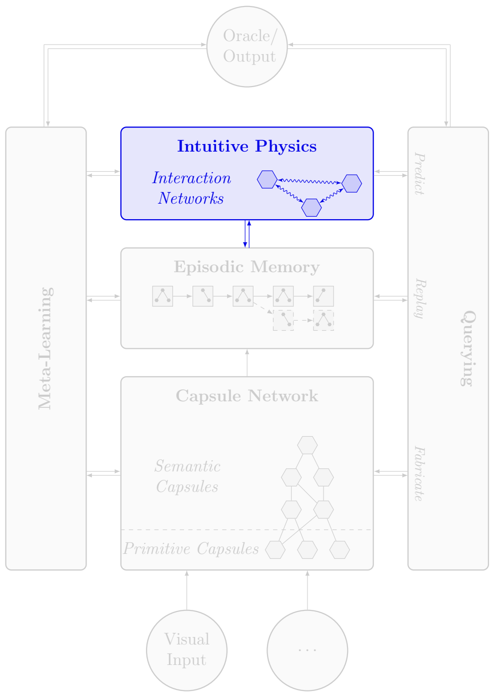


Figure 5.10: Two of many possible capsule network configurations the meta-learning agent might end up with, depending on the oracle and the decision matrix.

Chapter 6

Intuitive Physics

One of the advantages of a scene-graph representation of the visual semantics is that it is very versatile structure which allows for further exploration. In this chapter we discuss how to extract physical properties from an observation and how to predict future events based on this knowledge[1]. These processes of intuitive physics are a vital component to achieve our overall goal of mental simulation.



We approach the problem of predicting physics in three steps. First, we analyze the observations found in episodic memory to determine what objects in the scene are interactable. For this set of objects, we determine what kind of interaction they allow. This comprises, for example, is the object elastic or rigid or is the object connected to another by a joint? Finally, these properties allow us to predict future events using interaction networks.

[1]A preprint of the results in this chapter was previously uploaded to arXiv [Kissner and Mayer, 2019a].

6.1 Interaction Networks

Interaction networks were designed to learn and predict the relations and physical behavior of interacting bodies [Battaglia et al., 2016]. The overall idea is to learn this interaction for pairs of objects and to aggregate the results to determine the final effect these interactions have on an object.

To model an interaction pair, one object o_i is labeled as the *sender* and the other object o_j as the *receiver* of some physical effect e_k. To calculate this effect, more information about the interaction is required, such as all possible constraints between sender and receiver, which we encode in ρ. This gives us an interaction triplet $\langle o_i, o_j, \rho \rangle_k$. The resulting physical effect e_k of this interaction is calculated using a *relational model*:

$$e_k = \phi_R[\langle o_i, o_j, \rho \rangle_k] \quad . \tag{6.1}$$

In order to make a prediction for the next time-step of a receiver o_j, all interactions with all possible senders $\{o_i\}$ must be calculated to find the full set of effects $\{e_k\}$. Not all physical effects, however, arise from pairwise interactions. We must, therefore, include external effects x_j that may apply to o_j, such as gravity.

To aggregate all interaction effects that apply to the receiver, an aggregation function $a(\cdot)$ is introduced. In general, a linear combination of all effects $\{e_k\}$ is sufficient. To apply these effects to the o_j in a meaningful way, an *object model* is introduced as

$$o_j^{t+1} = \phi_O[o_j, a(\{e_k\}), x_j] \quad , \tag{6.2}$$

which calculates the receiver's attributes at the next time-step.

Both relational models ϕ_R and ϕ_O are the central part of interaction networks and are implemented using regression models trained on previously observed interactions. This allows us to both learn and predict intuitive physics.

6.2 Interactable Neural-Symbolic Capsules

Our observations in episodic memory are ideal candidates for interaction networks, as each object is a discrete entity and the most relevant physical properties are either known (position, rotation, ...), can be inferred explicitly (speed, ...) or estimated implicitly (mass, ...). In the following, we explore how to integrate interaction networks with our neural-symbolic capsule network.

For the physical interactions which we want to extract from the visual data, the distance and the orientation of the surfaces are important. While these quantities are, in some sense, encoded in the attributes, such as $\{$*size*$\}$ and $\{$*position*$\}$, and through their relation to other objects in the observed parse-tree, we extract this information explicitly using the object's visual representation, as:

1. Render objects A and B.

2. Find the minimal distance between the pixels (2D) or voxels (3D) of the two renderings using

$$d_{A,B} = \inf\left\{d(\vec{a}_i^{pos} - \vec{b}_j^{pos}) \mid \vec{a}_i \in A,\ \vec{b}_j \in B;\ a_i^{int} > 0,\ b_j^{int} > 0\right\} \quad , \tag{6.3}$$

   where $\vec{a}, \vec{b}$ are the attributes of individual pixels or voxels, *pos* and *int* refer to their positions and intensities and A, B on the sets of all pixels or voxels for each object.

3. Calculate the boundary normals $\vec{N}_A$ and $\vec{N}_B$ for the points of minimal distance between A and B.

6.3 Interactable Objects

Some features we require for the interaction networks must be extracted from past observation sequences and can't be inferred from a single image without prior information. We need to know about the object's ability to deform and move, in order to make accurate physical predictions.

For this analysis we need to identify those objects in the scene that are either non-interactable, independently interactable or constrained by a joint. For example, while a standard [*chair*] is made up of multiple parts, we only consider the [*chair*] as a whole interactable, as moving one [*leg*] causes the [*chair*] in its entirety to move. However, for an [*office-chair*] the [*chair-base*] and the [*seat*] are connected by a joint. Here, we consider [*chair-base*] and [*seat*] as interactable, because they have a degree-of-freedom (DOF) by which they can move independently of each other (namely rotation), but consider [*office-chair*] not to be interactable, as its interactions are fully determined by the two interactable parts. We see that the presence of a verb attribute, such as {*swivel*}, is the main factor that determines this joint/interactable relationship.

To find the interactable objects and the DOF of their movement, we propose the following steps (note that we assume g is continuous, as is the case for regression models based on neural networks):

1. Find a semantic capsule Ω that has verb attributes $\vec{\alpha}_\Omega^{k_1,\cdots,k_n}$ and one or more child capsules λ_i that do not have verb attributes. All child capsules λ_i that satisfy this condition form the set of interactable objects O.

2. Vary $\vec{\alpha}_\Omega^{k_1,\cdots,k_n}$ in the range $[0,1]^n$ using $g(\vec{\alpha}_\Omega) = \vec{\alpha}_{1,\cdots,|\lambda|}$ to find the spaces for scale, pose (position and rotation) and collision

$$\mathbf{S}_i = \{\vec{\alpha}_i^{size}\} \quad , \tag{6.4}$$

$$\mathbf{Q}_i = \{\vec{\alpha}_i^{pos} \oplus \vec{\alpha}_i^{rot}\} \quad , \tag{6.5}$$

$$\mathbf{Q}_{ij} = \{(\mathbf{R}_i \cdot (\vec{\alpha}_j^{pos} - \vec{\alpha}_i^{pos})) \oplus (\vec{\alpha}_j^{rot} - \vec{\alpha}_i^{rot})\} \quad , \tag{6.6}$$

$$\mathbf{D}_{ij} = \{d(\lambda_i, \lambda_j)\} \quad , \tag{6.7}$$

where $\mathbf{R}_i$ is the Euler rotation matrix for $-\vec{\alpha}_i^{rot}$, which is used to move the two objects into the same reference frame. $\mathbf{Q}_{ij}$ is also defined for two capsules which do not share a common parent, by instead finding the next common ancestor and successively applying g to vary over $[0,1]^n$.

3. For the spaces $\mathbf{S}_i$, $\mathbf{Q}_i$ and $\mathbf{Q}_{ij}$ we find the contracted block-counting dimensions $\dim_{CBC} S_i$, $\dim_{CBC} Q_i$ and $\dim_{CBC} Q_{ij}$.

In step 1, we are essentially splitting the objects in an observation into two groups. The first group consists of all capsules whose parts have been found to take on different poses described by a verb. An example is [*laptop*], where the [*screen*] and the [*keyboard*] poses are dependent on some {*open*} verb attribute. The second group consists of all capsules whose parts do not take on individual poses, such as a laptop's [*screen*] which itself is made out of parts, but their pose is fully described by the screen's pose and, thus, the [*screen*] does not have a verb attribute.

Following the identification of all the interactable objects, we explore how they move in relation to each other in step 2 and explicitly compute the DOF in step 3. From the derived quantities, we are able to infer some of the physical properties of the objects, such as:

A A verb-less child λ_i is a **rigid body** if $\dim_{CBC} S_i = 0$.
$\rightarrow$ *The size has never been observed to change over time.*

B A verb-less child λ_i is an **elastic/plastic body** if $\dim_{CBC} S_i > 0$.
$\rightarrow$ *The size has been observed to change over time.*

C A verb-less child λ_i is **static** if $\dim_{CBC} Q_i = 0$.
$\rightarrow$ *The position or rotation has never been observed to change over time.*

D A verb-less child λ_i is **dynamic** if $\dim_{CBC} Q_i > 0$.
$\rightarrow$ *The position or rotation has been observed to change over time.*

E Two verb-less descendants λ_i and λ_j are connected by a **joint** if they collide ($\sup \mathbf{D}_{ij} < \epsilon$) and their total DOF $\dim_{CBC} Q_{ij}$ in relation to each other is $3 > \dim_{CBC} Q_{ij} > 0$ (2D) or $6 > \dim_{CBC} Q_{ij} > 0$ (3D).
$\rightarrow$ *The two objects have been observed to move in relation to each other over time in a constrained way.*

Consider again our earlier example of the [*office-chair*] with the {*swivel*} verb attribute. Its parts, [*base*] and [*seat*], do not have this verb and are, according to step (1.) of our algorithm, interactable, whereas their parent [*office-chair*] is not. Now, by further analyzing the properties of [*base*] and [*seat*] using steps (2.) and (3.), we find that they are connected by a joint with one DOF ($\dim_{CBC} Q_{base,seat} = 1$). Further, both of these objects are dynamic ($\dim_{CBC} Q_{base} = 3$ and $\dim_{CBC} Q_{seat} = 3$), as they both are able to move around the room. Note, however, that we identify 3 DOF here: 2D movement across the floor and rotation around its own axis. This is the case, if the capsule network has never actually seen the [*office-chair*] being lifted up or tilted to the side and assumes that it is, in some sense, stuck to the floor (i.e., if [*office-chair*] were interactable, it would be considered to be connected by a joint to the floor). Assume for now, that the [*office-chair*] has a second verb attribute {*kicked*}, which is just a fast swivel. Now, we need to vary across two verbs using $[0,1]^2$ in step (2.) of the algorithm and one could assume that this traces out a plane in $\mathbf{Q}_{base,seat}$. However, by closer inspection we find that it does indeed still trace out the same curve as {*swivel*}, thus preserving our single DOF for this hinge.

6.4 Defining Relations

With all the required features in place, we are able to form the sender-receiver-relation triplets $\langle o_i, o_j, \rho \rangle_k$ that serve as the input for ϕ_R. In Table 6.1, the full definition of such a triplet is given, where we one-hot encode the symbol λ_i and encode the attributes such that across all objects, the position in the vector is the same if they have the same name. This is important, as the type of attribute might give indications of certain physical properties, such as {*metallic*} for magnetism.

Sender o_i					Receiver o_j					Relation ρ			
λ_i	$\vec{\alpha}_i$	$\frac{d\vec{\alpha}_i}{dt}$	static / dynamic	rigid / elastic	λ_j	$\vec{\alpha}_j$	$\frac{d\vec{\alpha}_j}{dt}$	static / dynamic	rigid / elastic	d_{o_i,o_j}	joint	$\vec{N}_{o_i}$	$\vec{N}_{o_j}$

Table 6.1: Full relation triplet used as input for ϕ_R.

Next, we aggregate the results of ϕ_R using $a(\cdot)$ and employ the format given in Table 6.2 as input for ϕ_O.

Receiver o_j					Effects	
λ_i	$\vec{\alpha}_i$	$\frac{d\vec{\alpha}_i}{dt}$	static / dynamic	rigid / elastic	Σe_k	X

Table 6.2: Aggregated data $a(\cdot)$ used as input for ϕ_O.

By being very explicit about the attributes and their lexical interpretation, the resulting interaction of ϕ_O exhibits the predicted characteristics: The regression model learns to associate attributes with physical properties. Revisiting our example, if the object has a high value for its {*metallic*} attribute, it is more likely heavy or magnetic, changing its physical interaction with the environment compared to an object with a low {*metallic*} value.

6.5 Implementation and Results

We implement the interaction network using a neural network for regression with six dense layers for ϕ_R and one with two dense layers for ϕ_O. For our test, we train the model on synthetic video data of collisions between circles. To calculate the contracted box-counting dimensions for $\mathbf{S}_i, \mathbf{Q}_i, \mathbf{Q}_{ij}$ and $\mathbf{D}_{ij}$, we employ a Monte-Carlo approach to avoid varying over all of $[0,1]^n$. By randomly selecting $[x_1, x_1+\Delta x] \times \cdots \times [x_n, x_n+\Delta x] \subseteq [0,1]^n$ boxes and feeding them into the decoder g, we get transformed volumes $\hat{V}$, which we contract using Equation 2.6 to find V. Repeating this process, we find a distribution for $\dim_{CBC} V$ and choose the maximum of this distribution as the true dimension for the space we are analyzing.

The distance in Equation 6.3 is calculated explicitly through the signed distance functions used in our capsule implementation.

For testing, the interaction network is fed with an observation from episodic memory. Figure 6.1 shows how our implementation predicts the interaction of two and three circles. This prediction is based on the capsule network having seen two frames with these circles and is able to simulate their interaction plausibly for ~ 50 frames thereafter.

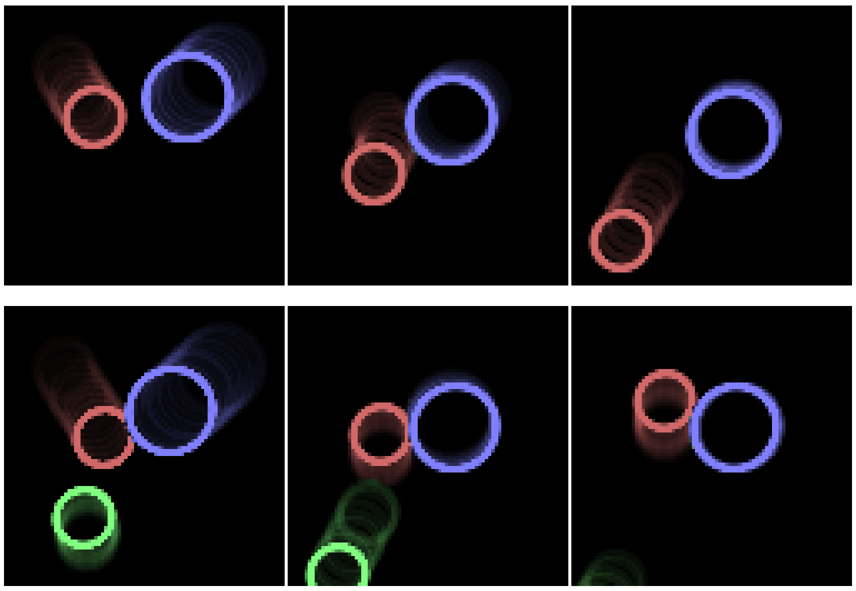


Figure 6.1: Top shows the prediction of two circles interacting. Bottom shows the prediction of three circles interacting, whereas the interaction network was only trained on two. Movement is illustrated using color and object trails.

In the next experiment, we use a figure-eight that is unable to move horizontally and vertically and only rotates like a windmill ($\dim_{CBC} Q_{eight} = 1$). Next, we launch a circle towards the figure-eight and predict the physical interaction, as seen in Figure 6.2, and see that it indeed behaves plausibly by rotating on collision.

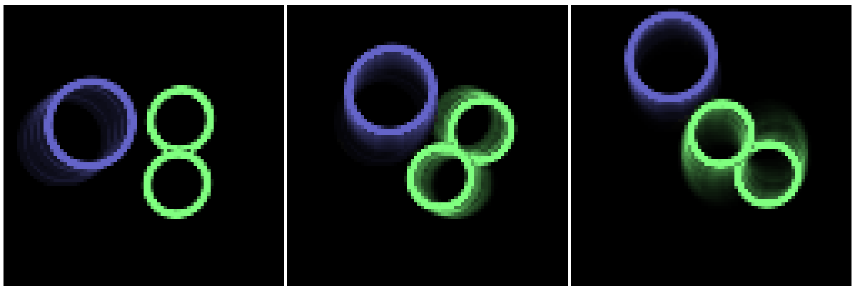


Figure 6.2: Interaction of a circle with a figure-eight, which is only able to rotate around its axis. Movement is illustrated using color and object trails.

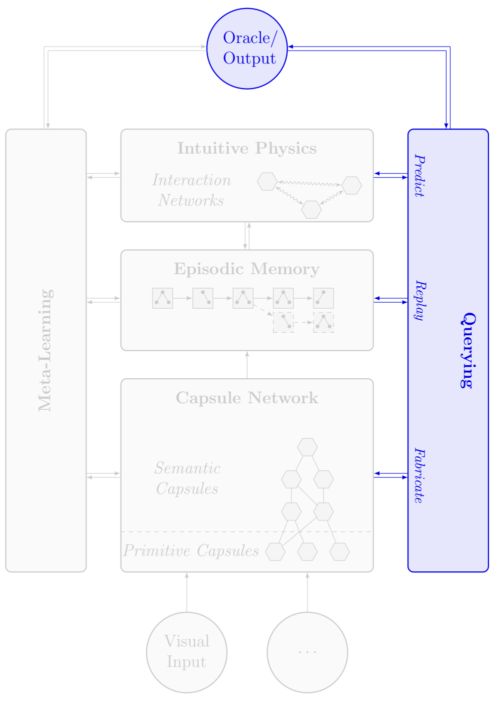
Chapter 7

Mental Simulation

With all elements needed to construct our mental simulation framework in place, we tie them together into a querying process. In this chapter we describe the extraction of knowledge from our architecture in detail. We subdivide the used queries into three types, which can be recombined in any way:

- **Replay:** Extracting existing information.
- **Predict:** Making a prediction based on a given state.
- **Fabricate:** Creating a fictive scenario.

Each of these query types is associated with one of the previously discussed parts: Replaying events targets episodic memory, predicting events relies on intuitive physics and fabricating events utilizes the capsule network itself. We begin by introducing a query language and proceed to explore some of the possible applications it enables. The presented framework is designed as a building block for planning agents [Russel and Novig, 2010] and the query language acts as its interface.



7.1 Querying

Mental simulation is the act of imagining an action and simulating the results in order to make a decision. To design our query mechanism, we must break down this task into basic components. First, to imagine something we need a starting point and some way to **replay** it. This starting point has to be modifiable in order to **fabricate** the effect of an action or a new situation. Finally, we **predict** future outcomes on which an agent would base its decision. The entirety of this process is our mental simulation. We now discuss these three individual actions in detail.

Replay: To replay an event, a sequence of time-ordered observations in episodic memory is chosen and returned or played. This is akin to video playback, but a replay on episodic memory is not limited to one type of observation or a single time-line.

Predict: Intuitive physics is used to predict the next time-step of some prior observation. Even though this prediction itself already has the structure of an observed parse-tree ready to be saved, it is instead fed into the capsule network and only the resulting observation is stored as a prediction in episodic memory. This final step might seem redundant, however, we illustrate by an example that it is indeed important to let the capsule network know about the semantics. Consider a half open door and a ball flying towards it. Intuitive physics then predicts that the ball hits the door, closing it, and then flies off at an angle. Our capsule network may have never seen a door in the closed state before. By feeding these new semantics back into the network, we force the meta-learning pipeline to act and learn a potentially new data point, prior to even observing it in a real scene and purely from mental simulation.

Fabricate: We fabricate events by altering given observations at certain nodes in the parse-tree that are of interest or by inventing completely new ones and feeding the result to the capsule network, as we did with predicted observations. For example, if the oracle received the description or proportions of a new object through a different source, such as a verbal explanation, the capsule network and its meta-learning pipeline are able to learn the new visual semantics by fabricating a scene and imagining it.

7.1.1 Expanded Episodic Memory

We expand our episodic memory to handle, apart from perceived observations from actual visual input, two more types of entries: fabricated and predicted observations.

The perceived observations form the main time-line of episodic memory. When we make a prediction based on such an observation, a new forked time-line is created. In this fork, new predictions can be made to further develop this time-line, or it may even be forked again. As predictions, fabrications are able to create a new fork, but may also spawn a completely separate time-line. Further, forks can be merged at any point. It is, however, forbidden that predictions and fabrications occur on the main time-line. Without such a restriction, it would be similarly to a human not able to distinguish reality from fiction.

To illustrate these three observation types, we consider the following example: A person sees a ball fly towards a wall (perceived observations). This person predicts that a few time-steps into the future the ball will bounce of the wall and fly towards the ground (predicted observations). If the predictions concur with what is observed shortly thereafter, a merge of the expected fork with the main time-line occurs, if not, the two branches diverge. Further, the person in our example may also imagine a second ball (fabricated observation) and how it flies and interacts with the first ball (predicted observations based on the fabricated observation). This branch begins completely independent of other time-lines, but may merge at some later point, if the set of events should ever occur in reality.

In Figure 7.1 a schematic depiction of how episodic memory stores these time-lines is given.

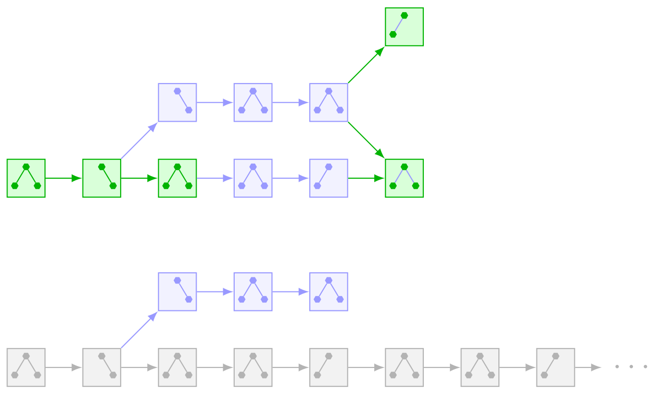


Figure 7.1: Example of episodic memory. The main time-line of perceived observations (gray) and a parallel time-line of fabricated observations (green), each with predicted observations (blue).

7.1.2 Visual Knowledge

Using a combination of the three queries (replay, predict, fabricate), we are able to extract visual knowledge that has been learned about the environment. For this task we outline a query syntax, inspired by other query languages such as SQL, GraphQL and SPARQL.

We first analyze in what tiers episodic memory is organized:

1. Tier: Time-line data is stored as a disconnected network graph of perceived, predicted and fabricated observations (cf. Figure 7.1).
2. Tier: Observations are stored together with meta-information and a tree-structure of observed capsules (cf. Figure 4.8).
3. Tier: Observed capsules are stored with their activation probability and a collection of attributes (cf. Equation 4.14).

Due to the heterogeneous nature of each tier, the query syntax we introduce must be flexible enough to produce these different representations, so that we do not have to introduce a separate syntax for each tier. The output data structure is chosen to be a disconnected network graph, as it can be organized as a graph-of-graphs (1. Tier), a tree-structure (2. Tier), as well as a pure collection (3. Tier). The content of this graph depends on the tier the query operates on. An example for the highest tier can be seen in Figure 7.2.

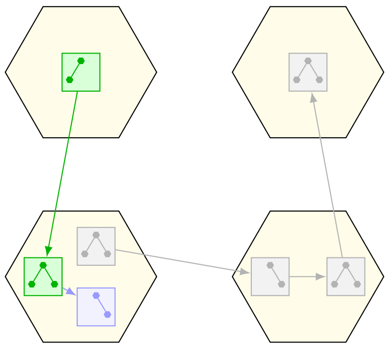


Figure 7.2: Creating a graph-of-graphs by clustering entire or partial observations filtered from episodic memory.

Next, we outline a simple query specification to extract knowledge given one of the three query types. In terms of SQL, replay can be seen as a `SELECT`, fabricate as an `INSERT` and predict as an `UPDATE`, although this analogy isn't entirely true, as a predict adds an updated observation to the memory instead of updating it in place.

We do not specify the exact grammar for this language, but rather keep it on a pseudo-code level. The structure for the queries is as follows:

$$Results, Frames \leftarrow \textsc{Replay}(Origin, Filters, Mappings, Groupings) \tag{7.1}$$
$$Observation \leftarrow \textsc{Predict}(Origin, Duration, Forces) \tag{7.2}$$
$$Observation \leftarrow \textsc{Fabricate}(Origin, Changes) \tag{7.3}$$

- *Results:* The results from a replay query, the tier format of it being dynamic and determined by the query itself.
- *Frames:* A series of frames rendered based on the results. If the results contain symbols instead of observations, each symbol is rendered individually, as if it were an observation of a single object. Queries that return non-renderable parts (attributes or meta-information) return a blank image.
- *Observation:* The observation that results from a predict or fabricate query.
- *Origin:* Starting point for a query, such as an observation, and acts like a `FROM` clause. May be empty for fabricate. In this case, a completely new time-line is created instead of a fork. For replay, it acts as a pre-filter and determines which observations are affected by the query.

- *Filters:* Generally, each query is concerned with all information found in memory, i.e., every symbol, attribute, connection (i.e., the edge between two consecutive observations) and piece of meta-information. Filters are designed to exclude uninteresting or redundant information. For example, if we are only interested in recent events, we define a filter to exclude all parse-trees before some specific time-stamp or only include symbols whose $\{red\}$ attribute is larger than some threshold. These filters act like a `WHERE` clause.
- *Mappings:* To allow for simpler grouping, we map semantic structures to new ones. For example, we could define an adverb-mapping that maps all adjectives and their values, such as $\{metallic : 0.8\}$, to an adverb-adjective combination if they exceed some threshold, e.g., $\{metallic : 0.8\}$ becomes $\{very\ metallic\}$. These mappings are equivalent to functions in other query languages.
- *Groupings:* For the final grouping, clauses may be added that act as a `GROUP BY` and a `HAVING` statement. They cluster together entire parse-trees or also parts of them in a graph-of-graphs structure.
- *Duration:* The duration intuitive physics is supposed to predict starting from the origin.
- *Forces:* External forces (cf. Equation 6.2) applied to an object or scene.
- *Changes:* Changes that are applied to the origin of a fabricated observation, or the new parse-tree that is created if no origin is provided.

Only a replay query returns processed information, namely the filtered, mapped and grouped data, as well as the corresponding rendered images. While the images aren't required, as all the requested information is provided through the semantics, they do serve a useful purpose to visualize the results and are employed in some of the applications we explore in the following section. Furthermore, they strengthen the notion of mental simulation and imagination.

7.2 Mental Simulation

With the ability to query our framework, it is able to function as a mental simulation engine. The capability for simulations is learned through observing the environment and occasionally interacting with an oracle. While our framework is not capable of performing any creative task by itself, its ability to simulate offers a platform for exploration for an oracle. We quote from [Hamrick, 2017]:

> "There is a sense in which mental simulation is not only useful for prediction and inference, but also for exploration. For example, Finke and Slayton [1988] showed that people can use mental simulation to discover novel and creative combinations of simple objects, such as combining the shapes `V`, `O`, and `D` into an ice cream cone (Rotate the `D` by 90° left, place it on the `V`, and then place the `O` on the very top). Given mental simulation's importance in planning, which

inherently must deal with the exploration-exploitation trade-off [Sutton and Barto, 1998], it seems unsurprising that the mind might be adapted towards using mental simulation for exploratory thought more generally. Moreover, although the way we choose which mental simulations to run is biased towards what is normally expected, we are able to modify our simulations to explore just outside the bounds of everyday life. For example, I can imagine riding an animal (which is not too far out of the ordinary), but gradually change the features of my mental simulation to make it unordinary (such as imagining riding a giant cat, or a flying alligator). The fact that our mental simulations tend to encode so much that is true about the world [Gendler, 1998] may be what makes them so perfect for creative thought: by tweaking them just by a small amount, we hit the sweet spot between novelty and utility that seems crucial for marking something as "creative" [Ward, 1994]. [. . .]"

Through the use of queries, we have abstracted away the need to visually understand a scene in many cases and an oracle can focus on analyzing answers to the questions it has based on symbolic information. In the following, we highlight some of the possible mental simulation tasks that an oracle might induce.

7.2.1 Sorting

Algorithm 1 Sorting apples

```
1: $apples \leftarrow$ REPLAY(**origin all**, **filter** [$apple$] **and** $time\text{-}stamp > t_0$
                              **and omit** $connections$,
                              **map** ([$apple$].{$red$}$< 0.5 \rightarrow$ {$reject$})
                              **and** ([$apple$].{$red$}$\geq 0.5 \rightarrow$ {$prefer$}),
                              **group by** {$prefer$}, {$reject$})
```

This query replays every recent apple symbol (after t_0) and maps the red attribute to whether to prefer or reject it. Finally, all apples are grouped and two collections (instead of graphs, as connections were filtered out) are produced, one with all preferred apples and one with all rejected apples. This could be a reasonable query at the supermarket for an agent that does not like green or discolored apples.

7.2.2 Mapping the Environment

For our next query, let G be some uniform grid in space given as a parameter to the query with $G_{i,j}$ its individual grid points.

Algorithm 2 Navigating by region

```
1: $map\text{-}graph \leftarrow$ REPLAY(**origin** $perceived\text{-}observations$, **filter** $time\text{-}stamp > t_0$,
                              **map none**, **group by** $\min_{i,j} \| camera\text{-}position - G_{i,j} \|$)
```

In Algorithm 2 we cluster all observations that are close to each of these grid points (cf. Figure 7.3). As all connections between the networks are preserved, the graph-of-graphs this query produces is essentially a navigation graph of walkable space [Werner, 2014] explored since t_0. We could extend this map purely based on "imagination" by performing predict-actions and including the predicted states. The map could be further improved by assuming that the aggregation of the connections between the nodes in the final graph-of-graphs represents the certainty that this route actually exists. Using this mental map, the oracle may then perform graph traversal to find the shortest route to some location based on the knowledge it has gained. Note that Algorithm 2 is overly simplified and does not take more complex effects into consideration, such as drift when it comes to loop closing.

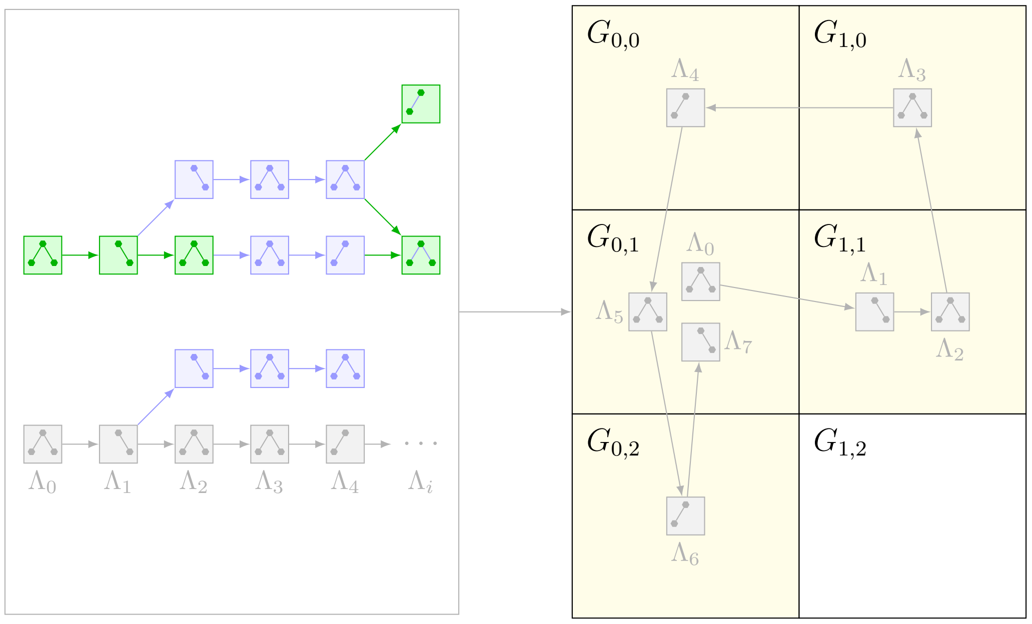


Figure 7.3: Transformation of episodic memory using Algorithm 2 to form a graph-of-graphs that represents the spatial map of the environment (size and distances not to scale), by only clustering perceived observations (gray) with a camera position similar to the grid points $G_{0,0}, \cdots, G_{1,2}$ (filled grid cells in yellow, empty grid cell in white).

7.2.3 Will They Collide?

Given a starting observation (*ball-obs*) of two moving balls ($[ball]_A$ and $[ball]_B$), we may ask ourselves, if they will collide in the next second? Employing the distance function used in intuitive physics (Equation 6.3), we incrementally predict the next time step of the two balls flying towards each other and check if they collide. This process is highlighted in Algorithm 3.

Algorithm 3 Will they collide?

```
1: $step \leftarrow 0$
2: **repeat**
3:     $ball\text{-}obs \leftarrow$ PREDICT(**origin** $ball\text{-}obs$, **duration** 5 ms, **force none**)
4:     $distance \leftarrow$ REPLAY(**origin** $ball\text{-}obs$, **filter** $[ball]_A$ **or** $[ball]_B$,
                          **map** $d([ball]_A,[ball]_B)$, **group by none**)
5:     $step \leftarrow step + 1$
6: **until** $distance < \epsilon$ **or** $step \geq 20$
```

7.2.4 Would it Fit?

Another common mental simulation task is to check if an [*object*] fits inside some [*container*]. We may do this by fabricating a scene with both objects in it and placing one inside the other. The distance function will then allow us to perform a simple check if the object is colliding, i.e., doesn't fit (cf. Algorithm 4).

Algorithm 4 Would it fit?

```
1: $fabricate\text{-}obs \leftarrow$ FABRICATE(**origin none**, **change add** [$object$] **and** [$container$])
2: $distance \leftarrow$ REPLAY(**origin** $fabricate\text{-}obs$, **filter** [$container$] **or** [$object$],
                          **map** $d([container],[object])$, **group by none**)
3: **if** $distance > \epsilon$ **then**
4:     DOESFIT()
5: **else**
6:     DOESNOTFIT()
```

7.2.5 Style Transfer

Thus far, our example queries have focused on extracting knowledge. We are, however, also able to input knowledge into our framework using queries. This is important, as an oracle or a planning agent may come to realize something through other means, which it then wishes to share with the framework.

We consider the following situation: a [*stool*] with the attribute {*modern*} produces [*leg*] symbols with a very {*wooden*} appearance and a [*top*] with a {*metallic*} appearance. The oracle has learned that this modern style is not unique to stools and also applies to lamps. So far, however, the capsule network has only encountered [*lamp*] symbols that produce a {*wooden*} [*stand*] and [*shade*] or a {*metallic*} [*stand*] and [*shade*], but never a mixture of both. The oracle may now impose the knowledge of a {*modern*} [*lamp*] through a query, such as in Algorithm 5.

Algorithm 5 Style Transfer

```
1: all-modern-stools ← Replay(origin none, filter [stool] and {modern} $> 0.8$,
                                   map none, group by none)
2: for each modern-stool $\in$ all-modern-stools do
3:     ... ← Fabricate(origin [lamp], change copy attribute values modern-stool)
```

We are in a sense implicitly copying the configuration space topology of one capsule into another by forcing the [*lamp*] capsule to simulate itself with the attributes of a {*modern*} [*stool*]. This copy process may also be applied to entire parts. We may, for example, simulate a lamp with its [*stand*] replaced by a [*leg*] trough fabrication. Here, not only is the style transferred, but also its configuration.

The truly interesting aspect to this is, that we may automate such a query and execute it for a wide range of capsules each time a new style is learned. This would mean an automated style transfer, but with the ability to control exactly the classes of objects it applies to.

7.2.6 Moving and Fixing Errors

For a robotics application, it is useful that a robot is able to simulate an action before executing it, in order to apply the correct movements with adequate forces. This is especially true if it needs to interact with its environment, such as moving an object without damaging it.

In our example, an arm needs to be rotated to a certain *target-angle*. Our framework has seen this arm in action before and learned some of the physics of it. Using this knowledge, the robot is able to perform a mental simulation of the intended action before executing it and assess an error it made afterwards for a possible correction (cf. Algorithm 6).

Algorithm 6 Moving and Fixing Errors

```
1: final-torque ← 0 N·m
2: repeat
3:     final-torque ← final-torque + 0.1 N·m
4:     predicted-obs ← Predict(origin current-obs, duration 5 ms,
                                   force apply final-torque to [arm])
5: until predicted-obs.[arm].{angle} $\geq$ target-angle
6: PerformAction(final-torque)
7: angle ← Replay(origin none, filter newest,
                       map [arm].{angle}, group by none)
8: angle-error ← target-angle - angle
```

7.2.7 Game Engine

As final application that combines all aspects of the framework, we investigate its use as a game engine. Based purely on the observations of its environment, it can simulate both the visual appearance, in addition to the physics, as shown in Algorithm 7.

Algorithm 7 Basic game engine

```
1: game-environment ← FABRICATE(**origin** *real-environment*, **change none**)
2: **while** *game-is-running* **is** *true* **do**
3:     *game-camera*, *modify-objects*, *forces* ← GAMEACTIONS(*game-environment*)
4:     *game-environment* ← FABRICATE(**origin** *game-environment*,
                                      **change apply** *modify-objects*)
5:     *game-environment* ← PREDICT(**origin** *game-environment*, **duration** 5 ms,
                                      **force apply** *forces*)
6:     ..., *frame* ← REPLAY(**origin** *game-environment*,
                          **filter** $arg\,min_{obs} \| obs.camera - game\text{-}camera \|$,
                          **map** ***obs.camera*** ← ***game-camera***, **group by none**)
7:     PRESENT(*frame*)
```

We begin by defining a subset of all observations as the *game-environment* to improve performance. For example, we may restrict the environment to the interior of some apartment. While the game is running, the player performs actions, such as moving around, throwing, adding or removing an object. What actions are performed is given by the GameActions function. Using these, the game environment is altered accordingly and the next time step is predicted. To simulate the player's movement, the observation from *game-environment* is selected whose camera position and rotation fit as good as possible and is used to provide the new parameters for rendering. Finally, the resulting frame is presented to the player (Present) and the next frame of the game loop starts.

7.3 Implementation and Results

We have implemented the inverse game engine as outlined in Algorithm 7 using our VividNet framework and Python mimicking the presented querying language.

Our test is conducted by training our framework on an Asteroids-like environment, letting it observe a limited amount of interactions of asteroids with the space shuttle and other asteroids. We then show it two new frames, in which the space shuttle is not moving, but the asteroids are, as shown in Figure 7.4. These new frames are sufficient for the framework to infer all the required information to begin simulation starting at that point.

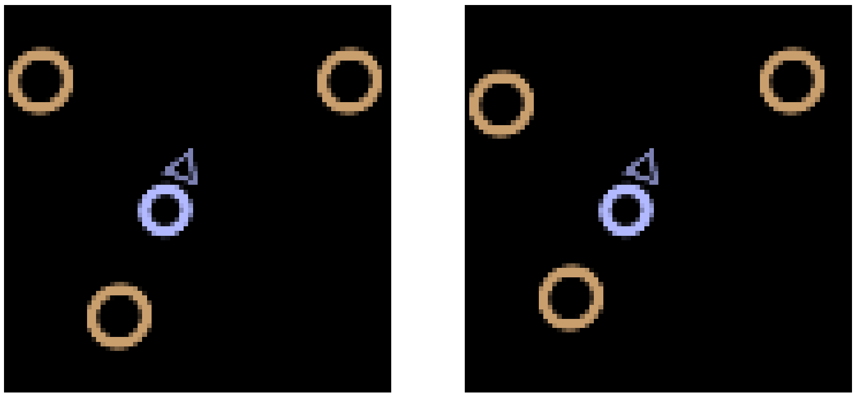


Figure 7.4: The two frames of slightly moving asteroids observed before starting the simulation. Asteroids are shown in brown and the shuttle in blue.

We produce three different simulation scenarios with 14 frames each. The oracle's input only controls the movement of the shuttle and nothing else. The shuttle's physics and all asteroids are simulated based solely on our VividNet framework and its past observations. All output is rendered using the feed-backward process of VividNet.

The first simulation is a simple prediction of what happens if nothing is changed, i.e., the asteroids move on their predicted paths, but the shuttle does not move (cf. Figure 7.5 **(a)**). Here we see the asteroids mutually bouncing off and eventually colliding with the ship, sending it to a new trajectory. One of the asteroids collides two times in this prediction causing this asteroid to grow in size because of the accumulated numerical errors of our implementation of the interaction network's predictor ϕ_R.

We repeat the same simulation, but this time take control of the shuttle and move it out of the way of all of the asteroids to avoid collision (cf. Figure 7.5 **(b)**). All asteroids fly on their expected paths, ignoring the shuttle.

In a final experiment, the shuttle deliberately strikes an asteroid to push it off its flight path (cf. Figure 7.5 **(c)**). The trajectories of the shuttle and asteroid after the collision correspond to the expected outcome.

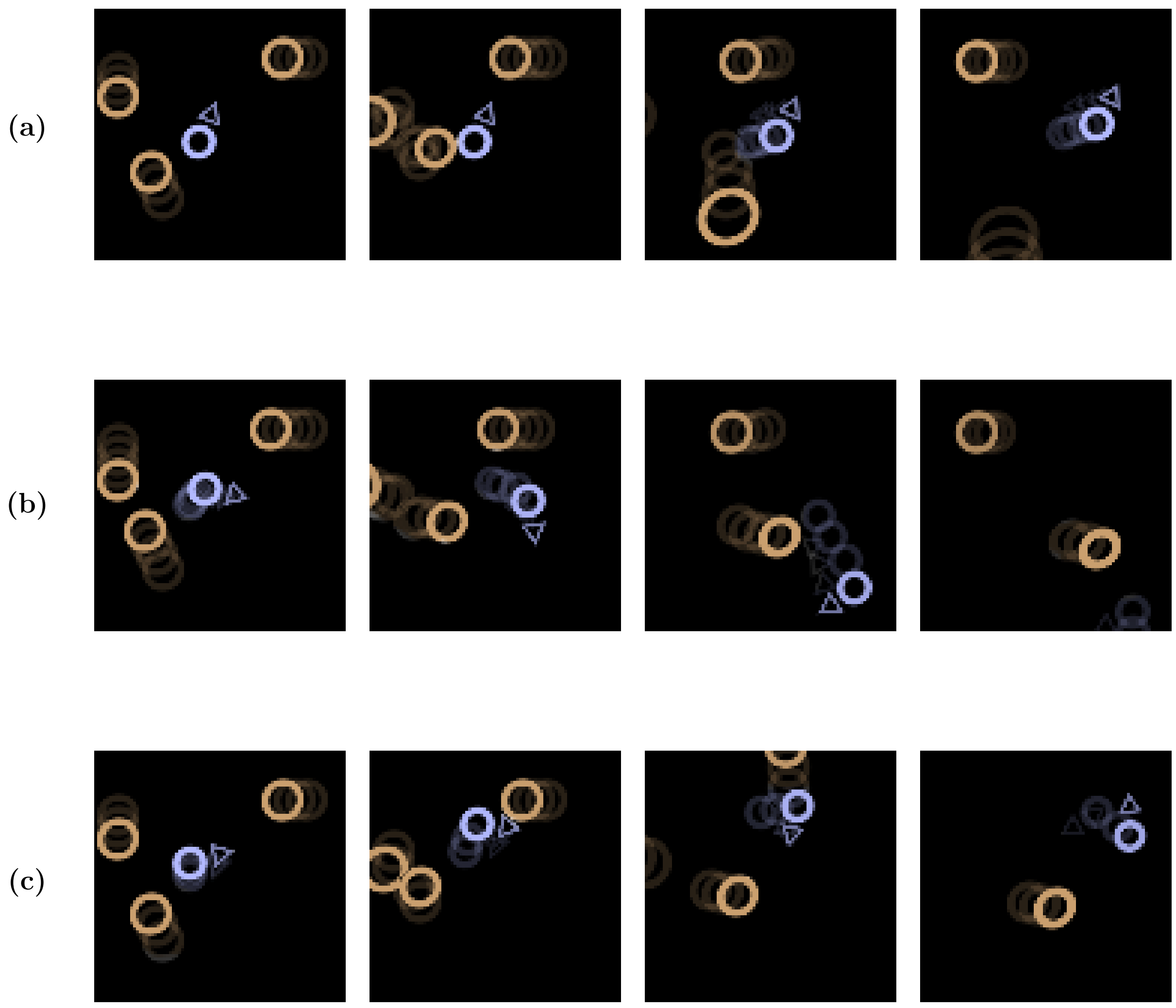


Figure 7.5: Three possible simulation scenarios with different behavior of the shuttle (blue) and its interaction with moving asteroids (brown). **(a)** shows the shuttle not moving, but eventually being hit and moved by an asteroid. **(b)** shows the shuttle moving out of the way of all asteroids. **(c)** shows the shuttle deliberately hitting an asteroid to move it. Trails of past frames are shown.

Chapter 8

Discussion

Our approach differs too much from current classification methods for a reasonable direct numerical comparison. The neural-symbolic capsule network expresses confidence, but has no notion of accuracy, as any inaccuracies are remedied by the meta-learning pipeline over time. A comparison would be to a subjective configuration of the capsule network at that point, eliminating the benefits of lifelong meta-learning.

Because of this, we abstain from a numerical comparison and instead discuss strengths and limitations of our method and compare them to the classical utilization of neural networks.

8.1 Versatility

We begin by giving a short definition of how we use the terms interpretable and explainable, analogous to [Gilpin et al., 2018]:

- Interpretability is to understand how a result is reached and the mechanisms employed on a mathematical or procedural level.
- Explainability allows a model's mechanisms to be explained to and understood by a non-expert.

Even though both terms are often used interchangeably in current literature, the subtle difference is important for our discussion. Often an explainable model is interpretable, but the opposite is seldom true.

8.1.1 Modularity

To avoid the need to constantly redesign and retrain models, the concept of modularity offers a way to simply combine existing parts in a novel way. As we discussed in previous sections, this is one of the key features of our neural-symbolic capsule's meta-learning algorithm. All attributes, routes and capsules can be recombined and allow for a dynamic structure.

For classical neural networks, modularity strongly hinges on interpretability and explainability. An example for interpretable modularity is neural architecture search (NAS) [Elsken et al., 2019]. In NAS, the idea is to have building blocks consisting of smaller neural networks (modules) and to use algorithms, such as evolutionary computation, to find the best architecture for a given use-case. While a NAS can be interpretable, the resulting architecture, however, is seldom explainable. Thus, while highly modular, we are so far unable to infer useful conclusions from the final design and need to repeat the costly process should the conditions change.

Explainable modularity for neural networks would eliminate these downsides. However, we are not aware of any classical architectures that are both modular and explainable. Current efforts have instead been to find explainable architectures and to use transfer learning [Tan et al., 2018] to take advantage of it in other domains.

Another challenge for the explainability of neural networks is the difficulty to pinpoint the neurons or region of neurons responsible for a specific result. This can be, for example, studied using saliency maps [Zeiler and Fergus, 2014, Bach et al., 2015, Bojarski et al., 2017], class activation maps [Zhou et al., 2016] or by identifying critical routing paths in the network [Wang et al., 2018a]. The results are often diffuse and span large regions of the network with only a small contribution from each neuron. This is not necessarily a bad property, because it is argued that this is what makes these networks so powerful by not focusing on exact features and instead also incorporating the context around these features. However, it does make it harder to explain in a reasonable matter why this is happening. With regard to this sort of explainability, Jaeger [2014] showed that it is indeed possible to extract symbolic meaning from the inner workings of recurrent neural networks, but so far this has not been transferred to convolutional neural networks.

In contrast, neural-symbolic capsules allow to precisely identify which region was influenced by which capsule, as the saliency maps (i.e., graphically rendering an object and using its silhouette) are exact. Together with the lexical interpretation of each parameter, we can explain the results in a more complete manner. Further, as each capsule is trained in isolation, we are able to add, remove, modify or move any capsule or attribute in the network, resulting in full modularity with an explainable outcome.

8.1.2 Adaptability

A neural network is believed to generalize well, if it has reached a locally flat minimum [Hinton and Camp, 1993, Hochreiter and Schmidhuber, 1996], where the information in the weights is limited and small perturbations barely influence the loss. New studies [Wu et al., 2017c, Novak et al., 2018, Li et al., 2018] have shown that this intuition goes in the right direction. However, computing the loss landscape for a large model and a wide variety of data points is computationally very expensive. In most cases, such a check is performed only after a model has been found using algorithms (such as NAS) to confirm its feasibility and ability to generalize.

By splitting the monolithic approach to neural networks into smaller networks for individual capsules, an analysis of the loss landscape becomes computationally much more feasible. Furthermore, as it is possible that all capsules use the same architecture internally, we only need to perform such a check on a single or some capsules. Through the modularity discussed above, we are also able to replace any neural network used inside the capsules at any point in the future with one that generalizes better.

Yet, we note that this only covers the generalization of individual capsules and not the capsule network as a whole. The network itself is very limited in its ability to generalize and delegates this task to the meta-learning pipeline. The meta-learning pipeline, however, is able to generalize well, as it simply attempts to learn the features of anything new.

By construction, our neural-symbolic capsules are able to learn the representation of any object or scene if an adequate set of primitive capsules was chosen. This is especially true if an atomic set of primitives was selected (for example edges combined with discrete cosine transforms) which also is suitable for organic surfaces. For the latter, we refer to the work by Omran et al. [2018], which use an architecture similar to that of the routes found in our primitive capsules to perform pose and shape estimation of humans. As such, their implementation could be directly used as a route for a [*human*] primitive capsule. Further, again taking advantage of modularity, it is always possible to add new primitive capsules at a later point by hand, expanding the set of learnable objects.

We also saw that through querying we are able to perform style transfers (cf. Section 7.2.5) and noted how this could also be automated. We consider the ability to explicitly perform style transfers as a strength, as it allows meta-learning or the oracle to control the process more accurately, instead of happening in unpredictable ways.

Our framework in its current state is, however, limited in its ability to handle transparency, refraction and reflection. We believe that this needs to be handled on a meta-level similar to intuitive physics, instead of directly inside our capsules. The objects still get detected, albeit with a different color grading (transparency), wrong position and inverted structure (reflection) or distortions of its shape (refraction), and must be put into the correct context through such a meta-process.

8.1.3 Context

The context of an object in a scene is often important or even necessary to infer its meaning or its type. A table will have the same appearance if it is in a room or in a miniature play-set, but they represent two very distinct object classes. CNNs are able to infer some context from nearby features on different scales, but it has been observed that this can become problematic on larger distances. Recently, there have been efforts to augment these networks to make them more context-aware [Liu et al., 2019b, Alamri and Pugeault, 2019].

In our framework, context is checked and provided by ancestor capsules. In our table example, a [*table*] capsule activates but we do not yet know if it is a life-sized or a play-set table. However, depending on the other objects in the scene, i.e., its context, either the [*room*] or [*play-set*] parent capsule will activate. This provides us with the answer to the context and what kind of table we were looking at.

A more concrete approach is to use a context-sensitive grammar as the basis for the neural-symbolic capsules, which we did not explore here. By context-sensitive we mean a grammar that allows production rules of the form

$$r : \lambda_l \Omega \lambda_r \to \lambda_1 \cdots \lambda_n \quad \text{where } \Omega \in V, \lambda \in \bigcup_{l,r,i \in \mathbb{N}\setminus\{0\}} (V \cup \Sigma) \quad , \tag{8.1}$$

where the left hand side needs a certain context given by λ_l and/or λ_r for the production rule to be valid. Such a grammar would result in a capsule's route not just checking if all parts are in the correct configuration, but also if the route itself fits into the current context. While this is certainly sensible, it is somewhat redundant in the feed-forward process of our capsules. In our context-free construction, the aforementioned route's check is performed by one of its ancestor capsules instead and even though our grammar is "context-free", a lot of the important context is still encoded this way. Yet, in a feed-backward operation, context-sensitivity does indeed become relevant, as it makes the choice of rule/route less ambiguous during a full rendering, producing more accurate images.

Furthermore, context provides additional semantic information by enabling the capsule network to fill in the gaps. As an example, consider a scene with [*house*][*car*][*house*]. If the two [*house*] symbols describe buildings in a wealthy neighborhood using its routes, it is expected that the [*car*] should also be using a route describing an expensive variety. In case the [*car*] should be occluded, through context-sensitivity it is possible to infer properties that might not be visible. This is in a sense akin to the ability to piece together a full picture from incomplete information, i.e., autoassociative memory. In our context-free grammar, we are also able to infer such properties if they are encoded in an attribute {*expensive*}, but not if it is described by a route itself. However, in spite of this we decided not to explore the option of context-sensitive grammars, as it would make the discussion of our approach much more complex.

8.2 The Binding Problem

In cognitive science, the binding problem [Revonsuo, 1999] plays an important role and has also gained some interest in the field of deep learning. Here, we address a part of the binding problem known as the segregation problem, which asks the following question:

> Which internal neurons or mechanisms encode the features of external physical objects?

This is similar to explainability in deep learning with a focus on identifying regions in the network that are specific to an object, as discussed in Section 8.1.1. However, it goes a step further, as the binding problem incorporates all sensory inputs. Two individual neural networks, one for image classification and one for sound classification, will even when running at the same time have their own understanding of the concept "dog" and there is no binding between the two. In such a case, the binding is performed using human input in hindsight.

We have addressed the binding problem early in the design of the capsules by letting each represent one specific symbol. As an example, let us assume we have a primitive capsule for every character $[a]$ to $[z]$. We require a semantic capsule for every word in a text. Yet, instead of adding a new capsule for a new word, we may also add it as a route to the existing semantic capsule that represents the object. For example, a $[cat]$ capsule may have a route for its visual representation and one for its text representation. The upside of this is that we have an explicit correspondence between the word and its image, something that is lacking in current neural network approaches. We may even go a step further and name the capsule itself by its text representation instead of querying the oracle, strengthening the autonomy of our meta-learning pipeline, as there is no need to ask for a name for every object.

Further, we may also add routes to a capsule that take an audio signal as input. A cow's "moo" and its image are then processed in two different routes of the same $[cow]$ capsule, which effectively binds these two concepts to the same physical entity. This is also the reason why we chose to incorporate the possibility of different inputs to our framework in Figure 1.1 in the introduction of this thesis.

8.3 Training

Training a neural network usually requires vast amounts of labeled data. Once it is finished, the network can be used in a production setting. For CNNs used in image classification tasks, this usually means fixing it in its current state and, thus, keeping it from learning anything new. Methods have been derived to overcome this problem, such as transfer learning [Tan et al., 2018] and lifelong machine-learning [Parisi et al., 2019]. However, most of these methods require large amounts of fully annotated data for continued learning and, generally, are not capable to learn directly from their environment.

Our neural-symbolic capsule network is designed to learn primarily through observation of its environment and the interaction with the oracle in the early phase. Through the proposed augmentation strategy we are able to do this with a one-shot / few-shot approach. Learning through observation of the environment is neither better nor worse than through pre-annotated data. On one hand, we forgo the costly process of hand-labeling data, on the other hand, we need to have an oracle present and a predefined set of primitive capsules. Simard et al. [2017] propose that the future of machine learning lies in simplifying the teaching process and making it accessible to non-experts. For this, our framework would be very suitable, as it asks simple-to-answer questions and handles all complex issues internally. Depending on the application, learning from large amounts of pre-annotated data might be preferred over the interaction with an oracle or vice versa. For example, a classifier intended to process well-defined data will not benefit as much from our approach compared to a robotics system that may find itself in unexpected situations.

One major hindrance for any neural network that attempts lifelong learning is catastrophic interference [McCloskey and J.Cohen, 1989, Ratcliff, 1990], i.e., the possibility of unlearning previous behavior through further training. A capsule's route based on neural networks is as susceptible to this phenomenon as any other network. However, as we only employ small neural networks and each capsule acts in isolation, capsules are only affected on an individual basis instead of on the network level and we can forgo the problem of catastrophic forgetting in an economical way, by retraining it from scratch or through rehearsing [Robins, 1995, 1996].

8.4 Performance

As our meta-learning pipeline adapts to a wide range of different scenes throughout its lifetime, the capsule network will add more and more semantic capsules. For a wider variety of scenes a network will grow both in width and depth, clustering related concepts (cf. Figure 8.1). For a particular scene, only capsules that are relevant to it need to activate which reduces the overall amount of computation required. This is in contrast to a block-style end-to-end network of neurons, where every node needs to be considered, even if some regions only give a negligible contribution to the result.

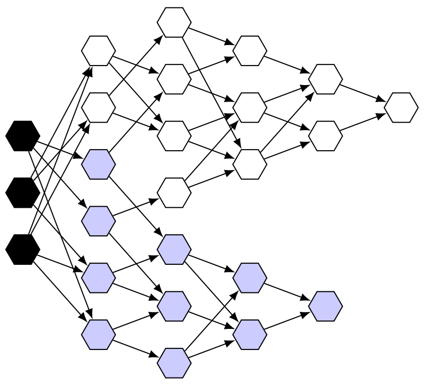


Figure 8.1: Only a part of the capsule network activates for each scene and needs computational resources. Primitive capsules are shown in black, activated semantic capsules in blue.

On the other hand we note, that while a classical neural network requires the computation of all nodes, it does not necessarily make it slower or use more resources than our framework. As there is essentially no branching taking place during execution, it is GPU-friendly and can be broken down mainly into simple linear algebra operations. With our framework, we do not only require branching operations, but also (in the case of a single GPU) context-switching between rendering and compute capabilities, as well as potentially switching to different kernels for each capsule. Here, our neural-symbolic capsules benefit from a multi-GPU setup not just from a raw computational perspective, but also by being able to dedicate specific tasks to each node.

Chapter 9

Conclusion

In this thesis, we presented a novel framework for mental simulation based on our interpretation of capsules using a neural-symbolic hybrid method that addresses the problems outlined in the introduction and repeated in the following. Through our VividNet implementation, we showed that our approach works on toy examples and is able to infer both the semantic information found in the scene as well as physical interactions (Problem 1: Unified Semantic Information). Based on the extracted knowledge, it is able to simulate the environment in different settings given by our querying algorithm (Problem 2: Querying and Simulation). Through the entire process, the binding between the semantic and visual representation is always clear and comprehensible (Problem 3: Explainability). We also showed that the meta-learning algorithm is capable of learning new semantics at any point in time by either inferring it automatically or asking the oracle about unknown elements in the scene (Problem 4: Lifelong Meta-Learning). Finally, we discussed the advantages and shortcomings of our approach and compared it, where possible, with classical neural networks.

Our neural-symbolic framework also tackles some of the other problems that plague deep learning in a novel way. We showed how our meta-learning process avoids the need for large amounts of data and how it can reduce the effects of catastrophic forgetting. Through the versatile nature of our capsule's routes, we were able to tackle the binding problem but also could explore how our framework is naturally invertible to be used both for rendering as well as inverse-graphics.

Looking forward, the true test for our framework would be to integrate it with an algorithm capable of replacing the oracle, such as the Rosie Soar cognitive architecture [Laird, 2012, Kirk and Laird, 2014], an imagination-based planner [Pascanu et al., 2017] or more general reinforcement learning algorithms. In such a pairing, the framework could potentially become fully unsupervised in its learning behavior. It would also be interesting to explore how the querying process can be further automated using recurrent neural networks, such as a neural Turing machine [Graves et al., 2014], or if it can be paired with a partially observable Markov decision process [Hamrick, 2017]. Further, disentangled variational auto-encoders [Kingma and Welling, 2014, Higgins et al., 2017] could provide an interesting alternative for the capsule's internal encoder-decoder pair we currently employ. Also, our implementation has mainly focused on 2D toy examples and it would be interesting to see its performance on 3D datasets, such as CLEVR [Johnson et al., 2017].

Finally, we believe that our approach is a suitable interface for any agent with its environment, so that this agent can perform all its actions based on purely symbolic information, i.e., as if in a classic symbolic artificial intelligence setting. As such, we hope to help bridge the gap between the connectionist and the symbolic approach to artificial intelligence.

Bibliography

ACCV	Asian Conference on Computer Vision
CVPR	Conference on Computer Vision and Pattern Recognition
ECCV	European Conference on Computer Vision
GCPR	German Conference on Pattern Recognition
ICANN	International Conference on Artificial Neural Networks
ICCV	International Conference on Computer Vision
ICLR	International Conference on Learning Representations
ICML	International Conference on Machine Learning
JMLR	Journal of Machine Learning Research
NeurIPS	Conference on Neural Information Processing Systems
SIGGRAPH	Special Interest Group on Computer Graphics and Interactive Techniques

A. Achille and S. Soatto. Emergence of invariance and disentanglement in deep representations. *JMLR*, 19(1):1947–1980, 2018.

F. Alamri and N. Pugeault. Contextual relabelling of detected objects. *International Conference on Development and Learning and on Epigenetic Robotics*, pages 313–319, 2019.

M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. *CVPR*, 2019.

D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *arXiv:1606.06565*, 2016.

S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *Public Library of Science ONE 10*, pages 1–46, 2015.

B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *ICLR*, 2017.

P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *NeurIPS*, pages 4509–4517, 2016.

P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45): 18327–18332, 2013.

T. R. Besold, A. d. Garcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K.-U. Kuehnberger, L. C. Lamb, D. Lowd, P. M. V. Lima, L. d. Penning, G. Pinkas, H. Poon, and G. Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv:1711.03902*, 2017.

M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv:1704.07911*, 2017.

M. Bokeloh, M. Wand, and H.-P. Seidel. A connection between partial symmetry and inverse procedural modeling. *ACM Transactions on Graphics*, 29(4):104–114, 2010.

D. Burago, Y. Burago, and S. Ivanov. *A Course in Metric Geometry*. American Mathematical Society - Graduate Studies in Mathematics, 2001.

D. P. Buxhoeveden and M. F. Casanova. The minicolumn hypothesis in neuroscience. *Brain*, 125(5):935–951, 2002.

F. Calakli and G. Taubin. SSD: Smooth signed distance surface reconstruction. *Computer Graphics Forum*, 30(7):1993–2002, 2011.

T. Chan and W. Zhu. Level set based shape prior segmentation. *CVPR*, 2005.

A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. *arXiv:1512.03012*, 2015.

M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *ICLR*, 2017.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. *NeurIPS*, page 2180–2188, 2016.

Z. Chen and B. Liu. *Lifelong Machine Learning*. Morgan and Claypool Publishers, 2018.

A. Dame, V. A. Prisacariu, C. Y. Ren, and I. Reid. Dense reconstruction using 3D object shape priors. *CVPR*, 2013.

J. J. DiCarlo and D. D. Cox. Untangling invariant object recognition. *Trends in Cognitive Science*, 11(8):333–341, 2007.

A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. *CVPR*, pages 4829–4837, 2016.

S. Ehrhardt, A. Monszpart, N. J. Mitra, and A. Vedaldi. Learning a physical long-term predictor. *arXiv:1703.00247*, 2017.

S. Ehrhardt, A. Monszpart, N. Mitra, and A. Vedaldi. Unsupervised intuitive physics from visual observations. *ACCV*, pages 700–716, 2018.

A. El-Nouby, S. Sharma, H. Schulz, D. Hjelm, L. E. Asri, S. E. Kahou, Y. Bengio, and G. W. Taylor. Tell, draw, and repeat: Generating and modifying images based on continual linguistic instruction. *ICCV*, 2019.

T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *JMLR*, 20:1–21, 2019.

K. Falconer. *Fractal geometry: mathematical foundations and applications*. Wiley, 1990.

P. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

R. A. Finke and K. Slayton. Explorations of creative visual synthesis in mental imagery. *Memory and Cognition*, 16(3):252–257, 1988.

O. Gafni, L. Wolf, and Y. Taigman. Vid2Game: Controllable characters extracted from real-world videos. *arXiv:1904.08379*, 2019.

A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. *CVPR*, pages 4340–4349, 2016.

A. S. d. Garcez, L. C. Lamb, and D. M. Gabbay. Neural-symbolic cognitive reasoning. *Springer*, 2009.

R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *ICLR*, 2019.

T. S. Gendler. Galileo and the indispensability of scientific thought experiment. *The British Journal for the Philosophy of Science*, 49(3):397–424, 1998.

L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. Explaining explanations: An overview of interpretability of machine learning. *IEEE International Conference Data Science and Advanced Analytics*, pages 80–89, 2018.

Godot Engine Team. Godot Engine, 2020. URL `https://godotengine.org`.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *NeurIPS*, pages 2672–2680, 2014.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, L. Wang, G. Wang, J. Cai, and T. Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

J. B. Hamrick. Metareasoning and mental simulation. *PhD Thesis – UC Berkeley*, 2017.

J. B. Hamrick, A. J. Ballard, R. Pascanu, O. Vinyals, N. Heess, and P. W. Battaglia. Metacontrol for adaptive imagination-based optimization. *ICLR*, 2017.

J. C. Hart. Ray tracing implicit surfaces. *SIGGRAPH 1993 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, pages 1–16, 1993.

J. C. Hart, D. J. Sandin, and L. H. Kauffman. Ray tracing deterministic 3-D fractals. *Computer Graphics*, 23(3):289–296, 1989.

F. Hausdorff. Dimension und äußeres maß. *Mathematische Annalen*, 79:157–179, 1918.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2017.

I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, and A. Lerchner. Towards a definition of disentangled representations. *arXiv:1812.02230*, 2018.

G. E. Hinton. What is wrong with convolutional neural nets? Presented at the Brain and Cognitive Sciences Fall Colloquium, 2014.

G. E. Hinton and D. v. Camp. Keeping the neural networks simple by minimizing the description length of the weights. *Computational Learning Theory*, pages 5–13, 1993.

G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. *International Conference on Artificial Neural Networks*, pages 44–51, 2011.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

G. E. Hinton, S. Sabour, and N. Frosst. Matrix capsules with EM routing. *ICLR*, 2018.

C. Häne, C. Zach, A. Cohen, and M. Pollefeys. Dense semantic 3D reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1730–1743, 2017.

S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1996.

D. A. Hudson and C. D. Manning. Compositional attention networks for machine reasoning. *ICLR*, 2018.

R. Iten, T. Metger, H. Wilming, L. d. Rio, and R. Renner. Discovering physical concepts with neural networks. *arXiv:1807.10300*, 2018.

H. Izadinia, Q. Shan, and S. Seitz. IM2CAD. *CVPR*, pages 5134–5143, 2017.

H. Jaeger. Controlling recurrent neural networks by conceptors. *Jacobs University Technical Report Nr 31*, 2014.

H. Jin, Q. Song, and X. Hu. Auto-Keras: Efficient neural architecture search with network morphism. *arXiv:1806.10282*, 2018.

J. Johnson, B. Hariharan, L. v. d. Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CVPR*, 2017.

M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *Robotics and Automation (ICRA)*, pages 746–753, 2017.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *ICLR*, 2014.

T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. *ICML*, 2018.

J. R. Kirk and J. E. Laird. Interactive task learning for simple games. *Advances in Cognitive Systems*, pages 13–30, 2014.

M. Kissner and H. Mayer. Adding intuitive physics to neural-symbolic capsules using interaction networks. *arXiv:1905.09891*, 2019a.

M. Kissner and H. Mayer. A neural-symbolic architecture for inverse graphics improved by lifelong meta-learning. *GCPR*, pages 471–484, 2019b.

H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

A. R. Kosiorek, S. Sabour, Y. W. Teh, and G. E. Hinton. Stacked capsule autoencoders. *NeurIPS*, 2019.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *NeurIPS*, pages 1097–1105, 2012.

T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. B. Tenenbaum. Deep convolutional inverse graphics network. *NeurIPS*, pages 2539–2547, 2015.

J. E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.

Y. LeCun, B. Boser, J. S. Decker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. E. Lenssen, M. Fey, and P. Libuschewski. Group equivariant capsule networks. *NeurIPS*, 2018.

A. Lerer, S. Gross, and R. Fergus. Learning physical intuition of block towers by example. *ICML*, 48:430–438, 2016.

M. Leyton. *A Generative Theory of Shape*. Springer, 2001.

H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *NeurIPS*, 2018.

Z. C. Lipton. The mythos of model interpretability. *Queue*, 16(3), 2018.

Y. Liu, Z. Wu, D. Ritchie, W. T. Freeman, J. B. Tenenbaum, and J. Wu. Learning to describe scenes with programs. *ICLR*, 2019a.

Z. Liu, W. T. Freeman, J. B. Tenenbaum, and J. Wu. Physical primitive decomposition. *ECCV*, 2018.

Z. Liu, Z. Jiang, W. Feng, and H. Feng. OD-GCN: Object detection boosted by knowledge GCN. *arXiv:1908.04385*, 2019b.

L. v. d. Maaten, E. Postma, and J. v. d. Herik. Dimensionality reduction: A comparative review. *Tilburg Centre for Creative Computing*, TR 2009–005, 2009.

A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. *CVPR*, pages 5188–5196, 2015.

S. Mahendran, H. Ali, and R. Vidal. 3D pose regression using convolutional neural networks. *ICCV*, pages 2174–2182, 2017.

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

A. Marchisio, M. A. Hanif, and M. Shafique. CapsAcc: An efficient hardware accelerator for capsulenets with data reuse. *Design, Automation and Test in Europe*, pages 964–967, 2019.

A. Martinovic. Inverse procedural modeling of buildings. *PhD Thesis - KU Leuven*, 2015.

A. Martinovic and L. V. Gool. Bayesian grammar learning for inverse procedural modeling. *CVPR*, pages 201–208, 2013.

M. McCloskey and N. J.Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.

G. F. Miller, P. M. Todd, and S. U. Hedge. Designing neural networks using genetic algorithms. *Proceedings of the third international conference on Genetic algorithms*, pages 379–384, 1989.

G. Montavon, W. Samek, and K.-R. Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

M. Mueller, V. Casser, J. Lahoud, N. Smith, and B. Ghanem. Sim4CV: A photo-realistic simulator for computer vision applications. *International Journal of Computer Vision*, 126(9):902–919, 2018.

M. Nakahara. *Geometry, Topology and Physics.* Institute of Physics Publishing, 2003.

T. Nguyen-Phuoc, C. Li, L. Theis, C. Richardt, and Y.-L. Yang. HoloGAN: Unsupervised learning of 3D representations from natural images. *ICCV*, 2019.

O. Niemitalo. A method for training artificial neural networks, 2010. URL `http://yehar.com:80/blog/?p=167`.

R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *ICLR*, 2018.

Nvidia Corporation. PhysX, 2018. URL `https://developer.nvidia.com/gameworks-physx-overview`.

M. Omran, C. Lassner, G. Pons-Moll, P. V. Gehler, and B. Schiele. Neural body fitting: Unifying deep learning and model-based human pose and shape estimation. *International Conference on 3D Vision*, pages 484–494, 2018.

G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia. Learning model-based planning from scratch. *arXiv:1707.06170*, 2017.

X. Peng, B. Sun, K. Ali, and K. Saenko. Learning deep object detectors from 3D models. *ICCV*, pages 1278–1286, 2015.

M. Pharr, G. Humphreys, and W. Jakob. *Physically Based Rendering 3rd Edition*. Morgan Kaufmann, 2016.

I. Quílez. Rendering signed distance fields, 2017. URL `http://www.iquilezles.org`.

D. Raposo, A. Santoro, D. Barrett, R. Pascanu, T. Lillicrap, and P. Battaglia. Discovering objects and their relations from entangled scene representations. *ICLR*, 2017.

R. Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308, 1990.

S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee. Deep visual analogy-making. *NeurIPS*, pages 1252–1260, 2015.

A. Revonsuo. Binding and the phenomenal unity of consciousness. *Consciousness and Cognition*, 8:173–185, 1999.

M. T. Ribeiro, S. Singh, and C. Guestrin. “Why should I trust you?” explaining the predictions of any classifier. *Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

L. G. Roberts. Machine perception of three-dimensional solids. *PhD Thesis*, 1963.

A. Robins. Catastrophic forgetting, rehearsal, and pseudorehearsal. *Connection Science*, 7:123–146, 1995.

A. Robins. Consolidation in neural networks and in the sleeping brain. *Connection Science*, 8(2):259–276, 1996.

L. Romaszko, C. K. Williams, and J. Winn. Learning direct optimization for scene understanding. *arXiv:1812.07524*, 2018.

S. Russel and P. Novig. *Artificial Intelligence: A Modern Approach*. Pearson, 2010.

S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. *NeurIPS*, pages 3859–3869, 2017.

A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. *ICML*, 2018.

L. K. Saul and S. T. Roweis. An introduction to locally linear embedding, 2001. URL `https://cs.nyu.edu/~roweis/lle/`.

J. Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879, 1992.

S. Shah, D. Dey, C. Lovett, and A. Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. *Conference on Field and Service Robotics*, pages 621–635, 2017.

P. Y. Simard, S. Amershi, D. M. Chickering, A. E. Pelton, S. Ghorashi, C. Meek, G. Ramos, J. Suh, J. Verwey, M. Wang, and J. Wernsing. Machine teaching: A new paradigm for building machine learning systems. *arXiv:1707.06742*, 2017.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034*, 2014.

S. Song and J. Xiao. Deep sliding shapes for amodal 3D object detection in RGB-D images. *CVPR*, pages 808–816, 2016.

O. Stava, S. Pirk, J. Kratt, B. Chen, R. Mech, O. Deussen, and B. Benes. Inverse procedural modeling of trees. *Computer Graphics Forum*, 33(6):118–131, 2014.

S. v. Steenkiste, M. Chang, K. Greff, and J. Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *ICLR*, 2018.

B. Sun and K. Saenko. From virtual to reality: Fast adaptation of virtual object detectors to real domains. *Proceedings of the British Machine Vision Conference 2014*, 2014.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1998.

R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. *ICANN*, pages 270–279, 2018.

O. Teboul, L. Simon, P. Koutsourakis, and N. Paragios. Segmentation of building facades using procedural shape priors. *CVPR*, pages 3105–3112, 2010.

J. B. Tenenbaum, V. d. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Y. Tian, A. Luo, X. Sun, K. Ellis, W. T. Freeman, J. B. Tenenbaum, and J. Wu. Learning to infer and execute 3D shape programs. *ICLR*, 2019.

J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *International Conference on Intelligent Robots and Systems*, pages 23–30, 2017.

G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.

G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119–165, 1994.

J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. *CVPR*, 2018.

S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik. Learning shape abstractions by assembling volumetric primitives. *CVPR*, pages 1466–1474, 2017.

T. D. Ullman, E. Spelke, P. Battaglia, and J. B. Tenenbaum. Mind games: Game engines as an architecture for intuitive physics. *Trends in Cognitive Science*, 21(9):649–665, 2017.

Y. Wang, H. Su, B. Zhang, and X. Hu. Interpret neural networks by identifying critical data routing paths. *CVPR*, pages 8906–8914, 2018a.

Y. Wang, X. Tan, Y. Yang, X. Liu, E. Ding, F. Zhou, and L. S. Davis. 3D pose estimation for fine-grained object categories. *ECCV*, pages 619–632, 2018b.

T. B. Ward. Structured imagination: the role of category structure in exemplar generation. *Cognitive Psychology*, 27(1):1–40, 1994.

N. Watters, A. Tacchetti, T. Weber, R. Pascanu, P. Battaglia, and D. Zoran. Visual interaction networks: Learning a physics simulator from video. *NeurIPS*, pages 4539–4547, 2017.

M. Werner. Indoor location-based services. *Springer*, 2014.

T. Willwacher. Basic group and representation theory. *ETH Zürich - Lecture*, 2014.

J. Wu, E. Lu, P. Kohli, W. T. Freeman, and J. B. Tenenbaum. Learning to see physics via visual de-animation. *NeurIPS*, pages 153–164, 2017a.

J. Wu, J. B. Tenenbaum, and P. Kohli. Neural scene de-rendering. *CVPR*, pages 7035–7043, 2017b.

L. Wu, Z. Zhu, and W. E. Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv:1706.10239*, 2017c.

J. Yang, S. Reed, M.-H. Yang, and H. Lee. Weakly-supervised disentangling with recurrent transformations for 3D view synthesis. *NeurIPS*, pages 1099–1107, 2015.

S. Yao, T.-M. H. Hsu, J.-Y. Zhu, J. Wu, A. Torralba, W. T. Freeman, and J. B. Tenenbaum. 3D-aware scene manipulation via inverse graphics. *NeurIPS*, 2018.

K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, and J. B. Tenenbaum. Neural-symbolic VQA: Disentangling reasoning from vision and language understanding. *NeurIPS*, 2018.

J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. *ICLR*, 2018.

J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *NeurIPS*, pages 3320–3328, 2014.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *ECCV*, pages 818–833, 2014.

Q. Zhang, Y. N. Wu, and S.-C. Zhu. Interpretable convolutional neural networks. *CVPR*, pages 8827–8836, 2018.

Y. Zhao, T. Birdal, H. Deng, and F. Tombari. 3D point-capsule networks. *CVPR*, 2019.

D. Zheng, V. Luo, J. Wu, and J. B. Tenenbaum. Unsupervised learning of latent physical properties using perception-prediction networks. *Conference on Uncertainty in Artificial Intelligence*, 2018.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. *CVPR*, pages 2921–2929, 2016.

Y. Zhou, Z. Zhu, X. Bai, D. Lischinski, D. Cohen-Or, and H. Huang. Non-stationary texture synthesis by adversarial expansion. *SIGGRAPH*, 37(4):49, 2018.

C. Zou, E. Yumer, J. Yang, D. Ceylan, and D. Hoiem. 3D-PRNN: Generating shape primitives with recurrent neural networks. *ICCV*, 313:504–507, 2017.