Sequence-to-Sequence Models for Visual Attention Prediction in a VR Museum using the EDVAM dataset

A. Goodall

September 3, 2021

1 Introduction

Sequence-to-sequence (seq2seq) models have been used widely and effectively for tasks such as natural language processing (NLP) (Mikolov et al. 2010), image captioning (Vinyals et al. 2015) and even for some real valued cases (Yang et al. 2019).

In this paper we will look at applying these model architectures to a different type of problem - visual attention prediction in virtual reality (VR) worlds. To do this we will use the EDVAM dataset which consists of 3D visual fixation point of gazes (PoG), camera positions, camera orientations and other pupil data that was collected while users interacted with and navigated through a VR museum. Refer to (Zhou et al. 2021) for more details.

The original publication (Zhou et al. 2021) proposed the use of recurrent neural networks (RNNs) for predicting subsequent visual attention and it formulates the problem as a classification task. It splits the VR museum into 12 different 3D spaces, time series data is then fed into the model and the model predicts which of the 12 3D spaces it expects to contain subsequent visual fixation.

However, we will formulate the problem in a different manner. Firstly, we will only consider the following 9 features from the data:

- 3D PoG data $\langle X, Y, Z \rangle$ - 3 features.
- Camera position data $\langle X, Y, Z \rangle$ - 3 features.
- Camera orientation data $\langle X, Y, Z \rangle$ - 3 features.

The 3D PoG data corresponds to the (x, y, z) coordinates of the user's current visual fixation (the point they are looking at in 3D space), the camera position data corresponds to the (x, y, z) coordinates of the user's viewport in the 3D space, and the camera orientation data corresponds to the unit vectors that represent the user's viewport orientation.

We will define a frame to be a 9-tuple of real values $\langle g_{x_t}, g_{y_t}, g_{z_t}, c_{x_t}, c_{y_t}, c_{z_t}, v_{x_t}, v_{y_t}, v_{z_t} \rangle$ for each of these features. Using the EDVAM raw subset which consists of time sequences of these frames collected from 63 participants navigating a VR museum, we hope to train a RNN to predict the next m frames from a sequence of n past frames.

The goal of redefining the problem is to hope to acheive a more precise prediction of subsequent visual attention and user orientation. Generating accurate PoG trajectories into the future could also have interesting use cases when it comes to context aware interaction.

2 Related Work

In this section we will discuss related work that has had a notable influence on the direction of this project.

2.1 Visual attention prediction

(Fan et al. 2017) is probably the most notable contribution for visual attention prediction while using head-mounted displays (HMDs). It presents a RNN realised with long short term memory (LSTM) cells that can accurately predict visual fixation while watching 360 videos in a HMD. It uses pre-computed image saliency maps, motion maps and tracks the user's current camera orientation to predict visual fixation for m timesteps into the future using n past timesteps. While the EDVAM dataset does not contain images and so image saliency maps and motion maps cannot be computed, we would still hope to acheive a similar level of perfomance as we are defining the task at hand in a very similar way.

2.2 Seq2seq models

Seq2seq models are a class of generative RNNs (Graves 2013) that learn the following conditional probability,

$$p(y|x) : y* = \arg\max_{y} p(y|x, \theta)$$

In other words we want to learn the most probable output sequence y given some source input x, that need not be the same length as y. This abstraction is quite straight forward for machine translation tasks; for example, we want

to translate an English sentence x to the corresponding French sequence y.

It is not difficult to see that seq2seq models can also be applied to our formulation of visual attention prediction. From a given sequence x of n past frames we want to predict the most likely sequence y of m frames into the future and use this as a prediction for subsequent visual attention and user orientation.

2.3 Encoder-Decoder Framework

The encoder-decoder model architecture (Sutskever et al. 2014) is a popular way of realising learning models for sequence-to-sequence tasks. The simplest way to implement this model is with two RNNs, one for the encoder and one for the decoder. The encoder processes the source x step by step and outputs a context vector, which can be thought of as a compressed representation of the input. The context vector is then passed as input to the decoder along with an input token, the decoder then outputs a predicted sequence y of possibly arbitrary length. To realise this, we typically consider last hidden RNN state of the encoder to be the context vector and we use it as the initial state for the decoder RNN.

In the context of our problem we pass a sequence x of n past frames through the encoder, we then pass the context vector as the initial decoder state and start the decoding process by passing as input the last frame of our input sequence x_{n-1}. The decoder then generates a predicted sequence of frames $\hat{y} = \hat{y}_0, \hat{y}_1, ..., \hat{y}_{m-1}$. Figure 1 illustrates this process.

2.4 Training

These models are trained as one would expect - to maximise the probability of generating a target sequence y given the source sequence x.

However, this isn't so straight forward, Figure 1 illustrates how the decoder generates sequences into the future - it takes the outputs of the previous decoder cell and passes it as input for the the next timestep (*auto-regressive* mode). If the decoder has poor performance then errors can quickly accumulate during the decoding process and as a result training can be slow and suffer from non-convergence.

A popular method of dealing with this issue is called *teacher forcing* (Williams & Zipser 1989). With *teacher forcing* we pass the ground truth targets as input to the decoder cells at each timestep instead of the decoder outputs during training. Typically this helps the model converge and learn token-to-token or frame-to-frame dependencies well, however models trained purely with *teacher forcing* do not often exhibit good sequence generation. As a result we will look at some more advanced learning methods such as *scheduled sampling* (Bengio et al. 2015) and *professor forcing* (Lamb et al. 2016) in this paper.

3 Simple Model

First we will look at a simpler formulation of the problem, so as to validate our approach. Instead of learning to generate sequences of frames into the future we want to just simply predict the next frame. We will train a model to learn the following conditional probability,

$$p(y|x_0, x_1, ..., x_{n-1}) : y* = \arg\max_y p(y|x_0, x_1, ..., x_{n-1}, \theta)$$

In other words we want to learn the most probable frame $y \in \mathbb{R}^9$ given a sequence of n past frames $x = x_0, x_1, ..., x_{n-1}$, where all $x_t \in \mathbb{R}^9$.

We realise this recurrent model using 2 layers of LSTM cells with hidden size set to 100 and no dropout. A single fully connected layer maps the last LSTM output to 9 real valued outputs which correspond to the next frame prediction $\hat{y} \in \mathbb{R}^9$. We train on examples that consist of 300 frames of input (10 seconds of user information) and 1 target frame. Using mini-batch gradient descent we train the model to minimse the mean-squared-error (MSE) loss between the predicted frame $\hat{y}$ and the single ground truth target frame y. Table 1 presents the regression metrics and error scores achieved for next frame prediction using this model.

Feature	MSE	MAE	R2
PoG X	0.71393	0.38115	0.96550
PoG Y	0.11824	0.15726	0.87537
PoG Z	0.26983	0.24346	0.97017
Cam X	0.00369	0.04344	0.99965
Cam Y	0.00003	0.00112	0.91226
Cam Z	0.00160	0.02673	0.99954
Vec X	0.00053	0.01360	0.99911
Vec Y	0.00019	0.00510	0.99377
Vec Z	0.00050	0.01218	0.99868

Table 1: Regression metrics for next frame prediction

While it is clear that this model exhibits excellent next frame prediction performance, this is not the whole story. Firstly, next frame prediction is not really that interesting, yes we can very accurately predict the user's visual attention and camera configuration in 0.033 seconds time, but what about in 3 seconds time? Using *auto-regressive* sampling we observed that this model almost spectacularly

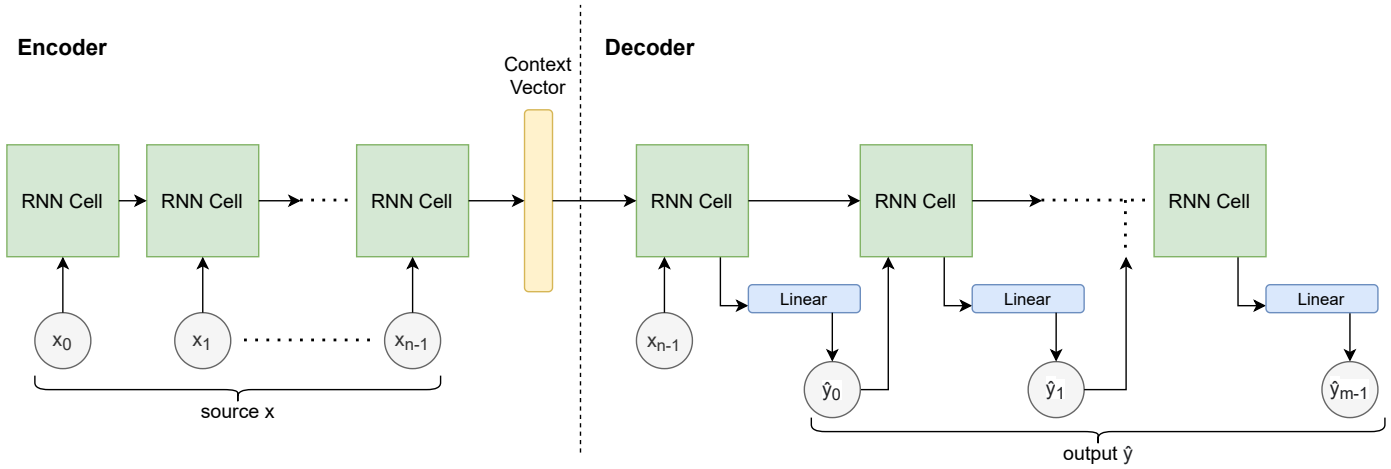

Figure 1: Encoder-decoder framework

fails at predicting much further than 5 frames into the future, which is not really a surprise because we didn't train it to do anything other than predict the next frame. Refer to Table 2 in Appendix A for the *auto-regressive* prediction metrics and errors.

In the following section we will apply the encoder-decoder architecure to create a model that has better generative properties and as a result it acheives a more accuracte prediction of subsequent visual attention and user orientation further into the future.

4 Seq2seq Model

We will now return to the original problem - predicting a sequence $\hat{y}$ of m frames into the future given some source x of n past frames.

For the following experiments we will use the encoder-decoder framework mentioned in Section 2, and we will introduce a new loss function called *frame loss* which proved to be more effective than MSE loss in our experiments. *Frame loss* is computed from a frame (9-tuple) of target values y_t and predicted values $\hat{y_t}$ as follows,

$$\begin{aligned} &\textit{frame loss}(y_t, \hat{y_t}) = \\ &\sqrt{(\mathrm{g}_{x_t} - \hat{\mathrm{g}_{x_t}})^2 + (\mathrm{g}_{y_t} - \hat{\mathrm{g}_{y_t}})^2 + (\mathrm{g}_{z_t} - \hat{\mathrm{g}_{z_t}})^2} + \\ &\sqrt{(\mathrm{c}_{x_t} - \hat{\mathrm{c}_{x_t}})^2 + (\mathrm{c}_{y_t} - \hat{\mathrm{c}_{y_t}})^2 + (\mathrm{c}_{z_t} - \hat{\mathrm{c}_{z_t}})^2} + \\ &\sqrt{(\mathrm{v}_{x_t} - \hat{\mathrm{v}_{x_t}})^2 + (\mathrm{v}_{y_t} - \hat{\mathrm{v}_{y_t}})^2 + (\mathrm{v}_{z_t} - \hat{\mathrm{v}_{z_t}})^2} \end{aligned} \quad (1)$$

where any frame y_t at timestep t is 9-tuple of real values for the 9 features $\langle g_{x_t}, g_{y_t}, g_{z_t}, c_{x_t}, c_{y_t}, c_{z_t}, v_{x_t}, v_{y_t}, v_{z_t} \rangle$ which correspond to the 3D PoG (x, y, z), 3D camera positions (x, y, z) and 3D camera orientation vectors (x, y, z) respectively.

This loss can be thought of as the 3D error between predicted 3D coordinates and ground truth 3D coordinates for each frame, and during training we sum them accross a time sequence of length m,

$$\text{loss seq} = \sum_{t}^{m} \textit{frame loss}(y_t, \hat{y_t})$$

4.1 Training with Scheduled Sampling

Scheduled sampling first introduced by (Bengio et al. 2015) is a curriculum learning approach for training seq2seq RNNs. It works by changing the decoding strategy between *teacher forcing* mode, where ground truth targets are supplied at each timestep into the future, and *auto-regressive* mode, where the model uses its own output as input at each timestep while decoding the context vector. Figure 2 illustrates the decoder's behaviour in both of the different modes.

Typically, during training we decode sequences in *teacher forcing* mode with probability ϵ and in *auto regressive* mode with probability $(1-\epsilon)$. For each mini-batch we calculate the loss and update the model in the same way regardless of whether a sequence was decoded in *teacher forcing* or *auto regressive* mode, and we anneal ϵ from 1 to 0 over time.

We used a slightly simplified version of *scheduled sampling* in our experiments and instead of annealing ϵ over time we set it to 0.2 for the entire duration of training.

Once again we trained the model using mini-batch gradient descent, except this time we calculate and backpropogate the *frame loss* over the entire sequence of predicted output frames. To realise this we construct examples from the EDVAM raw subset that consist of 300 input frames (10 seconds of user information) and 90 target frames

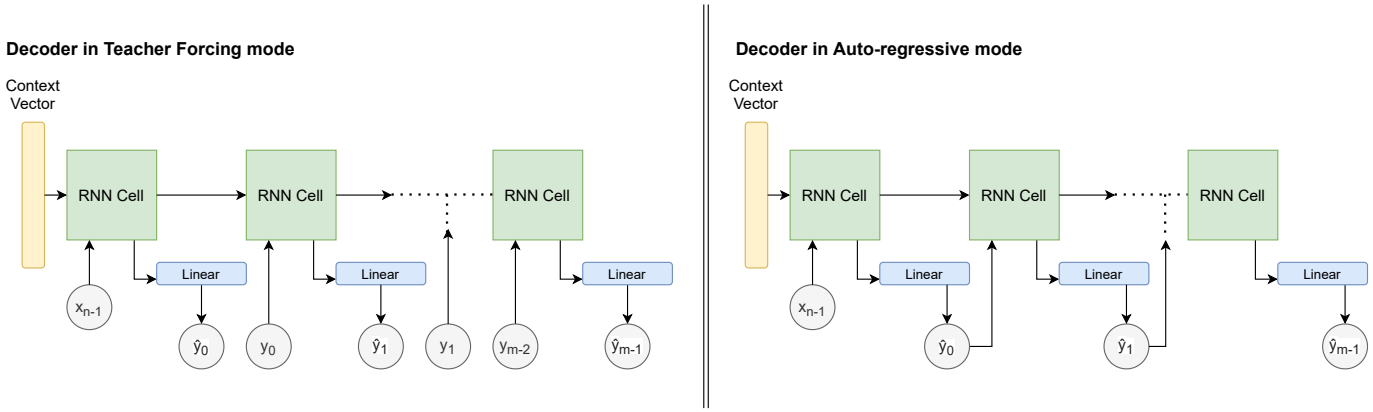

Figure 2: Decoding modes

(3 seconds into the future). We used 2 layers of Gated-recurrent units (GRUs) with hidden size set to 100 in the encoder, and we used just one GRU cell with hidden size set to 100 followed by a single fully conected layer for the decoder (we also used dropout of 0.4 for both the encoder and decoder).

The best validation (MSE) loss we acheived while training with *scheduled sampling* was 0.273577; note that when calculating this loss all 9 features were scaled to the standard normal distribution, the model was set to *auto-regressive* mode and the error was averaged over the entire generated output for every example in the validation set.

We achieved very similar results training just with *teacher forcing*; refer to Table 2 & 3 in Appendix A for the error scores and regression metrics for each of the 9 features in their original dynamic range.

4.2 Training with Professor Forcing

Professor forcing is a different method for training seq2seq RNNs based on the generative adversarial networks (GAN) framework (Goodfellow et al. 2014). Seq2seq RNNs are generative models and ideally we want sequences generated in *teacher forcing* mode to be indistinguishable from sequences generated in *auto-regressive* mode, so that at run time we can generate accurate sequences into the future without the need for ground truth targets. *Professor forcing* introduces this additional objective: make sure *auto-regressive* sequences look like *teacher forced* sequences.

So, in addition to training a generative encoder-decoder RNN, we train an additional discriminator model that will process output sequences and decide whether they have been produced in *teacher forcing* mode or in *auto-regressive* mode.

For the generator we realise the same encoder-decoder model, we used 2 layers of GRUs and set the hidden size to 512 for the encoder, and we used one GRU cell with hidden size set to 512 for the decoder (dropout was not used in either the encoder or decoder).

For the discriminator we process the generated sequences forward and backward through time using two RNNs (or bidirectional) implemented with 2 layers of GRUs (hidden size set to 512). After processing the generated sequence we concatenate the hidden states of the two RNNs and pass them through a Multi-layer preceptron (MLP) fitted with a sigmoid output layer (for binary classification). In our experiments we actually used the conditional GAN setup (Mirza & Osindero 2014), which means we provide the discriminator with the ground truth targets and we train the discriminator to distinguish between outputs generated in *teacher forcing* mode and *auto-regressive* mode given the ground truth sequence.

The training scheme is as follows:

1. Initialize the generator network G with parameters θ_g and the discriminator network D with parameters θ_d to random values.
2. Fetch a batch b of training data from the dataset and pass it through the generator in *teacher forcing* mode and then in *auto-regressive* mode, giving us two batches of generated sequences b' and $\hat{b}$.
3. Pass the *teacher forced* batch b' to the discriminator to get the *teacher forcing* probabilities.
4. Pass the *auto-regressive* batch $\hat{b}$ to the discriminator to get the *auto-regressive* probabilities.
5. If the discriminator accuracy is less than 99% back-propogate the conditional classification loss,

$$C(\theta_d \,|\, \theta_g) = \mathrm{E}_{(x,y)\sim data}[-\log D(G(x, y, \theta_g), \theta_d \,|\, y) - \log(1 - D(G(x, \theta_g), \theta_d \,|\, y))] \quad (2)$$

   and update the parameters of the discriminator θ_d.
6. Using the same batch b pass it through the generator again in *teacher forcing* mode and in *auto-regressive* mode to get b' and $\hat{b}$ respectively.

7. Pass the *teacher forced* batch b' to the updated discriminator to get the *teacher forcing* probabilities.
8. Pass the *auto-regressive* batch $\hat{b}$ to the updated discriminator to get the *auto-regressive* probabilities.
9. If the updated discriminator accuracy is greater than 75% accumulate the following loss to the generator only,

$$C_f(\theta_d \,|\, \theta_g) = \mathrm{E}_{(x,y)\sim data}[-\log D(G(x, \theta_g), \theta_d \,|\, y)] \tag{3}$$

this loss encourages the generator to match *auto-regressive* outputs to *teacher forced* outputs.
10. Backpropogate the frame loss for both the *teacher forced* batch b' and *auto-regressive* batch $\hat{b}$ to the generator,

$$\text{batch loss} = \sum_{i}^{|\hat{b}|} \sum_{t}^{m} \textit{frame loss}(y_t, \hat{y_t}) + \sum_{i}^{|b'|} \sum_{t}^{m} \textit{frame loss}(y_t, \hat{y_t}) \tag{4}$$

11. Finally, update the generator parameters θ_g w.r.t the accumulated loss.
12. Repeat from 2. until convergence.

With *professor forcing* we acheived a better validation (MSE) loss of 0.231359 during training and so validating this approach. Refer again Table 3 in Appendix A for the regression metrics and error scores for each of the 9 features.

5 Remarks

While we acheived better validation loss scores with *professor forcing* this is unfortunately not the whole story. Figure 3 in Appendix B compares the sequence generation capability of encoder-decoder model trained with *professor forcing* in both *teacher forcing* mode and *auto-regressive* mode.

It is evident that the *teacher forced* sequence is close to the ground truth - this is no surprise because we supply the model with the ground truth at each timestep. It is also clear to see that the *auto-regressive* behaviour of the model is far from the *teach forced* behaviour in terms of matching to the ground truth sequence.

In Figure 3 we see that the *auto-regressive* sequence captures the general downward trend of the PoG X feature, however it fails to capture the shorter term deviations and so matches to the ground truth sequence poorly. This is in general the case for most of the features in the test sequences we inspected after training. As a result it makes the discriminator's job very easy and it is no surprise that we observed high discriminator accuracy during training. There could be a number of reasons as to why the generator model fails to match its *auto-regressive* behaviour to its *teach forced* behaviour well:

- The problem is not very learnable, either because there is too much variance in the dataset or we require additional data such as image saliency maps or motion maps to capture all the patterns in the data.
- The adversarial training was not scheduled optimally so the generator model converged to a sub-optimal parameter space.
- The generator lacked the capacity to learn all the shorter-term patterns in the data.
- The recurrent encoder-decoder architecture is too naive of a model to learn all the patterns in the data accurately.

It is important to note that computing image saliency maps using pre-trained convolutional neural networks (CNNs) is not always feasible in real time, escpecially for VR systems. So, while (Fan et al. 2017) had some success using pre-computed image saliency maps for visual attention prediction, we may not be able to do the same thing depending on the potential applications for the system we are trying to build.

To conclude, while we have shown that RNNs can acheive excellent next frame prediction for this formulation of the visual attention problem, it remains unclear if they are able to fully capture visual attention patterns and exhibit good generative behaviours for generating visual attention and user orientation sequences into the future. Our best generative models appeared to fit a straight line of best fit to future trajectories and failed to learn short-term patterns. Potential improvements could be made by revisiting the formulation of the problem or by using more sophisticated models and training approaches.

References

Bengio, S., Vinyals, O., Jaitly, N. & Shazeer, N. (2015), 'Scheduled sampling for sequence prediction with recurrent neural networks', *arXiv preprint arXiv:1506.03099* .

Fan, C.-L., Lee, J., Lo, W.-C., Huang, C.-Y., Chen, K.-T. & Hsu, C.-H. (2017), Fixation prediction for 360 video streaming in head-mounted virtual reality, *in* 'Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video', pp. 67–72.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014), 'Generative adversarial nets', *Advances in neural information processing systems* **27**.

Graves, A. (2013), 'Generating sequences with recurrent neural networks', *arXiv preprint arXiv:1308.0850* .

Lamb, A. M., Goyal, A. G. A. P., Zhang, Y., Zhang, S., Courville, A. C. & Bengio, Y. (2016), Professor forcing: A new algorithm for training recurrent networks, *in* 'Advances in neural information processing systems', pp. 4601–4609.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J. & Khudanpur, S. (2010), Recurrent neural network based language model, *in* 'Eleventh annual conference of the international speech communication association'.

Mirza, M. & Osindero, S. (2014), 'Conditional generative adversarial nets', *arXiv preprint arXiv:1411.1784* .

Sutskever, I., Vinyals, O. & Le, Q. V. (2014), Sequence to sequence learning with neural networks, *in* 'Advances in neural information processing systems', pp. 3104–3112.

Vinyals, O., Toshev, A., Bengio, S. & Erhan, D. (2015), Show and tell: A neural image caption generator, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 3156–3164.

Williams, R. J. & Zipser, D. (1989), 'A learning algorithm for continually running fully recurrent neural networks', *Neural computation* **1**(2), 270–280.

Yang, C., Guo, Z. & Xian, L. (2019), Time series data prediction based on sequence to sequence model, *in* 'IOP Conference Series: Materials Science and Engineering', Vol. 692, IOP Publishing, p. 012047.

Zhou, Y., Tian, F., Shihui, S., Ziangdong, L., Sun, L. & Been-Lirn, D. H. (2021), 'Edvam: a 3d eye-tracking dataset for visual attention modeling in a virtual museum', *http://www.jzus.zju.edu.cn/article.php?doi=10.1631/FITEE.2000318* .

Appendices

Appendix A

	Simple Model						**Teacher Forcing**					
	1 future			*90 futures*			*1 future*			*90 futures*		
	MSE	**MAE**	**R2**	**MSE**	**MAE**	**R2**	**MSE**	**MAE**	**R2**	**MSE**	**MAE**	**R2**
PoG X	0.71393	0.38115	0.96550	36.36271	4.68496	0.28087	0.79100	0.24264	0.96280	10.02092	2.20432	0.61959
PoG Y	0.11824	0.15726	0.87537	1.16946	0.84612	0.15714	0.13518	0.11601	0.86472	0.77399	0.61757	0.30546
PoG Z	0.26983	0.24346	0.97017	10.18285	2.40818	0.32790	0.30782	0.15999	0.96705	3.89262	1.34779	0.63324
Cam X	0.00369	0.04344	0.99965	15.45370	3.30538	0.42252	0.00105	0.01384	0.99988	1.34856	0.80675	0.86356
Cam Y	0.00003	0.00112	0.91226	0.00849	0.06719	0.02963	0.00002	0.00028	0.91427	0.00057	0.01237	0.09974
Cam Z	0.00160	0.02673	0.99954	4.07758	1.58318	0.35525	0.00080	0.01058	0.99976	0.62672	0.54950	0.82430
Vec X	0.00053	0.01360	0.99911	0.65219	0.61175	0.25755	0.00039	0.00505	0.99928	0.16601	0.25263	0.72873
Vec Y	0.00019	0.00510	0.99377	0.10971	0.26524	0.21860	0.00017	0.00201	0.99410	0.02126	0.08875	0.42832
Vec Z	0.00050	0.01218	0.99868	0.34632	0.44144	0.31026	0.00044	0.00518	0.99881	0.15295	0.25822	0.61867

Table 2: Regression metrics and error scores for the simple model and *teacher forcing*

	Scheduled Sampling						**Professor Forcing**					
	1 future			*90 futures*			*1 future*			*90 futures*		
	MSE	**MAE**	**R2**	**MSE**	**MAE**	**R2**	**MSE**	**MAE**	**R2**	**MSE**	**MAE**	**R2**
PoG X	0.82850	0.41217	0.96165	8.99115	2.22310	0.58230	0.89974	0.53740	0.95700	8.03130	2.11479	0.62385
PoG Y	0.13188	0.13743	0.86570	0.80661	0.70800	0.20737	0.13251	0.20111	0.86096	0.69279	0.63884	0.30605
PoG Z	0.36165	0.37851	0.96460	3.95664	1.43894	0.60876	0.33008	0.32583	0.96343	3.16614	1.26286	0.66246
Cam X	0.00744	0.06671	0.99919	1.30474	0.83543	0.85763	0.01682	0.09562	0.99831	1.05169	0.71455	0.88465
Cam Y	0.00003	0.00147	0.90637	0.00032	0.00729	0.08111	0.00003	0.00173	0.90427	0.00038	0.00862	0.10322
Cam Z	0.00410	0.04709	0.99904	0.69338	0.58957	0.80520	0.00923	0.06815	0.99763	0.55462	0.52854	0.84018
Vec X	0.00134	0.02698	0.99804	0.20458	0.28267	0.67457	0.00190	0.03027	0.99734	0.15549	0.25616	0.73884
Vec Y	0.00021	0.00634	0.99321	0.01977	0.09600	0.39605	0.00021	0.00656	0.99287	0.01765	0.08791	0.41848
Vec Z	0.00179	0.03120	0.99687	0.16167	0.27768	0.61984	0.00185	0.02918	0.99528	0.12908	0.24597	0.66618

Table 3: Regression metrics and error scores for *scheduled sampling* and *professor forcing*

Appendix B

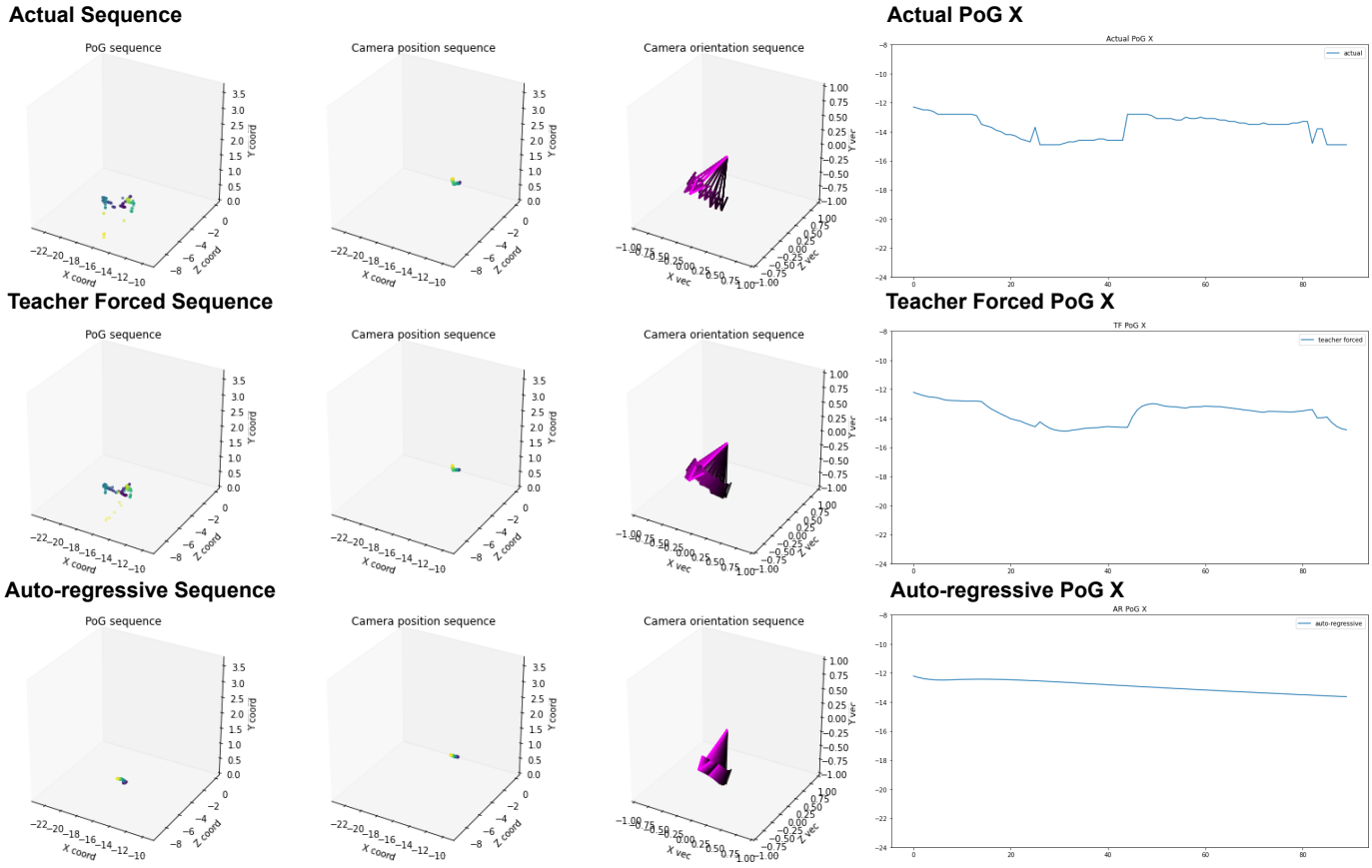

Figure 3: Sequence Generation