Action Prediction for Conversational Agents with BERT

Giovanni Cravero
Politecnico di Torino
s193598@studenti.polito.it

Eugenio Dosualdo
Politecnico di Torino
s271880@studenti.polito.it

Marco Rondina
Politecnico di Torino
s280096@studenti.polito.it

Abstract

Natural Language Processing is one of the most relevant frontiers of artificial intelligence. The BERT model has proven to be the state-of-the-art in this field. In this paper we will focus on studying how this model can be used in a specific challenge, SIMMC, by analysing different network configurations, different parameter configurations and their results.

1. Introduction

The goal of an artificial intelligence based conversational agent is to understand the meaning of a natural language input and to produce an adequate response. We focused on goal-oriented and multi-turn conversations between two agents: an user and an artificial assistant. The idea is to produce a model able to calculate the response on the basis of the actual request and the past interaction within the same conversation. This type of interaction allows a flexible and dynamic natural conversation. All the information needed can be provided in different turns and one of the tasks of the assistant is to drive the conversation toward the reaching of the goal.
In the context of our work, a shopping virtual assistant, the main goal is to help users do shopping, suggesting the appropriate item from the catalogue and clarifying any doubts about it. The specific objective of the model was to correctly identify the action to be performed to serve the requests received by the user. This can be seen as the first step in creating a virtual assistant capable of interacting with the user in a human way.
At each turn, the assistant has to:

- choose the most appropriate **action** to perform **related to the input received**, like the request to specify a particular information, the search in the database of some products or simply adding the chosen product to the shopping cart;
- attach the appropriate **attributes** at the chosen action (when needed), like color or material;

The work does not include the task of response generation and, therefore, this work can be categorised as a classification problem.

2. Background

In this project, there were two main goals:

- predict agent action in a conversation (multi class classification)
- predict action attributes (multi label classification)

These goals are language classification tasks, that are part of the more general problem of Natural Language Processing (NLP). We used a BERT model which we fine-tuned in order to accomplish both tasks at the same time. The concept of transfer learning, as pointed out in the work of Chi Sun et al. [5], is a key one when it comes to NLP problems.

The BERT model is built using a Transformer-based architecture, as described in the work of Vaswani et al. [7]. There is a stack of encoders, consisting of an attention level and a feed forward network, and a stack of decoders, consisting of two attention levels (one connected to the previous output and one connected to the encoder output) and a feed forward network. The basic concept involves calculating a score of each individual word with all the words in the input. By exploiting the multi-attention principle, the number of such parameters increases, being condensed using a softmax score before inserting the resulting matrix into the subsequent feed forward network.

The modern approach in several natural language understanding tasks consists in pre-training language models on a large network with a large amount of unlabeled data and fine-tuning in downstream tasks, such as and BERT (Devlin et al.,2018, [2]) and OpenAI GPT (Radford et al., 2018, [4]). Devlin et al. [2] shows that BERT obtains new state-of-the-art results on several natural language processing task.

The model is pretrained on a large unlabelled English language corpus (BookCorpus [8] and English Wikipedia) that contains a representation of the English language usable for several downstream tasks. Pre-training was carried out through two distinct tasks: Masked Language Modeling

(MLM) and Next Sentence Prediction (NSP). The objective of the first task is to predict a subset of the words of a sentence, while that of the second is to correctly classify whether two sentences are consequential to each other or not. One of the peculiarities of such a pretrained model is that it makes possible training based on a potentially very large input, because no prior labelling is required.

The model is therefore initialised with the pre-trained parameters, which are then calibrated by the downstream tasks. The architecture provides a multi-layer bidirectional transformer encoder capable of using bidirectional self-attention, so each token can attend to the context to its left (as in the OpenAI GPT transformer implementation) and to its right.

In its BASE version, the one used within the work described in this paper, the model has 12 layers (transformer blocks), a hidden size of 768 and 12 self-attention heads, for a total of 110M parameters.

Devlin et al [2] shows that BERT is a new kind of language representation model that can achieve state of the art result while being conceptually simple to use. It is a pre-trained model that can be easily fine-tuned with just the addition of a layer at the end. Moreover, quoting Chi Sun et al. [5], *"it's potential has yet to be fully explored."*

3. Dataset: SIMMC

Situated Interactive Multimodal Conversation (SIMMC) is a dataset created to develop virtual assistants capable of handling complex, task-oriented conversations in a co-observed situated multimodal context. In particular,the word *"situated"* implies that the user and assistant are continually co-observing the same context, and that context can be updated on each turn.

SIMMC tasks and datasets are regarded as a new direction towards the training of next generation of agents that can process multimodal inputs and provide multimodal outputs beyond the traditional NLP stack.

The paper by Moon et al. [3] proposes three tasks to be performed on two SIMMC datasets in order to measure the performance of a model in the simulated environment. This work focuses on the first one, **Structural API Call Prediction**.

This project involves predicting the assistant action as an API call along with the necessary arguments, using the dialog history, the multimodal context and the user utterance of that round as inputs (where a round consists in a new user utterance which will be followed by an assistant response). For example, enquiring about an attribute value (e.g., price) for an item is realized through a call to the *SpecifyInfo* API with the price argument.

In order to evaluate the results of the model, the evaluation framework provided in the paper mentioned above was used. The evaluation is based on three parameters:

- *Action accuracy*: represents the prediction accuracy regarding the actions
- *Perplexity*: represents how confident is the model of a choice it made: the higher it is, the less is the model evaluated
- *Attribute accuracy*: represents the prediction accuracy regarding the attributes

3.1. Data description

The project is based on the datasets provided by Situated and Interactive Multimodal Conversations (Moon et al. [3], 2020), totalling ∼13K human-human dialogs (∼169K utterances) collected using a multimodal Wizard-of-Oz (WoZ) setup, on two shopping domains:

- **furniture** – grounded in a shared virtual environment;
- **fashion** – grounded in an evolving set of images.

Datasets include multimodal context of the items appearing in each scene, and contextual NLU (natural language understanding), NLG (natural language generation) and coreference annotations using a novel and unified framework of SIMMC conversational acts for both user and assistant utterances.

The model was trained on the *fashion* dataset only, without considering the images of the items. Table 1 contains all types of actions in the dataset. It is specified which actions may or may not have attributes associated with them.

API Action and Attributes	
Action	Attributes
SearchDatabase	yes
SearchMemory	yes
SpecifyInfo	yes
AddToCart	-
None	-

Table 1. Action and relative attributes to predict

3.2. Used fields

In order to perform the set tasks, the fields used are: *'transcript'* and *'system_transcript'*. They represent the written text by the user and the agent respectively. The *'system_transcript'* can be seen as the natural language representation of the predictions on the actions (and related attributes) to be performed by the agent: for this reason only system transcripts related to previous turns are used, thus respecting the prescriptions of the 2020 SIMMC challenge and avoiding providing the model with the answer it should be able to predict.

3.3. Dataset splits

The division between train, dev, test-dev and test-std was not carried out by us independently, but took advantage of the random division into components already present in the SIMMC-dataset (tab 2).

Dataset split			
Split	Percentage	#dials	#samples
Train	60%	3.929	21.196
Dev	10%	655	3.513
Test-Dev	15%	982	5.397
Test-Std	15%	983	

Table 2. Percentage and Count of every dataset split

In our project, we left out the *Test-Std* dataset as it was not necessary for our task. The data was built exploiting the SIMMC script for the data extraction, as explained in appendix B.

4. Model

The implementation of our model (code available on GitHub [1]) started from the basic version of the Bert Model of Huggingface. The pretrained `bert-base-uncased` model was used, which makes no distinction between upper and lower case characters.
At the exit of the pre-trained model, we added an additional intermediate linear layer to the base model. We tested various configurations : the final one for our model is a 768-node layer that which takes as input the 768 weights (hidden size) of the first token (`[CLS]`) of each batch and outputs 768 features to the rest of the network.
After that, two more parallel layers are added. The first one outputs 5 features - where 5 is the number of actions to be interpreted - and it will be used for the multi-class classification task. The second, on the other hand, has as its output dimension the number of attributes to be associated with the actions: this layer will instead be used for multi-label classification. The number of the distinct attributes to predict has been extracted from the datasets.
The goals of choosing the action and choosing its attributes are obviously linked, while remaining separate tasks. A multi-task learning approach was therefore chosen to improve the performance of the model in terms of more precise results.

4.1. Input manipulation

4.1.1 Input composition

The context of the problem analysed in this study involves multi-turn dialogues between two subjects, a user and an assistant. One of the main issues to be resolved concerns the ability to include a representation of the dialogue history, so that the prediction of a given turn also takes advantage of the information previously collected. This requirement has to be balanced against the limits of the maximum number of tokens that the BERT network accepts as input, i.e. 512 tokens. Although the dataset contains an average of 5.39 ± 1.44 pairs of shifts, 11.2 ± 6 utterance length for the user and 11.1 ± 6.4 utterance length for the assistant, the maximum values can reach 12 utterance length shifts and 40 utterance length for both the user and assistant expression. One strategy may be to progressively include the entire history of previous turns within the same dialogue, truncating excess characters. In our work, in order to maintain a balance between accuracy and performance, we have instead used a different construction of the single row of data necessary for the classification tasks.

As described in the table 3, we opted to place alongside the transcript relating to the user's request (linked to the current round) a "dialogue memory" restricted to the question and answer of the previous round. To do this, we exploited the possibility of providing two distinct inputs to BERT, separating the two sentences with the token [SEP]. In the rounds following the first one, therefore, the first sentence will be a composition Transcript (T) and System Transcript (ST) of the previous round; the second sentence will be the Transcript (T) of the current round, which the class and the attributes being trained refers to. The first round of each dialogue, on the other hand, consists of a single input sentence, as there is no previous history.

Input composition				
Token	Previous Transcript	Previous System Transcript	Token	Transcript
[CLS]				T1
[CLS]	T1	ST1	[SEP]	T2
[CLS]	T2	ST2	[SEP]	T3
[CLS]	T3	ST3	[SEP]	T4

Table 3. Sentences composition

The records constructed in this way are then transformed into input tensors with the functions offered by the HuggingFace library. Although the entire past history of the dialogue is not present for each record, the network should still be able to recognise similarities between similar dialogue states without training too precisely on the dialogue itself. A fairly intuitive example concerns the similarity, inter dialogue, of the final thank you or acceptance phase before adding to the shopping cart.

4.1.2 Example of Input composition

1. (a) *User*: Is there a pattern on this one? It's hard to see in the image.
   (b) *Assistant*: I don't have any information on the pattern, but it has pointelle embellishments.

2. (a) *User*: That's fancy. Do you have anything in warmer colors like yellow or red?

   (b) *Assistant*: I have a crew neck sweater in red, would you like to see it?

3. (a) *User*: Yeah, that sounds good.

   (b) *Assistant*: This is $187 from Downtown Stylists with a 3.62 rating.

4. (a) *User*: Oh, I love that. Please tell me you have a small.

   (b) *Assistant*: It does come in small, shall I put one in your cart?

5. (a) *User*: Yes, please! Thank you for your help with this

Will result in the following input records:

Input composition				
Token	Previous Transcript	Previous System Transcript	Token	Transcript
[CLS]				1A
[CLS]	1A	1B	[SEP]	2A
[CLS]	2A	2B	[SEP]	3A
[CLS]	3A	3B	[SEP]	4A
[CLS]	4A	4B	[SEP]	5A

Table 4. Example of sentences composition

4.1.3 Label encoding

As described in the previous paragraphs, the two classification tasks involve the need to correctly predict an action and a number of attributes for each individual record (constructed as described in the previous paragraph). Action prediction is a multi-class classification task, because it is necessary to assign one, and only one, of the 5 different classes to the transcript of the user under analysis.

Attribute prediction, on the other hand, is a multi-label classification task, because it involves assigning from 0 to #attributes of attributes. To represent this type of information, a binary vector of dimension #attributes was used, where the position within the vector makes it possible to understand which label the present value refers to. The encoding provides for a 1 if the label is present, a 0 if the label is not present.

4.2. Intermediate layers

From the output of the BERT base model, the `[CLS]` token is usually used for classification purpose, since it is a special classification token [5] [2]. This practice is common because the `[CLS]` token is a sort of "summary" of the processed input, so it has information of the entire input sequence without the cost of using all of the values for all of the weights in further processing the results. The value of the `[CLS]` is passed through the fully-connected layer added to the model, and then the result is used in the last two layers: the one which predicts the class and the one which predicts the attributes of the class. Note that the last two layers are not cascaded but parallel: the input they receive is the same, obtained from the 768 features given from the hidden layer directly after BERT's architecture (fig. 1). The choice has been to use the same information for the multi-class classification and the multi-label classification, in order to emphasize the logical connection between these two tasks. Several configurations of additional layers were tested. These configurations aimed to measure whether performance was improved by a different network configuration, prior to the linear layer that precedes the task-specific activation function. The tests focused on configurations that gradually reduced the number of features, sometimes interspersed with several activating functions. For this purpose, we used a sequence of linear levels alternated by ReLU activation functions. Because of its peculiarities, we also tested the use of Leaky Relu. In all these configurations, the output was always used by two parallel linear layers that had the task of adapting the number of features to the number of actions and attributes. The following table highlights the results of this examination (epochs: 6; batch size: 16, learning rate: 5e-5). While the accuracy of the actions is almost comparable, the accuracy of the attributes is higher with the presence of a single intermediate linear layer with input dimension 768 and output dimension 768. The transformation applied by the linear layer, is the following one:

$$y = x \cdot A^T + b$$

Where y is the output, x is the input, A is the weights vector and b is the additional bias learned by the network.

Intermediate layers comparison			
Confs.	Action accuracy	Action perplexity	Attribute accuracy
1	84,82	5,53	77,93
2	84,70	5,31	77,53
3	84,53	8,63	75,16
4	84,40	5,82	73,56
5	84,33	9,54	74,14
6	84,44	7.56	72,88
7	84,70	9,69	75,19

Table 5. Examination of different layers between the last hidden output of BERT network and classification layers

Configurations:

1. linear(768,768)

2. linear(768,256)

3. linear(768,512) + linear(512,256) + linear(256,128)

4. linear(768,512) + ReLU + linear(512,256) + ReLU + linear(256,128) + ReLU

5. linear(768,768) + ReLU + linear(768,768) + ReLU + linear(768,768) + ReLU

6. linear(768,512) + LeakyReLU + linear(512,256) + LeakyReLU + linear(256,128) + LeakyReLU

7. linear(768,768) + LeakyReLU + linear(768,768) + LeakyReLU + linear(768,768) + LeakyReLU

4.3. Activation functions

There are two activation functions at the end of the model, one for each task on the respective layer. For the action prediction task, we chose to use the *softmax* activation function, while we use a *sigmoid* function for the attribute prediction.
Regarding the action prediction task, we used the *softmax* function, defined as:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{i=1} e^{z_i}}$$

This function is the standard choice for multi-class classification problems, since it normalizes the results given by the model to represent a probability distribution, hence indicating which class is most likely to be associated with the input. The activation function we used for predicting the attributes was a *sigmoid* function, again the common choice for this kind of problems. In particular, the model uses the *logistic sigmoid function* provided by the PyTorch library, defined as :

$$sigmoid(z_i) = \frac{1}{1 + e^{-z_i}}$$

4.4. Loss function

As mentioned above, multi-task learning was used to perform both tasks in parallel.This choice comes naturally since that the only answer considered right for the model is the correct action along with all the correct attributes.The model has to learn how to maximise the precision in the multi-class task while also being as good as possible in the multi-label problem. The best way to reflect this behaviour is the loss function is given by the sum of the single loss functions specific for each task.

4.4.1 Actions

For the calculation of the loss regarding the choice of the action (multi-class classification), the *CrossEntropyLoss* by PyTorch was used:

$$loss(x, class) = -log\left(\frac{e^{x[class]}}{\sum_j e^{x_j}}\right)$$

Where *x* represent the vector of probabilities calculated for the sample by the model, and *class* is the target class that should have been predicted. This is the standard choice for the loss function of a multi-class classification problem.

4.4.2 Attributes

For the calculation of the loss regarding the choice of the attributes (multi-label classification), the *BCELoss* by PyTorch was used. *BCELoss* stands for *Binary Cross-Entropy Loss*. Before the definition of the loss for the batch, *L* must be defined

$$L = \{l_1, ..., l_N\}^T$$

$$l_n = -w_n[y_n * logx_n + (1 - y_n) * log(1 - x_n)]$$

Where *N* is the batch size. The value of the loss is therefore:

$$loss(x, y) = mean(L)$$

5. Results

In this section the results obtained from the model are shown. For every parameter, we tried different combinations of the values. To the values suggested in Delvin et al. [2] we added new ones found empirically. Experimentally we note that the model performs better with 6 epochs instead of 4, and with a reduced batch size (from 16 to 12). These are the values we used for the parameters:

- epochs: 2,4,6;
- batches: 12,16,32;
- learning rate: 2e-5,5e-5;

In table 6, the best results are shown, ordered by attribute accuracy. The results for the chosen configuration of the model are highlighted.

5.1. Weight optimization for class balancing

The distribution of classes in the training dataset is not well balanced: there is a class, *SearchMemory*, that has substantially less samples than the other ones (695 out of 21196 total, $\sim 3\%$). This caused our model to avoid predicting that class, as can be seen in the confusion matrix (fig.2).
In order to overcome the problem, we based our work on

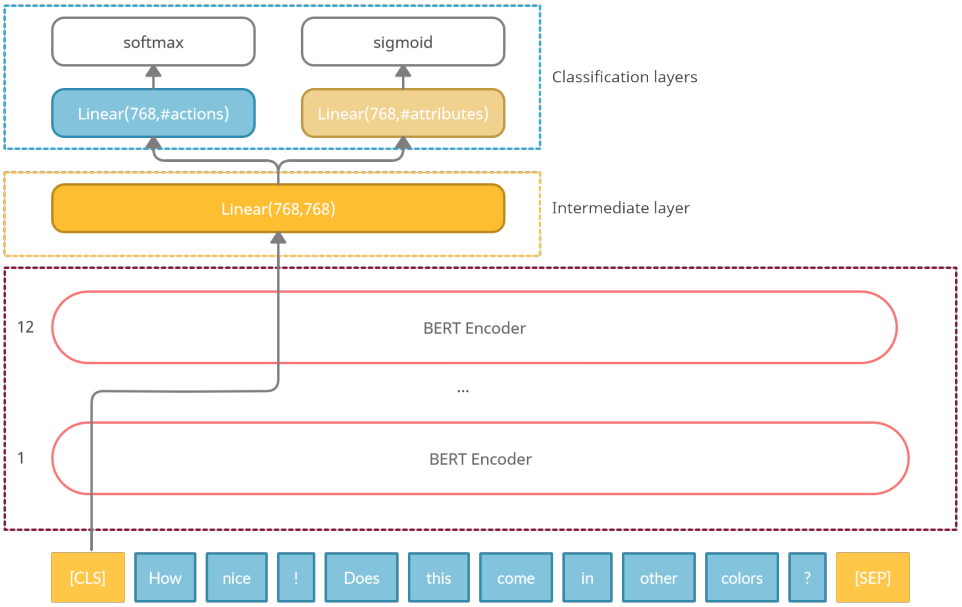

Figure 1. Model diagram

Best runs for finetuning					
E	B	LR	Act	P	Att
6	12	5e-5	85.14	6.30	79.67
6	16	5e-5	84.82	5.52	77.93
6	12	2e-5	84.95	5.11	77.91
4	16	5e-5	84.71	5.64	76.22
4	12	5e-5	84.62	5.51	75.67
6	16	2e-5	85.16	4.49	75.50
6	32	5e-5	84.82	4.91	74.84
4	12	2e-5	85.16	4.29	74.25
2	12	5e-5	84.47	4.53	74.09
4	32	5e-5	84.77	4.31	73.88
4	16	2e-5	84.96	3.83	73.78
2	16	5e-5	84.36	4.61	73.29
6	32	2e-5	85.03	3.61	72.99
2	12	2e-5	84.55	3.49	70.40
2	32	5e-5	84.42	3.73	68.21
4	32	2e-5	84.81	3.25	67.82
2	16	2e-5	84.68	3.35	66.34
2	32	2e-5	84.45	2.85	56.70

Table 6. Best runs for actions and attribute accuracy (ordered by attribute accuracy).
Legend: **B:** Batch dim., **E:** # of epochs,
LR: Learning rate,**Act:** Action accuracy,
P: Action perplexity, **Att:** Attribute accuracy

Madabushi et al. [6]: we passed to the *CrossEntropyLoss* function a vector of weights as input parameter, in order to give more importance to the less represented classes. Also, we tried decreasing some values for the most represented

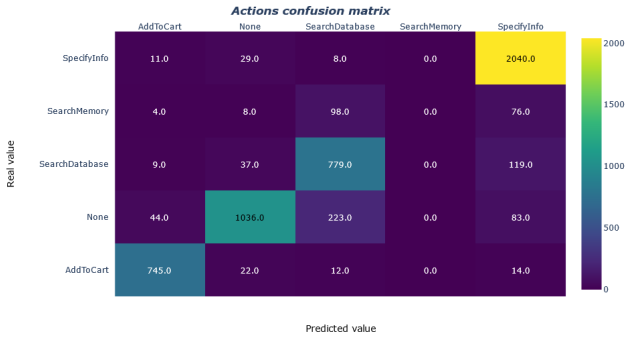

Figure 2. Actions confusion matrix before weights optimization (E:6/B:12/LR:5e-5). The *SearchMemory* class is never predicted

classes, in order to prevent the model from predicting those too often.
The weights we chose to be associated to the classes are:

- *None*=0.9;
- *AddToCart*=1;
- *SearchDatabase*=0.85;
- *SearchMemory*=3.0;
- *SpecifyInfo*=0.8;

After this addition to the model, we present the final results obtained in table 7. With this change the performance of the model is much improved regarding perplexity, with also a slight improvement in accuracy for actions and attributes.

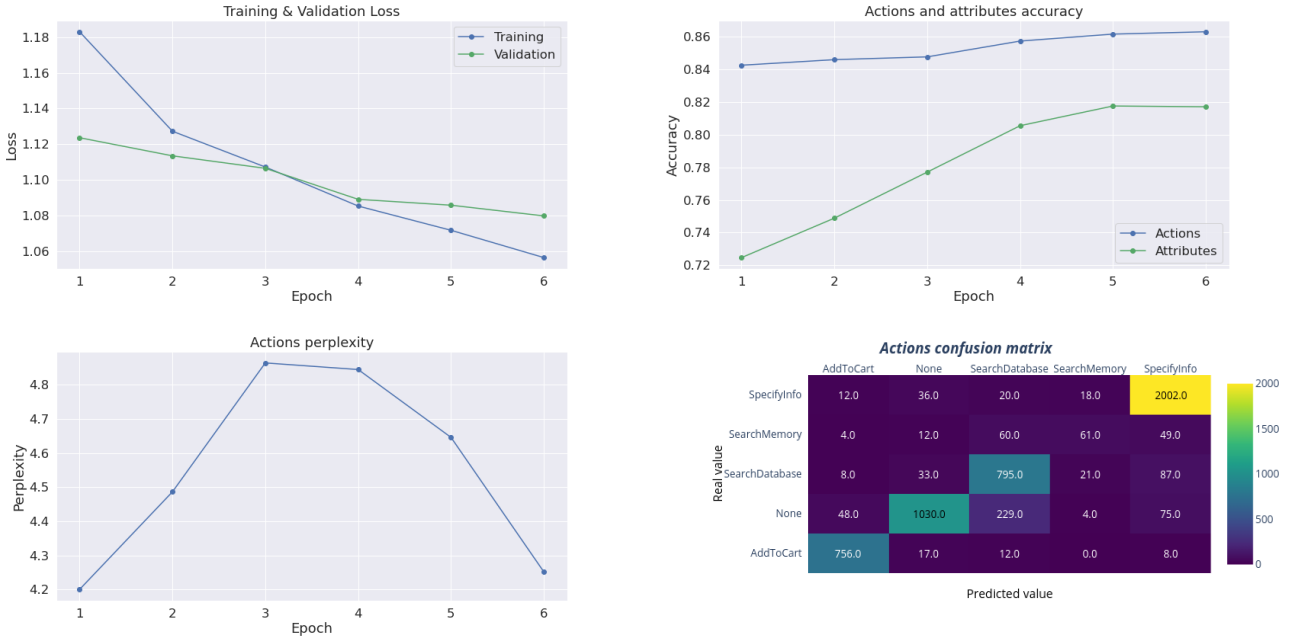

Figure 3. Best run stats

Figure 3 shows the plots of the loss, accuracy and perplexity functions. It can be seen that, although the plot of the loss functions seems to indicate a principle of overfitting from epoch 5 onwards, the loss of validation continues to fall slightly and there is a large reduction in perplexity between epochs 5 and 6.

Table 7. Best run accuracy after class balancing.

E	B	LR	Act	P	Att
6	12	5e-5	86.05	4.54	80.51

Legend: **B:** Batch dim., **E:** # of epochs,
Act: Action accuracy, **P:** Action perplexity,
Att: Attribute accuracy

Statistical distribution			
Metric	Action Accuracy	Action Perplexity	Attribute Accuracy
count	25	25	25
mean	85.56	4.48	79.62
std	0.23	0.19	0.53
min	85.16	4.15	78.41
25%	85.42	4.36	79.30
50%	85.53	4.47	79.59
75%	85.71	4.58	80.02
max	86.05	5.09	80.51

Table 8. Statics for chosen model

5.2. Statistical analysis on several executions

We built a set of 25 samples with the parameters indicated in section 5.1, from which we extracted the best run (table 7, fig 3). The action accuracy, action perplexity and attribute accuracy are distributed as indicated in table 8 and illustrated in fig.4.

6. Conclusion

Various considerations can be drawn from the results. The first one is that unfortunately the different configurations we tried for the hidden layer don't seem to affect the results much (at least for the accuracy of actions, on attributes there is a higher incidence). In addition to parameter finetuning, the most effective modification we made was class rebalancing: this allowed the model to learn to classify even the least represented action, instead of ignoring it altogether. Although the result does not appear very different in terms of accuracy on predictions, it is also important to consider that the reason lies in the scarce presence of samples of such class in the dataset. This is certainly a major improvement to the original model, making it more generalisable. For future implementations, one notable thing is the model's behavior of predicting so many samples of *'None'* as *'SearchDatabase'*; the correction of this issue might be a possible starting point for improvements. In conclusion, what this model certainly can confirm is the effectiveness of BERT in producing state-of-the-art results in NLP problems: despite the relative simplicity of the model,its scores seem to be comparable to those of the official SIMMC results.

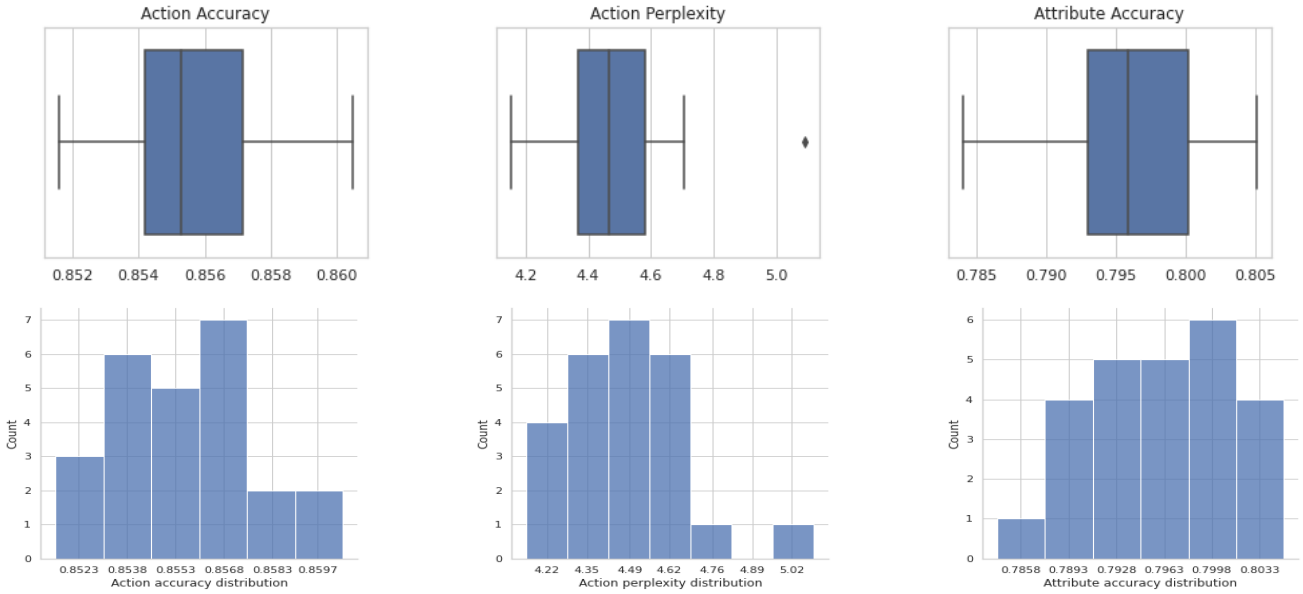

Figure 4. Accuracy and perplexity statistics

References

[1] https://github.com/jmcrav/ap4ca. 3

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. 1, 2, 4, 5

[3] Seungwhan Moon, Satwik Kottur, Paul A. Crook, Ankita De, Shivani Poddar, Theodore Levin, David Whitney, Daniel Difranco, Ahmad Beirami, Eunjoon Cho, Rajen Subba, and Alborz Geramifard. *Situated and Interactive Multimodal Conversations*, 2020. 2

[4] Alec Radford and Karthik Narasimhan. *Improving Language Understanding by Generative Pre-Training*. 2018. 1

[5] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. *How to Fine-Tune BERT for Text Classification?*, 2020. 1, 2, 4

[6] Harish Tayyar Madabushi, Elena Kochkina, and Michael Castelle. Cost-sensitive BERT for generalisable sentence classification on imbalanced data. In *Proceedings of the Second Workshop on Natural Language Processing for Internet Freedom: Censorship, Disinformation, and Propaganda*, pages 125–134, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. 6

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*, 2017. 1

[8] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*, 2015. 1

Appendix A. Alternative choice for activation function

One thing we noticed is that despite the good scores in the accuracy and attrubute predictions, the *perplexity* score was quite high. One possible solution we found was in using the *sigmoid* function as an activation function also for the class prediction. The reason we believe the *sigmoid* works best for *perplexity* computation is that we give the evaluator a vector of results not as polarized as in the case of *softmax*.
Experimental results have shown that using *sigmoid* for this classification task allows to significantly reduce the *perplexity* score of the model while keeping its accuracy approximately the same, as shown in table 9.
The *sigmoid* function usually is not taken in account for classification tasks because it is not mutually exclusive:there is the possibility of predicting more than one class for an input and the sum of all the predicted probability it is not 1, *i.e.* the probability values are indepentent of each other. For these reasons, we opted to retain the softmax as activation function.

Sigmoid activation f. for actions and attributes					
E	B	LR	Act	P	Att
2	14	5e-5	84.79	2.30	73.05
3	14	5e-5	84.73	2.45	73.12
4	12	2e-5	84.95	2.45	73.34
6	12	5e-5	85.23	2.88	77.88
6	12	2e-5	84.88	3.25	73.61

Table 9. Results with *sigmoid* as activation function.
Legend: **B:** Batch dim., **E:** # of epochs, **Act:** Action accuracy, **P:** Action perplexity, **Att:** Attribute accuracy

Appendix B. Code repository and reproducibility

The code behind what is described in the paper is available on github: https://github.com/jmcrav/AP4CA. Input data are extracted with the extract_actions_fashion.py script by SIMMC GitHub repository.
In order to ensure correct execution of the code, the Python scripts must be executed in an environment with a Python version ≥ 3 with these additional requirements:

```
pip install transformers==4.9.2
pip install absl-py
pip install pandas
pip install seaborn
pip install matplotlib
pip install numpy
pip install datetime
pip install torch==1.9.0
pip install sklearn
```

The runner can be executed with the command

```
python ap4ca/run.py
```

The optional command line arguments can be seen with the command

```
python ap4ca/run.py --help
```

and are further described in the linked repository (*readme* section.