Learning to Stop: A Simple yet Effective Approach to Urban Vision-Language Navigation

Jiannan Xiang[†]**, Xin Eric Wang**[‡]**, William Yang Wang**[§]

[†]University of Science and Technology of China
[‡]University of California, Santa Cruz
[§]University of California, Santa Barbara

szxjn@mail.ustc.edu.cn, xwang366@ucsc.edu, william@cs.ucsb.edu

Abstract

Vision-and-Language Navigation (VLN) is a natural language grounding task where an agent learns to follow language instructions and navigate to specified destinations in real-world environments. A key challenge is to recognize and stop at the correct location, especially for complicated outdoor environments. Existing methods treat the STOP action equally as other actions, which results in undesirable behaviors that the agent often fails to stop at the destination even though it might be on the right path. Therefore, we propose Learning to Stop (L2STOP), a simple yet effective policy module that differentiates STOP and other actions. Our approach achieves the new state of the art on a challenging urban VLN dataset TOUCHDOWN, outperforming the baseline by 6.89% (absolute improvement) on Success weighted by Edit Distance (SED).

1 Introduction

Vision-and-language navigation (VLN) aims at training an agent to navigate in real environments by following natural language instructions. Compared to indoor VLN (Anderson et al., 2018), navigation in urban environments (Chen et al., 2019) is particularly challenging, since urban environments are often more diverse and complex. Several research studies (Mirowski et al., 2018; Li et al., 2019; Bruce et al., 2018) have been conducted to solve the problem. In this paper, we also focus on the urban VLN task. As shown in Fig. 1, given a natural language instruction, the agent perceives local visual scene and chooses actions at every time step, learning to match the instruction with the produced trajectory and navigate to the destination. Existing VLN models (Wang et al., 2019; Tan et al., 2019; Ke et al., 2019; Ma et al., 2019b,a; Fried et al., 2018; Wang et al., 2018) seem to neglect the importance of the STOP action and treat
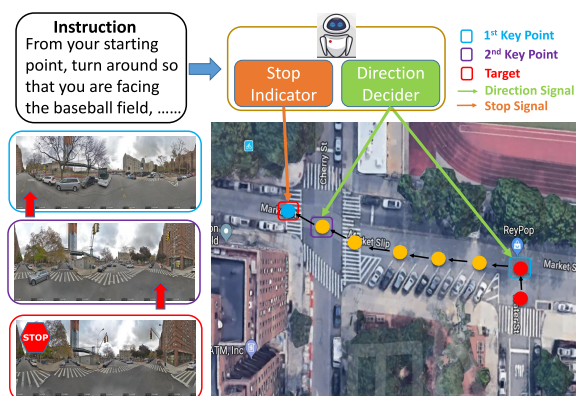

Figure 1: Vision-and-language navigation task in an urban environment. Our L2STOP agent chooses directions at key points and leverages a stop indicator to produce stop or non-stop signals.

all actions equally. However, this can lead to undesirable behaviors, also noticed in Cirik et al. (2018); Blukis et al. (2018), that the agent fails to stop at the target although it might be on the right path, because the STOP action is severely underestimated.

We argue that the STOP action in the urban VLN tasks is crucially important and deserves special treatment. First, in contrast to errors on other actions that are likely to be fixed later in the journey, the price of stopping at a wrong location is higher, because producing STOP terminates the episode, and there will be no chance to fix a wrong stop. Second, the statistical count of STOP is much lower than other actions as it only appears once per episode. Thus STOP will receive less attention if we treat all actions equally and ignore the difference of occurrence frequency. Moreover, STOP and other actions need different understandings of the dynamics between the instruction and the visual scene. Both require the alignment between trajectories and instructions, but STOP would emphasize the completeness of the instruction and the matching between the inferred target and the sur-

rounding scene, while choosing directions requires a planning ability to imagine the future trajectory.

Therefore, we introduce a Learning to Stop (L2STOP) module to address the issues. L2STOP is a simple and model-agnostic module, which can be easily plugged into VLN models to improve their navigation performance. As we demonstrate in Fig. 1, the L2STOP module consists of a *Stop Indicator* to determine whether to stop and a *Direction Decider* to choose directions when at key points. Besides, we weigh STOP action more than other actions in the loss function, forcing the agent to pay more attention to the STOP action. We conduct experiments on a language-grounded street-view navigation dataset TOUCHDOWN[1] (Chen et al., 2019). Extensive results show that our proposed approach significantly improves the performance over the baseline model on all metrics and achieves the new state-of-the-art on the TOUCHDOWN dataset. [2]

2 Approach

Fig. 2 illustrates the framework of our L2STOP model. Specifically, a text encoder and visual encoder are used to get the text and visual representations. Then the trajectory encoder uses the representations to compute the hidden context state, which is the input of the policy module. Unlike previous VLN models, which use one branch policy module, we use our proposed L2STOP module, a two-branch policy module that separates the policies for STOP and other actions. We detail each component below.

2.1 Visual and Text Encoder

As shown in the Fig. 2, we use two encoders as used in Chen et al. (2019) for encoding visual scene and language instruction respectively. For visual part, we apply a CNN (Krizhevsky et al., 2012) as the visual encoder to extract visual representation v_t from current visual scene at time step t. For text, we adopt an LSTM (Hochreiter and Schmidhuber, 1997) as the text encoder to get the instruction representation $X = \{x_1, x_2, ..., x_l\}$. We then use a soft-attention (Vaswani et al., 2017) to get the grounded textual feature x_t at time step t:

$$\alpha_{t,l} = softmax((W_x h_{t-1})^T x_l) \quad (1)$$
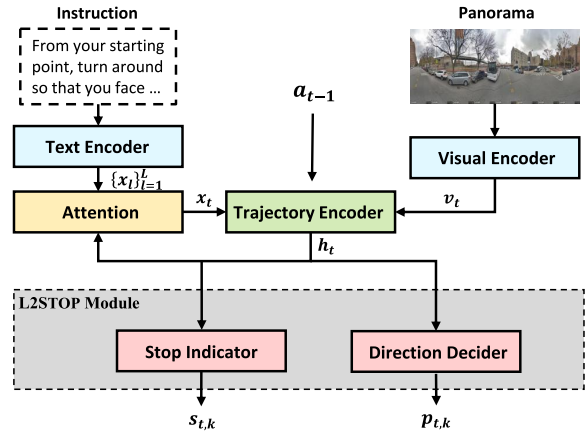
$$x_t = \sum_l \alpha_{t,l} x_l \quad (2)$$

Figure 2: Overview of our L2STOP model.

where W_x denotes parameters to be learnt, $\alpha_{t,l}$ denotes attention weight over l-th feature vector at time t, and h_{t-1} denotes the hidden context at previous time step. Then the agent produces the hidden context at the current step: $h_t = LSTM([x_t, v_t, a_{t-1}])$.

2.2 Learning to Stop Policy Module

Unlike the existing methods that view all the actions equally important, we propose the L2STOP module that helps the agent to learn whether to stop and where to go next with separate policy branches, *Stop Indicator* and *Direction Decider*.

Stop Indicator The stop indicator produces stop or non-stop signals at every time step. At time step t, the stop indicator takes the hidden context h_t and the time embedding t as input and outputs the probabilities of stop and non-stop signals:

$$s_{t,1}, s_{t,2} = softmax(g_2([h_t, t])) \quad (3)$$

where g_2 is a linear layer, and $s_{t,1}$ as well as $s_{t,2}$ are the probabilities of non-stop and stop signals at time step t, respectively. If the stop indicator produces stop signal, the agent will stop immediately. Otherwise, the direction decider will choose a direction to go next.

Direction Decider The direction decider is employed to select actions from a subset of the original action space. Specifically, the action subset includes all actions except STOP action (*go forward, turn left, and turn right*). Empirically, we observe that when navigating in urban environments, the agent only needs to choose directions at the intersections (nodes with more than two neighbors) it encounters in the journey. Therefore, we view

Method	Development					Test				
	TC↑	SPD↓	SED↑	CLS↑	SDTW↑	TC↑	SPD↓	SED↑	CLS↑	SDTW↑
Random	0.15	26.63	0.05	4.65	0.06	0.36	26.94	0.01	4.44	0.00
GA	9.85	21.43	9.50	46.86	9.44	9.65	21.46	9.21	47.34	9.15
RCONCAT	11.14	19.87	10.77	46.61	10.76	9.65	21.65	9.45	44.34	9.42
Ours										
ARC	15.33	18.61	14.62	48.56	14.48	14.13	19.41	13.62	48.02	13.50
GA + L2STOP	12.58	19.76	12.18	50.10	12.18	11.50	19.48	11.08	50.53	11.01
RCONCAT + L2STOP	13.01	19.28	12.69	50.86	12.66	12.63	19.45	12.31	50.66	12.28
ARC + L2STOP	**19.48**	**17.05**	**19.02**	**55.68**	**18.97**	**16.68**	**18.84**	**16.34**	**53.50**	**16.34**

Table 1: Experimental results on development and test sets.

these intersections as key points on the road and assume that the direction decider only needs to choose directions at key points and always goes forward otherwise. So at time step t, if the agent is at a key point, it will be activated and takes the hidden context h_t as well as a learned time embedding t as input and outputs the probability of each action in its action space:

$$p_{t,k} = softmax(g_1([h_t, t])) \qquad (4)$$

where g_1 is a linear layer and $p_{t,k}$ is the probability of each action at time step t.

2.3 Learning

We use Teacher-Forcing (Luong et al., 2015) method to train the model. We have two loss functions, $\mathcal{L}_{direction}$ and $\mathcal{L}_{stop}$, for direction decider and stop indicator, respectively. $\mathcal{L}_{direction}$ is a regular cross-entropy loss function,

$$\mathcal{L}_{direction} = -\sum_t \sum_k q_{t,k} log(p_{t,k}) \qquad (5)$$

Where $q_{t,k}$ denotes ground truth label for each action at time step t. For the stop indicator, we use a weighted cross-entropy loss, where we assign a greater weight for the stop signal in the loss function and therefore force the agent to pay more attention to the stop action, in formula,

$$\mathcal{L}_{stop} = \sum_t -o_t log(s_{t,1}) - \lambda(1 - o_t)log(s_{t,2}) \qquad (6)$$

where o_t are the ground-truth non-stop signals, and λ is the weight for the stop signal. Finally, the agent is optimized with a weighted sum of two loss functions:

$$\mathcal{L}_{loss} = \gamma \mathcal{L}_{direction} + (1 - \gamma)\mathcal{L}_{stop} \qquad (7)$$

where γ is the weight balancing the two losses.

3 Experiments and Analysis

3.1 Experimental Settings

TOUCHDOWN Dataset We evaluate our approach on the TOUCHDOWN dataset (Chen et al., 2019) for VLN in real-world urban environment. The navigation environment includes 29,641 panoramas and 61,319 edges from New York City. The dataset contains 9,326 examples of navigation tasks, which are pairs of ground-truth trajectory and instructions describing the trajectory. The dataset is split into training (6,526 examples), development (1,391) and test (1,409) sets.

Evaluation Metrics Following Chen et al. (2019), we report three evaluation metrics for the VLN task in urban environments: Task Completion (TC), Shortest-path Distance (SPD), and Success weighted by Edit Distance (SED). We also add another two metrics evaluating the alignment between the trajectories and the instructions: Coverage weighted by Length Score (CLS)[3] (Jain et al., 2019) and Success weighted by normalized Dynamic Time Warping (SDTW)[3] (Magalhaes et al., 2019).

Implementation Details The proposed framework and the baselines are implemented in PyTorch (Paszke et al., 2019), and the training of the models costs at average 6 hours. We use Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.00025 to train the model. The text encoder consists of a word embedding layer of size 32, and a bi-directional single-layer RNN with 256 hidden units. A single-layer fully connected layer of size 512 is used to map the previous hidden states, which is then used to compute the soft-attention to get the text representation. The visual encoder is

---

[3]https://github.com/google-research/google-research/tree/master/r4r

#	Model	TC↑	SPD↓	SED↑	CLS↑	SDTW↑
1	ARC +L2STOP	**19.48**	**17.05**	**19.02**	**55.68**	**18.97**
2	- one branch	15.40	18.33	14.92	52.00	14.86
3	- no key points	15.18	18.17	14.55	51.67	14.44
4	- no weighting	12.65	21.60	12.22	47.91	12.20

Table 2: Ablation study results for individual components on the development set.

#	Model	TC (Dev Set)
1	ARC + L2STOP	19.48
2	w/ Oracle Direction	30.63
3	w/ Oracle Stop	61.04

Table 3: Effect of oracle direction and stop.

a three-layer CNN. The first layer uses 32 8×8 kernels with stride 4, and the second layer uses 64 4×4 kernels with stride 4, applying ReLu non-linearities after each convolutional operation. Then a single fully-connected layer including biases of size 256 follows. An action embedding layer of size 16 is learned to map the previous action at every time step. Then, we concatenate the text representation, the visual representation, and the action embedding to get the input of the trajectory encoder. The trajectory encoder is a single-layer RNN with 256 hidden states. The time embedding layer is a single fully-connected layer including biases of size 32. Both of the stop indicator and the direction decider consist of a single-layer perceptron with biases and a SOFTMAX operation to compute the action probabilities.

3.2 Experimental Results

We compare the performance of our approach with the baselines: (1) Random: randomly take actions at each time step. (2) GA and RCONCAT: the baseline models reported in the original dataset paper (Chen et al., 2019). We adapt the RCONCAT model by equipping it with an attention mechanism on instruction representation to get our Attention-RConcat (ARC) model that outperforms RCON-CAT. Then we integrate ARC with the proposed L2STOP module, which further boosts the performances on all metrics and achieves the best results on both development and test sets.

In Table 1, our approach substantially outperforms the baseline models, improving SED from 9.45% to 16.34%. Significant improvements on both goal-oriented metrics (TC, SED) and path alignment metrics (CLS, SDTW) demonstrate the effectiveness of L2STOP model in instruction following and goal achievement, which also validate that L2STOP learns not only where to go but also where to stop better.

3.3 Modularity

We compare the performance between the baseline models with and without L2STOP module. The results are shown in Table 1. Integrated with the L2STOP module, both of the baseline models show improvements on all the metrics. It demonstrates that our approach is model-agnostic and generalizable: the L2STOP module can be plugged into other VLN models and enhance their navigation performance in the urban environment.

3.4 Ablation Study

Effect of Individual Components We conduct an ablation study to illustrate each component's effect on the development set in Table 2. Row 2-4 shows the influence of each component by removing them respectively from the final model (ARC with L2STOP module). Removing any of the components results in worse performance, proving the indispensability of all components in our model. Row 2 shows the results of ARC with only one policy module, which will disable *turn left* and *turn right* actions when the agent is not at key points. The results evaluate the effectiveness of the two-branch structure for providing different sub-policies for STOP and other actions. Row 3 shows the results of the model whose Direction Decider makes decisions at every time step instead of only at key points. The results validate the effectiveness of only choosing directions at key points. Row 4 shows the results where the stop signal's weight is the same as the non-stop signal in the loss function of Stop Indicator. The worst results validate the importance of STOP action. When stop and non-stop signals are treated equally, the agent will prefer non-stop because of its higher occurrence frequency.

Which Is More Important, Stop or Direction? In Table 3, we study the effect of making either Direction Decider or Stop Indicator an Oracle to see to what extent the model can be improved. *Oracle Direction* means the Direction Decider always chooses correct directions, and *Oracle Stop* means the Stop Indicator always produces ground

Red: Trajectory and actions of our two-branch model; Green: Trajectory and actions of baseline model; ★: Ground truth target

Instruction: Turn so that you're going in the same direction as the cars. Walk forward until you pass the row of blue bikes and reach the next intersection. Turn right at the intersection and continue straight until the next intersection. At the intersection, turn right again. There will be a black fire hydrant on the left sidewalk.

Instruction: Turn so you'll be walking in the same direction as traffic on this one-way street. Go to the first intersection and turn right. After you turn, keep an eye on the right side. You'll pass a theater with a bunch of stuff on the marquis. Then you'll pass a bike locked up at a small bike rack. After the bike, look for a small parking pay station on the right (it's like a chunky square pole, with a blue sign/sticker on it). Stop when you're pretty much even with that pole/kiosk (you'll be ever so slightly past it)

(a) A case where baseline stops too late　　　(b) A case where baseline stops too early

Figure 3: Case study. We choose two cases from the development set, where our proposed model is successful, but the baseline stops either too late or too early. Red boxes show the key items to recognize the target.

truth stop signals as long as the agent reaches there. **First**, Row 2 shows the stop branch has about 30% chance to stop at the right position when the agent is on the right path. **Second**, The performance in Row 3 is much greater than that in Row 1, indicating that although our approach improves agent's stop ability, the performance is still seriously limited by the wrong stop problem. This indicates that the wrong stop problem in VLN deserves more attention and further study.

3.5 Case Study

We provide visualizations for two qualitative examples to further illustrate how our L2STOP model learns to stop better in Figure 3. In both cases, our model and the baseline model are on the right path to the target. However, the baseline stops either too late or too early. Specifically, In (a), the baseline agent fails to recognize the black fire hydrant on the target but stops at a place where another black fire hydrant is visible. In (b), the baseline agent successfully recognize the parking pay station on the right, but it ignores the instruction "slightly past it"

and just stops immediately. In contrast, our agent stops in the right place.

4 Conclusion

We investigate the importance of the STOP action and study how to learn a policy that can not only make better decisions on where to go but also stop more accurately. We propose the L2STOP module for the vision-language navigation task situated in urban environments. Experiments illustrate that L2STOP is modular and can be plugged into other VLN models to further boost their performance in urban environments.

Acknowledgement

703

References

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.

Valts Blukis, Dipendra Misra, Ross A Knepper, and Yoav Artzi. 2018. Mapping navigation instructions to continuous control actions with position-visitation prediction. *arXiv preprint arXiv:1811.04179*.

Jake Bruce, Niko Sünderhauf, Piotr Mirowski, Raia Hadsell, and Michael Milford. 2018. Learning deployable navigation policies at kilometer scale from a single traversal. *arXiv preprint arXiv:1807.05211*.

Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. 2019. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12538–12547.

Volkan Cirik, Yuan Zhang, and Jason Baldridge. 2018. Following formulaic map instructions in a street simulation environment. In *2018 NeurIPS Workshop on Visually Grounded Interaction and Language*.

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. 2018. Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems*, pages 3314–3325.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Vihan Jain, Gabriel Magalhaes, Alex Ku, Ashish Vaswani, Eugene Ie, and Jason Baldridge. 2019. Stay on the path: Instruction fidelity in vision-and-language navigation. *arXiv preprint arXiv:1905.12255*.

Liyiming Ke, Xiujun Li, Yonatan Bisk, Ari Holtzman, Zhe Gan, Jingjing Liu, Jianfeng Gao, Yejin Choi, and Siddhartha Srinivasa. 2019. Tactical rewind: Self-correction via backtracking in vision-and-language navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6741–6749.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Ang Li, Huiyi Hu, Piotr Mirowski, and Mehrdad Farajtabar. 2019. Cross-view policy learning for street navigation. *arXiv preprint arXiv:1906.05930*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, Ghassan Al-Regib, Zsolt Kira, Richard Socher, and Caiming Xiong. 2019a. Self-monitoring navigation agent via auxiliary progress estimation. *arXiv preprint arXiv:1901.03035*.

Chih-Yao Ma, Zuxuan Wu, Ghassan AlRegib, Caiming Xiong, and Zsolt Kira. 2019b. The regretful agent: Heuristic-aided navigation through progress estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6732–6740.

Gabriel Magalhaes, Vihan Jain, Alexander Ku, Eugene Ie, and Jason Baldridge. 2019. Effective and general evaluation for instruction conditioned navigation using dynamic time warping. *arXiv preprint arXiv:1907.05446*.

Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. 2018. Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*, pages 2419–2430.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037.

Hao Tan, Licheng Yu, and Mohit Bansal. 2019. Learning to navigate unseen environments: Back translation with environmental dropout. *arXiv preprint arXiv:1904.04195*.

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

Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. 2019. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6629–6638.

Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. 2018. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 37–53.

Jiannan Xiang, Xin Wang, and William Yang Wang. 2019. Not all actions are equal: Learning to stop in language-grounded urban navigation. In *ViGIL@ NeurIPS*.

A Appendices

A.1 Analysis of Model Structure

In Fig. 4, we examine four model structures to evaluate the interactions between the two branches: (1) **Separate Enc-Dec** model, where two encoder-decoder models are trained separately for two branches. (2) **Shared Enc** model, which has a shared encoder but uses two different decoders for two branches. (3) **Shared Dec** model, which has different encoders for both linguistic and visual input but shared trajectory decoder. (4) **shared Enc-Dec** model, which shares both the encoder and the decoder. Note that this is the final architecture we use, which is demonstrated in Sec. 2. Table 4 shows the performance of four architectures on the development and test set. **First**, despite worse performance on other metrics, Separate Enc-Dec can achieve competitive performance on SPD and CLS against other two-branch shared models. The results show that the Separate Enc-Dec agent can produce high-fidelity trajectory matched with instruction but fail to stop at the correct location. This shows that to stop better, the stop indicator requires the information from the direction branch. **Second**, compared with Shared Enc model, Shared Dec performs competitively on SPD and CLS while much worse on other metrics, indicating that the stop branch learns better from the direction branch in the encoder phase. **Third**, both Shared Enc and Shared Dec show stronger ability to learn to stop; thus we use Shared Enc-Dec model, which requires fewer parameters. Improved performance shows the Shared Enc-Dec model learns to stop better than other architectures.

A.2 Hyper-Parameters Sensitivity Analysis

Threshold for Stop Signal   We study the sensitivity of the threshold for stop signals on the development set. The result is shown in Fig. 5 (a). Task-Completion (TC) is consistent in a large range of thresholds, with a slight drop when the threshold is getting higher than 0.7 and sharp decreases when the threshold is close to 0 and 1. The results demonstrate that our approach is insensitive to the change of threshold for stop signals. The consistency of the performance means that the scores of stop signals are either low or high, rarely intermediate. This proves that our approach enables the agent to pay more attention to STOP; that is, the agent is cautious about deciding to stop and only stop when it is highly confident it reaches the goal.
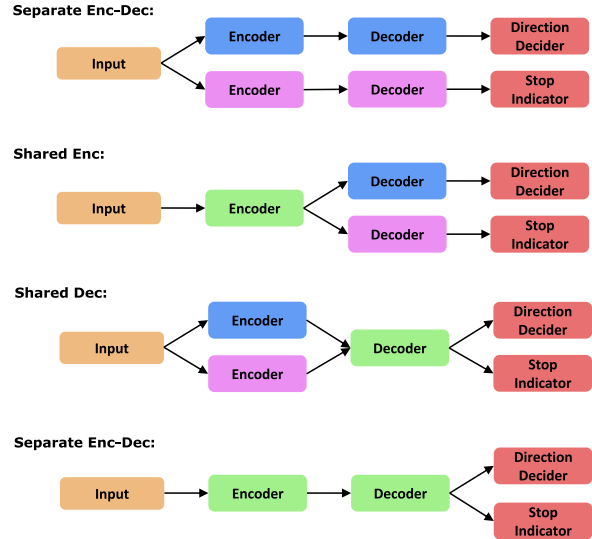

Figure 4: Two-branch VLN models. Input includes language instruction and local visual scene. one Encoder consists of a Visual Encoder and a Text Encoder in Fig. 2, and Decoder represents Trajectory Encoder in Fig. 2.

Direction Branch Weight   We study the sensitivity of direction branch weight γ on the development set. The optimal value for γ is 0.6, as depicted in Fig. 5 (b), which demonstrates that the balance between the loss functions of two branch enables the agent to not only select correct directions at key points but also stop at the right place. As shown in the figure, smaller γ (0-0.5) results in relatively worse performance than higher γ, indicating that small γ enforces the agent to concentrate too much on STOP but ignore the choice for direction. Consistently good performance with larger γ (0.6-0.85) shows that only a small weight for the stop branch can significantly improve the agent's stop ability.

Stop Signal Weight   We study the sensitivity of stop signal weight λ on the development set. As shown in the Fig. 5 (c), the optimal value for λ is 20. We can see that when $\lambda = 0$, our model's performance is similar to the ARC model (15.53 as shown in Table 1). However, when setting greater λ, the TC shows fluctuations, but is consistently better than ARC's performance. Only when λ increases to a large number of 80 does the performance decline sharply. This demonstrates the effectiveness of our proposed Weighted Cross-Entropy loss function, which consistently improves the agent's stop ability with a large range of λ.

	Development					Test				
Model	TC↑	SPD↓	SED↑	CLS↑	SDTW↑	TC↑	SPD↓	SED↑	CLS↑	SDTW↑
Separate Enc-Dec	13.71	<u>17.67</u>	13.35	<u>55.24</u>	13.32	14.14	**17.40**	13.71	<u>54.56</u>	13.61
Shared Dec	14.43	18.45	14.05	52.90	14.00	12.29	<u>17.87</u>	11.86	**54.86**	11.74
Shared Enc	<u>18.75</u>	18.19	<u>18.32</u>	52.42	<u>18.27</u>	<u>15.55</u>	18.31	<u>15.21</u>	52.87	<u>15.19</u>
Shared Enc-Dec	**19.48**	**17.05**	**19.02**	**55.68**	**18.97**	**16.68**	18.84	**16.34**	53.50	**16.34**

Table 4: Performance comparison for four different architectures of the two-branch model.
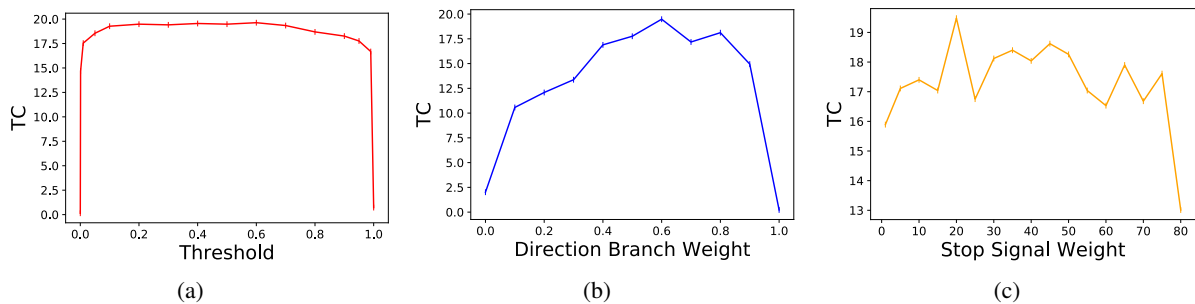

(a) (b) (c)

Figure 5: (a) Task Completion (TC) scores with different thresholds for the stop signal ($s_{t,2}$ in Equation 6). TC shows insensitivity to different thresholds. (b) TC scores with different direction branch weights γ in Equation 7. $\gamma = 0.6$ gives the highest TC. (c) TC scores with different stop signal weight λ in Equation 6. $\lambda = 20$ gives the highest TC. All the experiments are done on the development set,