Cells2Vec: Bridging the gap between experiments and simulations using causal representation learning

Dhruva Abhijit Rajwade[1]
rajwadedhruva@gmail.com

Atiyeh Ahmadi[2]
atiyeh.ahmadi@uwaterloo.ca

Brian Ingalls[3]
*bingalls@uwaterloo.ca

Abstract

Calibration of computational simulations of biological dynamics against experimental observations is often a challenge. In particular, the selection of features that can be used to construct a goodness-of-fit function for agent-based models of spatiotemporal behaviour can be difficult (Yip et al. (2022)). In this study, we generate one-dimensional embeddings of high-dimensional simulation outputs using causal dilated convolutions for encoding and a triplet loss-based training strategy. We verify the robustness of the trained encoder using simulations generated by unseen input parameter sets. Furthermore, we use the generated embeddings to estimate the parameters of simulations using XGBoost Regression. We demonstrate the results of parameter estimation for corresponding time-series real-world experimental observations, identifying a causal relationship between simulation-specific input parameters and real-world experiments. Our regression approach is able to estimate simulation parameters with an average R^2 metric of 0.90 for model runs with embedding dimensions of 4,8,12 and 16. Model calibration led to simulations with an average cosine similarity agreement of 0.95 and an average normalized Euclidean similarity of 0.69 with real-world experiments over multiple model runs.

1 Introduction

We investigate the learning of fixed-size representations from high-dimensional time-series data collected from bacterial population growth simulations. These simulations, produced by an agent-based model, generate time series with variable lengths. The representations learned from these simulations can be used to calibrate model behaviour against corresponding experimental data sets. In our approach, the agent-based models serve as computational frameworks where individual agents representing a bacterial cell interact within a defined environment. This modeling is particularly effective for simulating complex biological systems, such as bacterial colonies, as it captures the dynamic interactions and behaviours at the cellular level. Our simulations begin with a single-cell agent. This agent grows and divides over time, leading to the development of a microcolony composed of a few hundred cells, all within a monolayer. During this growth, cells experience forces due to their own growth and the shoving from neighbouring growing cells.

'Calibration' for our pipeline involves fine-tuning the parameters of our agent-based model to ensure that the outputs of our simulations align closely with the behaviours observed in real-world experimental data on bacterial growth. This includes adjusting growth rates, division times, etc. Through this calibration process, we aim to enhance the model's accuracy and predictive power, thereby ensuring that the learned representations from these simulations are reliably indicative of actual bacterial population dynamics. Cellular adhesion also plays a crucial role in determining colony structure. These interactions, along with energy costs of cell-cell overlaps, influence the overall spatial distribution within the colony.

*[1]Department of Biotechnology, Indian Institute of Technology Kharagpur, [2]Department of Biology, University of Waterloo, [3]Department of Applied Mathematics, University of Waterloo

Accepted to the NeurIPS 2023 Workshop on Causal Representation Learning.

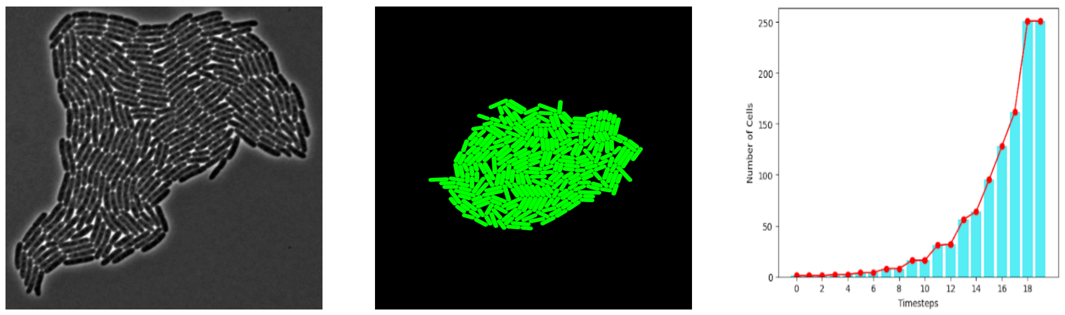


Figure 1: (a) Image taken the last timestep of a sample real-life experiment (b) Image generated after the last timestep of a sample simulation (c) Cells vs timesteps (for a sample simulation)

We take inspiration from Cess & Finley (2022), which describes the use of deep learning to represent on-lattice agent-based model simulations that align pixel-by-pixel to experimental imaging observations.

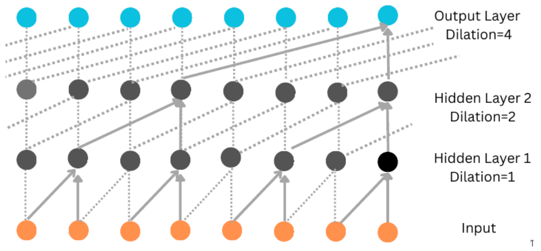
Simulations are valuable tools for exploring the behaviour of complex systems, which often do not produce easily reproducible outcomes. We view each experiment and corresponding simulation as samples from an underlying distribution of biological behaviours. Drawing from these patterns, we develop a model to capture the 'true' essence of cellular actions. This model can then be a compass for comparing and interpreting real-world experiments. However, challenges arise when we consider the application of these models. Ensuring they predict accurately in unfamiliar scenarios and synchronizing the vast data from real-world experiments with the more condensed outputs of our simulations are hurdles we aim to overcome. To tackle this, we refine our models to deliver outputs that align with our experimental observations. We also note that our data(simulations and real-world experiments) grows with time; that is, at every timestep, the number of cells increases due to cell division (Figure 1(c)), and along with it, the observation size for each consecutive timestep. We tackle this 'evolving' time-series data by using causal convolutions, which are, in turn, stacked and dilated to increase the receptive field of the overall model and capture fixed-size embeddings from simulations with a wide range of sequence lengths.

2 Related Work

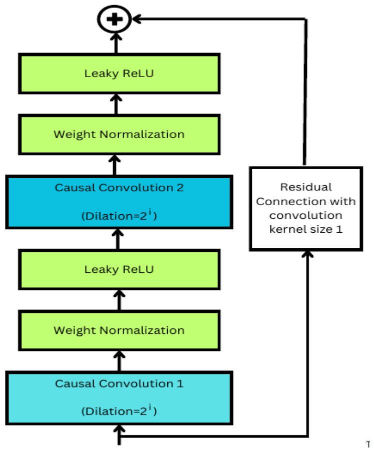
The topic of calibrating computer-generated simulations, including agent-based models, with real-world experiments has been well-studied. Traditional methods for parameter calibration include sensitivity testing (Gutenkunst et al. (2007)) and Bayesian calibration (Kennedy & O'Hagan (2001), McCulloch et al. (2022)). Machine learning has been employed to build surrogate models to achieve efficient exploration of high-dimensional parameter spaces (Lamperti et al. (2017)).

In the context of Representation Learning and Biological data, Soelistyo et al. (2022) use variational autoencoders with a temporal component (τ-VAE) to learn representations of cells across time from time-lapse microscopy images and consequently predict whether a cell is subject to mitosis, apoptosis or other cell fates. Szubert et al. (2019) use Siamese networks for dimensionality reduction of single-cell expression data for interpretation and visualization. Cess & Finley (2022) use three disparate computational models to learn representations in a contrastive fashion. They use these representations to compare different model outputs. One of the model types they use for their analyses is an agent-based model of tumour-immune interactions. They treat this data as images, leveraging the fact that the simulation dimensions resemble the dimensions of a multi-channel image, using the approach introduced by Chen et al. (2020) to learn representations in a contrastive fashion. Yip et al. (2022) provide a comprehensive review of traditional methods for agent-based model calibration.

Our goal is to learn representations from *one dimensional* multivariate time series data extracted from time series microscopy simulations and real-world experiments of growing E.coli microcolonies. Further, unlike Cess & Finley (2022), we do not have a clear method to augment our data for training using the SimCLR (Chen et al. (2020)) approach. Hence, we focus on introducing similarity through stochasticity, which is inherent in our agent-based model simulations.



(a)



(b)

Figure 2: (a) Illustration of three stacked dilated causal convolutions, figure adapted from van den Oord et al. (2016) (b) Composition of the i^{th} layer of the Encoder, figure adapted from Franceschi et al. (2019), each layer consists of two stacked *Conv-Weight Norm-Leaky ReLU* blocks

3 Methods

3.1 Data

3.1.1 Simulations

For this study, we use the *Cellmodeller* (Rudge et al. (2012)) agent-based modelling package to generate simulations of microcolony formation from a single cell. Cell morphology parameters are set to means of observed values. The model parameters to be inferred are gamma, which quantifies cell rigidity, and Reg_param, which characterizes the energy cost of cell-cell overlap (with higher values resulting in more stringent resolution of overlaps). We highlight again the dynamic nature of our data over consecutive timesteps (with each timestep, our cells divide and increase in number).

We selected 100 parameter pairs (gamma, Reg_param) from a uniform distribution (sampling details in the Supplement). We then produced 10 simulations for each parameter pair, resulting in a dataset with a total of 1000 simulations. Note that the simulations are stochastic in nature, meaning iterations of the same parameter set are not identical but exhibit similar behaviour. Each simulation began with the same initial condition: a single cell of fixed length. Figures 1(b) and 1(c) show a sample simulation snapshot at the last time point and the distribution of cells with time, respectively.

3.1.2 Real-World Experiments

Overnight cultures of *E. coli* MG1655 were diluted in LB Miller medium and incubated until an OD600 of 0.4-0.6 was reached. Following protocols from Young et al. (2012), samples were diluted, and 2.25 uL was placed on LB pads with 1.5% agarose. After 5 minutes, the pads were sealed in a 50 mm coverslip dish. They were then imaged at 37°C in a GeneFrame chamber on a Zeiss Axio Observer microscope. Using a 63x oil objective, images were taken every 3 minutes at various locations, with intensity and exposure set at 50% and 150 ms, respectively. We then processed the images using Ilastik (Berg et al. (2019)), followed by CellProfiler (McQuin et al. (2018)), and a custom-written package (CellProfilerAnalysis v1.0.0). The outputs from these tools provide a representation that mirrors the ABM, capturing a low-dimensional portrayal of each cell, which includes details such as frame number, ID, centroid, orientation, length, and parent ID. Figure 1(a) shows a sample experiment snapshot taken at the last time-point of the experiment.

3.1.3 Data Sampling

Our goal was to learn meaningful representations from simulations in a supervised fashion. We employed a Triplet Loss (Hoffer & Ailon (2014)) based training strategy for our encoder. We refer to

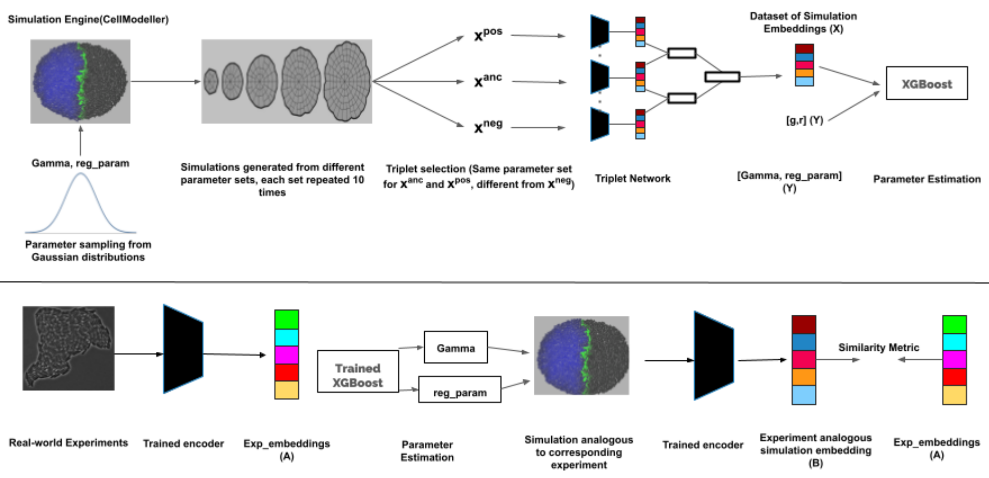

Simulation Engine(CellModeller)
Gamma, reg_param
Parameter sampling from Gaussian distributions
Simulations generated from different parameter sets, each set repeated 10 times
x^{pos}
x^{anc}
x^{neg}
Triplet selection (Same parameter set for x^{anc} and x^{pos}, different from x^{neg})
Triplet Network
Dataset of Simulation Embeddings (X)
[g,r] (Y)
[Gamma, reg_param] (Y)
XGBoost
Parameter Estimation
Real-world Experiments
Trained encoder
Exp_embeddings (A)
Trained XGBoost
Gamma
reg_param
Parameter Estimation
Simulation analogous to corresponding experiment
Trained encoder
Experiment analogous simulation embedding (B)
Similarity Metric
Exp_embeddings (A)

Figure 3: Flowchart of our Pipeline

Algorithm 1 Algorithm to sample triplets X^{pos}, X^{anc}, and X^{neg} from a dataset

```
Require: Dataset $D$, Number of triplets $N$
    $n \leftarrow 0$
    while $n < N$ do
        Randomly select a class indices $i$ and l from $D$ such that $i \neq l$
        Randomly select an anchor example index $j$ from $D[i]$
        Randomly select a positive example index $k$ such that $k \neq j$ from $D[i]$
        Randomly select a negative sample index $m$ from $D[l]$
        $X^{pos}[n] \leftarrow D[i][j]$ , $X^{anc}[n] \leftarrow D[i][k]$ ,$X^{neg}[n] \leftarrow D[l][m]$
        $n \leftarrow n + 1$
    end while
    return $X^{pos}$, $X^{anc}$, $X^{neg}$
```

each unique parametrization of the simulation model as a 'class'. We start with a training dataset containing 1000 simulations. We remove all simulations for 5 randomly selected classes from the training dataset and use these to evaluate our trained encoder to check how robust our encoder is at generalizing unseen classes. We call this the testing dataset. From the training dataset, we also remove 3 simulations from each class (out of 10 simulations per class) and use them to form a validation dataset to be used to prevent our model from overfitting. We construct triplets by selecting two simulations X^{anc} and X^{+} from a class , and a simulation X^{-} from a different class. We randomly sampled 10000 triplets from our training dataset and 2500 from our validation dataset. For each training epoch, we pass the 10000 triplets to our encoder, backpropagate and compute loss for the validation dataset of 2500 triplets to check for overfitting. The pseudo-code for the process of triplet generation can be found in Algorithm 1.

3.2 Encoder Architecture

For the base architecture for our encoder, we used deep neural networks with exponentially dilated causal convolution, as proposed by Franceschi et al. (2019). The model is based on stacks of dilated causal convolutions, which map a given sequence to a sequence of the same length, such that the ith element of the output sequence is determined using only values up until the ith element of the input sequence. Each layer of our network combines causal convolutions, weight normalizations (Salimans & Kingma (2016)), leaky ReLUs and residual connections. Each of these layers is given an exponentially increasing dilation parameter. The output of this stacked network is transformed into a fixed-size vector using a Global Max Pooling layer, which aggregates all of the temporal information. This vector is further used for a linear transformation to produce the final embedding of our network. We create a triplet network (Hoffer & Ailon (2014)) using the base encoder described in Figures 2(a) and 2(b). Our simulations are growing with time, i.e. different time steps contain different numbers

Table 1: R^2 Score and RMSE for Different Embeddings

Embedding Size	R^2 Score		RMSE	
	Original	New	Original	New
4	0.9831	0.9565	17.3402	35.1357
8	0.9877	**0.9638**	**7.8902**	21.0414
12	0.9786	0.9570	8.69	19.27
16	**0.9889**	0.9517	9.0044	**13.2809**

Table 2: ARI and AMI averaged over 10 runs

Embedding Size	ARI	AMI
4	0.882	0.926
8	**0.950**	**0.934**
12	0.857	0.902
16	0.898	0.930

Table 3: Experiment vs Simulation Similarity

Embedding Size	Similarity Scores	
	Cosine	Euclidean
4	0.948	0.661
8	0.984	**0.743**
12	0.963	0.618
16	**0.992**	0.732

of cells. This provided more motivation to use a sequence length invariant model consisting of 1D convolutions rather than traditional LSTMs or RNNs.

3.3 Supervised Training

We aim to train an encoder-only architecture without the use of a decoder. We choose a vanilla Triplet Loss, first introduced by Schroff et al. (2015) for training our Encoder. As introduced by Hoffer & Ailon (2014), we use a Triplet Network consisting of three Encoder units with shared weights. Let X^{anc} represent an anchor sample, X^{pos} represent a positive sample (same class as the anchor), and X^{neg} represent a negative sample (different class from the anchor). Triplet loss is based on the intuition that if two samples are similar (positive and anchor), their embeddings will be close in the embeddings space, but if they are not (negative and anchor), their embeddings will be far from one another. Let $f(.,\theta)$ denote an encoder that maps samples to a feature space. The objective to be minimized using Euclidean distance to quantify the magnitude of similarity, $L(X^{anc}, X^{pos}, X^{neg})$, where the embedding is represented by $f(x) \in \mathbb{R}^d$ and $f(.,\theta)$ encodes a simulation X into a d-dimensional Euclidean space:

$$\max\big(0, \|f(X^{anc},\theta) - f(X^{pos},\theta)\|^2 - \|f(X^{anc},\theta) - f(X^{neg},\theta)\|^2 + \alpha\big) \tag{1}$$

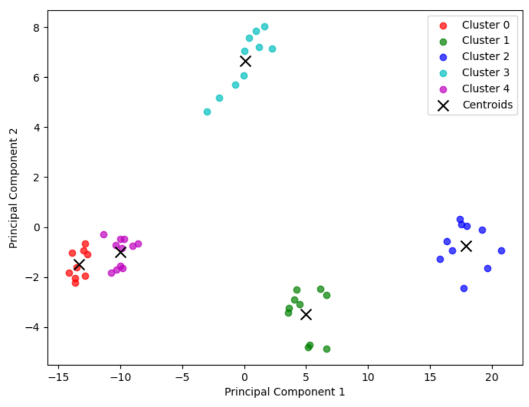
Where α is the margin parameter that enforces a minimum separation between classes.

Note, we *did not use padding* to normalize sequence lengths; the encoder is trained using the original uneven sequence lengths. To counter overfitting, we trained the model until convergence using a held-out validation set described in section 3.1.3. We fix a single hyperparameter set (except the final latent dimension)for all our training pipeline experiments. We carried out *no hyperparameter tuning*, as in Franceschi et al. (2019).

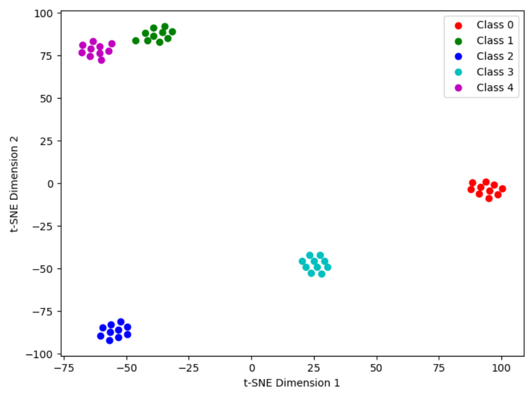
4 Results

Here, we present the results of assessments of the utility of the learned representations. We used Python 3 for the implementation with PyTorch. 1.13.0 (Paszke et al. (2019)) for neural networks and scikit-learn (Pedregosa et al. (2011)) for K-Means, TSNE and PCA analyses. [2] We also used XGBoost (Chen & Guestrin (2016)) for regression to estimate parameters. Each encoder was trained using the Adam optimizer (Kingma & Ba (2017)) on a single Nvidia T4 GPU with CUDA 11.0 unless

[2]The code for this work is available at https://github.com/ingallslab/Cells2Vec



(a)



(b)

Figure 4: (a) PCA+KMeans plot With cluster centres marked for embedding size 8, for unseen parameter set(s) simulations. (b) t-SNE scatter plot with perplexity 10 for embedding size 8 and unseen parameter set(s) simulations

stated otherwise. Opting for appropriate hyperparameters in supervised methods is challenging due to the prevalent nature of supervised downstream tasks. Hence, as proposed by Wu et al. (2018) and used by Franceschi et al. (2019), we select a single set of hyperparameters for all our analysis. Our complete pipeline can be understood through Figure 3.

4.1 Evaluating Quality Of Learned Representations

We trained our models until convergence or until an early-stopping condition was triggered (validation loss increasing in more than 5 epochs continuously). We evaluated our encoder using simulations from our evaluation dataset, the classes unseen by the model while training. We obtained embeddings for these simulations, clustered them using K-Means clustering, and computed Adjusted Mutual Information(AMI) and Adjusted Rand Index scores(ARI).

We further validated the results using a Principal Component Analysis in conjugation with a K-Means clustering analysis (Figure 3(a)) and a t-SNE dimensionality reduction (Figure 3(b)). For every embedding size, we trained and assessed 10 models, in each case randomly sampling 5 classes to be used for evaluation as described above. We present the 10-fold averaged metrics in Table 2. We find that the model with embedding size 8 is best able to cluster the dataset and that our model is able to generalize on parameter sets unseen while training. Overall, our ARI and AMI results show that our encoder is able to generalize between unseen classes and hence is able to learn meaningful representations from our simulations.

4.2 Parameter Estimation from Embeddings

We used XGBoost regression (Chen & Guestrin (2016)) to predict values of the two parameters (gamma and Reg_param). We use the embeddings generated from our trained encoder as inputs for the regression model and absolute parameter values as labels, with the task being estimating parameter values from low dimensional embeddings.

We present results for two Regression models, the first being passed the *original set* used to train the encoder, and the second being passed a *new set* of 300 simulations with randomly sampled input parameter values. We use a train: test split of [0.5:0.5] for both models. We present regression metrics (R^2 scores) for both models in Table 1.

4.3 Evaluating On Real-World Experiments

We obtain embeddings for data processed from 13 real-world experiments performed in-house. Using a standard pipeline, we obtain data in a format similar to simulations (number of cells, number of features). We use these embeddings to estimate 13 parameter sets, one for each experiment. Using these parameter sets, we generate simulations, which we expect to be analogous and similar to our experiments. This leads us to an interesting caveat: Our real-world experiments have, on average, 3500 timesteps, while our simulations have, on average, 1300 timesteps. To verify this, we pass our experiment-analogous simulations through our encoder to obtain embeddings for the simulations and compare these embeddings with the experiment embeddings to compute two similarity scores. We decided to use cosine similarity as our first similarity metric, which is magnitude invariant. We also normalize the embeddings and use them to compute Euclidean similarity scores. The cosine and Euclidean similarity values for different embedding sizes are presented in Table 2. We note that an encoder with tightly clustered outputs would also result in high cosine similarity. Hence, we also test another metric for a thorough validation. Our results (Table 3) indicate that our pipeline is successfully able to generate simulations analogous to experiments, as shown by the Euclidean and Cosine similarity values.

5 Discussion and Conclusion

Here, we present a pipeline to learn embeddings from simulations of cellular population dynamics. We used a trained encoder to infer simulation-specific parameters. This approach uses causal dilated convolutions to infer a single-dimensional embedding from high-dimensional multivariate simulations. The nature of these simulations, intricately woven with complex biological dynamics like cell orientation changes, shoving, and cellular adhesion, provides a detailed vision into the world of cell colonies.

To illustrate some applications of our encoder, we formulated a regression task and a simulation-experiment inference task. While prior works, e.g. Cess & Finley (2022), build towards using Representation Learning to compare complex model outputs, they involve leveraging simulation data as 2D images and augmenting these images to train an Encoder in a Siamese fashion. In contrast, our method uses only simulations to train an encoder to extract meaningful representations and uses this encoder directly on data from experiments to estimate simulation-specific parameters via regression. We evaluate our learned representations using methods such as K-Means clustering and principal component Analysis, using ARI and AMI as metrics, and visual validation through t-SNE analysis.

We estimated simulation-specific parameters from experiment embeddings using an XGBoost model trained using simulation embeddings and corresponding parameters as labels. Our parameter estimation methodology emerged successful at generating simulations aligned with experiments, as inferred through cosine similarity and normalized Euclidean similarity metrics. These results show that our pipeline was successful in capturing the causal relationship between parameters and simulations, as well as experiments, to a good extent. The results indicate that our formulation is a viable way to estimate simulation-specific parameters from experiments (through similarity scores, Table 3) and generate good-quality embeddings from temporal time series (through ARI and AMI scores, Table 2) for interpretation.

In the future, we will aim to extend our pipeline using a much larger dataset, which would cover a rich sampling of real-world parameterizations. We are facing some out-of-distribution errors, where our encoder fails to capture good quality representations from simulations with parameter values different than that of the training parameter distributions, and we hope to fix this by including all biologically plausible parameter sets in our training data. We also plan to implement different triplet-mining strategies to learn better quality embeddings. Additionally, the idea of regenerating the original length sequences from fixed-size embeddings is one we are actively pursuing, using a transformer-based or RNN-based sequence generation model.

Acknowledgments

We would like to acknowledge Aryan Satpathy for valuable comments, suggestions and discussions. We would also like to thank Aaron Yip for his suggestions and for providing Figure 1(a). We thank as well our anonymous reviewers for their constructive suggestions. We acknowledge financial support from the MITACS Globalink Program and from the Canadian Natural Sciences and Engineering Research Council (NSERC, RGPIN-2018-03826).

References

Berg, S., Kutra, D., Kroeger, T., Straehle, C. N., Kausler, B. X., Haubold, C., Schiegg, M., Ales, J., Beier, T., Rudy, M., et al. Ilastik: interactive machine learning for (bio) image analysis. *Nature methods*, 16(12):1226–1232, 2019.

Cess, C. G. and Finley, S. D. Representation learning for a generalized, quantitative comparison of complex model outputs, 2022.

Chen, T. and Guestrin, C. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. doi: 10.1145/2939672.2939785. URL `https://doi.org/10.1145%2F2939672.2939785`.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations, 2020.

Franceschi, J.-Y., Dieuleveut, A., and Jaggi, M. Unsupervised scalable representation learning for multivariate time series. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/53c6de78244e9f528eb3e1cda69699bb-Paper.pdf`.

Gutenkunst, R. N., Waterfall, J. J., Casey, F. P., Brown, K. S., Myers, C. R., and Sethna, J. P. Universally sloppy parameter sensitivities in systems biology models. *PLoS Computational Biology*, 3(10):e189, October 2007. doi: 10.1371/journal.pcbi.0030189. URL `https://doi.org/10.1371/journal.pcbi.0030189`.

Hoffer, E. and Ailon, N. Deep metric learning using triplet network, 2014.

Kennedy, M. C. and O'Hagan, A. Bayesian calibration of computer models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 63(3):425–464, September 2001. doi: 10.1111/1467-9868.00294. URL `https://doi.org/10.1111/1467-9868.00294`.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017.

Lamperti, F., Roventini, A., and Sani, A. Agent-based model calibration using machine learning surrogates, 2017.

McCulloch, J., Ge, J., Ward, J. A., Heppenstall, A., Polhill, J. G., and Malleson, N. Calibrating agent-based models using uncertainty quantification methods. *Journal of Artificial Societies and Social Simulation*, 25(2), 2022. doi: 10.18564/jasss.4791. URL `https://doi.org/10.18564/jasss.4791`.

McQuin, C., Goodman, A., Chernyshev, V., Kamentsky, L., Cimini, B. A., Karhohs, K. W., Doan, M., Ding, L., Rafelski, S. M., Thirstrup, D., et al. Cellprofiler 3.0: Next-generation image processing for biology. *PLoS biology*, 16(7):e2005970, 2018.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Rudge, T. J., Steiner, P. J., Phillips, A., and Haseloff, J. Computational modeling of synthetic microbial biofilms. *ACS Synthetic Biology*, 1(8):345–352, August 2012. doi: 10.1021/sb300031n. URL `https://doi.org/10.1021/sb300031n`.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks, 2016.

Schroff, F., Kalenichenko, D., and Philbin, J. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015. doi: 10.1109/cvpr.2015.7298682. URL `https://doi.org/10.1109%2Fcvpr.2015.7298682`.

Soelistyo, C. J., Vallardi, G., Charras, G., and Lowe, A. R. Learning biophysical determinants of cell fate with deep neural networks. *Nature Machine Intelligence*, 4(7):636–644, June 2022. doi: 10.1038/s42256-022-00503-6. URL `https://doi.org/10.1038/s42256-022-00503-6`.

Szubert, B., Cole, J. E., Monaco, C., and Drozdov, I. Structure-preserving visualisation of high dimensional single-cell datasets. *Scientific Reports*, 9(1), June 2019. doi: 10.1038/s41598-019-45301-0. URL `https://doi.org/10.1038/s41598-019-45301-0`.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio, 2016.

Wu, L., Yen, I. E.-H., Yi, J., Xu, F., Lei, Q., and Witbrock, M. Random warping series: A random features method for time-series embedding, 2018.

Yip, A., Smith-Roberge, J., Khorasani, S. H., Aucoin, M. G., and Ingalls, B. P. Calibrating spatiotemporal models of microbial communities to microscopy data: A review. *PLOS Computational Biology*, 18(10):e1010533, October 2022. doi: 10.1371/journal.pcbi.1010533. URL `https://doi.org/10.1371/journal.pcbi.1010533`.

Young, J. W., Locke, J. C., Altinok, A., Rosenfeld, N., Bacarian, T., Swain, P. S., Mjolsness, E., and Elowitz, M. B. Measuring single-cell gene expression dynamics in bacteria using fluorescence time-lapse microscopy. *Nature protocols*, 7(1):80–88, 2012.

A Supplementary Material

A.1 Multi-Variate One dimensional Time-Series

Our simulations were multi-variate and one-dimensional (time). From Cellmodeller simulations, multiple features can be inferred at every timestep. Here, we only selected the features that were dynamic with time. A list of our features with short descriptions is presented in Table A1.

Table A1: Features and Descriptions

Feature	Description
ID	Unique identifier to keep track of a cell through time
Parent	Parent identifier to track lineage
CellAge	Age of the cell
Start_vol	Length of the bacterium when it was first formed.
pos	Position of the cell at current timestep
length	Length of cell at current timestep
dir	Orientation of cell along X-Z axis
ends0	Termini 1 of the cell
ends1	Termini 2 of the cell
strainRate_rolling	The average change in bacterial length over its lifespan

A.2 Data Processing

We preprocess both our simulations and experiments so that the set of time series values for each dataset has zero mean and unit variance. For each simulation and experiment, each dimension of the time series was preprocessed independently from the other dimensions by normalizing in the same way its mean and variance.

A.3 Sampling of parameters

We conducted 1000 simulation sets, where key parameters were varied to study their impact on the discriminant analysis. The parameter values gamma and Reg_param were sampled from Gaussian distributions (generalized equation given below) with values given in Table A2(rounded off).

$$\mathcal{N}(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Table A2: Parameter Sampling from Gaussian Distribution

Parameter	Mean (μ)	Standard Deviation (σ)	Sampled Value
`gamma`	507.478	267.824	$\sim \mathcal{N}(507.478, 267.824)$
`reg_param`	0.635	0.303	$\sim \mathcal{N}(0.635, 0.303)$

A.4 Triplet Network Architecture

We used a Triplet architecture consisting of three encoders, as described in Section 3.2. We depict the architecture of our Triplet network in Figure A1.

A.5 Network training

We used an early stopping condition on the Validation loss to halt training and prevent overfitting. Specifically, we set a patience parameter value of 5, meaning if validation loss increased for more than 5 epochs continuously, we would halt training. We present training and validation loss curves for all embedding sizes used in the study in Figure A2.

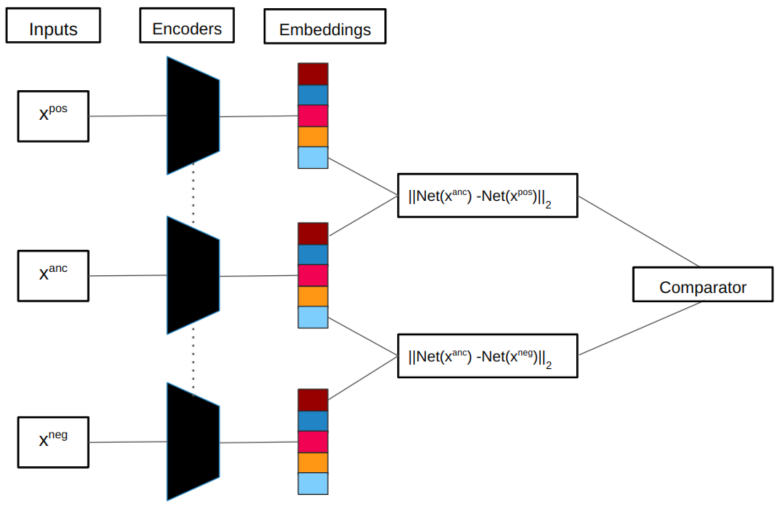

Inputs
Encoders
Embeddings
x^{pos}
x^{anc}
x^{neg}
$||Net(x^{anc}) - Net(x^{pos})||_2$
$||Net(x^{anc}) - Net(x^{neg})||_2$
Comparator

Figure A1: (a): Triplet Network, Image adapted from Hoffer & Ailon (2014), dotted lines indicate weight sharing between the encoder instances

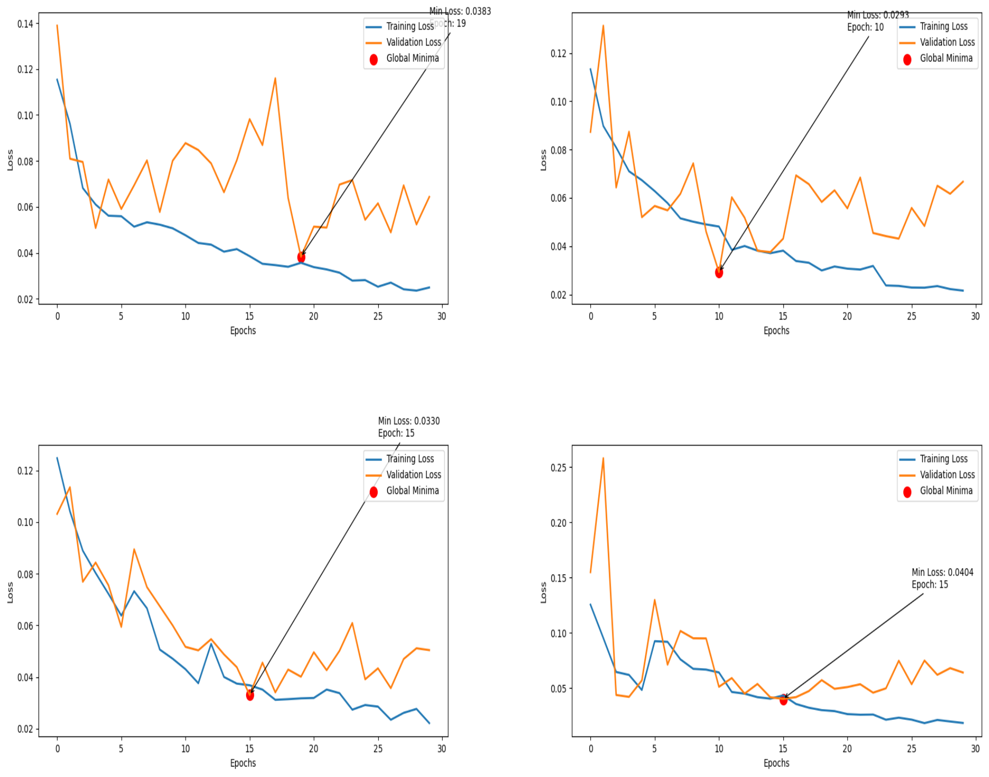


Figure A2: (a): Top Left, Embedding Size 4. (b): Top Right, Embedding Size 8. (c): Bottom Left, Embedding Size 12. (d): Bottom Right, Embedding Size 16.

A.6 Encoder Hyperparameters

We choose a single set of parameters for our Encoder, presented in Table A3. We chose a depth of 5 for our Encoder to make training faster yet retaining sufficient depth to capture features efficiently.

Table A3: Encoder Hyperparameter Choices

Hyperparameter	**Description**	**Value**
`num_samples`	Number of triplets sampled	10000
`num_val`	Number of validation classes	5
`num_epochs`	Number of training epochs	30
`learning_rate`	Learning rate, for Adam Optimizer	0.001
`in_channels`	Input channels for the network	10
`channels`	Number of channels for convolution operation	10
`depth`	Depth of the network (number of stacked encoder blocks)	5
`reduced_size`	Simulation reduced to this size, input to GlobalAvgPooling layer	200
`out_channels`	Embedding size, output of GlobalAvgPooling layer	8

A.7 Evaluation Metrics

A.7.1 Adjusted Rand Index (ARI)

The ARI quantifies the similarity between two data clusterings, considering the possibility of random agreement. It adjusts the Rand Index for chance, providing a more robust metric.

Given a set of n elements $S = \{o_1, \ldots, o_n\}$ and two partitions $X = \{X_1, \ldots, X_r\}$ and $Y = \{Y_1, \ldots, Y_s\}$ of S, define the following:

a, the number of pairs of elements in S that are in the same subset in X and in the same subset in Y.

b, the number of pairs of elements in S that are in different subsets in X and in different subsets in Y.

c, the number of pairs of elements in S that are in the same subset in X and in different subsets in Y.

d, the number of pairs of elements in S that are in different subsets in X and in the same subset in Y.

The Rand index (R) is given by:

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}}$$

Where $\binom{n}{2}$ represents the number of possible pairs of elements from a set of size n.

A.7.2 Adjusted Mutual Information (AMI)

Adjusted Mutual Information (AMI) is an adjustment of the Mutual Information (MI) score to account for chance. It accounts for the fact that the MI is generally higher for two clusterings with a larger number of clusters, regardless of whether there is actually more information shared.

For two clusterings U and V, the AMI is given as:

$$AMI(U, V) = \frac{MI(U, V) - E(MI(U, V))}{avg(H(U), H(V)) - E(MI(U, V))}$$

Where:

$MI(U, V)$ is the Mutual Information between clusterings U and V.

$E(MI(U, V))$ is the expected Mutual Information under random labeling.

$H(U)$ and $H(V)$ are the entropies of clusterings U and V.

$avg(H(U), H(V))$ is the average of the entropies of U and V.

A.7.3 Cosine Similarity

Cosine Similarity is a metric used to measure the similarity between two non-zero vectors in an inner product space, typically a high-dimensional space represented as vectors. It quantifies the cosine of the angle between the vectors, which is a measure of their orientation or direction relative to each other.

The Cosine Similarity ($\cos(\theta)$) between two vectors A and B is calculated as:

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Where:

$A \cdot B$ is the dot product of vectors A and B.
$\|A\|$ and $\|B\|$ are the magnitudes (Euclidean norms) of vectors A and B.

A.7.4 Euclidean Similarity

Euclidean Similarity is a similarity measure that quantifies how close two data points are in Euclidean space.

The Euclidean Similarity between two data points X and Y is calculated as:

$$Euclidean\ Similarity = \frac{1}{1 + Euclidean\ Distance}$$

Where: $Euclidean\ Distance$ is the Euclidean distance between data points X and Y, computed as:

$$Euclidean\ Distance = \sqrt{\sum_{i=1}^{n} (X_i - Y_i)^2}$$

This similarity measure returns values between 0 and 1, where 0 indicates maximum dissimilarity (infinite Euclidean distance), and 1 indicates maximum similarity (Euclidean distance of 0, meaning the points are identical).