Generate Symbolic Plans for Long-Horizon Robot Manipulation Tasks

Stanford CS224N Custom Project

Karen Yang
Department of Computer Science
Stanford University
kaiyuany@stanford.edu

Toki Migimatsu
(External collaborator)
Department of Computer Science
Stanford University
takatoki@stanford.edu

Abstract

Developing generally intelligent robots to handle a wide range of complex real-world tasks requires the ability to sequence a set of motor skills to reach the goal. We aim to explore GPT-2's language modeling capability to generate goal-directed symbolic plans in the format of a formal language called planning domain definition language (PDDL). To this end, we finetuned a pre-trained GPT-2 model on a custom-made PDDL dataset constructed from STRIPS planner. We demonstrated that our model could simultaneously adapt to the PDDL language in robotic manipulation context, and understand the underlying dynamics and state transitions to produce valid symbolic plans. Given prompts unseen during training, our model is able to produce valid symbolic plans 58% of the time. We see a potential in applying language models to do generalizable task-level planning for long-horizon manipulation tasks.

1 Introduction

Humans are incredibly fast and intelligent learners when it comes to acquiring new knowledge and skills. We have gradually developed strong language capability and constructed a scientific framework to understand the dynamics of the world around us. Humans' ability to represent, summarize the abstract dynamics of the environment helps us handle a diverse set of environments and tasks, and perform complex tasks with a long horizon.

Providing robots with similar ability to generalize across a wide range of long-horizon tasks remains a long-standing challenge. A robot needs to sequence several motor skills to reach its goal to handle long-horizon jobs in homes, factories, hospitals, and public places. This requires a complex planning system that integrates motion planning and control, perception of the environment as well as the symbolic reasoning of the given task. While all of the three are important areas of research in robotics and computer vision communities, in this paper, we primarily focus on the symbolic reasoning part.

We propose to use language models to generate goal-directed symbolic plans in planning domain definition language (PDDL)[1]. Our hypothesis is that large generative models like GPT-2 [2] pre-trained on large general domain corpora can adapt to the grammar of PDDL, which shares similar semantics but structurally different from natural language. Additionally, we hypothesize that the fine-tuned language model can capture the underlying dynamics involved in robot manipulation environments.

We constructed a PDDL symbolic plan dataset using STRIPS planner [3]. We fine-tuned a GPT-2 model on this dataset that can generate a symbolic plan consisting of intermediate actions and states, given an input prompt of the goal and initial state of the environment. We demonstrate that our model can model the PDDL language and generate valid symbolic plans from input prompts. It can generalize to novel prompts where the goal and initial states contain unseen objects and similar actions.

To our knowledge, this is the first approach that explores the language model's ability to directly generate goal-directed symbolic plans, without any additional structured processing or constraint optimization.

2 Related Work

The first automated symbolic task planner is STRIPS (STanford Research Institute Problem Solver) [4]. STRIPS represents a world model by a set of logical operators from which the hierarchy of goal, sub-goals (intermediate states) and actions can be represented by a search tree. It deploys a search strategy (say Breadth-first-search) to fine a feasible symbolic plan. Many hierarchical planning networks today are still constructed using STRIPS. For example, [3] solves for the symbolic plans in a discrete domain, and solve a nonlinear trajectory optimization problem in a continuous domain over relative object poses which makes re-planning very fast. One issue with the STRIPS planner is that it requires manual definition of the initial world model and specific rules for all operators, actions, and objects. Thus it cannot generalize to unseen words without expansive manual definition. Another issue is that when the planning domain becomes more extensive and more complicated, the search-based planning can become intractable.

Recently, reinforcement learning and imitation learning have been applied successfully to teach robots to acquire manipulation skills [5, 6, 7, 8]. However, beyond simple motor skills such as (pick up an object, stack blocks, etc.), they often become intractable once the tasks become more complex or have a longer horizon. This is because the dimensionality of the state and action space grows exponentially. To this end, researcher studies Hierarchical Reinforcement Learning methods to learn multiple levels of policies, each responsible for control at a different level of temporal abstraction. For example, the Options framework [9] jointly learns a top-level policy and bottom level policy, where the top-level policy network outputs abstract sub-task as sub-policies, and the bottom level policy network takes in the sub-policy and environment observation to outputs the primitive actions. The FUN (FeUdal Networks) [8] adopts a modular architecture where the "manager" network chooses a direction in a high-level latent state space, and the workers learn to achieve that direction through actions. There is a promising outlook for learning-based hierarchical planning to tackle large state and action space. However, there are still many challenges in designing an efficient hierarchical structure and representing the re-usable domain knowledge across tasks.

Inspired by the recent advancement of language representation learning, recent works have extensively studied using language features to ground and guide robot behaviors [10, 11, 12]. For instance, [12] uses a language encoder LSTM to encode the textual instructions into language features and inform the navigation policy in an attention mechanism. In general, natural language can be a powerful and intuitive interface for robotic tasks. Most adoption of language embeddings in problem-solving is to use them as features for an existing task-specific model for a task-specific objective. Language-based context vectors such as GloVe [13], Word Vectors [14] have also been widely used for this purpose.

Since the development of transformers, large-scale pre-trained language models like BERT[15] and GPT[2] have achieved SOTA performance on a variety of language tasks and shown zero-shot transfer learning ability in downstream language tasks. In particular, the authors of GPT [2] propose that a multitask, unsupervised language modeling training, along with a large dataset (WebText), can improve general performance. The resulting GPT-2 model, a 1.5B parameter multi-layer transformer, achieves the state of the art results on seven language modeling datasets with zero-shot task transfer. This line of work facilitates the use of contextual language representation in many applications beyond common language datasets with minimal or no supervised training. We think large generalized models like GPT-2 also has the potential to model the domain-specific language used in sequential planning tasks.

3 Approach

3.1 Problem Formulation

We aim to generate task-level symbolic plans for robotic manipulation tasks with language models.

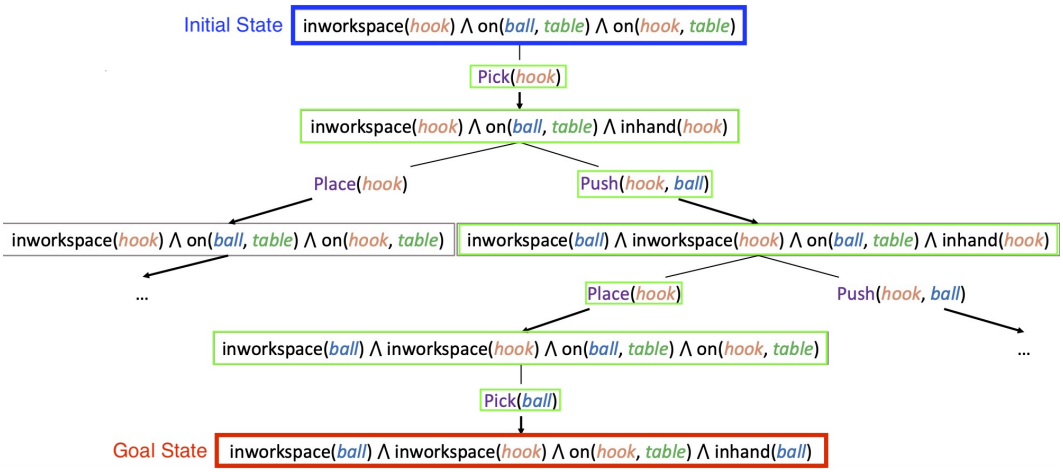

Figure 1: A symbolic plan corresponds to a path from the initial state (marked blue at the top) to the goal state (marked red at the bottom) with a sequence of $(action, state)$ pairs (marked green). Actions and states are described following the grammar of Planning Domain Definition Language (PDDL)[1]. We used a STRIPS planner[3] (oracle) that runs tree search algorithms to find feasible solutions to generate our dataset.

The input to the model is (1) the goal state that describes the desired outcome for a given task, and (2) the initial state of the environment. The output of the model is a generated sequence of $(action, state)$ pairs in PDDL format until it reaches an end-of-text token or reaches maximum length.

The input and output sequence are described in the format of Planning Domain Definition Language (PDDL) [3]. PDDL is semantically similar to natural language, but has structural differences given its domain-specific grammar. See Figure 1 for an example of action plan in PDDL.

$$(a_t, s_t) = f((a_{1:t-1}, s_{1:t-1})|s_T, s_0) \tag{1}$$

In this problem, the model needs to generate a symbolic plan that respects the grammar of PDDL and learns the transition dynamics as described in Equation 1. We will train our model with dual objectives of language modeling in PDDL language, and generating correct sequence of $(action, state)$ pairs that leads to the goal state.

3.2 Input Embeddings

We used GPT-2's pre-trained Byte Pair Embedding[16] to perform subword tokenization and encode the input sequences.

Most word-based embedding methods are limited by the pre-defined vocabulary and do not generalize to rare and unknown words, and character-based embeddings are not well compressed and slower to train. Motivated by the intuition that uncommon and unknown words can often be decomposed into multiple subwords, BPE finds the best word segmentation by iteratively and greedily merging frequent pairs of characters. BPE initializes vocabulary with the character vocabulary from the training corpus, and then iteratively count all symbol pairs and replace each occurrence of the most frequent pair a new symbol until it reaches the desired vocab size.

To encourage the model to make use of the order of the sequence, we added position embeddings as proposed in original transformer paper [17]. We used a learned embedding layer based on position in the entire sequence rather than a sinusoid function to represent the position embedding of each token. We then sum the position embeddings and input embeddings to acquire the initial hidden states.

3.3 Model Architecture

The model architecture is GPT-2 (small)[18] with a Language Modeling head and a multi-class classification head, as in Figure 4. We chose the double-head model because we want the model to

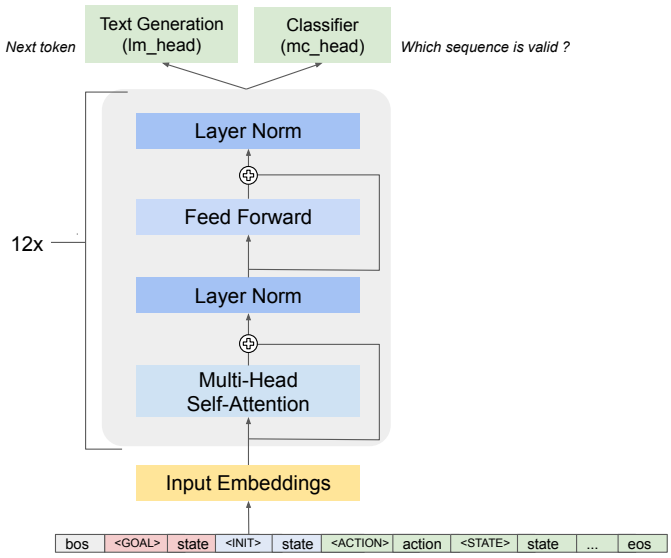

Figure 2: The model architecture is GPT-2[2] with a Language Modeling head and a Multi-class Classification head.

simultaneously learn how to model the PDDL language in robotic manipulation context, and how to understand the state transitions and produce correct symbolic plans.

GPT-2 uses multi-head masked self-attention to learn the different representation or context of the word. [17] Self attention mechanism allow the model to see the context of current token, by assigning scores to how relevant each previous tokens in the segment is, and adding up their vector representation. The concatenation of multi-head attention allows the model to jointly attend to information from different representation sub-spaces at different positions. [17] Applying masks is necessary to prevent the model from attending to future decoder inputs.

GPT-2 model applies layer normalization[19] in between each attention and feed forward layers in its transformer blocks. Layer normalization normalizes the input across each feature and are independent of other examples within the batch.

3.4 Finetuning on GPT-2

We finetuned on Hugging Face's pre-trained GPT-2 model weights[18]. It is the small version of the GPT-2 model, consisting of a stack of 12 transformer blocks, 768-hidden, 12-heads, and 117M parameters. The GPT-2 model is trained on a large corpus and is expected to have the transfer learning capability to capture learn the semantic meaning of the vocabulary in PDDL language and generate coherent symbolic plans after finetuning.

We implemented negative sampling during training to make the model classify a valid sequence of symbolic plans. Thus, the input to the model is a pair of a positive sample and a negative sample of PDDL plan. The negative inputs are corrupted by reshuffling the state and action sequences from positive samples.

We pre-process each input sample with special tokens indicating the conditional segments (goal and initial state), the action segments, and the state segments. The initial hidden states then go through a stack of 12 transformer blocks. Finally, the final hidden states are passed to the language modeling head and the classifier head of the model to generate two parts of the training loss.

The language modeling head generates the next token as output. It projects final hidden states onto the token's embedding matrix, generate logits for the probability for each possible token, and compute the cross-entropy loss based on the ground truth input sequence. During our training process, the model only learns from the positive samples' sequence of $(action, state)$ pairs. The conditional

segments (the goal and initial state) and the negative samples are masked out and not accounted for the language modeling loss.

Inspired by the Sentence-Order Prediction (SOP) task for unsupervised training in GPT-2 and Bert models, we trained the classifier head to determine which one of the input pair is the valid symbolic plan. It assigns logits to the positive and negative sequences to determine which one is the valid symbolic plan and generates a binary cross-entropy loss from the classification result.

$$L_{total} = c_{lm} \cdot L_{lm} + c_{mc} \cdot L_{classifer} \tag{2}$$

The training loss of the model will be a weighted sum of language modeling loss and classification error, as in Equation 2.

3.5 Oracle

In order to generate the training dataset and check for the correctness of our model's output, we implemented a Stanford Research Institute Problem Solver (STRIPS)[3] as the oracle.

The input to the STRIPS planner is a domain definition and a problem definition, both in PDDL format. The domain configures the set of the workspace environment and transition dynamics, which includes a set of predicates (e.g., inhand, inworkspace), actions (e.g., pick, place), the pre-conditions, and post-conditions for each action to be valid. A pre-condition defines the required symbolic states before an action can be performed, and a post-condition states the change of symbolic state after an action is performed. Each problem configures a set of objects (e.g. hook, table, shelf) that are present in the workspace, the initial state, and the goal state.

Our implementation of STRIPS planner uses Breadth First Search (BFS) to exhaustively iterates all possible paths, and checks whether they can reach the goal state from initial states. We use it to generate all valid action plans given a maximum action steps T_{max}. We will describe more details of dataset generation in section 4.1.

3.6 Baselines

We implement a LSTM-based model as a weak baseline. Its architecture consists of an Embedding layer, 2 stacks of LSTM layers, and a dense layers to produce the likelihood of next token based on its softmax value. It uses the same pre-trained tokenizer.

For a stronger baseline, we finetuned a GPT-2 GPT2LMHeadModel model from Hugging Face's repository with the same pre-trained weights [18]. We also use BPE embedding for this baseline model. The main different is that this model only has a language modeling head where the training loss is the simply language modeling cross-entropy loss. It is only trained on the positive samples. It does not have the ability to classify whether a sequence is valid or not, thus the sequence it generates are not guaranteed to be correct.

4 Experiments

4.1 Data

We used our STRIPS planner (oracle) to generate the training data. The input to the STRIPS planner is a domain and a problem; both defined in PDDL.

To generate our training dataset, we defined one domain that consists of five actions (pick, place, push, open, close), and five predicates that describe spatial relations between objects. We defined a total of 8 problem templates, such as place [sth] on the shelf from the table, open [sthB] and place [sthA] in [sthB]. We use [sth] as an object placeholder for various objects that fit well with the manipulation tasks. Using STRIPS planner, we generated action skeletons with BFS for each domain and problem pairs. We used STRIP planner (orable) to generate all feasible action plans that are within 7 action steps.

In order to train a model that can generalize to similar actions and more objects, we augmented our datasets by randomly replacing semantically synonymous action words, for example, replacing

"place" with "put" or "open" with "uncover." We replaced the object placeholders with a set of objects commonly seen in manipulation tasks to diversify our datasets.

To let the model classify valid and invalid action plans in a negative sampling process, we generated negative training examples with a ratio of 1:1. We randomly shuffled the order of actions and states of valid action plans we already generated. The resulting negative samples have similar grammar, but their $(action, state)$ pairs are not all valid, and overall plan are invalid as checked by our STRIPS planner.

We generated a total of 71128 pairs of symbolic plans (about 76MB in size). We split them in the ratio of 5:1 for training and evaluation datasets.

When constructing the input sequence, we added delimiters and segment indicators as special tokens in input sequence to inform the model about the structure of the PDDL language. We added <GOAL>, <INIT>, <ACTION>, <STATE>, as well as bos and eos tokens into the model's vocabulary.

4.2 Evaluation Method

We evaluate the model performance based on three metrics: perplexity, the accuracy of classifying the valid plans, and ability to generate valid plans given novel prompts.

$$perplexity = 2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

We used perplexity for evaluating language modeling capability. It measures how well a language probability model predicts a sample, where lower values imply more confidence in predicting the next token in the sequence. It is two to the power of the cross-entropy loss from the language modeling head.

We measure the classification accuracy as the second evaluation metric. It is the accuracy of the classifier head in classifying which of the input training sample is the positive one. It shows whether the model can tell which symbolic plan is valid from the last hidden states. The evaluation will be the percentage of correct classification in test datasets. This metric will only apply to our GPT-2 double head model since the single head model, and LSTM baseline will not have a classifier head nor negative training samples.

The third metric is the percentage of correct generation of symbolic plans. Correctness is measured by whether the new generated sequence of actions leads to the goal state, which we can check using our STRIPS symbolic planner. To test whether our model can generalize to unseen input prompts (new goal and initial conditions), we let the model generate action plans given a set of novel prompts unseen in training sets. We change the objects or actions vocabularies in the input sequence and check whether the model still generate correct action plans with no additional learning. We used a beam search of $top_k = 10, top_p = 0.9$, with maximum generation length of 500 tokens, and <|endoftext|> as the stop token.

4.3 Experimental details

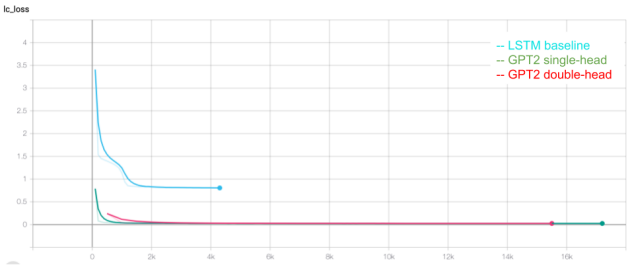

Figure 3: The training curve of language modeling loss for the baselines and GPT-2 Double-head model.

We trained on our PDDL datasets with pre-trained GPT-2 model (2 stacks of transformers, 768-hidden, 12-heads, 117M parameters) and pre-trained GPT2 tokenizer with a voacb size of 50264.

We trained on a single NVIDIA Quadro P5000 GPU for 15000 steps for 5 hours, each step taking about 1 second. For GPT-2 Double Head model, each batch consists of 4 pairs of positive and negative samples. We used the Adam optimizer [20] with $lr = e^{-5}, \beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 1e^{-8}$. For regularization, we set the attention layer dropout and residual layer dropout rate at 0.1. The gradient norm is clipped at 1.0. We set the $c_{lm} = 1.0, c_{mc} = 2.0$ when computing the training loss.

For both baselines, we trained on only positive samples. We use a batch size of 8 for GPT-2 Single Head mode, and a batch size of 128 for LSTM baseline model. We use the same Adam optimizer and dropout parameters. We only considered the language modeling loss.

4.4 Results

Model	perplexity	classification accuracy	% of correct generations
LSTM (weak baseline)	92527.0234	-	0%
GPT-2 w/ LM Head (strong baseline)	1.0228	-	0.6%
GPT-2 w/ LM + Classifier Heads	1.0228	100%	58%

Table 1: Evaluation result on LSTM baseline, GPT-2 single head baseline and GPT-2 double head model.

Table 4.4 shows the evaluation results. The LSTM baseline with a pre-trained tokenizer fails to learn the language model of PDDL. It has a very high perplexity, and its generation results are completely messy and not following the PDDL grammar.

The GPT-2 single head model and double head model both have low perplexity. They adapt to the PDDL grammar and generalize to some unseen vocabularies in test datasets. The GPT-2 Double head model's classifier head can classify positive samples in the test dataset 100% of the time.

Only GPT-2 double-head model can generate valid symbolic plans given unseen prompts for 58% of the time. LSTM model could not even generate sentences with correct PDDL grammar. GPT-2 single-head model produces paragraphs with proper PDDL grammar, but the generated action sequence does not lead to the goal state majority of the time.

5 Analysis

We analyzed the generated outputs from the two baselines and the finetuned GPT-2 double-head model. An example is shown in Figure 5. In this example, the manipulation tasks involves an underlying action sequence of moving cup from shelf to table, openning the cup, and putting a teabag from shelf onto the cup. This description is not shown to the model and we only gave the models the initial and goal states.

LSTM completely failed to generate meaningful sentences, as expected. The GPT-2 with single head is able to generate sentences that seems like a PDDL plan, however, when testing against the STRIPS planner's validator, the generated action sequence mostly fail to achieve the goal state. As seen in this example, the generated action sequences sometimes does not make sense (i.e. close(cup)), and does not lead to the goal state before finishing the sentence. This shows that despite learning the correct grammar and structure of PDDL, GPT-2 single-head by itself is not able to reason about the underlying dynamics of the environment and produces logical action sequences.

GPT-2 double-head can produce a valid action plan given unseen prompt for 58% of the time. We showed a correct symbolic plan in this example. It is able to produce a correct set of actions as well as predict the corresponding state changes, as highlighted in the figure. The model does not always generate correct answers. Still close to 40% of them are invalid action sequences checked by STRIPS validator. Among these, around 5% of them did not finish the sequence (did not end with the stop token). There are sometimes repetitive action sequences consecutively that are not necessary. This model generally captures the state changes given an action well, but it still sometimes fails to output a valid plan that matches the final goal state.

Figure 4: Example of generation outputs in a long-horizon task. The action and changes between states are highlighted. Here our model generates a valid symbolic plan and captures the state transitions correctly. The GPT-2 single head baseline generates a plan but fails to reach the goal. The LSTM baseline completely fails to model the PDDL language.

6 Conclusion

We explored the power of generalized language models for directly generating task-level robotic manipulation plans. We fine-tuned a GPT-2 double-head model on a custom-made PDDL dataset. We trained it to generate a sequence of actions and intermediate state given only the goal and initial state of the task. We demonstrated that our model can simultaneously adapt to the PDDL language in robotic manipulation context, and understand the underlying dynamics and state transitions to produce a valid symbolic plan. To our knowledge, this is the first approach that explores the language model's ability for robot planning without any additional structured processing or constraint optimization. We think there is potential in utilizing language models to solve the generalization challenges in long-horizon robotic planning.

In the future, we will continue exploring ways to add more structured domain knowledge from robot planning into language models, and make the accuracy of generated plans higher. We would like to try integrating the language models into the current learning-based robot planning network by using it as a generator for high-level instructions to guide and compose low-level motor skills in long-horizon manipulation tasks.

7 Acknowledgement

We would like to thank CS224N faculty Prof. Chris Manning, all the teaching staff, our TA who provides project feedbacks Arnaud Autefand, and our external mentor Prof. Jeanette Bohg. We appreciate your support and guidance throughout the course and the final project.

References

[1] Marc Toussaint, Kelsey Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners.

[3] Toki Migimatsu and Jeannette Bohg. Object-centric task and motion planning in dynamic environments. *IEEE Robotics and Automation Letters*, 5(2):844–851, 2020.

[4] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[5] Alexandre Attia and Sharone Dayan. Global overview of imitation learning. *arXiv preprint arXiv:1801.06503*, 2018.

[6] Mathew Botvinick, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement learning, fast and slow. *Trends in cognitive sciences*, 2019.

[7] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.

[8] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017.

[9] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[10] Yuuya Sugita and Jun Tani. Learning semantic combinatoriality from the interaction between linguistic and behavioral processes. *Adaptive behavior*, 13(1):33–52, 2005.

[11] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. 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, 2018.

[12] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. 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, 2019.

[13] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[14] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

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

[18] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[19] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

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