Finding Sub-task Structure with Natural Language Instruction

Ryokan Ri[1]
li0123@logos.t.u-tokyo.ac.jp*

Yufang Hou[2]
YHou@ie.ibm.com

Radu Marinescu[2]
radu.marinescu@ie.ibm.com

Akihiro Kishimoto[2]
Akihiro.Kishimoto@ibm.com

[1]The University of Tokyo [2]IBM Research

Abstract

When mapping a natural language instruction to a sequence of actions, it is often useful to identify sub-tasks in the instruction. Such sub-task segmentation, however, is not necessarily provided in the training data. We present the A2LCTC (Action-to-Language Connectionist Temporal Classification) algorithm to automatically discover a sub-task segmentation of an action sequence. A2LCTC does not require annotations of correct sub-task segments and learns to find them from pairs of instruction and action sequence in a weakly-supervised manner. We experiment with the ALFRED dataset and show that A2LCTC accurately finds the sub-task structures. With the discovered sub-tasks segments, we also train agents that work on the downstream task and empirically show that our algorithm improves the performance.

1 Introduction

Building computational agents that execute actions given natural language instruction has a great deal of potential in real-world applications. One common approach is to cast the problem as mapping from instruction text into action sequence, and train an agent with supervised learning (Chen and Mooney, 2011; Mei et al., 2016). A challenge on effective machine learning stems from a long horizon of the tasks. Typical navigation tasks often involve more than a paragraph of instructions (Chen and Mooney, 2011; Misra et al., 2017; Shridhar et al., 2020). In such cases, many existing approaches exploit the task hierarchy, *e.g.*, decompose one episode of the task into sub-tasks and treat them as separate training instances (Mei et al., 2016), incorporate the hierarchical information into the model (Zhu et al., 2020), or aid the learning process with progress monitoring (Ma et al., 2019).

However, annotations for such a decomposition are not necessarily available in training data. In this case, previous work has attempted to perform sub-task segmentation by augmenting data through crowdsourcing (Hong et al., 2020), or developing a heuristic algorithm (Zhu et al., 2020).

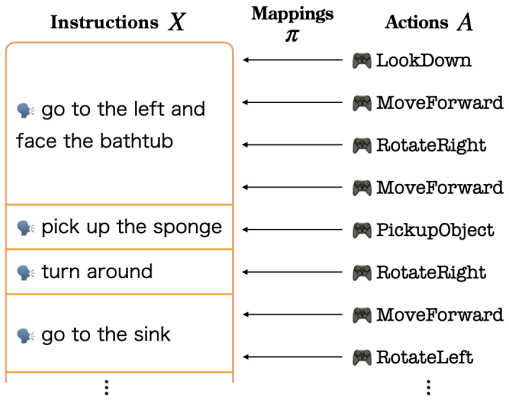

Figure 1: Example illustrating a mapping from each action to a corresponding fine-grained instruction

In this paper, we present A2LCTC (Action-to-Language Connectionist Temporal Classification), an unsupervised algorithm that *automatically* discovers a sub-task segmentation of an action sequence. Given pairs of a natural language instruction and action sequence, A2LCTC maps each action to a fine-grained instruction (Figure 1).

We formulate the problem of sub-task segmentation as classification of each action into a fine-grained instruction. Inspired by the connectionist temporal classification algorithm (Graves et al., 2006), we consider an objective function that maximizes the log-likelihood of the coarsely aligned data. This formulation allows us to learn the classification in a weakly-supervised manner without any need of the ground truth mapping.

We experiment with the ALFRED dataset which involves navigating a robot to perform household task (Shridhar et al., 2020). A2LCTC successfully discovers the sub-task information in the unsegmented training data (§3), which are shown to be useful for the downstream task (§4).

* Work done as an intern at IBM Research.

Proceedings of the First Workshop on Learning with Natural Language Supervision, pages 1 - 9

2 Learning Sub-task Segmentation

Given a training instance of a navigation task which consists of language instruction X and action sequence $A = [a_1, ..., a_T]$, we aim at finding a decomposition of the instance $(X, A) = [(X_1, A_1), ..., (X_L, A_L)]$, which is semantically plausible and useful for learning the task.

Our approach starts with dividing the instruction into fine-grained segments. In the experiment, the segment X_i corresponds to a verbal phrase (*e.g.*, "*go to the desk*") extracted from the instruction using a simple rule-based algorithm (Appendix A.1).

2.1 Formulating the Task as Temporal Classification

We formulate the decomposition task as classification of actions into one of the fine-grained instructions: our algorithm predicts a mapping $\pi = [\pi_1, ..., \pi_T]$ where $\pi_t \in [1, L]$.

We assume that the alignment is monotonic, *i.e.*, the actions and instructions are both arranged in a chronological order, which typically holds for step-by-step instruction text[1]. This formulation allows us to develop an *unsupervised* learning method to effectively solve the task without ground-truth label mappings. We introduce A2LCTC (Action-to-Language Connectionist Temporal Classification), which is based on the CTC algorithm originally used for speech recognition (Graves et al., 2006) or action labeling in video (Huang et al., 2016).

Our model attempts to maximize the following likelihood for each training instance:

$$\sum_{\{\pi|\mathcal{B}(\pi)=[1,2,..,L]\}} P(\pi|X, A) \qquad (1)$$

where $\mathcal{B}$ is the operator to remove repeated labels, *e.g.*,, $B([1, 1, 2, 2, 2, 3]) = [1, 2, 3]$. That is, the objective is the sum of the probabilities over all the possible assignments under the monotonic constraint. Under the conditional independence assumption, we further decompose the likelihood into $P(\pi|X, A) = \prod_t^T P(\pi_t|X, A)$.
With this decomposition, we employ the forward algorithm (Stratonovich, 1965) to efficiently calculate the sum over all possible paths.

[1]In case that the monotonic assumption does not hold, we could reorder instructions with some sentence-ordering algorithm such as (Ghosal et al., 2021) as preprocessing.

2.2 Modeling with Neural Network

We model the probability computation using neural networks. In our approach, each fine-grained instruction and action are represented as feature vectors. We then define the probability $P(\pi_t|X, A)$ as the softmax of the dot product of the feature vectors of the action $\boldsymbol{a}_t$ and the instruction $[\boldsymbol{x}_1, ..., \boldsymbol{x}_L]$:

$$P(\pi_t = i|X, A) = \frac{\exp(\boldsymbol{a}_t \cdot \boldsymbol{x}_i)}{\sum_{j=1}^{L} \exp(\boldsymbol{a}_t \cdot \boldsymbol{x}_j)}. \qquad (2)$$

In our implementation, a fine-grained instruction X_i is tokenized into words and $\boldsymbol{x_i}$ is computed as the average of the word embeddings followed by a linear layer. The action feature vectors are computed by feeding action embeddings $[\hat{\boldsymbol{a}}_1, ..., \hat{\boldsymbol{a}}_T]$ into a one-layer bidirectional LSTM (Hochreiter and Schmidhuber, 1997) to capture the semantics of actions in the context.

At inference time, the model induces the best mapping π that maximize $P(\pi|X, A)$ with the viterbi algorithm to form decomposed instances (X_i, A_i).

2.3 Stabilization of Unsupervised Learning

Optimizing the aforementioned objective function with neural networks turns out to be highly unstable. Therefore, we further employ the following techniques to better guide the training process.

Length-based curriculum learning. The training objective (1) is defined as the sum of the probabilities over all possible assignments, whose size grows at the speed of $\mathcal{O}(L^T)$. When L and T are large, too many degenerated assignments take up a large part of the probability mass, which hinders the training especially at the beginning of training.

To address this problem, we adopt the curriculum learning framework (Bengio et al., 2009). At the beginning of training, we restrict the training instances with a shorter action sequence and gradually expose the model to longer instances (Spitkovsky et al., 2010). Details are shown in Appendix A.4.

Initializing with pre-trained word embeddings. We initialize the word embeddings $\boldsymbol{w}$ with the Glove embeddings (Pennington et al., 2014) to inject a model with the prior knowledge of words.

Initializing with pre-trained action embeddings. We also initialize the action embeddings $\hat{\boldsymbol{a}}$ with pre-trained embeddings. We obtain action embeddings from the action sequences in the training data with Skip-gram (Mikolov et al., 2013).

3 Experiments

We test A2LCTC with the ALFRED dataset (Shridhar et al., 2020) [2], where the agent performs household tasks in a simulated indoor environment given English instruction. We choose this dataset because (1) the data involves relatively long sequence of actions (over 100); (2) it offers ground-truth sub-task segmentation data which allows the evaluation of our algorithm. Although the ALFRED task itself involves understanding visual inputs, which is currently beyond the scope of A2LCTC, the action set defined in the dataset is semantically rich enough for A2LCTC to solve the segmentation problem.

3.1 Experimental Setups

Training and Evaluation Data. The dataset contains expert demonstrations, which are split into the training set and two validation sets: *valid_seen* and *valid_unseen*. In *valid_seen*, the ALFRED tasks are given in the same set of environments as the training data, while unseen environments are used for *valid_unseen*. We use *valid_seen* as the validation set during training and report the results evaluated with both of them.

The task defines 12 types of actions: five for navigation (e.g., `LookUp`, `MoveForward`, `RotateRight`) and seven for interaction (e.g., `Pickup`, `Slice`). To simplify training and evaluation, we preprocess data by merging the same consecutive navigation actions into a single action (see Appendix B for detailed statistics on the data).

Evaluation Metrics. Each training instance in the ALFRED dataset contains ground-truth sub-goal segments for instructions and actions. We use them to evaluate the learned sub-task segments.

Note that our algorithm operates on finer-grained instructions segmented into verbal phrases, while the ground-truth segments are coarser; some sentences contain a couple of verbal phrases (*e.g.*, "*Rinse the sponge out in the sink, and pick it up again*"). In our evaluation, we first merge the fine-grained instructions into their corresponding sub-goal instruction, together with the mapped actions, and then compare the overlap with the gold sub-goal segments of actions.

We report the sub-goal exact match (EM) score defined as the percentage of the sub-goal segments perfectly reconstructed. We also report the sub-goal F1 score defined as the macro average of the F1 scores calculated by taking the overlap between the predicted and ground-truth segment.

[2] `https://askforalfred.com/`, MIT License

3.2 Baselines

Under our task formulation, A2LCTC offers an advantage that it can take into account the temporal constraint and the semantics of instructions. However, many existing unsupervised sequence alignment algorithms (*e.g.*, IBM models) operate only between discrete symbols and are not directly applicable in this situation where we need to align actions to text (or a bag of words in our model). Thus, we compare A2LCTC with two baselines that do not consider the textual information.

Uniform. The uniform baseline assigns an equal number of actions to each of the fine-grained instructions.

Byte-pair Encoding. The byte-pair encoding (BPE) baseline is based on a data compression algorithm that finds repeated patterns in the data and merges them into chunks (Gage, 1994; Sennrich et al., 2016). Concretely, we repeat the following two steps until each action sequence is split into the number of the corresponding fine-grained instructions: (1) count bigrams in the action sequences; (2) merge the most frequent bigrams.

3.3 Results

Table 1 shows that A2LCTC significantly outperforms the baselines, which indicates that our model successfully leverages textual information and learns meaningful alignments between the fine-grained instructions and actions.

Although the BPE baseline does not use textual information, it exhibits reasonable F1 scores (61.9 points in *valid_unseen* and much higher EM scores than UNIFORM (27.1 vs. 9.3 points). This reflects the characteristics of the ALFRED dataset. As each episode in the dataset is generated from specific templates, the actions follow specific patterns, which enable the BPE to learn correct segmentation to some extent.

3.4 Ablation Study

In A2LCTC, we utilize several techniques to stabilize the training process. Table 2 shows the effects of ablating one stabilization method from the full A2LCTC model. Our results indicate that curriculum learning is most essential to successful training. Without curriculum learning, A2LCTC suffers from significant performance degradation (41.2 $\rightarrow$ 14.6 in EM and 78.2 $\rightarrow$ 36.7 in F1 score).

	valid_seen		*valid_unseen*	
	EM	F1	EM	F1
UNIFORM	8.8	54.4	9.3	55.5
BPE	22.8	61.9	27.1	65.9
A2LCTC	**61.2**	**85.3**	**58.5**	**85.1**

Table 1: Performance for sub-task segmentation. The value of A2LCTC is the best among 10 runs with different random seeds.

	EM	F1
Full	41.2 ± 10.2	78.2 ± 10.8
- pre. language	33.5 ± 14.1	70.2 ± 18.0
- pre. action	41.1 ± 13.4	76.9 ± 18.2
- curriculum	14.6 ± 5.7	36.7 ± 8.0

Table 2: Ablation model performance (*valid_seen*). The values show the mean and standard deviation of 10 runs.

Ablating pre-training language or action embeddings still obtains mean values comparable to the full A2LCTC but yields much larger standard deviations. This indicates that these two stabilization methods are also beneficial for A2LCTC.

4 Evaluation with the Downstream Task

We evaluate the effectiveness of sub-task segments induced by A2LCTC on the downstream ALFRED task.

Models. Our baseline agent (BASELINE) is based on the CNN-LSTM sequence-to-sequence architecture in Shridhar et al. (2020), which takes the whole instruction and current state as input and then predicts an action at each time step. Unless specified, we use the same hyperparameters as the original implementation.

To incorporate the sub-task information, we extend the baseline with the progress monitoring module (Ma et al., 2019). We use two progress monitoring schemes from Shridhar et al. (2020), which estimate the current time step and the completed sub-tasks. Specifically, the modules are trained to predict the proportion of elapsed steps or completed sub-tasks to the total numbers.

We evaluate the segmentation of A2LCTC as well as the two baseline methods (UNIFORM and BPE). Those algorithms are applied on the training data and the agents are trained with the progress monitoring according to the segmentation.

Metric. We evaluate the agents with the subgoal sequence accuracy. The agent predicts the next action given the history from an expert trajectory. The metric measures how many subgoal chunks of actions, which is defined by the ground-truth segmentation of the dataset, are successfully predicted in the evaluation data. Note that this metric simplifies the original task in that it ignores the object interactions and focuses on action prediction[3].

	valid_seen	*valid_unseen*
BASELINE	57.6 ± 2.0	29.6 ± 1.5
UNIFORM	63.6 ± 2.5	38.8 ± 1.4
BPE	66.0 ± 2.0	39.1 ± 0.4
A2LCTC	$\mathbf{69.7 \pm 1.4}$	$\mathbf{41.4 \pm 0.8}$

Table 3: Performance on the ALFRED task measured by the subgoal sequence accuracy. The values show the mean and standard deviation of 5 runs.

Results. Table 3 summarizes the result. On both *valid_seen* and *valid_unseen* splits, the agents trained with fine-grained instruction (UNIFORM, BPE and A2LCTC) significantly outperform BASELINE. The fact that UNIFORM achieves improvement indicates that keeping track of detailed progress is helpful even if it inaccurately performs the fine-grained task segmentation (see Section 3.3). A2LCTC performs best because of the better accuracy of the segmentation than the others. This demonstrates that A2LCTC successfully provides more informative instructions for the agent in solving the downstream task.

5 Conclusion

We presented A2LCTC, which finds a hierarchical structure of an action sequence by mapping each action to fine-grained natural language instructions without ground-truth mapping data. We demonstrated that A2LCTC successfully learns meaningful segments and training the ALFRED agents with these segments leads to improved performance.

A2LCTC currently relies only on semantic correspondence between actions and text. Applying A2LCTC to the tasks with low-level actions is an important extension, *e.g.*, actions specifying the direction to move. Furthermore, the instruction may often describe the visual input whose information is not encoded in the actions. Another important future direction is to incorporate visual or additional information to tackle a broader range of domains.

[3]We find the original navigation task is too difficult for the baseline model: the success rate is very low with high variance, which prevents meaningful comparison among the variants of the model (Appendix D).

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *ICML*, volume 382 of *ACM International Conference Proceeding Series*, pages 41–48. ACM.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *AAAI*.

P. Gage. 1994. A new algorithm for data compression. *The C Users Journal archive*, 12:23–38.

Deepanway Ghosal, Navonil Majumder, Rada Mihalcea, and Soujanya Poria. 2021. Stack: Sentence ordering with temporal commonsense knowledge. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Alex Graves, Santiago Fernández, Faustino J. Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 369–376. ACM.

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

Yicong Hong, Cristian Rodriguez, Qi Wu, and Stephen Gould. 2020. Sub-instruction aware vision-and-language navigation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3360–3376, Online. Association for Computational Linguistics.

De-An Huang, Li Fei-Fei, and Juan Carlos Niebles. 2016. Connectionist temporal modeling for weakly supervised action labeling. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 137–153. Springer.

Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, G. Al-Regib, Z. Kira, R. Socher, and Caiming Xiong. 2019. Self-monitoring navigation agent via auxiliary progress estimation. *ArXiv*, abs/1901.03035.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *AAAI*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Dipendra Misra, John Langford, and Yoav Artzi. 2017. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1004–1015, Copenhagen, Denmark. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2010. From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 751–759, Los Angeles, California. Association for Computational Linguistics.

Ruslan Leont'evich Stratonovich. 1965. Conditional markov processes. In *Non-linear transformations of stochastic processes*, pages 427–453. Elsevier.

Wang Zhu, Hexiang Hu, Jiacheng Chen, Zhiwei Deng, Vihan Jain, Eugene Ie, and Fei Sha. 2020. BabyWalk: Going farther in vision-and-language navigation by taking baby steps. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2539–2556, Online. Association for Computational Linguistics.

A Implementation details of A2LCTC

A.1 Instruction Decomposition

We split a whole instruction into fine-grained instructions each of which is a verbal phrase. Our current implementation employs a simple rule-based algorithm, which segments a whole instruction at periods, commas, phrases such as *and* or *then*. For example, "*Turn around and go back to the table*" will be segmented into ["*Turn around*", "*go back to the table*"], while we also write few rules to handle erroneous splits such as conjunctions between nouns (["*take the apple*", "*(and) banana*"]), commas before a prepositional phrase (["*Put the bowl*", "*(,) on the coffee table to the left of the statue*"]), or verbal phrases that cannot be instruction by itself (["*go to the left*", "*(and) face the bathtub*"]).

A.2 Neural Network Architecture

Instruction Feature Vectors

The feature vector of a fine-grained instruction $\boldsymbol{X_l}$ is simply modeled by taking the average of the word embeddings $[\boldsymbol{w}_1, ..., \boldsymbol{w}_N]$ followed by a linear layer (the results from different encoding strategies are shown in Appendix C).

$$\boldsymbol{x} = \tanh(\text{Linear}(\frac{1}{N}\sum_{i}^{N}\boldsymbol{w}_i)). \tag{3}$$

In our experiment, the dimension of word embeddings, the input and output size of the linear layer is all set to 50. The total number of parameters of A2LCTC is about 44K. The training takes approximately one hour with a single GPU.

Action Feature Vectors

The action feature vectors are computed through feeding embeddings for primitive actions $[\hat{\boldsymbol{a}}_1, ..., \hat{\boldsymbol{a}}_T]$ to a one-layer bidirectional LSTM.

$$\boldsymbol{a}_1, ..., \boldsymbol{a}_T = \text{LSTM}(\hat{\boldsymbol{a}}_1, ..., \hat{\boldsymbol{a}}_T) \tag{4}$$

where $[\hat{\boldsymbol{a}}_1, ..., \hat{\boldsymbol{a}}_T]$ are embeddings for primitive actions.

The size of the action embeddings and the hidden size of the LSTM are set to 50. The outputs of LSTM in the forward and backward directions are summed to merged into one single feature vector.

A.3 Hyperparameters for training

optimizer	Adam
learning rate	0.001
batch size	128
validation metric	the sub-goal F1 score
patience	5

A.4 Stabilization of Unsupervised Learning

Length-based curriculum learning

We set the maximum length of training instances for each training epoch according to a schedule. In our experiment, the maximum length starts at 20, and linearly increases to 60 with 30 steps.

Initializing with pre-trained action embeddings

We use the `gensim` library[4] to train action embeddings. We use the Skip-gram algorithm and the hyperparameters are shown in Table 4

[4] https://radimrehurek.com/gensim/

embedding size	50
window size	1
# of iterations	15
# of negative samples	5

Table 4: The hyperparameters for training action embeddings

B Data Statistics

Our experiments are based on the expert demonstration data in the ALFRED dataset (Table 5).

training data	valid_seen	valid_unseen
21,023	820	821

Table 5: The number of expert demonstrations in the ALFRED dataset.

After the instruction decomposition, the instructions contain 10 fine-grained instructions on average. The entire distribution is shown in Figure2.

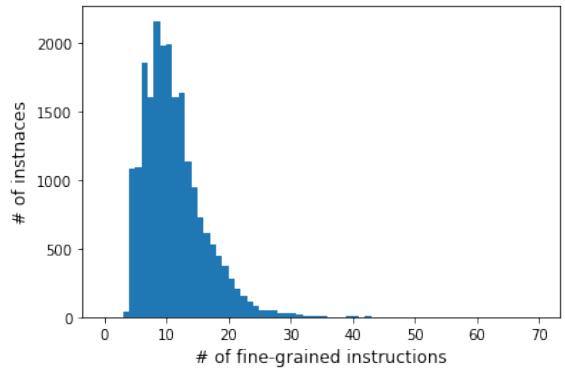

Figure 2: The distribution of the number of fine-grained instructions in the training split

To simplify training and evaluation, we preprocess data by merging the same consecutive actions into one single action. This results in the average sequence length of 25. The entire distribution before and after the merge preprocessing is shown in Figure3 and 4.

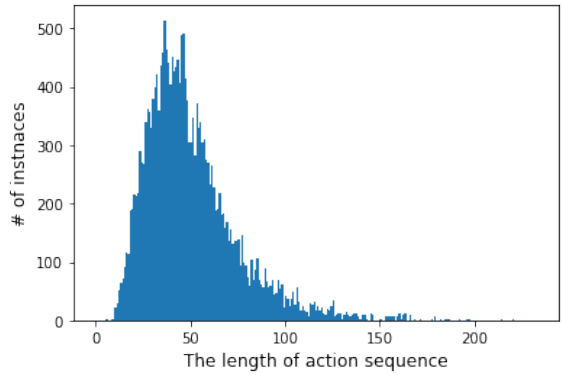

Figure 3: The distribution of action sequence length in the training split.

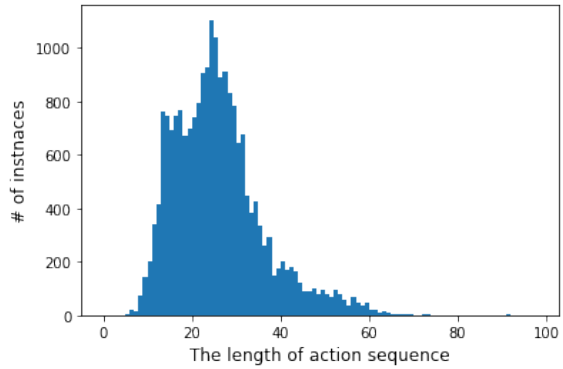

Figure 4: The distribution of action sequence length in the training split after the merge preprocessing.

C Additional Results for the Segmentation Task

Here we compare different strategies for encoding instruction segments. Besides the mean pooling of word embeddings followed by a linear layer in A2LCTC (MEAN), we also tried the summation of word embeddings (SUM), the mean pooling of one-layer Bi-LSTM outputs LSTM. The hidden size of LSTM is set to 50, which is the same as the word embeddings, and the vectors of the forward and backward directions were summed to form the output vectors. The result is shown in Table 6.

	valid_seen		*valid_unseen*	
	EM	F1	EM	F1
MEAN	41.2 ± 10.2	78.2 ± 10.8	44.5 ± 12.1	79.0 ± 11.5
SUM	34.5 ± 13.8	70.7 ± 15.2	36.8 ± 8.4	73.3 ± 12.1
LSTM	18.9 ± 10.8	51.8 ± 20.3	33.8 ± 18.2	65.1 ± 20.6

Table 6: Performance for sub-task segmentation. The value of A2LCTC is the best among 10 runs with different random seeds.

We find that MEAN gives the most stable result. LSTM exhibits the worst performance, indicating that it is hard to optimize the unsupervised objective in A2LCTC with an overly complex architecture.

D Additional Results for the Downstream Task

D.1 Results with the ground-truth sub-goal annotation

In section 4, we compared the models trained with automatically generated fine-grained sub-task segments. Here we provide the results from the model trained with the ground-truth sub-goal segments (SUBGOAL) in Table 7. Note that the granularity of SUBGOAL is coarser than the other models.

	valid_seen	*valid_unseen*
BASELINE	57.6 ± 2.0	29.6 ± 1.5
SUBGOAL	59.5 ± 2.1	31.4 ± 1.4
UNIFORM	63.6 ± 2.5	38.8 ± 1.4
BPE	66.0 ± 2.0	39.1 ± 0.4
A2LCTC	69.7 ± 1.4	41.4 ± 0.8

Table 7: Performance on the ALFRED task measured by the subgoal sequence accuracy. The values show the mean and standard deviation of 5 runs.

SUBGOAL provides better results than BASELINE, which demonstrates the benefit of the ground-truth sub-gold segmentation in the dataset. However, the improvement is limited compared to the UNIFORM segmentation, which segments an action sequence into the chunks of the same size. The model benefits from inaccurate but finer-grained segmentation more than accurate but coarse segmentation.

We hypothesize that this reflects the characteristics of the dataset. The ALFRED dataset is created by generating expert trajectories from task templates. As a result, the type of actions are somewhat correlated with the time step within an episode. For example, navigation actions such as `MoveForward` are more likely to be executed at the beginning of the episode, whereas interactive actions such as `PutObject` are at the end. Adding finer-grained progress monitoring supervision at training time can help the agent learn the correlation between time steps and actions better than coarser progress monitoring.

D.2 Success Rates of the Downstream Task

In our preliminary experiments, we find the original navigation task is too difficult for the baseline model: the success rate is very low with high variance, which prevents meaningful comparison among the variants of the model. The success rate (SC) and goal condition success rate (GC) are provided on Table 8. With multiple runs, we did not observe any significant difference ($p > 0.05$ in the Welch's t-test) among the models.

	SC		GC	
	valid_seen	*valid_unseen*	*valid_seen*	*valid_unseen*
Baseline	2.4 ± 0.9	0.0 ± 0.0	9.5 ± 0.5	6.7 ± 0.3
SUBGOAL	2.6 ± 0.5	0.0 ± 0.0	8.9 ± 0.8	6.6 ± 0.4
A2LCTC	2.3 ± 0.7	0.0 ± 0.0	9.4 ± 0.8	6.7 ± 0.3

Table 8: Performance on the ALFRED task measured by the task success rate (SC) and goal condition success rate (GC). The values show the mean and standard deviation of 5 runs.