Syntactic Aware Cross Modality Alignment for Vision Language Navigation

Anonymous ACL-IJCNLP submission

Abstract

Vision-language navigation (VLN) is an embodied AI task which requires an agent to navigate 3D environments through following language instructions. While numerous algorithms have been proposed to provide better learning paradigms, more intelligent path planning, extra supervision signals, and more efficient multi-modal embedding schemes for VLN, few work consider integration of syntactic information for better cross modality alignment and decision making. As it is quite intuitive that the algorithms will be able to 1) identify important words, 2) extract sub-instructions, and 3) learn a better alignment from instructions to visual clues and actions with syntactic information captured by dependency parse tree, in this paper, we proposed to study how to incorporate syntactic information to get syntax-aware instruction representations and improve the algorithm's performance on the VLN task. First, we will replicate an existing work to test if tree-LSTM Tai and Manning (2015) (long short-term memory network) is effective in providing syntax information for LSTM-based VLN models. Then, we will adopt the idea from Tree transformer to integrate tree structures into self-attention in transformers. We will implement self attention with hierarchical constraints for a transformer-based VLN model, recurrent VLN BERT (Bidirectional Encoder Representations from Transformers), and we will test if such implementation can incorporate syntactic information and lead to enhanced performance for the VLN task.

1 Introduction

1.1 Goal and Problem Statement

The goal of this project is to improve cross-modality alignment in vision language navigation by incorporation of syntactic information. We will focus on incorporation of syntactic information for 1) LSTM-based and 2) transformer-based VLN models.

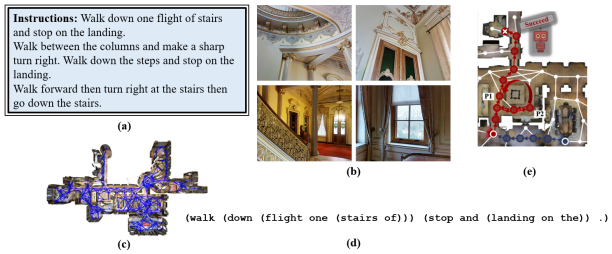

Figure 1: Input and Output Examples for VLN Tasks: (a) Language Instructions; (b) Images from Matterport3D; (c) Navigation Graph; (d) Linearized Parse Tree Generated by CoreNLP 4.5.3; (e) Output. 1 (c) is reproduced from Anderson Peter and Hengel (2018), and 1 (e) is reproduced from Wang Hanqing and Shen (2021)

To be specific, our goals are: 1) add tree LSTM to LSTM-based VLN systems to incorporate syntactic information; 2) test if syntactic information leads to greater performance for LSTM-based VLN systems; 3) incorporate tree transformer's structure, or high level ideas, with transformer-based VLN pretrained models to incorporate syntactic information; 4) test if syntactic information leads to greater performance for transformer-based VLN pretrained models.

1.2 Input and Output

For the VLN task, the inputs are textual instructions, visual features of the environment, and connectivity graph of the scenes. For the VLN model with tree-LSTM, the inputs also include the dependency parsing trees of the instructions.

For the VLN task, the output is a sequence of the agent's actions.

1.3 Motivation

In the natural language instructions, some of the words are better aligned with visual clues, whereas

others are better aligned with actions and orientation information. By generating the dependency parsing tree for the instructions, we can extract syntactic information, which reflects the relations between actions and visual clues. In the parse tree, we can also recognize sub-instructions, which corresponds to the steps in the navigation instructions. In this project, we are curious if we can incorporate syntactic information in VLN frameworks for better cross-modality alignment and boosted VLN performance.

An existing work shows that we can use tree LSTM to incorporate syntactic information to LSTM-based VLN systems and achieve better performance. Since recent VLN pretrained models are transformer-based models, in this project we are curious if we can use tree transformers Wang Yaushian and Chen (2019) to incorporate syntactic information to transformer-based VLN pretrained models and achieve better performance. Can we design novel network structures to add tree transformer's *constituent attention* module to the VLN BERT, so that we can combine the advantages of both recurrent VLN BERT (which can benefit from pretrained V & L BERT Devlin and Toutanova (2018) with reduced computational burden) and tree transformer(which improves interpretable attention for pretrained language models)? This is what we want to explore in this project.

1.4 Hypothesis

We hold the hypothesis that syntactic information can help recognize sub-instructions and provide better alignments with visual clues and actions, which might lead to enhanced performance in VLN tasks.

2 Related Work

2.1 Tree-LSTM for Multi-Modality Alignment with Syntactic Information

Li Jialu and Bansal (2021) proposed to incorporate dependency trees into VLN tasks by a Tree-LSTM, which provides better phrase-level alignment between the visual environment and language instructions. According to Li Jialu and Bansal (2021), explicit incorporation of syntactic information and syntax-aware instruction representations can help the model generalize well to unseen environments. Our project will replicate the results of Tree-LSTM for multi-modality alignment with syntactic information.

2.2 Tree Transformer

Wang Yaushian and Chen (2019) proposed the tree transformer, which inserts a *constituent attention* module to the original transformer encoder to promote interpretable, syntactic aware self attention. Following the same pretraining procedure identical to BERT, with the *constituent attention* module, tree transformer is able to perform unsupervised parsing and improve the learning of language models. In our project, we will try to use tree transformer to provide syntactic information for transformer-based VLN models.

2.3 Recurrent VLN BERT

Unlike other vision-and-language tasks, VLN can be modeled as a partially observable Markov decision process, in which future observation are dependent on the current state and action. To deal with partially observable and temporal-dependent inputs, recurrent VLN BERT Hong Yicong and Gould (2021) proposed to integrate recurrence into BERT. To reduce memory consumption, recurrent VLN BERT modified the self-attention mechanism to only consider language tokens as keys and values (not queries) during navigation, which reduces memory usage so that the model can be trained on a single GPU. In our project, we will try to incorporate syntactic information with recurrent VLN BERT.

2.4 Our Work

To incorporate syntactic information to LSTM-based VLN models, we will 1) replicate the work of Li Jialu and Bansal (2021) and 2) conduct the experiments to compare its performance with the baseline model Tan Hao and Bansal (2019) which has no access to syntactic information.

To incorporate syntactic information to transformer-based pretrained models, we will try our own ideas. We will add the *constituent attention* module to recurrent VLN BERT to promote interpretable and syntactic aware attention.

3 Approach

3.1 LSTM-Based VLN with Syntactic Information

The architecture of the syntax-aware LSTM-based VLN agent is shown in figure 2.

The architecture contains a LSTM-based language encoder, a visual encoder, and a navigation

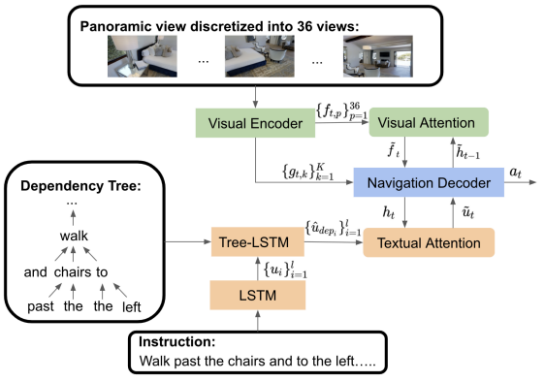

Figure 2: Syntax Aware LSTM-Based VLN. The figure is reproduced from Li Jialu and Bansal (2021)

decoder. First, we use the language encoder to generate syntax aware representation $\{\hat{u}_{dep_i}\}_{i=1}^{l}$ from instructions $\{w_i\}_{i=1}^{l}$:

$$\hat{w}_i = \text{Embedding}(w_i) \tag{1}$$

$$u_1, u_2, ..., u_l = \text{BiLSTM}(\hat{w}_1, \hat{w}_2, ..., \hat{w}_l) \tag{2}$$

$$\{\hat{u}_{dep_i}\}_{i=1}^{l} = \text{TreeLSTM}(\{u_i\}_{i=1}^{l}) \tag{3}$$

Then, we use syntax aware representation $\{\hat{u}_{dep_i}\}_{i=1}^{l}$ of the instructions and the decoder's hidden state h_t to compute textual attention $\tilde{u}_t$ and context aware hidden state $\tilde{h}_t$:

$$\gamma_{t,i} = \text{softmax}_i(\hat{u}_{dep_i}^T W_U h_t) \tag{4}$$

$$\tilde{u}_t = \sum_i \gamma_{t,i} \hat{u}_{dep_i} \tag{5}$$

$$\tilde{h}_t = tanh(W_M[\tilde{u}_t; h_t]) \tag{6}$$

Then, we use panoramic features $\{f_{t,p}\}_{p=1}^{36}$ at time step t and context aware hidden state $\tilde{h}_t$ to compute visual attention $\tilde{f}_t$ and the decoder's next hidden state h_{t+1}:

$$\beta_{t+1,p} = \text{softmax}_p(f_{t+1,p}^T W_F \tilde{h}_t) \tag{7}$$

$$\tilde{f}_{t+1} = \sum_p \beta_{t+1,p} f_{t+1,p} \tag{8}$$

$$h_{t+1} = \text{LSTM}([\tilde{f}_{t+1}; \tilde{a}_t], \tilde{h}_t) \tag{9}$$

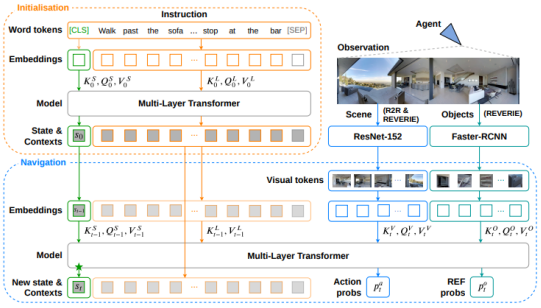

Figure 3: Architecture of Recurrent VLN BERT. The figure is reproduced from Hong Yicong and Gould (2021)

Finally, given visual representations $\{g_{t,k}\}_{k=1}^{K}$ and context aware hidden state $\tilde{h}_t$, we pick the next viewpoint from the K navigable locations:

$$p_t(a_t = k) = \text{Softmax}_k(g_{t,k}^T W_G \tilde{h}_t) \tag{10}$$

For this approach, the core idea is to use Tree-LSTM in the language encoder to provide syntax-aware representation of instructions to improve cross-modality alignment.

3.2 Transformer-Based VLN with Syntactic Information

The architecture of recurrent VLN BERT is shown in figure 3. In this project, we will combine recurrent VLN BERT with constituent attention proposed in tree transformer.

Constituent Prior In transformers, the scaled dot-product attention is computed as:

$$E = \text{softmax}(\frac{QK^T}{d}) \tag{11}$$

where Q is the query matrix, K is the key matrix. Q and K consist of query vectors and key vectors with dimension d_k, and $d = \sqrt{d_k}$ is the scaling factor. In E, the element $E_{i,j}$ is the probability that the position i attends to the position j.

To make each position attend only to the positions in the same constituent, tree transformer proposed to modulate self attention by a constituent prior C:

$$E = C \odot \text{softmax}(\frac{QK^T}{d}) \tag{12}$$

Here $\odot$ is the element-wise multiplication. $C_{i,j}$ describes if position i and position j belongs to the same constituent. If $C_{i,j}$ is small, it means that position i and j might belong to different constituents,

and there shouldn't be attention between different constituents.

In tree transformer, $C_{i,j}$ is determined by the learnable parameters a_k:

$$C_{i,j} = \prod_{k=i}^{j-1} a_k = e^{\sum_{k=i}^{j-1} log(a_k)} \quad (13)$$

where a_k describes the linkage probability between the word w_k and w_{k+1}. If a_k is small, it implies that there is a break point between the word w_k and w_{k+1}. To obtain a_k, two additional mechanisms, *Neighboring Attention* and *Hierarchical Constraint*, are implemented.

Neighboring Attention

For each word w_i at the l-th layer of the transformers, we compute the scores $s^l_{i,i-1}$ and $s^l_{i,i+1}$ by scaled dot-product attention:

$$s^l_{i,i+1} = \frac{q^l_i \cdot k^l_{i+1}}{d} \quad (14)$$

where q^l_i is the query vector of w_i, and k^l_{i+1} is the key vector of w_{i+1}. Here, the attention is masked so that w_i can only attend to its neighbors, w_{i-1} and w_{i+1}.

Then, we apply the softmax function to obtain:

$$p^l_{i,i+1}, p^l_{i,i-1} = \text{softmax}(s^l_{i,i+1}, s^l_{i,i-1}) \quad (15)$$

where $p^l_{i,i+1}$ is the probability that w_i attends to w_{i+1}. Since $p^l_{i,i+1}$ and $p^l_{i+1,i}$ might have different values, we compute their average:

$$\hat{a}^l_i = \sqrt{p^l_{i,i+1} \times p^l_{i+1,i}} \quad (16)$$

$\hat{a}^l_i$ represents the probability that w_i is linked to w_{i+1} at layer l. We will use *hierarchical constraint* to recursively obtain a^l_i from $\hat{a}^l_i$.

Hierarchical Constraint

To form tree structures in self attention, tree transformer introduced hierarchical constraints for a_k, such that the linkage probability at layer l is strictly larger than the linkage probability at layer $l-1$:

$$a^l_k = a^{l-1}_k + (1 - a^{l-1}_k)\hat{a}^{l+}_k \quad (17)$$

Initially, different words are considered as different constituents. By computing the linkage probability a_k and the constituent attention C, different constituents are gradually linked together with shared attention. Hence, the self attention forms a tree structure in tree transformer, which is helpful for incorporation of syntactic information into transformers. Here a_k and C are learnt in an unsupervised manner from the dataset.

For this approach, our plan is:

1) Implement constituent attention for recurrent VLN BERT.

2) Train recurrent VLN BERT with and without constituent attention. Compare the 2 models' performance on VLN tasks.

3.3 Code Use and Implementations

For replication of the LSTM-based baseline method in experiment 1) we use existing code from github repositories[1]. For replication of Tree-LSTM for syntactic information alignment in experiment 2), we use existing code from github repositories[2]; For replication of transformer-based baseline method in experiment 3, we use existing code from github repositories[3]. For implementation of tree-transformer based syntactic information alignment, we will bring our own implementations, however the building blocks might come from recurrent VLN BERT and tree transformer[4].

4 Experimental Setup

4.1 Experiments

In this project, we will evaluate 4 methods (2 baseline methods and 2 improved methods with syntactic information alignment) for their performance on VLN tasks:

Experiment 1) Evaluation of LSTM-based baseline method Tan Hao and Bansal (2019) on VLN tasks;

Experiment 2) Evaluation of Tree-LSTM for syntactic information alignment Li Jialu and Bansal (2021) on VLN tasks;

Experiment 3) Evaluation of transformer-based baseline method Hong Yicong and Gould (2021) on VLN tasks;

Experiment 4) Evaluation of tree transformer Wang Yaushian and Chen (2019) based syntactic information alignment on VLN tasks.

[1] https://github.com/jialuli-luka/SyntaxVLN

[2] https://github.com/jialuli-luka/SyntaxVLN

[3] https://github.com/YicongHong/Recurrent-VLN-BERT

[4] https://github.com/yaushian/Tree-Transformer

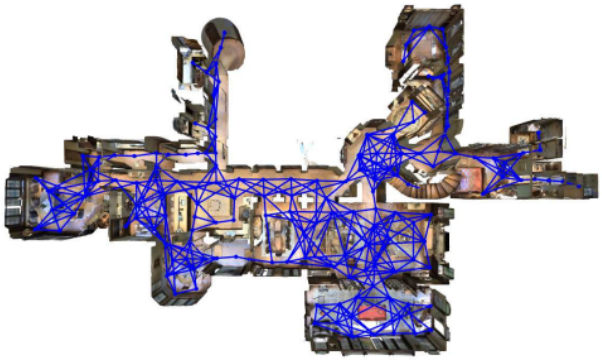

Figure 4: Visualization of Navigation Graph: blue lines show navigable paths. The figure is reproduced from Anderson Peter and Hengel (2018)

4.2 Dataset

We will use Room-to-Room (R2R) dataset Anderson Peter and Hengel (2018) which is built on top of Matterport3D dataset Chang Angel and Zhang (2017).

Matterport3D is a large-scale RGB-D dataset which contains 10,800 panoramic views from 194,000 RGB-D images of 90 building-scale scenes Chang Angel and Zhang (2017). 61 scenes are for training, 11 scenes are for validation, and 18 scenes are for testing. The Matterport3D dataset contains various types of annotations, and its total size is about 1.4TB. For training purpose, we can use the ResNet-152 image features extracted from the Matterport3D environments. For visualization purpose, we need the *matterport_skybox_images* part of the dataset, which is about 23GB. For training and visualization, Matterport3D simulator Anderson Peter and Hengel (2018) is required, which allows interaction between the agent and the environments.

R2R dataset Anderson Peter and Hengel (2018) consists of the navigation graphs, the language instructions, and the reference navigation trajectories.

The navigation graphs define the navigable paths between panoramic viewpoints. On average, each graph contains 117 viewpoints. A visualization of the navigation graphs can be found at figure 4.

R2R has a total of 21,567 navigation instructions with an average length of 29 words. Specifically, it contains 14,025 instructions for the train dataset, 1,020 instructions for the val seen dataset, 2,349 instructions for validating in unseen environments, and 4,173 instructions for test set. Figure 5 shows the distribution of navigation instructions based on the first 4 words.

Corresponding to language instructions, reference trajectories are provided in R2R dataset. Figure 6 shows the distribution of instruction length and navigation trajectory length in the R2R dataset.

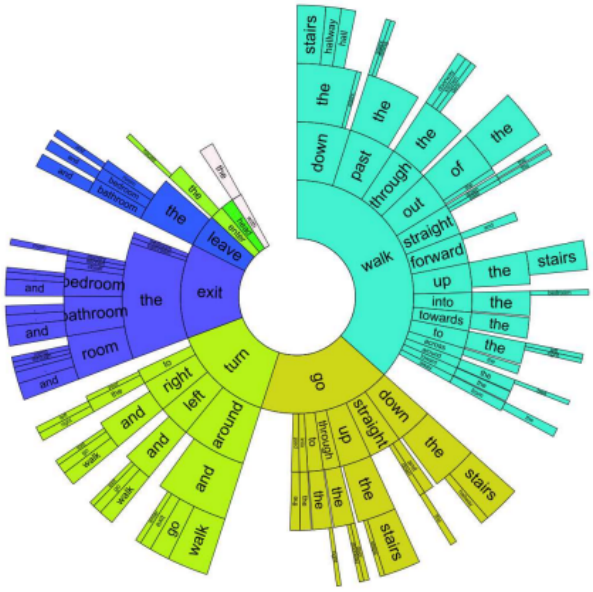

Figure 5: Distribution of navigation instructions based on their first 4 words. The figure is reproduced from Anderson Peter and Hengel (2018). Instructions are read from the center outwards

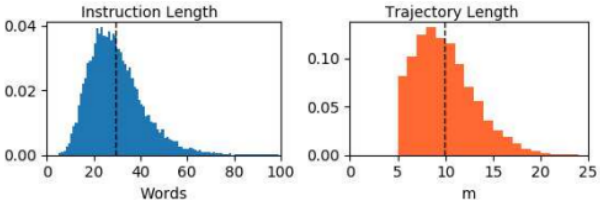

Figure 6: Distribution of instruction length and navigation trajectory length in R2R dataset. The figure is reproduced from Anderson Peter and Hengel (2018)

For a concrete example from R2R dataset, please refer to figure 1.

4.3 Training Details

The models are trained on a single NVIDIA A100 GPU from Colab platform. The LSTM-baseline and LSTM-syntactic models are trained for 80,000 iterations. The LSTM-baseline model is trained for about 1.7 days, and the LSTM-syntactic model is trained for about 4.2 days. The transformer-baseline and transformer-syntactic models are trained for 150,000 iterations. The transformer-baseline model is trained for about 1.4 days, and the transformer-syntactic model is trained for about 1.4 days.

4.4 Evaluation Metrics

For evaluation of the models, we will use **success rate** (SR), **success rate weighted by path length** (SPL) Anderson Peter (2018), **normalized dynamic time warping** (nDTW) Ilharco and Baldridge (2019), **success rate weighted by dynamic time warping** (sDTW) Ilharco and Baldridge (2019), and **coverage weighted by length score** (CLS) as the evaluation metrics:

(1) Success Rate (SR): if the agent stops within 3 meters from the target location, we consider that the VLN task is successfully completed. Higher SR indicates better model performance.

(2) Success Rate Weighted by Path Length (SPL): this metric is similar to SR, the difference is that it penalizes long trajectories. Higher SPL indicates better model performance.

(3) Normalized Dynamic Time Warping (nDTW): This metric penalizes any deviation from the ground truth path. Higher nDTW indicates better model performance.

(4) Success Rate Weighted by Dynamic Time Warping (sDTW): This metric considers both path fidelity and the successful completion of the VLN task. That is, sDTW puts an additional constraint on nDTW such that only successful navigation examples are calculated. Higher sDTW indicates better model performance.

(5) Coverage Weighted by Length Score (CLS): This metric encourages path fidelity. Higher nDTW indicates better model performance.

4.5 Comparison

We will compare two existing methods in experiment 1) and 2) to see if tree-LSTM is effective in providing better syntax-aware cross-modality alignment for LSTM-based VLN models which leads to better performance on VLN tasks. The LSTM-baseline and LSTM-syntactic models will be trained for 80,000 iterations.

We will compare our method with an existing method in experiment 3) and 4) to see if our proposed ideas are effective in providing better syntax-aware cross-modality alignment for transformer-based VLN models which leads to better performance on VLN tasks. The transformer-baseline and transformer-syntactic models will be trained for 150,000 iterations.

Models	SR	SPL	nDTW	sDTW	CLS
LSTM-Baseline	0.989	0.979	0.973	0.967	0.965
LSTM-Syntactic	**0.998**	**0.991**	**0.982**	**0.980**	**0.978**

Table 1: Comparison of LSTM-Based Models on R2R Training Set

Models	SR	SPL	nDTW	sDTW	CLS
LSTM-Baseline	**0.610**	**0.583**	**0.696**	**0.548**	**0.692**
LSTM-Syntactic	0.578	0.547	0.679	0.522	0.677

Table 2: Comparison of LSTM-Based Models on R2R Validation Seen Set

5 Experimental Results

5.1 Quantitative Results

5.1.1 Comparison of LSTM-Based Models

We have trained the LSTM-baseline model and the LSTM-syntactic model for 80,000 iterations on the R2R dataset. The results are summarized in table 1, 2, and 3.

5.1.2 Comparison of Transformer-Based Models

We have trained the transformer-baseline model and the transformer-syntactic model for 150,000 iterations on the R2R dataset. The results are summarized in table 4, 5, and 6.

5.2 Qualitative Results

5.2.1 Learning Curves

We compared LSTM-baseline model and LSTM-syntactic model's success rate on *training*, *validation-seen* and *validation-unseen* datasets for the whole training period. The result is shown in figure 7.

We compared Transformer-baseline model and Transformer-syntactic model's success rate on *training*, *validation-seen* and *validation-unseen* datasets for the whole training period. The result is shown in figure 8.

5.2.2 Self Attention Generated by Tree Transformers

To examine what *constituent prior* the tree transformer learns for the Transformer-Syntactic model,

Models	SR	SPL	nDTW	sDTW	CLS
LSTM-Baseline	**0.476**	**0.443**	**0.587**	**0.414**	**0.587**
LSTM-Syntactic	0.463	0.430	0.586	0.403	0.586

Table 3: Comparison of LSTM-Based Models on R2R Validation Unseen Set

Models	SR	SPL
Transformer-Baseline	**0.967**	0.944
Transformer-Syntactic	**0.967**	**0.946**

Table 4: Comparison of Transformer Based Models on R2R Training Set

Models	SR	SPL
Transformer-Baseline	0.746	**0.700**
Transformer-Syntactic	**0.750**	0.699

Table 5: Comparison of Transformer Based Models on R2R Validation Seen Set

Models	SR	SPL
Transformer-Baseline	0.613	**0.556**
Transformer-Syntactic	**0.622**	0.549

Table 6: Comparison of Transformer Based Models on R2R Validation Unseen Set

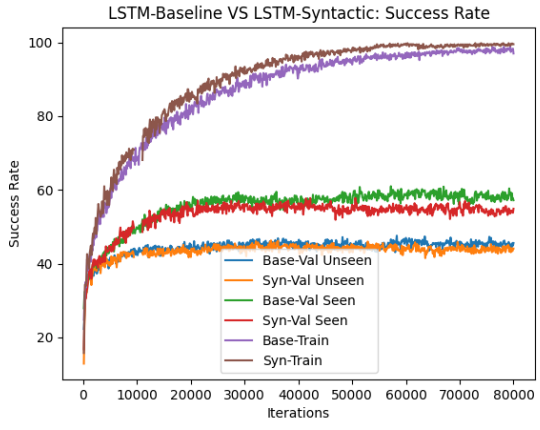

Figure 7: Success Rates Comparison Between LSTM-Baseline and LSTM-Syntactic Models on Training, Validation-Seen and Validation Unseen Datasets

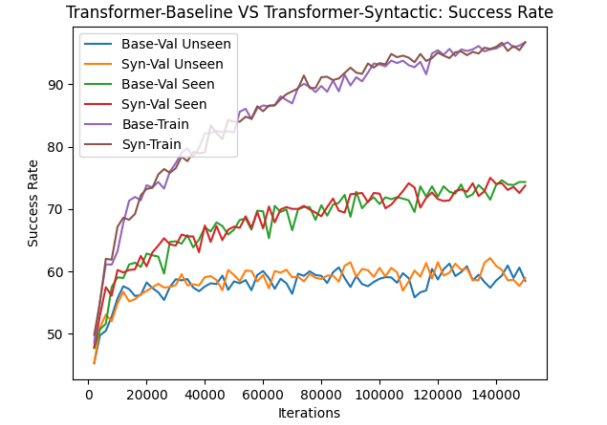

Figure 8: Success Rates Comparison Between Transformer-Baseline and Transformer-Syntactic Models on Training, Validation-Seen and Validation Unseen Datasets

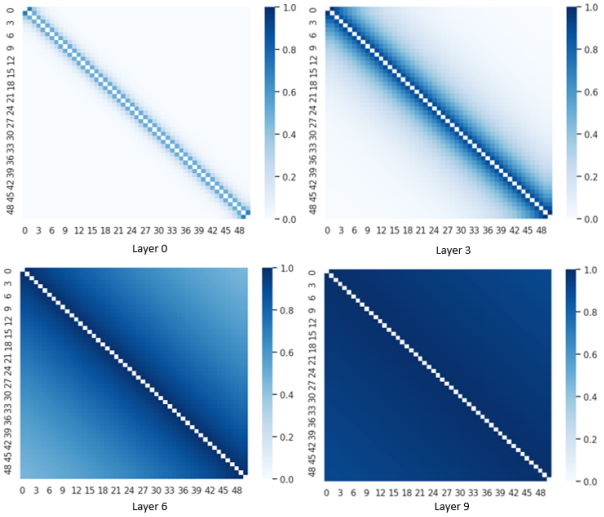

Figure 9: Heatmaps of Constituent Priors for an Example Instruction

we plotted the heatmaps of the *constituent priors* at layer 1, 3, 6, 9 for the instruction *Turn around and enter the house. Head past the blue chairs. When you are behind the red chair on the left, turn and enter the bathroom to the left. Stop inside the bathroom right in front of the sink facing the sink and mirror*. The result is shown in figure 9.

To examine if the *constituent prior* has learned any tree structures, we plotted the heatmaps of the *constituent priors* at layer 1, 3, 6, 9 for the first part of the above instruction: *Turn around and enter the house*. The result is shown in figure 10.

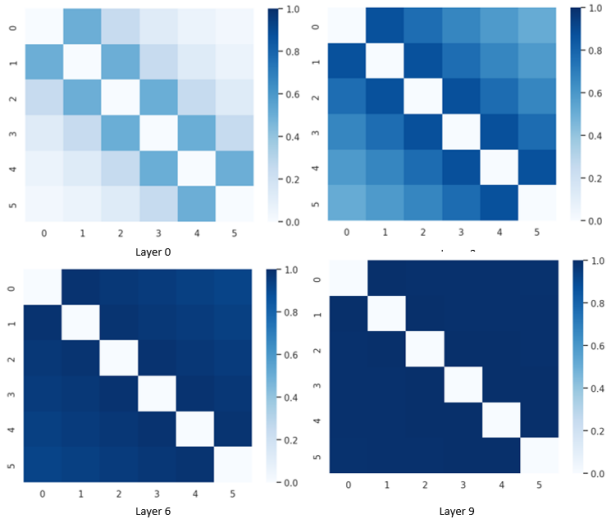

Figure 10: Heatmaps of Constituent Priors for the First Part of the Example Instruction

6 Analysis

6.1 LSTM-Syntactic: Poor Generalization Ability

The original paper Li Jialu and Bansal (2021) reported that, comparing with the model with 2 layers of LSTMs (LSTM-Baseline), replacing the 2nd LSTM by the tree-LSTM (LSTM-Syntactic) leads to +2.3% (62.6% versus 60.3%) increase of success rate on Validation-Seen dataset, and +2.6% (49.0% versus 46.4%) increase of success rate on validation-unseen dataset.

However, our finding is *not* consistent with the results reported in the original paper: While the original paper reported +2.3% and +2.6% success rates of LSTM-syntactic model on validation-seen and validation-unseen dataset, our experiments show that LSTM-syntactic model leads to -3.2% and -1.3% success rate on validation-seen and validation-unseen datasets.

Since the learning curve in figure 7 shows that the LSTM-Syntactic model consistently outperforms the LSTM-Baseline model on the training dataset, and the LSTM-Baseline model consistently outperforms the LSTM-Syntactic model on the validation-seen and validation unseen datasets, what we can conclude from our own observation is that the LSTM-Syntactic model shows larger performance gaps between training and validation datasets, which indicates that LSTM-Syntactic model might have poorer generalization ability compared with the LSTM-Baseline model.

6.2 Transformer-Syntactic: Possibly Better Success Rate and Longer Paths?

According to the *best-so-far* SR (success rate) and SPL (success rate weighted by path length), the Transformer-Syntactic model tends to outperform Transformer-Baseline model in success rate. However, it seems that the Transformer-Syntactic model tends to generate longer paths, and its advantage in success rate diminishes if we take the path length penalty into account. The performance difference between the models is negligible on training and validation-seen datasets. The difference only becomes prominent on the validation-unseen dataset (+0.9% SR and -0.7% SPL for Transformer-Syntactic model).

From the *best-so-far* evaluation metrics, we might preliminarily conclude that the Transformer-Syntactic model tends to achieve higher success rate at the cost of longer paths, especially on the validation-unseen dataset. Since the Transformer-Syntactic model shows smaller gap of success rate between the training and validation datasets, we might conclude that Transformer-Syntactic model tends to have better generalization ability than the Transformer-Baseline model.

6.3 What does the Tree-Transformer Learn?

In this project, our hypothesis is that tree-Transformer can learn the tree structure of the syntactic constituents, which helps improve the cross-modality alignment. However, from the heatmaps of the constituent priors we can conclude that the tree transformer didn't learn tree structures. Suppose that the tree transformer does learn attention with tree structures, then the heatmaps should show significant asymmetry (if the model recognizes that word w_i is with the left neighbor w_{i-1} and not with the right neighbor w_{i+1}, then the constituent priors $C_{i-1,i}$ and $C_{i,i+1}$ should be very different, which produces asymmetry in the heatmaps). However, our heatmaps indicate that the local attention in lower layers spread symmetrically to global attention in higher layers, which means that in our VLN scenarios, the tree-transformer didn't learn the structures of the syntactic constituents. What the tree-transformer actually did is to implement local attention in lower transformer layers and global attention in higher transformer layers.

6.4 Why Tree-Transformer leads to Better Success Rate and Longer Paths?

Since the tree transformer didn't learn tree structures, we need to propose alternative explanations why the tree transformer leads to better success rate and longer paths.

At this time, our preliminary explanation is that, tree transformer implements local attention in the lower transformer layers, which exerts higher dropout rates that helps the model generalize well to unseen datasets. This might be why the Transformer-Syntactic model did well on the validation-unseen dataset in the success rate. However, the local attention implemented in the lower layers might also hinder transformer's ability to model long range dependencies, which means that the tree transformer might be unable to learn the most effective representations. This might be a potential reason why the paths generated by the Transformer-Syntactic model is not as concise as the paths generated by the Transformer-Baseline model.

7 Limitations

7.1 Limitations

In this project, we reached some conclusions that are *not* consistent with what were reported in the original papers. For example, we found that LSTM-Syntactic model didn't outperform LSTM-Baseline model, and the tree-transformer didn't actually learn tree structures. We still need to check the implementation details and replicate the experiments for multiple times to confirm the validity of our conclusions.

In this project, we use several evaluation metrics, such as success rate, as mild indicators of how well the multi-modal information is represented and aligned. However, we didn't show direct evidence of how well the multi-modal information is represented and aligned.

7.2 Future Directions

We might want to show how the attention weights are distributed during the navigation process to evaluation how well the cross modality information is aligned.

Since the tree transformer didn't actually learn tree structures, we might want to study the reason why there is a performance difference between the Transformer-Baseline and Transformer-Syntactic models.

8 Conclusion

Based on the experiment results shown in this project, we can draw the following conclusions:

1. The LSTM-Syntactic model consistently shows larger performance gap between the training and validation datasets, which means that the LSTM-Syntactic model suffers from poorer generalization ability. This finding is *not* consistent with what was reported in the original paper Li Jialu and Bansal (2021) Wang Yaushian and Chen (2019).

2. The Transformer-Syntactic model tends to outperform the Transformer-Baseline model in success rate, however the Transformer-Baseline model tends to generate more concise paths. The difference between the two models is more prominent on the validation-unseen dataset.

3. In the VLN scenarios, the tree transformer didn't learn tree structures of syntactic constituents for better cross modality alignment.

4. Tree transformer implements local attention and higher dropout rates at lower transformer layers, which might be a potential explanation why the Transformer-Syntactic model tends to generalize well and have higher success rates at the cost of longer paths.

9 Contribution

This is a single person project and Sihui Wang is responsible for all components.

References

Damien Teney Jake Bruce Mark Johnson Niko Sünderhauf Ian Reid Stephen Gould Anderson Peter, Qi Wu and Anton Van Den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3674–3683.

Devendra Singh Chaplot Alexey Dosovitskiy Saurabh Gupta Vladlen Koltun Jana Kosecka et al Anderson Peter, Angel Chang. 2018. On evaluation of embodied navigation agents. In *arXiv preprint arXiv:1807.06757 (2018)*.

Thomas Funkhouser Maciej Halber Matthias Niebner Manolis Savva Shuran Song Andy Zeng Chang Angel, Angela Dai and Yinda Zhang. 2017. Matterport3d: Learning from rgb-d data in indoor environments. In *In 2017 International Conference on 3D Vision (3DV)*, pages 667–676.

Ming-Wei Chang Kenton Lee Devlin, Jacob and Kristina Toutanova. 2018. Bert: Pre-training of deep

bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805 (2018)*.

Yuankai Qi Cristian Rodriguez-Opazo Hong Yicong, Qi Wu and Stephen Gould. 2021. Vln bert: A recurrent vision-and-language bert for navigation. In *In Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 1643–1653.

Vihan Jain Alexander Ku Eugene Ie Ilharco, Gabriel and Jason Baldridge. 2019. General evaluation for instruction conditioned navigation using dynamic time warping. In *arXiv preprint arXiv:1907.05446 (2019)*.

Hao Tan Li Jialu and Mohit Bansal. 2021. Improving cross-modal alignment in vision language navigation via syntactic information. In *In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1041–1050.

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

Licheng Yu Tan Hao and Mohit Bansal. 2019. Learning to navigate unseen environments: Back translation with environmental dropout. In *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2610–2621.

Wei Liang Caiming Xiong Wang Hanqing, Wenguan Wang and Jianbing Shen. 2021. Structured scene memory for vision-language navigation. In *In Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 8455–8464.

Hung-Yi Lee Wang Yaushian and Yun-Nung Chen. 2019. Tree transformer: Integrating tree structures into self-attention. In *In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1061–1070.