Visual Parsing and Reconstruction with Stochastic Grammars and Recursive Networks

B.Tech Project Report

by

Uday Kusupati(140050048)

under the guidance of

Prof. Siddhartha Chaudhuri

Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai 400 076

Abstract

The inherent hierarchical nature of the visual data we interact with it can be put to very good use for generation of new data as well as reconstruction of existing data. These global level relationships captured through grammars and conditional models help reconstruct incomplete data, even by learning from very less training data. The aim of our project is to exploit these relationships to obtain a compact but efficient model or in other words a very high level understanding of scenes/hierarchical structures.

Acknowledgments

I wish to sincerely thank my guide Prof. Siddhartha Chaudhuri for his guidance and unwavering support, without whom this endeavour would have remained fruitless.

Uday Kusupati(140050048)
IIT Bombay
May 09, 2018

Contents

Abstract i

Acknowledgments ii

1 Introduction 1

2 Literature Survey 2
- 2.1 Creating Consistent Scene Graphs Using a Probabilistic Grammar . 2
- 2.2 Probabilistic Reasoning for Assembly-Based 3D Modeling 3
- 2.3 GRASS: Generative Recursive Autoencoders for Shape Structures . 5

3 Use of Probabilistic Graphical Models with Neural Networks 8
- 3.1 Method 8
  - 3.1.1 Parsing 9
  - 3.1.2 Learning parameters from data 10
- 3.2 Inference at test time 12
  - 3.2.1 Parsing: 12
  - 3.2.2 Reconstruction: 12

4 Implementation Details 13

5 Recursive Neural Networks to model conditional relationships 14
- 5.1 Training 14
  - 5.1.1 Toy Grammar 15
- 5.2 Testing 16
  - 5.2.1 Parsing 16
  - 5.2.2 Merge Estimators 16
  - 5.2.3 Local Reconstructions 17
  - 5.2.4 Results 21

5.3 Reconstruction of missing leaves 23
5.3.1 Training 23
5.3.2 Testing 24
5.3.3 Results 25

6 Conclusion and Future Work **27**

Chapter 1

Introduction

Much of the visual data we interact with have an inherent hierarchical structure embedded within. We intend to learn these hierarchical relations efficiently to parse and reconstruct scenes or any hierarchical structure with discrete entities. We use probabilistic grammars with the conditional relationships between the parent and child modeled by neural networks to parse and thereafter reconstruct the complete structure. To train the neural networks and learn to model the conditional relationships efficiently, we use various Bayesian inference techniques and optimization methods. Parsing a scene is more difficult than parsing a sequential structure like a sentence. To tackle this, we intend to exploit the conditional relationships between the features of various nodes in the hierarchy through graphical models or through encoding methods that model the global information using the lower-level/local information.

Chapter 2

Literature Survey

2.1 Creating Consistent Scene Graphs Using a Probabilistic Grammar

The paper by Liu et al. [Liu+14] aims to develop algorithms that build a consistent representation for the hierarchical decomposition of a scene into semantic components. It is done in two stages. First, given a collection of consistently-annotated scene graphs representing a category of scenes (e.g., bedroom, library, classroom, etc.), they learn a probabilistic hierarchical grammar that captures the scene structure. Second, they use the learned grammar to hierarchically segment and label newly downloaded scenes. Let us deal with the second part that is given a probabilistic hierarchical grammar, how do we parse/ hierarchically segment and label new scenes.

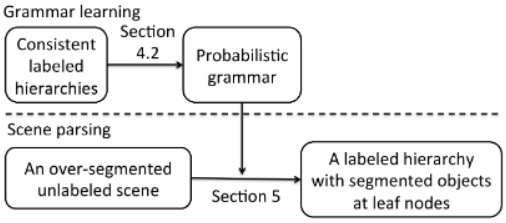
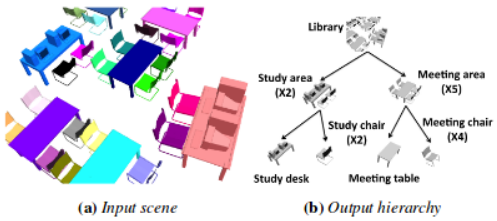


Figure 2: *Flow chart of our approach. We learn a probabilistic grammar from consistently annotated training hierarchies. We then leverage this grammar to parse new scenes (which might include over-segmented objects). The output is a labeled hierarchy consistent with the grammar and assigned a high probability by it.*



(a) *Input scene* **(b)** *Output hierarchy*

Figure 3: *An example library scene. By grouping objects, we are not only able to detect interesting intermediate-level structures, e.g. study area and meeting area, but also distinguish objects based on their functionalities, e.g. study chair and meeting chair.*

Figure 2.1

Given a learned grammar G and an input scene graph S, the goal is to produce an annotated hierarchy H on scene geometry that is a valid parse of S according to G. They first extract the set of leaf nodes S_{leaf} from S, which forms a partition of the scene. These leaves do not necessarily correspond to semantic objects or groups. We assume that H has S_{leaf} as leaf nodes, assigning them special object-subpart labels from the grammar in the case of oversegmentation.

Given a grammar G and scene S, the goal is to produce an annotated hierarchy $H^* = argmax_H P(H|S, G)$. This probability is re-written using bayes rule as product of $P(H|G)$ and $P(S|H, G)$, which depend on the rule generation probabilities and the data likelihood respectively. The rule generation probabilities can be obtained from the learnt grammar and the data likelihood is the product of per-node likelihoods which are derived from the geometric features. The negative log of this likelihood is modeled as energy to be minimized.

To achieve this, they propose candidate groups for grouping/merging at the lower levels to avoid exponential blow-up. The candidate groups are generated using a heuristic that only shapes that are close to one another produce semantically meaningful groups. Also they simplify the grammar by binarization i.e splitting every longer rule into sets of binary rules. Finally the solution space is explored by dynamic programming over the energies of the possible subtrees.

2.2 Probabilistic Reasoning for Assembly-Based 3D Modeling

In this paper by Chaudhuri et al. [Cha+11], they use a probabilistic graphical model called a Bayesian network to represent semantic and stylistic relationships between components in a shape database. When new models are assembled, inference in the Bayesian network is used to derive a relevance ranking on categories of components, and on individual components within each category. For example, when a user begins a modeling session by placing an airplane fuselage, the probabilistic model identifies components in the repository that are more likely to be adjacent to a fuselage, such as wings, stabilizers, and engines. The presented components continue to be dynamically updated as the current model is constructed.

The Probabilistic Model:

The probabilistic model is trained on a set of segmented and labeled shapes. Each

shape is represented by the existence, adjacencies, geometric style, symmetries, and number of components from each category. These attributes are represented as random variables. The probabilistic model encodes the joint distribution $P(X)$. The purpose of the model is to support estimation of compatibility between each component from the repository and a given 3D model. The given model is the current 3D model observed during the runtime stage. The given model imposes specific values on some random variables in X. For example, if the current model contains a component from a particular category or style, the corresponding random variables are set appropriately. Such variables are said to be observed. The rest of the random variables are unobserved. In addition, the cardinality random variables are never observed, since we also do not know the final number of components in the assembled shape; although unobserved, cardinality variables are important for querying the model. By performing inference in the probabilistic model, we will compute probability distributions on some of the unobserved variables given the observed ones, and thus estimate the compatibility of various components with the currently assembled shape.

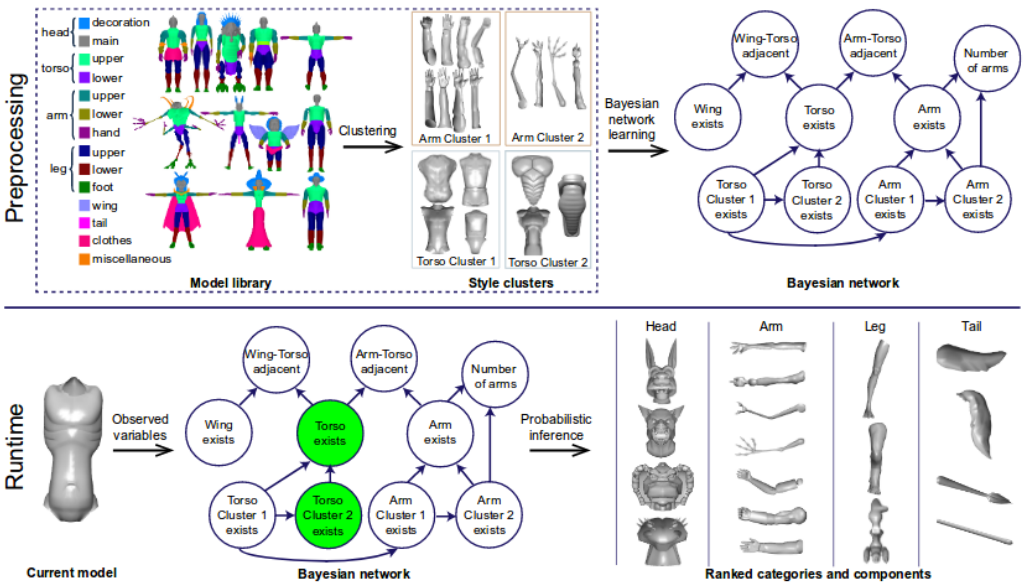


Figure 2: *Overview of our approach. The* **preprocessing** *stage (top) begins with a library of models, segmented and labeled using the technique of Kalogerakis et al. [2010]. The components extracted from the models are further clustered by geometric style. A Bayesian network is then learned that encodes probabilistic dependencies between labels, geometric styles, part adjacencies, number of parts from each category, and symmetries. The figure shows a subset from a real network learned from a library of creature models. The* **runtime** *stage (bottom) performs probabilistic inference in the learned Bayesian network to generate ranked lists of category labels and components within each category, customized for the currently assembled model.*

Figure 2.2: Bayes net structure and Inference

2.3 GRASS: Generative Recursive Autoencoders for Shape Structures

In this paper by Li et al. [Li+17], the key insight is that most shape structures are naturally hierarchical and hierarchies can jointly encode structure and geometry. Most importantly, regardless of the variations across shape structures, a coding scheme that recursively contracts hierarchy or tree nodes into their parents attains unification at the top any finite set of structures eventually collapses to root node codes with a possibly large but fixed length. They learn a neural network which can recursively encode hierarchies into root codes and invert the process via decoding. Then, by further learning a distribution over the root codes for a class of shapes, new root codes can be generated and decoded to synthesize new structures and shapes in that class. In particular, the training and testing setups are described in particular below.

For training, the input hierarchy is used to merge nodes and appropriately they use the corresponding encoder(adjacency/symmetry). The entire tree is formed by encoding till root and the root is recursively decoded to give a decoded tree. Loss is formulated as reconstruction error and the entire encoder/decoder tree is trained through end-to-end back-propagation. During testing, greedy strategy is adopted for recursive merging of nodes to apply encoders upon and while decoding, a node classifier that was pre-trained to detect the decoder to use to expand a code is used at every level to decode the code.

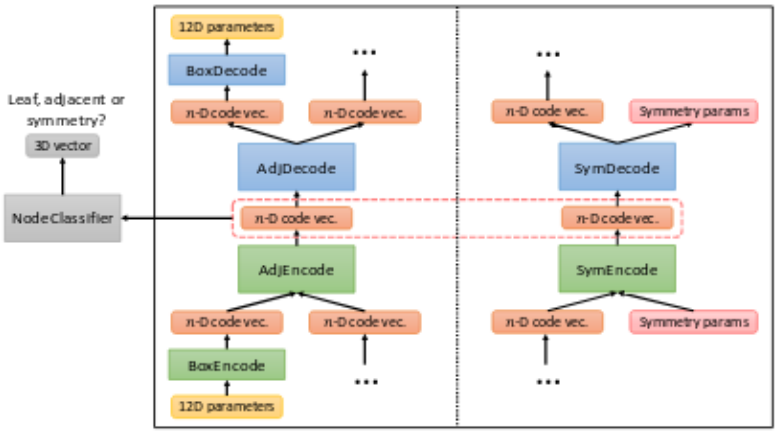


Fig. 4. Autoencoder training setup. Ellipsis dots indicate that the code could be either the output of BoxEnc, AdjEnc or SymEnc, or the input to BoxDec, AdjDec or SymDec.

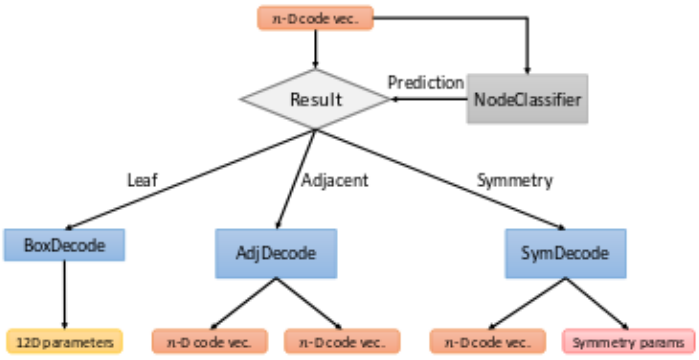


Fig. 5. Autoencoder test decoding setup.

Figure 2.3: Training and Testing Setups

One more interesting thing is they use this autoencoder architecture to build a generative adversarial network that generates a novel shape given a root code trained using a discriminator that outputs the real/fake nature of the structure. The loss of discriminator trains the discriminator to recognize real structures from fake structures . At the same time it trains the generator to produce better structures.

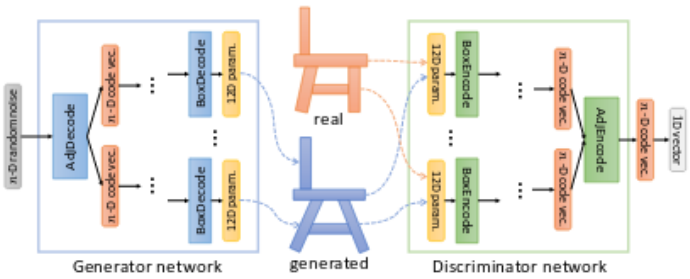


Fig. 9. Architecture of our generative adversarial network, showing reuse of autoencoder modules.

Figure 2.4: GAN Setup

Even after taking into structural priors and constraints, it is difficult to train the generator to produce real structures from random codes. Therefore, instead of directly feeding it random codes from a normal or uniform distribution, they feed it codes drawn from Gaussians around the training samples x, whose mixture approximates the standard normal distribution. Specifically, G takes samples from a multivariate Gaussian distribution that models the mean and variance of the encoded structures/root codes of the structures.

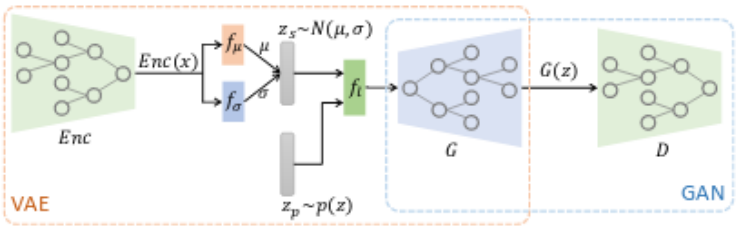


Fig. 11. Confining random codes by sampling from a learned Gaussian distributions based on learned root codes $Enc(x)$. Jointly learning the distribution and training the GAN leads to a VAE-GAN network.

Figure 2.5: VAE-GAN Setup

Chapter 3

Use of Probabilistic Graphical Models with Neural Networks

3.1 Method

Given the grammar representing the type of structures we are dealing with, the scene/structure can be parsed to give a parse-tree. We assume the parse-tree to be a Bayes' net with the features/attributes of the nodes of parse tree representing the random variables at the nodes of the Bayes' net. To model the conditional relationships, we cannot use a discrete CPT(Conditional Probability Table) as the features of a node are assumed to be modeled by a continuous random variable. To model the conditional distribution with limited parameters, we fit a neural network between the parent and child features, that takes the parent features as input and gives the parameters that model the conditional distribution on the child's features. Since we are dealing with a CFG, no child can have more than one parent. One of the simple but efficient modeling method is to assume the feature distributions to be following a normal distribution with mean modeled by the neural network output. For this we need to train all the neural networks in the Bayes' net to learn to model the means of the Gaussians of the children.

3.1.1 Parsing

The given scene with label probabilities of the terminal nodes is parsed using the known grammar in two stages:

- Bottom-up Inference in which the parent probabilities are estimated from the child probabilities, starting with the priors at leaf levels. No higher level rule outside the current subtree influences the estimates.
- Top-down Inference in which the estimated probabilities of a node are adjusted given its parent's probabilities.

The label probabilities at each node at the end of the bottom-up and top-down passes represent the parse of the scene.

Bottom-Up Propagation:

The joint probability of the sub-tree($B \leftarrow S \rightarrow A$) rooted at a current node S, can be estimated as:

$$P(A, B, S, f_A, f_B, f_S) \propto P(A, f_A)P(B, f_B)P(S, f_S)P(rule(S \rightarrow AB)) \tag{3.1}$$

But since f_S is unknown, we marginalize over all possible f_S and $P(S|f_S)$ is 1 as we don't take into account any rules that influence S from the top.

$$P(A, B, S, f_A, f_B) \propto \int_{f_S} P(A, f_A)P(B, f_B)P(f_S)P(rule(S \rightarrow AB)) \tag{3.2}$$

To calculate this integral analytically is difficult as the term $P(A, f_A)$ depends on the output of the corresponding neural network given f_S as input. To approximate this marginal we use the maximum probable f_S estimated through Gibbs Sampling(described later) and find the joint probability.

Top-Down Propagation:

In the top-down pass, the label probabilities at each node are updates using the parent's probabilities.

$$P(S_{new}) \propto P(S_{old})P(P_S)P(rule(P \rightarrow S...)) \tag{3.3}$$

where P_S is the parent of S and $P \rightarrow S...$ is the rule between P and S.

During Parsing, we use the parameters of the neural network which may not be thoroughly learnt yet. For initial learning of the nets, parse trees constructed just from label probabilities obtained from terminal node classifiers are only used along with the rule probabilities to construct the parse tree. The features and the conditional dependence between nodes is not used.

3.1.2 Learning parameters from data

Given the parse tree T_i of a given data instance i, we construct a Bayes net on the nodes of the parse tree and learn the parent-child relationships from the data instance. A parse tree P is represented by terminals T, non terminals N, rule-set R. All the nodes have an attribute vector of k-dimensions. For every parent(Z) - child(X) relationship given the rule, we fit a neural network or decoder that gives the distribution of the child attributes given the parent attribute vector. Let the parameters of the net between parent Z and child X be $W^{Z \to X}$. Given $z \in Z$, $NN^{Z \to X}(z)$ gives θ_z^x, the parameters of the distribution of x. To learn the weights of these decoders for every parent-child relationship in the net, likelihood of the data given the joint distribution of all nodes should be maximized. But since only the terminals are observed, the other variables are hallucinated. Also, the root non terminal S is solely dependent on θ^s which is independent.

Expectation step:

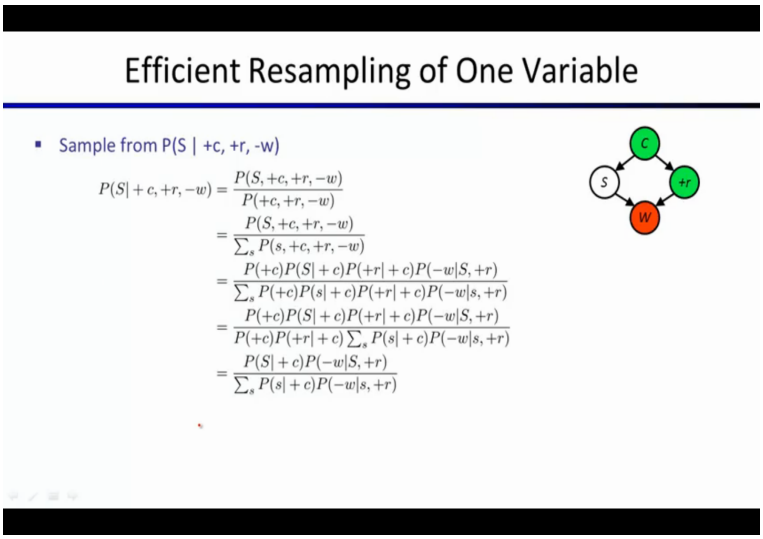
Assuming W_t is known, we can generate the conditional distributions of all nodes starting from the root. Using these conditional distributions at each node, Gibbs sampling is performed to find the most likely unobserved variables / latent variables given the evidence terminals. For each tuple $< i, T_i >$, we sample the most likely unobserved variables Z_i given the observed X_i.

Gibbs Sampling:

For Gibbs Sampling, we freeze the evidence/observed variables, randomly initialize the latent variables, and iteratively resample each latent variable keeping all other variables fixed. From the figure below for the sample example, it is clear that

$$p(Z|z') \propto p(Z|p)p(x|Z) \tag{3.4}$$

where Z is the variable being resampled, z' being all other variables, p the parent of Z and x the child of Z



To sample z from $p(Z|z')$ we use Metropolis-Hasting sampling as the denominator is intractable due to the non-gaussian nature of $p(x|Z)$:

- Sample z from $q(z|z_*)$ which is a gaussian around previous sample of z with a variance that needs to be tuned to have the acceptance ratio around 0.3
- Accept z with probability $p(z|z')/p(z_*|z')$

After generating significant amount of samples, we compute the mode(or mean as an estimate of the mode) of them to obtain and approximate estimate of the most likely sample.

Maximization step:

Given each Z_i, we compute the negative log likelihood loss over all data instances for each neural network $NN^{Z\to X}$ and back-propagate to update θ_z^x thereby maximizing the log likelihood of the data instance.

Variance Estimation: While the child's mean is estimated from the parent's features using the neural network, the variance is estimated by keeping track of the MSE(mean squared error) of the training examples. The MSE is nothing but the variance as the true value is a prediction of a gaussian with variance 0, and the prediction is assumed to be a sample from the gaussian around the true value. So MSE gives an estimate of sample variance.

3.2 Inference at test time

3.2.1 Parsing:

The parsing is done same as stated before but in order to compute the joint probability during the bottom-up parse, we need the features of the children. But if there are missing nodes in the input scene, the most probable feature of the child along with that of the parent are estimated by Gibbs Sampling.

3.2.2 Reconstruction:

For a given parse tree T_i for a test instance i with possible missing labels, reconstruction of labels is equivalent to Bayesian inference and Gibbs sampling can be used to give the most likely missing labels.

Chapter 4

Implementation Details

The entire code is implemented in C++ and the external requirements are very limited. The neural network part is accomplished using FANN, a free open source neural network library.

Chapter 5

Recursive Neural Networks to model conditional relationships

One more method we propose is the use of an common encoder E_c for all c-cardinal rules i.e the rules with c labels in RHS. E_c takes the features of the c children as input and outputs the features of the parent. Also a decoder D_c is used to obtain the children's features given the parent features depending upon the cardinality of the rule. Instead of using Bayes' nets and Gibbs Sampling coupled with Metroplis Sampling which is expensive, we use the encoder networks to do the upward inference. This method is much faster to learn as we avoid sampling and also use the same encoder/decoder networks for different equi-cardinal rules, thus making them learn the parent-child relationships not only locally but globally. The parsing step is the same with the encoders used to obtain the most likely parent features. The training in the learning part is now made end-to-end on the parse-tree.

5.1 Training

The data present during training are the features of the nodes at leaf level which essentially can be the bounding box features of the objects for simplicity. Along with the leaf features we also have the true parse trees for the training data. Given the context free grammar that models the data, we initialize one encoder E_c for all the c-cardinal rules and one decoder D_c similarly.

- An encoder is a fully connected network with tanh activations. The inputs are concatenated and the output is a single feature vector. Similarly decoders are fully connected with tanh activations giving outputs of size c×feature-size

depending on the decoder D_c

- The entire tree is encoded using the encoders starting from the leaf levels. Given the parse-tree we choose E_c corresponding to the cardinality of the rule in the tree.
- The root code is obtained and using the root code and the decoders, the entire parse tree is reconstructed using the decoders. The encoder tree and the decoder tree are joined at the root, with the loss for the entire setup modeled as the reconstruction loss over the terminal feature codes.
- One implementation choice is to shuffle the inputs to the encoder. We will discuss about the consequences of this choice later.

5.1.1 Toy Grammar

The following simple grammar is used to evaluate the method.

$$G \rightarrow 0.5 \quad Sun \quad Tree \quad House$$

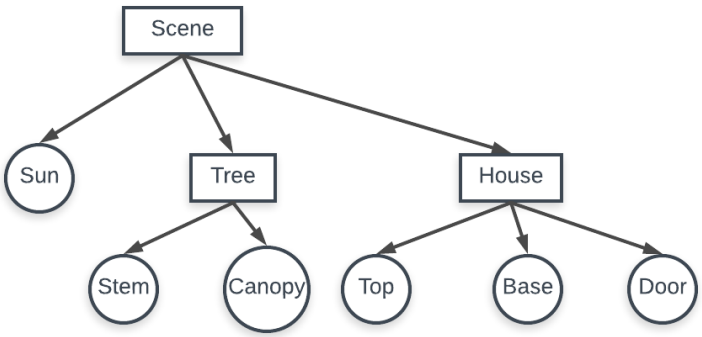
$$G \rightarrow 0.25 \quad Sun \quad House$$

$$G \rightarrow 0.25 \quad House$$

$$Tree \rightarrow 1 \quad Stem \quad Canopy$$

$$House \rightarrow 0.5 \quad Top \quad Base \quad Door$$

$$House \rightarrow 0.5 \quad Base \quad Door$$



5.2 Testing

During the test phase we are presented only the leaf features with no relationships such as a parse-tree between them.

5.2.1 Parsing

As before, we can parse the given leaves to give a parse-tree solely using the grammar rules and searching through all possible merges. Elaborately,

- At every stage, choose a subset of nodes from the frontier nodes. Initially only leaves are in the frontier set.
- Merge the chosen subset as a parent node and estimate its probabilities using the child probabilities using the bottom up propagation as discussed earlier.
- The frontier loses the child nodes and gains the parent node
- Each merge gives a new candidate with a new frontier
- Continue the same till the frontier is a single node with non-zero label probabilities(i.e the root)

This method is really expensive as for each frontier state, there are exponential possible merges which in turn give exponential possible frontiers. Multiple solutions are possible and if we do not have any priors on the leaf nodes(i.e uniform priors) then there is no way that the multiple solutions can be distinguished with. Also note that this method doesn't use the feature vectors in any way. To tackle this and to use the conditional relationships we modeled on the features, we propose the following method.

5.2.2 Merge Estimators

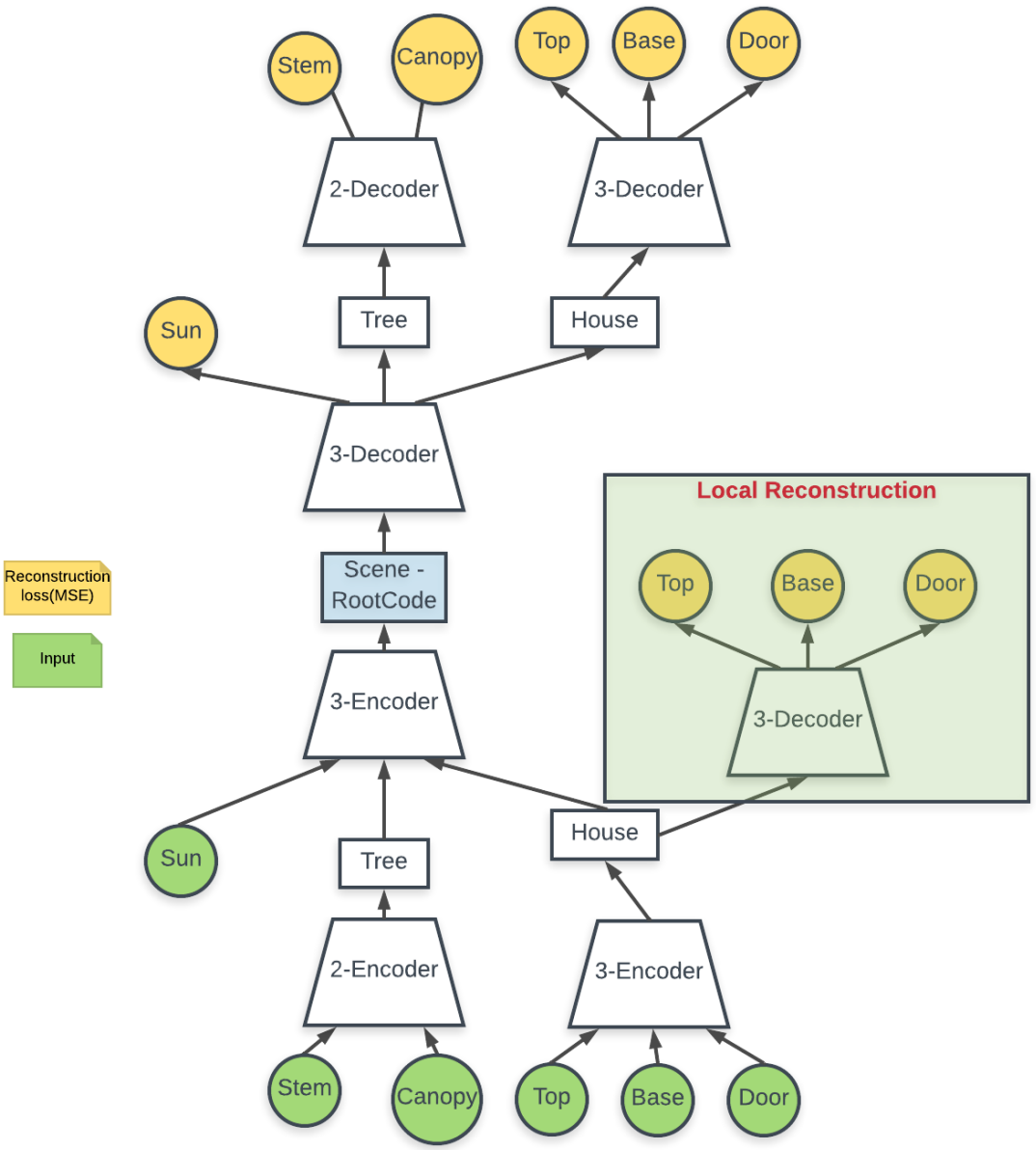
An approach to tackle the above problem is to choose some form of pruning over the exponential possible merges so that there are only a non-exponential number of frontiers at the next step. A naive way to do so is to train recursive neural networks similarly as done before to estimate the probability of a merge. During the encode-decode training performed earlier, at each encode step, we can jointly train a similar RNN(say C_c for each c-cardinal encode) whose goal is to provide the probability that the given c inputs actually constitute a c-cardinal rule. We use these binary merge classifiers during parsing in the following way:

- At every stage, choose a subset of nodes from the frontier nodes. Initially only leaves are in the frontier set.
- For each possible subset of cardinality c, find the probability that it constitutes the RHS of a c-cardinal rule using C_c.
- Pick the top-k subsets depending on the probability obtained from the classifiers
- Merge the chosen subset as a parent node and estimate its probabilities using the child probabilities using the bottom up propagation as discussed earlier for each subset.
- The frontier loses the child nodes and gains the parent node.
- Each merge gives a new candidate with a new frontier.
- Continue the same till the frontier is a single node with non-zero label probabilities(i.e the root)

This method has a bound over the number of candidates at every time i.e k. The bound also has a meaning attached to it that the candidates are scored by a classifier.

5.2.3 Local Reconstructions

One easier implementation is to use the same feature encoder networks to model the merge probability as done by the classifiers above. This is accomplished by training the encoder(sayE_c) on local reconstructions too during the train phase. During the encode phase during training, we incorporate a one-level local reconstruction on the child nodes that are to be encoded using the same E_c and D_c that are used for encoding/decoding globally. The end-end to end encode/decode network that is determined by the parse-tree is trained jointly using the global reconstruction loss at the leaves and the local reconstruction loss at every node. This accomplishes the requirement of some scoring method over the candidate merges without training other networks explicitly for this purpose.

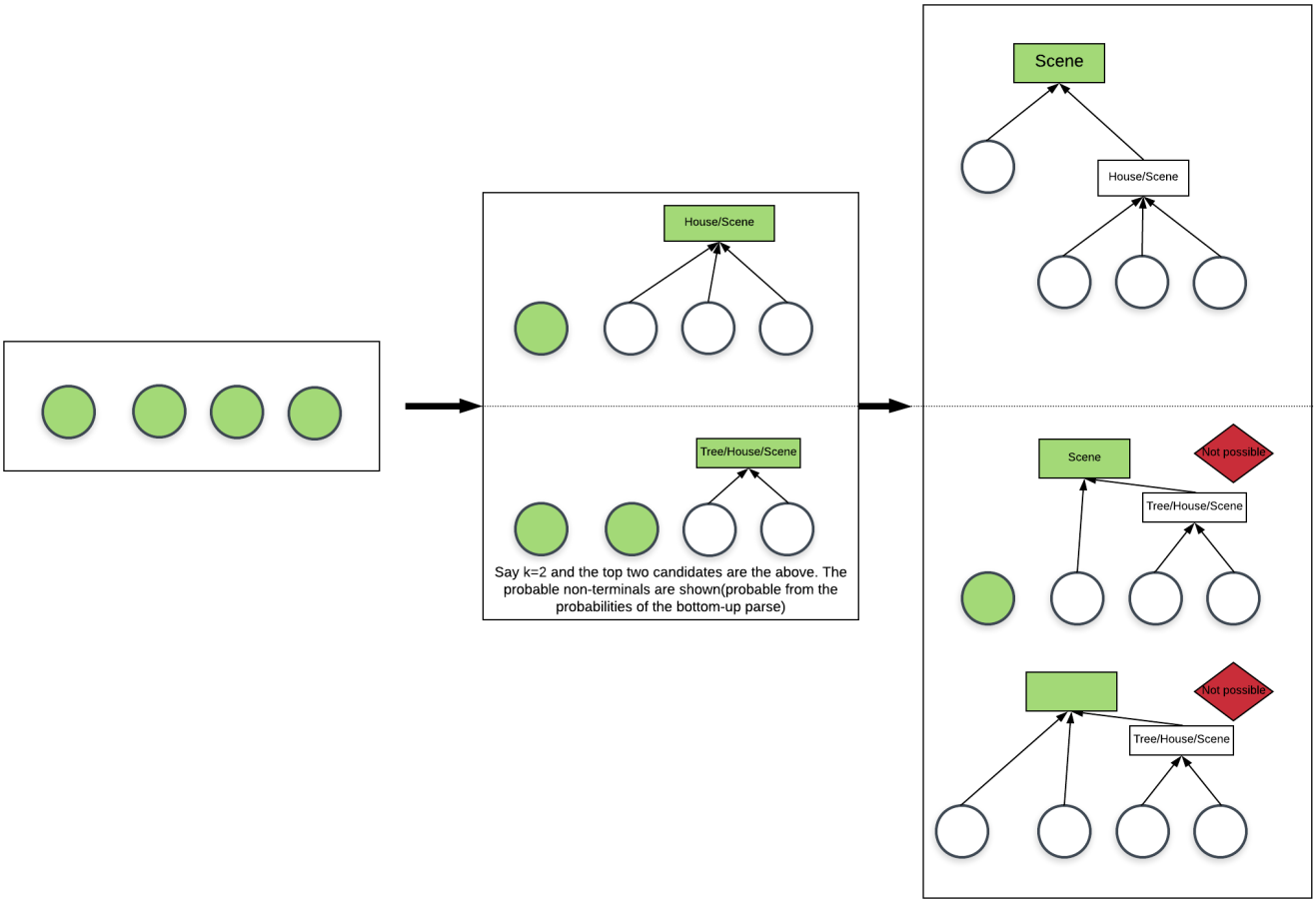


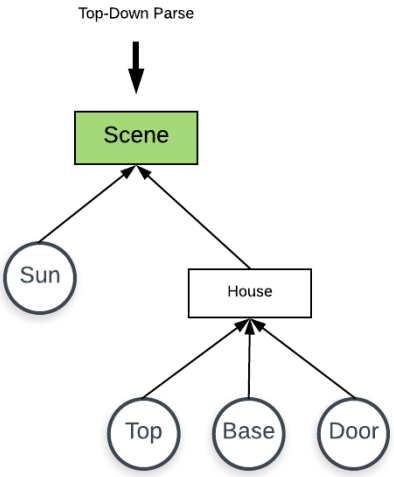
Using the Local reconstructions and the Grammar based pruning, we implement the following method for parsing during testing.

- At every stage, choose a subset of nodes from the frontier nodes. Initially only leaves are in the frontier set.
- For each possible subset of cardinality c, find the local reconstruction error by encoding the inputs and decoding them using E_c and D_c.
- Pick the top-k subsets for each cardinality depending on the local reconstruc-

tion error obtained.

- Merge the chosen subset as a parent node and estimate its probabilities using the child probabilities using the bottom up propagation as discussed earlier for each subset.
- Do further pruning on the candidates using a lower bound on the probability of the parent node(the sum of the probabilities that the parent corresponds to a particular label over all labels)
- The frontier loses the child nodes and gains the parent node.
- Each merge gives a new candidate with a new frontier.
- Continue the same till the frontier is a single node with non-zero label probabilities(i.e the root)
- Once possible candidates for root are obtained:
  - Perform the top down parse i.e update the child probabilities using the parent's.
  - Perform the global reconstruction and update features across the tree.
  - Choose the tree with the best global reconstruction error.


House/Scene
Tree/House/Scene
Say k=2 and the top two candidates are the above. The probable non-terminals are shown(probable from the probabilities of the bottom-up parse)
Scene
House/Scene
Scene
Not possible
Tree/House/Scene
Not possible
Tree/House/Scene


Top-Down Parse
Scene
Sun
House
Top
Base
Door

Here, Let us talk about shuffling the inputs to the encoder during training. If the inputs are shuffled, it gives us a computational gain because we only pass the merge candidate subsets to the encoder. Otherwise, if the encoder is trained on specific orders of inputs, then the candidate subsets must be permuted and all possible permutations should be passed to the encoder resulting in higher number of computations. One gain of this method is since the encoders are trained over specific orders, the encoder gives information about the label probabilities of the children i.e the encoder interprets the children order as the label order in the rule. This helps when we don't have any priors on the leaf nodes as the encoder-decoder reconstructions are order-specific and the leaf labels are thus directly assigned.

5.2.4 Results

k_i is the initialization of the pruning parameter k. If there is no possible solution using the present value of k, it is incremented till a parse-tree is found. Then it is checked whether accurate or not. Total number of test examples = 1000

- $k = 1$, Order-specific Encoders

k	Ex. with accurate soln.	Ex. with a soln.	Accuracy
1	552	686	80.4
2	170	170	100
3	79	79	100
4	27	27	100
5	21	21	100
6	8	8	100
7	0	0	-
8	6	6	100
9	1	1	100
10	1	1	100

- $k = 2$, Order-specific Encoders

k	Ex. with accurate soln.	Ex. with a soln.	Accuracy
2	852	852	100
3	83	83	100
4	27	27	100
5	21	21	100
6	8	8	100
7	0	0	-
8	6	6	100
9	1	1	100
10	1	1	100

- $k = 1$, Order-independent Encoders

k	Ex. with accurate soln.	Ex. with a soln.	Accuracy
1	464	598	77.6
2	305	305	100
3	55	55	100
4	16	16	100
5	25	25	100
6	0	0	-
7	0	0	-
8	0	0	-
9	1	1	100

- $k = 2$, Order-independent Encoders

k	Ex. with accurate soln.	Ex. with a soln.	Accuracy
2	875	903	96.8
3	55	55	100
4	16	16	100
5	25	25	100
6	0	0	-
7	0	0	-
8	0	0	-
9	1	1	100

- $k = 9$, Order-independent Encoders

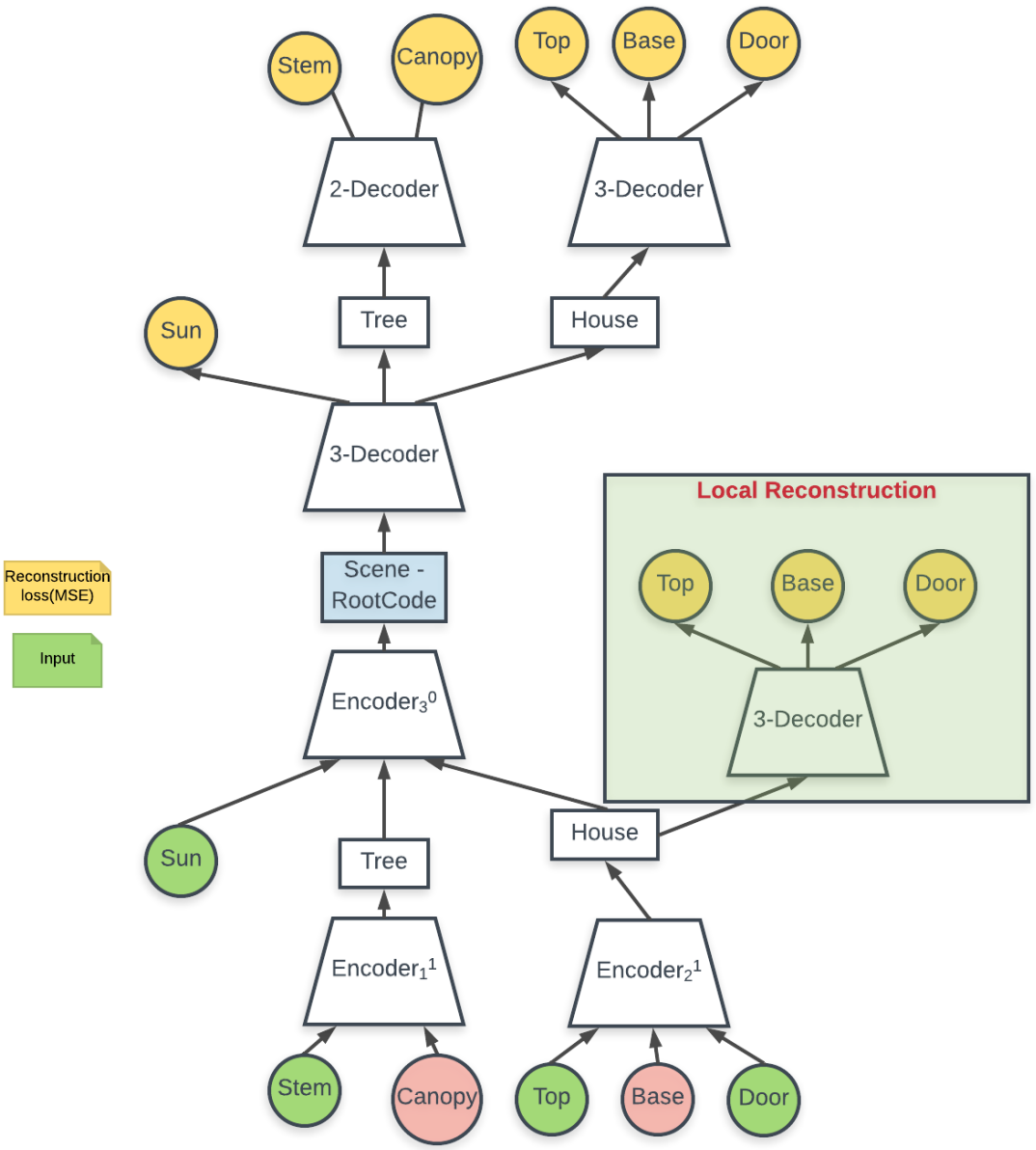
k	Ex. with accurate soln.	Ex. with a soln.	Accuracy
9	972	1000	97.2

5.3 Reconstruction of missing leaves

We assume that some of the leaves are missing during test time. The target is not only to parse the incomplete set of leaves but to predict the missing leaf labels as well as reconstruct them. In the lines of the method above for non-missing cases, we now incorporate a few more encoders for the missing case. Encoders are $E_{c_i}^{c_m}$ where c_i is the cardinality of the inputs to the encoder and c_m is the number of missing leaves. For example, if a k-cardinal rule's LHS is to be encoded using only k-1 of its children E_{k-1}^{1} is used. The number of encoders is still small as the cardinality of grammars is generally small.

5.3.1 Training

We randomly make some leaves missing during training and since we know which ones are missing, we train the encoder-decoder framework as before using the new encoders.



5.3.2 Testing

The testing method is again the same as before with a few changes to incorporate the missing cases.

- For each candidate merge subset of size k, the inputs are passed into the encoders E_k^r for each $0 \leq r \leq max - cardinality - k$
- Assuming the case for a particular r, the positions of the missing nodes is to be found too.

- This is done by hallucinating the missing nodes in all possible subsequences of the input and choosing the one that gives the least reconstruction error.
- If the minimal reconstruction error corresponds to E_k^r then r leaf nodes are hallucinated and their features are obtained by decoding the encoded feature.
- Again, training the encoders over shuffled inputs helps here as iterating over subsequences of inputs is an expensive operation.

5.3.3 Results

k_i is the initialization of the pruning parameter k. If there is no possible solution using the present value of k, it is incremented till a parse-tree is found. Then it is checked whether accurate or not. Total number of test examples = 1000

- $k = 2$, each leaf can be missing with probability 0.1

k	Ex. with accurate soln.	Ex. with a soln.	Accuracy
2	444	590	75.2
3	19	73	26
4	39	72	54.2
5	38	82	46.3
6	15	43	34.8
7	6	10	60
8	9	9	100

- $k = 2$, with relaxation between the rules expanding the non terminal $House$,each leaf can be missing with probability 0.1

k	Ex. with accurate soln.	Ex. with a soln.	Accuracy
2	466	590	78.9
3	24	73	32.8
4	55	72	76.4
5	76	82	92.7
6	41	43	95.3
7	10	10	100
8	9	9	100

- $k = 5$, with relaxation between the rules expanding the non terminal $House$, each leaf can be missing with probability 0.1

k	Ex. with accurate soln.	Ex. with a soln.	Accuracy
5	668	846	79
6	31	31	100
7	7	7	100
8	7	7	100
9	8	8	100
10	8	8	100

Chapter 6

Conclusion and Future Work

In this project, we proposed methods to model the conditional relationships between features of various nodes in a scene hierarchy. We also dealt with exploiting those relationships to parse/generate the hierarchy itself. In this context, recursive neural networks are very useful to learn the conditional relationships between the nodes generated by a context free grammar with few parameters. We not only parse the leaf nodes to find the hierarchy but complete the hierarchy if there are any missing leaves. This is really helpful to annotate the real world scene data with hierarchies automatically given just the object features in it. Future work has a lot of scope as follows:

- Train and test using real world data
- Predict entire missing sub-trees: Till now, we only dealt with missing leaves but there can be cases where all the leaves of a parent are missing and thus it can propagate higher up the tree to make an entire sub-tree missing. The same method as above works in theory for non-missing nodes too.
- Till now, we use the reconstruction errors and the grammar-based pruning independently during parsing. Merging reconstruction error into the label probabilities obtained from the grammar by interpreting the reconstruction error as some form of probability(e^{-x}?) should be done and pruning will only depend on the parent probability.

Bibliography

[Cha+11] Siddhartha Chaudhuri et al. "Probabilistic Reasoning for Assembly-Based 3D Modeling". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 30.4 (2011).

[Liu+14] Tianqiang Liu et al. "Creating Consistent Scene Graphs Using a Probabilistic Grammar". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 33.6 (Dec. 2014).

[Li+17] Jun Li et al. "GRASS: Generative Recursive Autoencoders for Shape Structures". In: *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017)* 36.4 (2017), to appear.