A Preliminary Study on BERT applied to Automated Planning

Lorenzo Serina[1], Mattia Chiari[1], Alfonso E. Gerevini[1], Luca Putelli[1] and Ivan Serina[1]

[1]*Università degli Studi di Brescia, Brescia, Italy*

Abstract

Despite some similarities that have been pointed out in the literature, the parallelism between automated planning and natural language processing has not been fully analysed yet. However, the success of Transformer-based models and, more generally, deep learning techniques for NLP, could open interesting research lines also for automated planning. Therefore, in this work, we investigate whether these impressive results could be transferred to planning. In particular, we study how a BERT model trained on plans computed for three well-known planning domains is able to understand how a domain works, its actions and how they are related to each other. In order to do that, we designed a variation of the typical masked language modeling task which is used for the training of BERT, and two additional experiments into which, given a sequence of consecutive actions, the model has to predict what the agent did previously (*Previous Action Prediction*) and what it is going to do next (*Next Action Prediction*).

Keywords

Automated Planning and Natural Language Processing, Deep Learning, BERT

1. Introduction

In 2007, the work in [1] drew an interesting parallel between automated planning and natural language processing (NLP), presenting a new formulation for plan recognition (i.e. the task of inferring an agent's plan observing some of its actions) based on grammars. The authors claimed that the two disciplines could share some of the research results, but they stated that much of the recent work in both fields had gone unnoticed by the researchers in the other field. In the following years, the separation between these two fields has not diminished. For NLP, deep learning techniques such as word embedding [2], recurrent neural networks, the attention mechanism [3] and pre-trained Transformer-based architectures [4, 5] have revolutionised the field, reaching a completely new state of the art in many different tasks [6, 7]. Although automated planning is a key and current research field in artificial intelligence [8, 9, 10], deep learning had a limited impact on it. In fact, it is mostly used for predicting heuristics [11], processing sensor data in order to create a symbolic representation of a planning problem [12] and for goal recognition tasks [13, 14] or in specific applications [15].

In this work, we focus on one of the most recent deep learning architectures for NLP: BERT [5]. Processing a huge quantity of sentences and documents, BERT is able to learn how the language works and its capabilities can be exploited for text classification [16], sentiment

10th Italian Workshop on Planning and Scheduling (IPS-2022)

lorenzo.serina@unibs.it (L. Serina); m.chiari017@unibs.it (M. Chiari); alfonso.gerevini@unibs.it (A. E. Gerevini); luca.putelli1@unibs.it (L. Putelli); ivan.serina@unibs.it (I. Serina)

analysis [17], question answering [18] and other NLP tasks. This is done by training the model to perform the so-called *masked language modeling* task. Basically, the input of the BERT model is an incomplete sentence, into which some words are replaced by a special marker, and the model has to reconstruct the complete sentence, predicting the missing words from context.

Inspired by the parallelism pointed out in [1], in this work we aim to transfer these techniques in the automated planning field. We claim that plans and actions can be seen similarly to sentences and words, and we have designed a *planning language modeling* task into which the model has to reconstruct an incomplete sequence of actions to train the model. With this technique, we train a single BERT model using actions from three well-known planning domains: Logistics, Satellite and Blocksworld. Next, we evaluate the performance of the model in this task and in two additional experiments: *Next Action Prediction*, into which the model has to predict the action that follows a sequence of actions (which are the input of the model) extracted from a complete plan, and *Previous Action Prediction*, into which the predicted action is the one that precedes the input sequence. These tasks were made to demonstrate how well the model understands about how a planning domain works, its rules, its actions and their effects. Moreover, in this work, we present a detailed report of the main difficulties in the training process, the necessary choices the user has to make in order to train a similar model, the dimension of the datasets required to obtain good results, etc. Finally, we discuss the positive and negative aspects of applying these techniques to the planning domain.

2. Background on BERT

BERT (Bidirectional Encoder Representations from Transformer) [5] is a deep learning architecture based on Transformer [4] composed by several layers which progressively analyse a sequence of elements, named tokens, in order to understand how they relate to each other, their meaning and the overall properties of the entire sequence. Although BERT is a model typically used for Natural Language Processing, recently the same techniques have been applied in other contexts such as the analysis of programming languages [19], graphs [20] and images [21].

At first, each token is associated to an index. Next, an embedding layer calculates a vector of real numbers for each token, as in initial representation of its meaning. Considering a sequence of tokens S of length N, each token $t \in S$ is represented by a vector $x_t \in \mathbb{R}^d$. We denote as $X \in \mathbb{R}^{N \times d}$ the overall matrix of token embeddings.

After calculating this representation, these vectors are processed by several encoding layers. Each layer applies in parallel multiple self-attention mechanisms, called *heads*, into which the relation between each possible pair of tokens is computed. This mechanism produces a matrix $A_{i,j} \in \mathbb{R}^{N \times N}$, into which i is the number of the encoding layer and j is the head number. For each token $w \in S$, the vector $a_w \in A_{i,j}$ contains the attention weights that represent how much w is related to the other tokens in S. In order to calculate these weights, in each head the input representation of the token sequence $X \in \mathbb{R}^{N \times d}$ is projected into three new representations called key (K), query (Q) and value (V) with three weight matrices W_k, W_q and W_v:

$$K = X \times W_k,\ Q = X \times W_q,\ V = X \times W_v$$

Then, the self-attention weights are calculated using the softmax function on the scaled dot-product between Q and K. Finally, this mechanism computes a new vector representation of

the input tokens (Z) by multiplying the attention weights for V.

$$A = softmax\left(\frac{Q \times K^{\intercal}}{\sqrt{d}}\right), \quad Z = A \times V$$

Given that in each encoding layer there are several heads (typically 8, 12, 16 or 24), in order to create a single representation provided by the *multi-head attention mechanism* the result of each head is concatenated and then passed to a feed-forward layer.

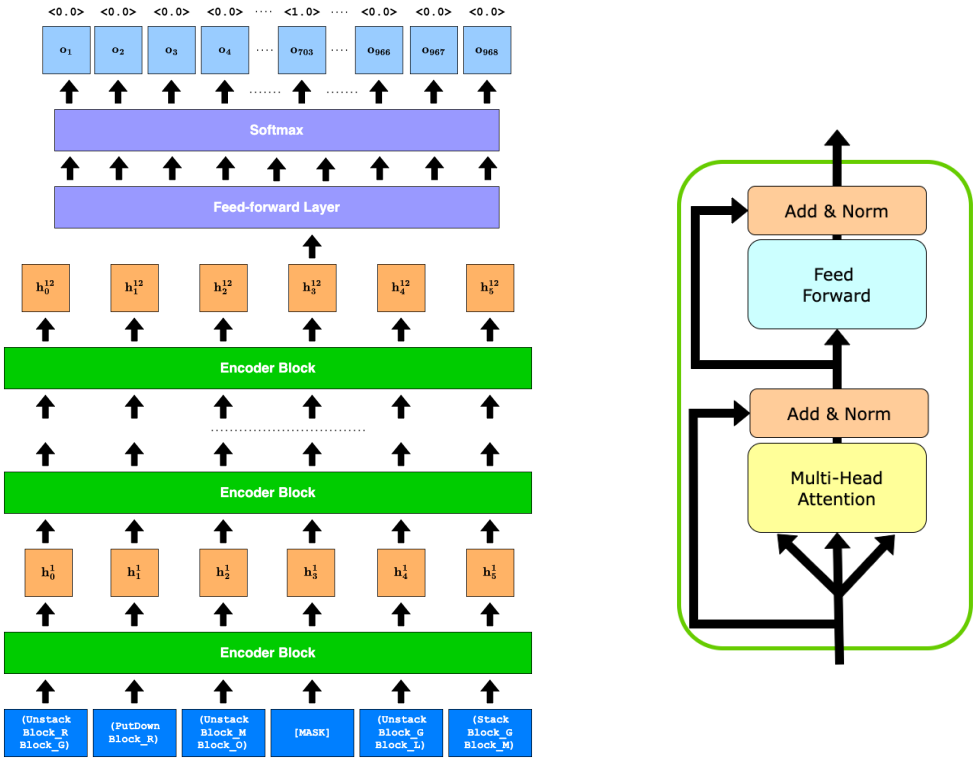
As described in [4], the multi-head attention mechanism is followed by a residual connection which adds the input representation with the one calculated by the multi-head attention. Next, the results is fed to a feed-forward layer and another residual connection. The output of the encoding layer is also the input of the next encoder. Typically, a BERT model is composed by 8, 12 or 16 encoding layers. A schematic representation of an encoding layer is shown in Figure 1b.

In the original model, designed for NLP tasks, the training was performed using 16GB of documents for a total of more than 3 billions of words [5]. However, more recent implementations exploit even larger datasets (for instance, RoBERTa [22] uses about 165GB of data). The training task is called *masked language modeling* and works as follows. Given a sequence of words (typically the maximum length is 512 tokens), a certain percentage of them is replaced with a special token called [MASK]. The encoding layers process this sequence and learns how to predict the masked words from the context formed by the remaining ones. In order to perform this operation, the output of the last encoding (i.e. the final representation of each token produced by the architecture) is connected to a fully-connected layer with softmax activation, into which each neuron represents a word from the vocabulary. Supposing that the overall corpus of documents contains $100K$ different words, this layer will be formed by $100K$ neurons.

3. Planning Language Modeling with BERT

In order to perform PLM, we approach the planning domains as a NLP task. In fact, our BERT model is trained using a slight variation of the typical language modeling task that we call *Planning Language Modeling*. In this technique, given a sequence of contiguous actions $\bar{p}$ composed of n actions a_i, with $i \in [1, n]$, we consider each action as a separated token, like words in a sentence. First, we divide the action sequence into separated tokens, using the WordPiece tokenizer [23]. Next, we substitute a certain percentage of actions with the special token [MASK] and give this incomplete sequence as input. The BERT model has the task of predicting the missing actions from the overall context of the action sequence. If the model is capable of performing this operation, we claim that it has the ability to understand how the planning domain works, the impact of the actions, when they are performed by an agent, and which effects they have. In order to perform the planning language modeling, the BERT model needs processing a large quantity of these training instances, progressively adjusting the training weights of the model with the backpropagation algorithm until it reaches a satisfying level of accuracy.

Our model architecture is very similar to the one proposed in [5]. We use a total of 12 encoding layers, each one of them with 12 heads, and we use 768 as the embedding size, i.e. actions are represented as vectors of 768 real numbers across the model.



(a) (b)

Figure 1: On the left (a) , schematic representation of a BERT model performing the PLM task. On the right (b), a simplified view of the structure of an encoder block.

Our choice is to develop a multi-domain model, considering Logistics, Blocksworld and Satellite. With a single model, following the current trend of multi-language models, we want to verify whether BERT could understand different domains at the same time. Next, we aim to study the training process for a model which was already trained on a different domain. The model was first trained on Logistics. It took 40 epochs to understand the domain and reach a good accuracy. Then we trained the model on Blocksworld and Satellite and it took less than 10 epochs to achieve good performance. Therefore, we verified that adding another domain to a pre-trained model does not require another, full training procedure. Instead, the model can easily adapt to the new domain as in fine-tuning operations, requiring only a few epochs to achieve a good accuracy on the new domain.

3.1. A Blocksworld Example

In this section, we present a simple example of how we structure our planning language modeling task. For simplicity's sake, let us assume that our BERT works only for this domain. We use a very simple plan instance in the well-known Blocksworld domain. In this domain, the agent has the goal of building one or more stacks of blocks, and only one block may be moved at a time. The agent can perform four types of actions: `Pick-Up` a block from the table,

`Put-Down` a block on the table, both these operations take only one argument, `Stack` which puts a block on top of another one and `Unstack` which removes a block that is on another one, both have two arguments. We assume that our plans involve at most 22 blocks, therefore we have 22 possible Pick-Up action, 22 Put-Down actions, $22 * 21$ Stack actions and $22 * 21$ Unstack actions, for a total number of 968 actions.

Let's consider the plan formed by the following 6 actions: `(Unstack Block_R Block_G)`, `(PutDown Block_R)`, `(Unstack Block_M Block_O)`, `(Stack Block_M Block_R)`, `(Unstack Block_G Block_L)`, `(Stack Block_G Block_M)` with ids corresponding to indices 59, 462, 121, 703, 309 and 240. Next, we mask 1 action (17%) chosen randomly. Assuming that the special token [MASK] has index 0, the input of our BERT model will be the following sequence: [59, 462, 121, 0, 309, 240].

This sequence is then processed by the embedding layer and by all the encoding layers, producing a final representation of each element as a vector of 768 real numbers. Given that the fourth element of the sequence is masked, its embedding vector is passed to a feed-forward $f$layer composed of 968 neurons (one for each possible action of the domain) with the softmax activation function. Given that the original action had index 703, we want in output a vector formed by all zeroes except for a 1 at index 703. Therefore, the result produced by the feed-forward layer is compared with the desired output. This procedure is repeated for every training instance, evaluating the overall error made by the architecture and calculating a loss function, which will be used by the backpropagation algorithm to adjust the model weights.

In Figure 1a we show a simplified representation of the model architecture performing the planning language modeling task. The actions are encoded into vectors by several encoding blocks and, in the last step, the vectorial representation of [MASK] is passed to a feed-forward layer with softmax activation function that predicts a 1 at the index 703 and 0 for all the other indexes.

3.2. Training Technique

In order to properly train the model, there are some necessary choices that the user must make. Although typically BERT models treat a specific language, there are several multi-language models. Given that a planning domain has its own set of actions, predicates, and rules, we can consider it as a sort of an independent language. Therefore, the user should evaluate whether training a BERT model for each domain, perhaps specialising and improving its predictive capabilities, or building a multi-domain model, selecting which domains to consider and their quantity.

Next, an important constraint of the BERT model is that it must have a predefined vocabulary of actions. Therefore, during the training process the user has to define a predefined set of objects for each domain and generate all the possible actions that an agent could perform with them. Although the predefined vocabulary is sensitive to the object names, this operation only requires to set a maximum number of objects. In fact, a mapping algorithm can be easily implemented for translating new names into predefined ones.

As with all deep learning techniques, a crucial aspect for obtaining good performance is the number of training instances, i.e. the number of plans necessary to train the BERT model. The choices of how many domains to consider for training and how many objects to include have a

Domains	Objects	Vocab. size	Plans	Mean length	Max length	Augmented instances
Logistics	52	15154	200k	29	382	360k
Blocksworld	22	968	180k	39	338	430k
Satellite	65	33225	140k	17	33	140k

Table 1
Characteristics of the dataset we used to train the BERT model in terms of overall number of objects considered, number of actions in the vocabulary, number of generated plans and their mean and maximum length. In the column Augmented instances we report the dimension of the dataset after the data augmentation process.

serious impact on the number of plans which have to be collected or generated for the training of the model, the amount of time required to this process, and its difficulty.

In our model, we have selected three well-known planning domains: Logistics, Blocksworld and Satellite. In Table 1 we report the characteristics of our dataset. Please note that while in Blocksworld the only objects present in the domain are the blocks, in the other two domains there are several types of objects (for instance, Logistics includes airplanes, airports, locations, cities, trucks and packages) and we report the overall sum of them for simplicity's sake.

In order to train the multi-domain model, the typical approach is simply considering a single training set composed by the plans for all the three domains. However, we have also experimentally verified that it is possible to add a domain to a pre-trained model extending its vocabulary and continuing its training with additional epochs using the plans generated from the new domain. In this phase, however, we have noticed that it is useful to include also a percentage (such as 25%) of plans belonging to the already processed domains.

The overall dimension of our data set is about 350 MB of data; therefore, it is definitively smaller than standard BERT architectures [5, 22].

3.3. Data Augmentation

Another technique that can be useful in training a BERT model is data augmentation. In this procedure, we divide the longest plans in several subsequences with random dimension and add them to the dataset. With this technique, the training set can contain not only entire plans but also smaller sequences of actions. In order to avoid including a large number of very similar instances, which could create overfitting issues to the model, we assure that plans which are too similar to the ones already present in the dataset are not added. In our context, for Logistics and Blocksworld we divided the plans with more than 20 actions, randomly selecting a subsequence of the plan. The subsequence is added to the training set if the similarity calculated by the Ratcliff-Obershelp algorithm [24] with the other plans is lower than 0.4, otherwise it isn't. While this operation increases the predictive performance of the model for these two domains, in Satellite data augmentation did not improve the results. We claim that this is due to the smaller size of the Satellite plans, with respect to the other two domains. In Table 1 we report the dimension of the original dataset (with complete plans) and the augmented dataset, with the smaller subsequences generated by the data augmentation process.

4. Experiments and Results

In this section, we present the set of learning tasks we designed for our model. The model was trained on 2 NVIDIA V100, for 120 epochs, with a training time of almost 130 hours.

4.1. Designed Tasks and Evaluation Metrics

First of all, we test the capabilities of our model to perform the planning language modeling using a test set composed by plans which were not used during the training of the algorithm. In this experiment, we check if the predictions of the masked input actions are correct. Given that the softmax activation function of the output layers actually predicts a score between 0 and 1 for all the possible actions of our three domains, we evaluate the planning language masked in two configurations. In the first one (*Top-1* accuracy), we consider only the action predicted with the highest score. Given the very high number of possible classes in output (one for each action, i.e. more than 40 thousands in our multidomain model), we also use as an evaluation metric the *Top-5* accuracy [25] into which we consider the set of the 5 actions with the highest score, and, if the original action belongs to the set, we consider that the model correctly predicts the masked action. The planning language modeling is evaluated using the following percentages of masked actions: 10%, 15%, 25% and 50%.

We also design two additional tasks to test the capabilities of the model to handle sequences of actions and to infer some of the domain knowledge such as actions' preconditions and effects. For both of these tasks, we use the same plans used for the planning language modeling. In the first task, called *Next Action Prediction*, we ask our model to predict the action that follows an input action sequence. In the second, called *Previous Action Prediction*, we ask our model to predict the action that precedes an input action sequence. In detail, given an action sequence of length l, for *Next Action Prediction*, we pass the first $l-1$ actions to the model and ask the model to predict the l-th action. Similarly, for *Previous Action Prediction* the model should predict the first action of the sequence while the remaining actions are passed in input. We evaluate *Next Action Prediction* and *Previous Action Prediction* using different input sequence lengths (i.e. 3, 5, 10, 20) and with the entire plan without the last and first action, respectively (Tot).

For the evaluation of these tasks, in our opinion, verifying whether the predicted action matches the one in the input plan is not enough. In fact, the input action sequence filled with the predicted action might be valid even though the predicted action does not match the one in the label. Moreover, we are not providing the model the initial state or the goal (see Section 5), therefore the model cannot always predict the best action based only on an incomplete set of actions. For this reason we use VAL [26] to check if, starting from the initial state of the problem, the action sequence, which also comprehends the action predicted by the model, can be executed or not, providing insight into how the model is able to learn the inner or working of a planning domain[1].

[1]Please note that VAL does not always validate our plan as the goal of the problem is often not reached; nonetheless, we use VAL errors to determine whether the sequence is valid or not

Task	10 %	15%	25%	50%
Top 1	0.710	0.671	0.574	0.333
Top 5	0.821	0.802	0.725	0.486

Table 2
Results for the Planning Language Modeling task in terms of *Top-1* and *Top-5* accuracy. On the columns, we report the percentage of masked actions in the test sequences.

Length	3	5	10	20	Tot
Top 1	0.234	0.291	0.200	0.333	0.596
Top 1 (V)	0.650	0.675	0.454	0.664	0.759
Top 5 (V)	0.885	0.906	0.755	0.928	0.929

Table 3
Results for the *Next Action Prediction* task. Length stands for the number of input actions (column tot means that we pass an entire plan to the model except for the last action). On the rows, we report the *Top-1* accuracy considering only the correct actions and *Top-1* and *Top-5* accuracy considering also the valid actions (V).

4.2. Experimental Results

Considering all three domains, in Table 2 we show the results, in terms of *Top-1* and *Top-5*, of the planning language modeling task, using 3000 plans, 1000 for each domain, as a test set. In this experiment, we masked different percentages of actions: 10%, 15%, 25% and 50%. As it should be expected, the accuracy decreases proportionally to the increasing of the percentage of masked actions (i.e. less input data). The overall performances are more than acceptable, in particular for the 50% in *Top-5*, where the model, analyzing just half of the plan, is able to correctly predict almost half of the actions, highlighting a deep comprehension of the context. With less masked actions, the accuracy is very high, with more than 80% in *Top-5* for 10% and 15% and good performance also in terms of *Top-1*.

The results for the *Next Action Prediction* task in terms of *Top-1*, *Top-5* are shown in Table 3. In each column, we consider a different length for the input sequence. For instance, if the length is 5 we have 5 consecutive actions in input and we want to predict the 6-th.

In each row we can notice the same trend: the accuracy increases with the length of the plan, except for the plans with length 10, where there is a drop of the performances. However, in terms of *Top-5* we have very good results even with small action sequences, underlining a deep understanding of the inner workings of the planning domain. Focusing only on the *Top-1* correct, the results are much lower. That could be due to the fact that our model does not receive any information about the agent's goal and therefore it can only predict a set of executable actions, into which it is possible to find the right one.

To better understand these results, we study the performance for each considered domain for the prediction of valid actions. In Figure 2, we analyze the accuracy in terms of *Top-1* and *Top-5* accuracy. On the left, we can see how *Top-1* accuracy increases proportionally to the length of the plans in the Logistics domain, from 51% to 92%, while it's almost constant in Satellite

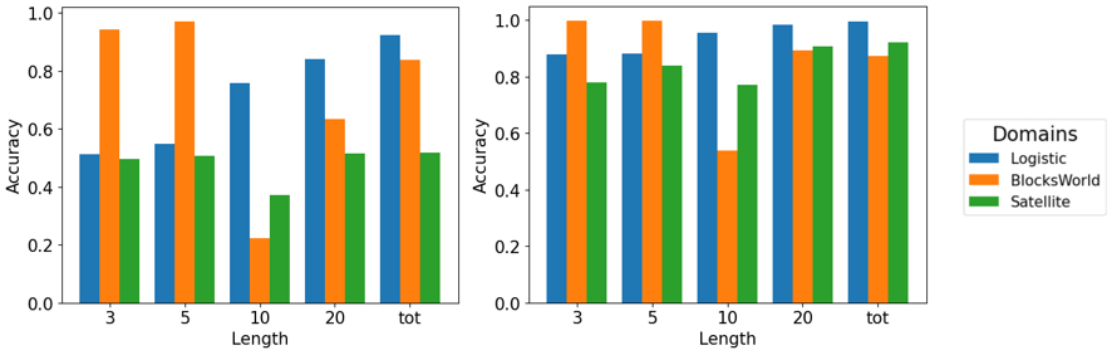


Figure 2: Results for the *Next Action Prediction* task in terms of *Top-1* (left) and *Top-5* (right) for each domain. In this experiment we considered the 5 most probable predictions of the model. On the x-axis we found the length of the action sequence and on the y-axis we found the accuracy of the predictions, computed as validated answers on total instances in the specific domain.

Length	3	5	10	20	Tot
Top 1	0.287	0.385	0.511	0.545	0.566
Top 1 (V)	0.423	0.510	0.594	0.600	0.602
Top 5 (V)	0.637	0.740	0.834	0.843	0.854

Table 4
Results for the *Previous Action Prediction* task. Length stands for the number of input actions (column tot means that we pass an entire plan to the model except for the first action). On the rows, we report the *Top-1* accuracy considering only the correct actions and *Top-1* and *Top-5* accuracy (V) considering also the valid actions.

domain. In the BLOCKSWORLD domain the results are excellent with shorter plans, with 95%, while they drop with length equals to 10, to 22%. Then they recover with longer plans, arriving to 84%. As it can be seen in Figure 2, the performance drop for length 10 regards almost only the BLOCKSWORLD domain. We will address this issue in Section 5. In terms of *Top-5* accuracy, on the right of Figure 2 we can see the same trend as the previous task on the performances, with good results in predictions, on LOGISTICS up to to 99% and on SATELLITE up to 92% .

In Table 4 we report the results for the *Previous Action Prediction* task using the same metrics and format used in the evaluation of the *Next Action Prediction* task. Similarly to the *Next Action Prediction* results, we can notice that, for each row, the accuracy increases as the input length increases. These results are very promising, even if they are slightly lower than the ones obtained in *Next Action Prediction*. The performances increase with the length of the plan and the collapse on length 10, seen in the previous task, is not present.

As we did previously, in Figure 3 we show the results for each domain in terms of *Top-1* and *Top-5* accuracy (considering also the valid actions) on the *Previous Action Prediction* task. In terms of *Top-1*, we can notice that the performances on the LOGISTICS and the BLOCKSWORLD domains remains quite constant, (with a maximum of 47% and 57% respectively), while on the SATELLITE domain they increase proportionally with the length of the plans, from 33% to 84%. In

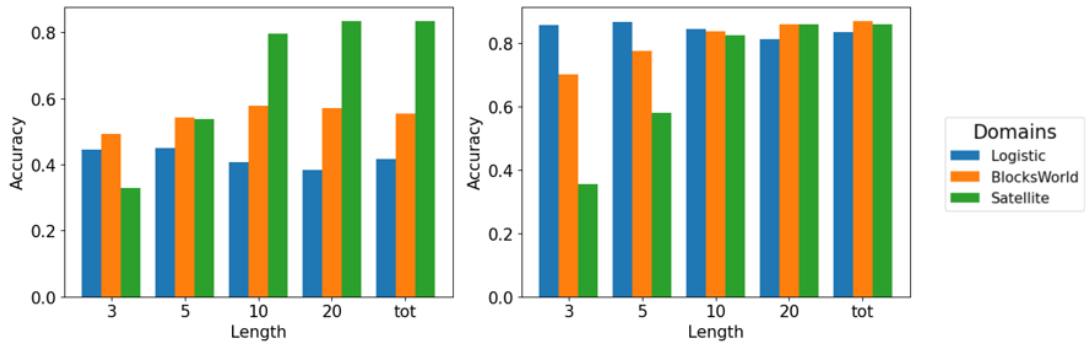


Figure 3: Results for the *Previous Action Prediction* task in *Top-1* (left) and *Top-5* (right) for each domain. In this experiment we considered the 5 most probable predictions of the model. On the x-axis we found the length of the action sequence and on the y-axis we found the accuracy of the predictions, computed as validated answers on total instances in the specific domain.

terms of *Top-5*, we have higher results for all three domains. Howevever, while the performance in the Logistics domain remains almost the same for all lengths, we can notice a small increase in Blocksworld and a strong increase in Satellite. This domain in particular has some issues with actions sequences of length 3 (36%) but reaches 86% at length 10, showing that for this domain we need an adequate number of actions in order to have significant predictive results.

5. Discussion

The results presented in Section 4.2 show that a deep learning model such as BERT, trained on a set of action sequences from well-known planning domains, can effectively accomplish three predictive tasks: the planning language modeling, i.e. predicting actions from context, the prediction of the next action given sequences of different length and, in the same conditions, the prediction of the previous action.

In our opinion, this demonstrates that BERT is able to understand how a planning domain works and which rules are followed by the action sequences. Despite a great number of possible actions to predict (especially for the Satellite domain, which has more than 33k actions in our configuration), our results in terms of *Top-1* and *Top-5* accuracy show that the model is able to identify which are the most probable ones. Although we have relatively bad results in terms of *Top-1* for the *Next Action Prediction* and *Previous Action Prediction* tasks considering only if the action belonged to the original plan, the performance are much higher considering the valid actions. This can be also an indicator that our model does not simply memorize a lot of action sequences, but it is really capable to adapt its knowledge with new plans, avoiding overfitting issues.

A problem of this first implementation is that we do not provide any data on the initial state and the goal to our model. This could be the reason for the performance drop in Blocksworld with sequences of length 10 in the *Next Action Prediction* task. In fact, while the model seems capable of predicting the next action with smaller sequences, when the complexity increases

it could be necessary to provide more data in order to direct its reasoning. However, it is interesting to note that for Logistics this does not happen. This may be due to the fact that information about initial state could be indirectly contained in the action sequences. While we will conduct a further analysis of these results, considering also other domains, we think that including the initial state and the goal in the model could improve the results and avoid this performance drop.

While we have verified that a BERT model has this learning capability, we are also conscious of several limits. For instance, as we show in Table 1, the number of plans necessary to train such a model is very high. In a real-world application, this can be a huge problem because it would require a long time to collect the necessary data to exploit this kind of model. While our data augmentation technique shows interesting results, perhaps similar results could be obtained with simpler models, which could require less training data. Another related problem with this kind of architectures is the training time and the computational resources required. An interesting line of research could be apply knowledge distillation techniques and build smaller versions of our model [27].

Moreover, the original BERT for natural language processing [5] is a pre-trained model and adapted for other tasks such as document classification. To do that, the authors introduced a special token called [CLS] for representing the entire document. In our context, we could have the same token for obtaining a vector representation of the entire plan or action sequence and use it for several tasks such as heuristic prediction or plan and goal recognition. However, training [CLS] in BERT required another task, called Next Sentence Prediction, into which the model learns the ability to predict whether two sentences are consecutive in a document. Although this is a very intuitive operation in NLP, the same concept is not present in automated planning. Thus, there is the need to design a completely new task which could lead the model to learn an informative representation of a sequence of actions.

6. Related work

In the last few years, deep learning techniques started to surface across several contexts and tasks related with automated planning.

For instance, the work in [12] exploits variational autoencoders based on neural networks to create latent and symbolic representations of actions starting from pairs of images which describe a transition between two states. This representation can be further exploited for other purposes, such as Goal Recognition, which is defined as the task of recognising the goal that an agent is trying to achieve from observations about the agent's behaviour in the environment [28]. Using only image-based domains such as MNIST or 8-Puzzle, the work [13] first computes a latent representation using the tool proposed by [12] and then analyses the sequence of actions using an LSTM Neural Network [29] for predicting the goal.

Goal Recognition has seen several other applications of deep learning techniques. For instance, the work [15] presents an LSTM network to infer player goals from the observation of their low-level actions in the context of digital games. However, more general works such as [30] and [31] used the classical domains of automated planning and, starting from partial observations of states or actions represented symbolically, trained neural networks or other machine learning

algorithms in order to predict the agent's goal.

Deep learning techniques are also used in many path planning applications with robots [32]. For example, in [33] a neural network is used to process data from robot's sensor in order to get an approximate direction. This approximation is then used to build a navigation system. Furthermore, the controller for an autonomous mobile robot presented in [34] implements a feed forward neural network which predicts the steering angle given the obstacles distances from the left, right and front directions.

Another problem which has been addressed by deep learning techniques is the learning of heuristic functions for classical planning. For instance, the authors of [11] trained a neural network to compute the distance between two states and used the output measure as a heuristic. However, one of the drawbacks of their approach is the need to train a neural network for each planning task. The work in [35] solves this issue for a subset of planning domains by learning a strong heuristic function for PDDL domains with an underlying grid structure. The authors implement a convolutional neural network structure that takes the current state and the goal state as input and predicts both the next action and the heuristic value.

To the best of our knowledge, ours is the first implementation of a Transformer-based architecture such as BERT in the context of automated planning. While most of the works show slight variations or improvements [22, 36], versions for some specific contexts [37] or jargons [38], the work in [19] shows how these techniques can be useful even for artificial languages such as programming languages, proposing a BERT model for evaluating the semantic similarity between two portions of code. Their model is trained using publicly available open source code repositories from GitHub. In [39], a similar model for code summarization and generation is proposed. An interesting study about the capabilities of such models is shown in [40].

7. Conclusions and Future work

In this work, we trained a BERT model in the context of automated planning. While this type of deep learning architecture has completely changed the state-of-the art for several NLP tasks [5, 22], at the best of our knowledge this is the first implementation of such technique in the planning context. We designed an adaption of the masked language modeling task, called *planning language modeling* into which, using plans generated from three well-known planning domains (Logistics, Satellite and Blocksworld), this multi-domain model receives in input an action sequence with a percentage of missing actions and learns how to identify them from context. We have also presented a detailed account of the operations necessary to train such a model, in terms of pre-processing choices, vocabulary definition, data augmentation, etc.

In our experimental evaluation, we designed two tasks in order to verify if the model has the capability of understanding the inner working a planning domain: *Next Action Prediction* and *Previous Action Prediction*, into which given a sequence of actions the model has to predict the action that immediately follows or precedes the sequence. Our models obtains very promising results and we show that it has a high accuracy in predicting action that can be actually executed by an agent, before or after an action sequence.

However, our research is still preliminary. As future work, we will further investigate the capabilities of our model, including new domains and more complex configurations, with more

objects and longer plans. Moreover, we will study how this pre-trained model can be exploited and adapted for other tasks such as goal recognition or heuristic prediction.

References

[1] C. W. Geib, M. Steedman, On natural language processing and plan recognition, in: M. M. Veloso (Ed.), IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007, 2007, pp. 1612–1617.

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: C. J. C. Burges, L. Bottou, Z. Ghahramani, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, 2013, pp. 3111–3119.

[3] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, 2015.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 5998–6008.

[5] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), Association for Computational Linguistics, 2019, pp. 4171–4186.

[6] L. Putelli, A. E. Gerevini, A. Lavelli, R. Maroldi, I. Serina, Attention-based explanation in a deep learning model for classifying radiology reports, in: A. Tucker, P. H. Abreu, J. S. Cardoso, P. P. Rodrigues, D. Riaño (Eds.), Artificial Intelligence in Medicine - 19th International Conference on Artificial Intelligence in Medicine, AIME 2021, Virtual Event, June 15-18, 2021, Proceedings, volume 12721 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 367–372. URL: https://doi.org/10.1007/978-3-030-77211-6_42. doi:10.1007/978-3-030-77211-6_42.

[7] L. Putelli, A. Gerevini, A. Lavelli, I. Serina, Applying self-interaction attention for extracting drug-drug interactions, in: M. Alviano, G. Greco, F. Scarcello (Eds.), AI*IA 2019 - Advances in Artificial Intelligence - XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19-22, 2019, Proceedings, volume 11946 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 445–460. URL: https://doi.org/10.1007/978-3-030-35166-3_32. doi:10.1007/978-3-030-35166-3_32.

[8] A. Gerevini, A. Saetti, I. Serina, P. Toninelli, Fast planning in domains with derived predicates: An approach based on rule-action graphs and local search, in: M. M. Veloso, S. Kambhampati (Eds.), Proceedings, The Twentieth National Conference on Artificial

Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA, AAAI Press / The MIT Press, 2005, pp. 1157–1162. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-183.php.

[9] A. Gerevini, I. Serina, Planning as propositional CSP: from walksat to local search techniques for action graphs, Constraints An Int. J. 8 (2003) 389–413. URL: https://doi.org/10.1023/A:1025846120461. doi:10.1023/A:1025846120461.

[10] L. Bonassi, A. E. Gerevini, E. Scala, Planning with qualitative action-trajectory constraints in PDDL, in: L. D. Raedt (Ed.), Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022, ijcai.org, 2022, pp. 4606–4613. URL: https://doi.org/10.24963/ijcai.2022/639. doi:10.24963/ijcai.2022/639.

[11] P. Ferber, M. Helmert, J. Hoffmann, Neural network heuristics for classical planning: A study of hyperparameter space, in: ECAI 2020, IOS Press, 2020, pp. 2346–2353.

[12] M. Asai, A. Fukunaga, Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI Press, 2018, pp. 6094–6101.

[13] L. Amado, J. P. Aires, R. F. Pereira, M. C. Magnaguagno, R. Granada, F. Meneguzzi, Lstm-based goal recognition in latent space, CoRR abs/1808.05249 (2018).

[14] M. Chiari, A. E. Gerevini, L. Putelli, F. Percassi, I. Serina, Goal recognition as a deep learning task: the grnet approach, 2022. URL: https://arxiv.org/abs/2210.02377. doi:10.48550/ARXIV.2210.02377.

[15] W. Min, B. W. Mott, J. P. Rowe, B. Liu, J. C. Lester, Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks, in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, IJCAI/AAAI Press, 2016, pp. 2590–2596.

[16] L. Putelli, A. E. Gerevini, A. Lavelli, T. Mehmood, I. Serina, On the behaviour of bert's attention for the classification of medical reports, in: C. Musto, R. Guidotti, A. Monreale, G. Semeraro (Eds.), Proceedings of the 3rd Italian Workshop on Explainable Artificial Intelligence co-located with 21th International Conference of the Italian Association for Artificial Intelligence(AIxIA 2022), Udine, Italy, November 28 - December 3, 2022, volume 3277 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 16–30. URL: http://ceur-ws.org/Vol-3277/paper2.pdf.

[17] M. Hoang, O. A. Bihorac, J. Rouces, Aspect-based sentiment analysis using bert, in: Proceedings of the 22nd nordic conference on computational linguistics, 2019, pp. 187–196.

[18] C. Qu, L. Yang, M. Qiu, W. B. Croft, Y. Zhang, M. Iyyer, Bert with history answer embedding for conversational question answering, in: Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval, 2019, pp. 1133–1136.

[19] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, Codebert: A pre-trained model for programming and natural languages, in: T. Cohn, Y. He, Y. Liu (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020, volume EMNLP 2020 of *Findings of ACL*, Association for Computational Linguistics, 2020, pp. 1536–1547.

[20] J. Zhang, H. Zhang, C. Xia, L. Sun, Graph-bert: Only attention is needed for learning graph

representations, CoRR abs/2001.05140 (2020).
[21] H. Bao, L. Dong, S. Piao, F. Wei, Beit: BERT pre-training of image transformers, in: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022, OpenReview.net, 2022.
[22] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized BERT pretraining approach, CoRR abs/1907.11692 (2019).
[23] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144 (2016).
[24] J. W. Ratcliff, D. E. Metzener, Pattern-matching-the gestalt approach, Dr Dobbs Journal 13 (1988) 46.
[25] A. Thilagar, R. M. Frongillo, J. Finocchiaro, E. Goodwill, Consistent polyhedral surrogates for top-k classification and variants, in: K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, S. Sabato (Eds.), International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA, volume 162 of *Proceedings of Machine Learning Research*, PMLR, 2022, pp. 21329–21359.
[26] R. Howey, D. Long, M. Fox, Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl, in: 16th IEEE International Conference on Tools with Artificial Intelligence, IEEE, 2004, pp. 294–301.
[27] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, Q. Liu, Tinybert: Distilling BERT for natural language understanding, in: T. Cohn, Y. He, Y. Liu (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020, volume EMNLP 2020 of *Findings of ACL*, Association for Computational Linguistics, 2020, pp. 4163–4174.
[28] F. A. Van-Horenbeke, A. Peer, Activity, Plan, and Goal Recognition: A Review, Frontiers Robotics AI 8 (2021) 643010.
[29] S. Hochreiter, J. Schmidhuber, Long Short-term Memory, Neural computation 9 (1997) 1735–80.
[30] M. Maynard, T. Duhamel, F. Kabanza, Cost-Based Goal Recognition Meets Deep Learning, Proceedings of the AAAI 2019 Workshop on Plan, Activity, and Intent Recognition, PAIR 2019 (2019).
[31] D. Borrajo, S. Gopalakrishnan, V. K. Potluru, Goal recognition via model-based and model-free techniques, Proceedings of the 1st Workshop on Planning for Financial Services at the Thirtieth International Conference on Automated Planning and Scheduling, FinPlan 2020 (2020).
[32] A.-M. Zou, Z.-G. Hou, S.-Y. Fu, M. Tan, Neural networks for mobile robot navigation: a survey, in: International Symposium on Neural Networks, Springer, 2006, pp. 1218–1226.
[33] S. H. Dezfoulian, D. Wu, I. S. Ahmad, A generalized neural network approach to mobile robot navigation and obstacle avoidance, in: Intelligent autonomous systems 12, Springer, 2013, pp. 25–42.
[34] M. K. Singh, D. R. Parhi, Path optimisation of a mobile robot using an artificial neural network controller, International Journal of Systems Science 42 (2011) 107–120.
[35] L. Chrestien, T. Pevny, A. Komenda, S. Edelkamp, Heuristic search planning with deep

neural networks using imitation, attention and curriculum learning, arXiv preprint arXiv:2112.01918 (2021).

[36] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: K. Inui, J. Jiang, V. Ng, X. Wan (Eds.), Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019, Association for Computational Linguistics, 2019, pp. 3980–3990.

[37] Y. Peng, S. Yan, Z. Lu, Transfer learning in biomedical natural language processing: An evaluation of BERT and ELMo on ten benchmarking datasets, in: Proceedings of the 2019 Workshop on Biomedical Natural Language Processing (BioNLP 2019), 2019, pp. 58–65.

[38] M. Polignano, P. Basile, M. De Gemmis, G. Semeraro, V. Basile, Alberto: Italian bert language understanding model for nlp challenging tasks based on tweets, in: 6th Italian Conference on Computational Linguistics, CLiC-it 2019, volume 2481, CEUR, 2019, pp. 1–6.

[39] W. U. Ahmad, S. Chakraborty, B. Ray, K. Chang, Unified pre-training for program understanding and generation, in: K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tür, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, Y. Zhou (Eds.), Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021, Association for Computational Linguistics, 2021, pp. 2655–2668.

[40] R. Sharma, F. Chen, F. H. Fard, D. Lo, An exploratory study on code attention in BERT, CoRR abs/2204.10200 (2022).