DeepStories - A generative model for interactive fiction

Simge Ekiz, Daniele Maccari

***Abstract*—Interactive Fiction (IF) is computer- and text-based interactive narratives in which players can directly affect the way the story develops by typing commands into an interpreter. Standard IF works are static, as the story is determined as *a priori* by the author. The aim of this project was hence to investigate a generative model for IF, i.e. a model in which the user's past decisions influence the system's reply, thus providing players with a unique experience every time. To this end, we frame the problem in terms of a neural conversational model. We then implement and compare different instantiation of this paradigm, trained on a dataset consisting of playthroughs collected on a variety of IFs. While we cannot claim any astounding result, we still provide a few reflections regarding neural conversational models.**

I. Introduction

In their most general form, Interactive Fiction (IF) games can be understood as computer- and text-based interactive narratives. The user interacts by typing in commands which an interpreter parses, generating an appropriate response in return. This special combination of interaction and narrative requires authors to not only consider the story to be told, but also the set of available interactions. For instance, going left instead of right will present the user with two very diverse descriptions. Or, the set of object currently available to the player will influence possible actions in the same environment. Different scenes also depend on each other in a constrained fashion, as actions will change the state of the world, affecting the progress of the story as it is being told. As such, IF constitutes a hard test-bed for natural language processing techniques. Of course, IF's narrative-based nature also poses limits to the possibilities offered to the user. There is a story to be told, and the choice of actions and scenes, as large as it may be, is still confined to what the developer allowed for in the first place. Very popular in the 1980s and 1990s, new IFs are still being published nowadays, with a loyal fanbase and dedicated communities[1].

In this project, we set forth to investigate the use of a neural conversational model [1] based on sequence-to-sequence framework [2] to reproduce this duality in a dynamic setting. In other words, we asked ourselves the question: can we create believable interactions in real time?

The background literature that motivated us during the development is reviewed in Section 2. In section 3, we mainly describe our approach and our design to solve the problem and the results of our experiments. A summary of the project is given in Section 4. Finally, in Section 5, we conclude our study and also discussed the different approaches that can improve the results as a future work.

The project is made open source. The code can be found in a GitHub repository[2].

II. Background

While there has been some research on interactive narratives, most of the previous studies focused on building agents capable of *playing*, rather than *narrating*, the games. In [3], He et. al. use deep reinforcement learning in what they name a deep reinforcement relevance network (DRRN). Here, despite part of their architecture's resemblance with our approach, their ultimate goal is to approximate state-action pairs values in order for the agent to learn how to play. In their work, Narasimhan et al. [4] follow a similar approach, although they rely on a slightly different architecture. Some example of the latter also exists. In [5] Chourdakis and Reiss use both reinforcement learning and generative artificial neural networks. However, their goal is not to generate entirely new narratives, but to learn how to recombine sentences found in the corpus in novel ways.

In terms of neural conversational models, the main example is probably [1], in which the authors trained a sequence-to-sequence network [2] on a series of conversational tasks such as e.g. IT helpdesk, showing interesting results. Other work include dynamic memory networks [6], though their architecture is considerably more complex.

III. Project Description

As stated in Section I, the scope of Interactive Fiction games is much broader than just one-off interactions. They comprise a complex description of objects, environments, characters, and their interdependencies. We therefore limit our investigation to a simpler question: "Can we generate believable interactions on-the-fly?". This translates to simple triples of the form `<scene, command, reply>` where the system first presents a user with a scene description, to which the user replies with a command. The system then generates the next scene based on the joint representation of `scene + command`. To this end, we adapt the conversational model as presented in [1], by training the system on `<scene + command, reply>` input-output pairs. To assess the performance of the model, we compare slight variations thereof. Specifically, we train and evaluate a basic network and an attention-powered sequence-to-sequence network. Both networks are further trained on sequences of length of 200 and 50 words to see how this influence the generation process. Finally, during the tests,

s4706757. email: sekiz@student.ru.nl
s4711262. email: d.maccari@student.ru.nl

[1] See e.g. `http://ifdb.tads.org` and `http://www.ifarchive.org`

[2] `https://github.com/macco3k/deepstories`

we compare simple greedy decoding with beam-search decoding.

The next section describes the dataset of input-output pairs that the models are trained on in more detail.

A. Dataset

As previous work focused on a few, manually-selected games, and given their different objectives, we have to look elsewhere for a dataset suitable to our task. While ideally we would like to generate training data by actually playing the games, as this would allow for the generation of far more interactions, the limited amount of time and resources force us to turn our attention to pre-generated data. In particular, we use transcripts of recorded game sessions collected and made available by the ClubFloyd community[3]. This collection consists of playthroughs of 241 IF games played over the course of more than 10 years. Although not all of them is usable, as some contains specially-formatted data or present text in languages other than English, they still provide us with a wealth of data. After removing unusable parts, total of 228 files to process is left.

B. Preprocessing

As different games are played on different interpreters with different formats, the processing of the raw transcripts prove quite challenging. Indeed, these are characterized by meta data such as the '`Floyd |`' interpreter prompt or chats between the players. Moreover, the text can contain noise in the form of shortcut commands, about or help text. We thus proceed to clean the data in order to obtain the canonical `<scene, command, scene>` structure (see Figure 2 in the appendix for an example of text pre- and post-processing). We also expand contractions via the PyContractions package[4]. To recover the triples necessary for the training of the network, we slide a 3-lines wide window over the text with a stride of 2.[5]

C. Design

To account for the relationship between scenes and commands, we use a shallow sequence-to-sequence architecture. While the literature usually suggests deeper architectures, their dilated training time persuaded us not to use them. As depicted in Figure 1, the input to the encoder is the concatenation of `scene` and `command`, while the decoder learns to predict `reply`. Before tokenization, each line is further preprocessed to remove capitalization and punctuation. We also use pre-generated word embeddings trained on Google News [7] to map each word to a 300-dimensional vector[6]. In addition to providing us with a representation which captures word similarity, using these word embeddings also save us some time during training, as the network does not have to learn new embeddings. We set the vocabulary size to 20k words to keep the number of parameters for the final prediction layer small enough. OOV words are mapped to the special `UNK` token, which is assigned the zero-vector. Words in the vocabulary for which embeddings cannot be found are treated the same way. A special all-ones `EOS` token is used to signal the end of the sentence. To enforce sequence length, we proceed as follows; first, all input lines longer than the maximum length are converted to sub-sequences. Each such sub-sequence then becomes a new line. Unfortunately, for longer sequences this means losing the relation with the corresponding command, as a legit triple could be replaced by a `<sub-sequence, sub-sequence, sub-sequence>` one in which all components are from the same original sequence. We discuss this at more length in Section VI.

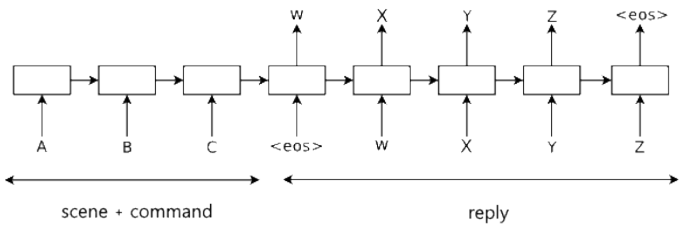

Figure 1: The architecture of the network.

Predictions are carried out in the usual way, by first encoding the input sequence and subsequently feeding the encoder's final states into the decoder together with a `EOS` token. The decoder then generates one input at a time, reusing its own output and states for the next prediction. In addition to a greedy *argmax* approach which is choosing the most likely word for the next step , a simple beam-search decoding strategy is implemented as well. The beam search expands the all possible next steps by keeping a list of k candidate sequences with their respective scores, computed as the product of the log-likelihoods of items in the sequence [8]. The maximum output length is set in both cases to 50 words and k is set to 5.

The attention-based models implement the simple dot-product attention mechanism provided in [9]. While more advanced version of attention exists (e.g. [10]), they are also more complex and require far longer training time.

All networks are trained for a total of 5 epochs on categorical cross-entropy. The optimizer is `rmsprop` with standard settings. Data is shuffled and 5% of it is reserved for validation. For each epoch, we generate 1000 batches[7] of 32 and 128 samples for 200 words and 50 words long sequences, respectively. Table I provides a summary of these settings.

D. Evaluation

While BLEU scores are the norm in many sequence-to-sequence papers, we did not deem such metric appropriate

[3] `http://www.allthingsjacq.com/interactive_fiction.html#clubfloyd`

[4] `https://github.com/ian-beaver/pycontractions`

[5] Some very wordy commands appeared to be valid in some games and were left as-is.

[6] We set the dimensionality of the internal representation to the same for consistency and to avoid longer training times.

[7] We settled upon this value after noticing the loss plateauing around it.

length	**epochs**	**batches**	**b. size**	**train-val**
200	5	1000	32	96,697-5,090
50	5	1000	128	115,773-6,093

TABLE I: Training settings. The train-val column refers to the number of triples.

in our case, as we are not dealing with a translation task. In order to provide some performance measures, we report the loss and accuracy of all models. As our goal is to generate realistic interactions, we also present some sample outputs on a few hand-crafted inputs. To avoid cluttering the exposition, we leave those for the appendix (see Section VII).

IV. Results

In Table II, we present a comparison of different networks. In addition to training and validation loss, we also report accuracy. Although it is not very meaningful for our goal, it still provides some insight about the differences between the models' behaviour. All reported numbers refer to the last epoch.

model	**loss**	**acc.**	**val. loss**	**val. acc.**
basic-200	0.4971	0.0251	0.4988	0.0256
basic-50	1.4515	0.1141	1.4967	0.1149
attention-200	0.7125	0.0407	0.6908	0.0396
attention-50	1.4479	0.1152	1.4225	0.1144

TABLE II: Training and validation loss and accuracy for all models.

At first sight, we can observe how longer sequences appears to help in decreasing the loss at the expense of accuracy. However, we believe the latter could be explained by the fact that the network trained on shorter sequences has simply less room for error. If we compare the relative improvement over chance, we can actually see a 5x increase in both settings. Interestingly enough, attention does not impact much, with performance comparable to or even worse than the no-attention counterparts. The reasons behind this behaviour can be twofold: first, the chosen attention mechanism is very simplistic, acting as a basic similarity-weighting layer before the softmax scores are computed. In this sense, the decoder is still being fed the raw encoder output + states, with no knowledge of which input it should be attending to. We are confident that more advanced techniques would be a better fit in this regard. Second, albeit some alignment of sort can probably be found for some input-output pairs, we question whether this occurs often enough for the network to capture such dependencies. In Section VI, we offer an alternative for the creation of samples which might help overcome this flaw.

When looking at the output produced by the different models (see Figures 3 through 6 in the appendix), we can again note a pattern. The basic models, i.e. with no attention, seem to generally perform better, especially when using beam-search decoder. We suspect the almost-replicas replies generated by the attention-based models can be attributed to the dot-product scoring function, which might reward similar input-output sequences too much. Within no-attention models, the one trained on longer sequences manages to produce intelligible output even in a simple greedy decoding setting. We interpret this as a sign that, in this case, the network had enough "context", both in the scene and in the subsequent reply, to actually learn some dependency, while the basic-50 model seems to get stuck in a typical loop by predicting the `UNK` token over and over again. Single commands such as `inventory` provide some variation, although this does not influence the output of beam-search significantly.

In general, though, all networks appear to learn to associate the same reply independent of the command attached to the scene. In hindsight, this is not surprising, as for many such inputs the command only contributes to the entire sequence for a few words at the end. We can think of the rest of the input as "washing away" this contribution.

V. Summary

In this paper, we study an approach to generate Interactive Fiction stories in real-time based on user input. In order to achieve this, we present a neural conversational model inspired by sequence-to-sequence architectures. While the rationale underlying the proposed model is intuitively simple, its concrete formulation proved unsuited for such a difficult task. Likewise, some limitations of RNNs – training time and sequence length above all – force us to lower the complexity of the models we can test. Despite interesting results, the goal stated in the introduction still appears quite afar.

In the last section, we provide some discussion on this and give some suggestions on improvements towards this end.

VI. Conclusions & Future Work

From the results presented in Section IV, it is clear that the proposed architecture, despite its intuitive appeal, necessitates further work before it can capture the original intuition behind its inception. As is clear, the model is completely unaware of any inner structure to the game, only having access to external information in the shape of interactions with the user. This means that for many outputs which are based on knowledge of the game internal state such as current scene, the player's inventory, etc., this relation is not available. Also, the fact that we trained on a corpus built from all games, while providing us with more training data, added a lot more variation to the dataset.

On the technical side, in addition to all of a series of tweaks (e.g. a deeper architecture or higher-dimensional embeddings) and tricks (e.g. gradient clipping) we think two modifications would have a major impact on the performance of the models.

First of all, the restriction to sequences of a fixed length, as anticipated, gave rise to a number of spurious training samples, either by truncating longer paragraphs in the middle of a sentence, or worse by entirely destroying the `<scene, command, scene>` structure. This most likely had a huge influence on the concrete patterns presented to the network during training, thus negatively affecting it. An alternative method would require a sampling strategy to select random sub-sequences while maintaining the canonical triple structure. Of course, a simple random strategy will only makes things slightly better, since we still have no way of knowing which part of the scene really matches the input command. In this respect, a similarity measure between the sub-sequence and the command[8] could provide a first approximation which more sophisticated attention mechanisms could take advantage of. The use of hierarchical architectures could also help in capturing longer-term dependencies for longer sequences, allowing to remove the sequence length limit altogether.

With respect to the models' inability to tell apart the same scene when paired with different commands, a more clever architecture could combine the scene and the command in a different way in order to avoid this phenomenon, e.g. by providing the command as a separate input to an attention-based decoder.

On a more general note, we point out the extremely small size of our dataset, especially when compared to the millions of pairs usually available for translation tasks. An automatic agent capable of replaying games many different times would virtually solve this problem by providing an almost endless set of samples. This would also facilitate the creation of training samples by doing away with the need for preprocessing. Finally, it would allow to control the extraction process in a more precise manner, e.g. by handling multiple commands referring to the same scene, which at the moment break the assumptions behind the use of a fixed sliding window.

Some inspiration may also come from previous work such as [3]. Indeed, excluding the additional reinforcement learning layer, their model also built upon a way to represent state-action pairs in interactive fiction games. A reinforcement learning framework could provide additional guidance for the network's predictions by enforcing the choice of relevant descriptions.

References

[1] O. Vinyals and Q. Le, "A neural conversational model," *arXiv preprint arXiv:1506.05869*, 2015.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[3] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf, "Deep reinforcement learning with a natural language action space," *arXiv preprint arXiv:1511.04636*, 2015.

[4] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," *arXiv preprint arXiv:1506.08941*, 2015.

[5] E. T. Chourdakis, J. D. Reiss *et al.*, "Constructing narrative using a generative model and continuous action policies," 2017.

[6] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," in *International Conference on Machine Learning*, 2016, pp. 1378–1387.

[7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[8] M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," *arXiv preprint arXiv:1702.01806*, 2017.

[9] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[10] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[11] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International Conference on Machine Learning*, 2014, pp. 1188–1196.

[8] E.g. a dot-product between their paragraph vectors, see [11]).

VII. Appendix

This appendix contains additional material which could not fit into the pages count.
Figure 2 shows an example of the input text before and after preprocessing.

```
Floyd | Polished carbuncles have been firmly fixed into the eye-sockets of
↪ this
Floyd | seemingly ordinary skull. The words "Saint Jean le Baptiste" have been
Floyd | roughly carved into the jaw.
Floyd |
Floyd | >
[clubfloyd] inky says, "or they could be in a dufflebag"
Johnny says (to Floyd), "x germany"
Floyd | Dried skin is stretched tightly over the skull, and two plain-cut
↪ green
Floyd | gemstones, possibly agate, fill the eye-sockets. A brass plaque
↪ riveted
Floyd | to the forehead reads "Sankt Johannes der T?ufer".
Floyd |
Floyd | >
[clubfloyd] Jacqueline says, "Maybe we're wearing cargo pants with lots of
↪ pockets."
[clubfloyd] DavidW says, "Or in an oversized Pez dispenser."
Binder says (to Floyd), "x israel"
Floyd | This battered head has seen better days. The mummified flesh shows
Floyd | rough abrasions, and it is free of adornments, lacking even glass eyes
↪ .
Floyd | Someone has scrawled a Hebrew word in pencil across the forehead.
```

(a) Before preprocessing.

```
Polished carbuncles have been firmly fixed into the eye-sockets of this
↪ seemingly ordinary skull. The words "Saint Jean le Baptiste" have been
↪ roughly carved into the jaw.
Examine germany
Dried skin is stretched tightly over the skull, and two plain-cut green
↪ gemstones, possibly agate, fill the eye-sockets. A brass plaque riveted to
↪ the forehead reads "Sankt Johannes der T?ufer".
Examine israel
This battered head has seen better days. The mummified flesh shows rough
↪ abrasions, and it is free of adornments, lacking even glass eyes. Someone
↪ has scrawled a Hebrew word in pencil across the forehead.
```

(b) After preprocessing.

Figure 2: Extract from a transcript before (a), and after (b) preprocessing.

All models were tested on the same `scene + command` sequence. The scene description:

```
You find yourself in a dimly lit room. Before you you can spot a table. On the other side of the
↪ room a poster hangs askew on the wall. To the east a door leads to what seems to be a large
↪ living room.
```

was combined with the following set of commands:

```
examine table
look
east
```

The stand-alone `inventory` command was also used with no scene description, to assess the model's response. Figure 3 shows sample outputs for the basic, 200-words sequence model. Figure 4 shows instead the results for the attention-based network. Figure 5 and 6 show the output for the basic-50 and attention-50 models, respectively.

```
> examine table.
you are in a small cave a large room with been been been used to use a few
↪ feet of the room there is a small room to the north and a small room to the
↪  north is a small window leading into the room to the north is a small

> look
you are in a small cave a large room with been been been used to use a few
↪ feet of the room there is a small room to the north and a small room to the
↪  north is a small window leading into the room to the north is a small

> east
you are in a small cave a large room with been been been used to use a few
↪ feet of the room there is a small room to the north and a small room to the
↪  north is a small window leading into the room to the north is a small

> inventory
the UNK is not a little more than a few feet away EOS
```

(a) Greedy search.

```
> examine table
you cannot see any such thing you can hear the crashing and clanging of a
↪ sword fight coming from the tavern common room EOS

> look
you cannot see any such thing you can hear the crashing and clanging of a
↪ sword fight coming from the tavern common room EOS

> east
you cannot see any such thing you can hear the crashing and clanging of a
↪ sword fight coming from the tavern common room EOS

> inventory
you cannot see any such thing you can hear the crashing and clanging of a
↪ sword fight coming from the tavern common room EOS
```

(b) Beam search with $k = 5$

Figure 3: Sample outputs for the basic-200 model with greedy (a) and beam (b) search.

```
> examine table
examine me EOS

> look
examine me EOS

> east
examine me EOS

> inventory
you are not wearing the key EOS
```

(a) Greedy search.

```
> examine table
south EOS

> look
south EOS

> east
south EOS

> inventory
look
```

(b) Beam search with $k = 5$

Figure 4: Sample outputs for the attention-200 model with greedy (a) and beam (b) search.

```
> examine table
a UNK of UNK and a UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK
↪  UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK tia UNK UNK UNK UNK
↪ UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK

> look
a UNK of UNK and a UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK
↪  UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK tia UNK UNK UNK UNK
↪ UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK

> east
a UNK of UNK and a UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK
↪  UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK tia UNK UNK UNK UNK
↪ UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK UNK

> inventory
you are carrying a black rubber strap EOS
```

(a) Greedy search.

```
> examine table
you can hear the crashing and clanging of a sword fight coming from the tavern
↪  common EOS

> look
you can hear the crashing and clanging of a sword fight coming from the tavern
↪  common EOS

> east
you can hear the crashing and clanging of a sword fight coming from the tavern
↪  common EOS

> inventory
you EOS
```

(b) Beam search with $k = 5$

Figure 5: Sample outputs for the basic-50 model with greedy (a) and beam (b) search.

```
> examine table
examine me EOS

> look
examine me EOS

> east
examine me EOS

> inventory
examine bag EOS
```

(a) Greedy search.

```
> examine table
inventory EOS

> look
inventory EOS

> east
inventory EOS

> inventory
west EOS
```

(b) Beam search with $k = 5$

Figure 6: Sample outputs for the attention-50 model with greedy (a) and beam (b) search.