ACCELERATION OF PDE-BASED BIOLOGICAL SIMULATION THROUGH THE DEVELOPMENT OF NEURAL NETWORK METAMODELS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Lukasz Burzawa

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Burzawa, Lukasz M.S., Purdue University, May 2020. Acceleration of PDE-based Biological Simulation through the development of Neural Network metamodels. Major Professors: David Umulis, Charles Bouman.

PDE models are a major tool used in quantitative modeling of biological and scientific phenomena. Their major shortcoming is the high computational complexity of solving each model. When scaling up to millions of simulations needed to find their optimal parameters we frequently have to wait days or weeks for results to come back. To cope with that we propose a neural network approach that can produce comparable results to a PDE model while being about 1000x faster. We quantitatively and qualitatively show the neural network metamodels are accurate and demonstrate their potential for multi-objective optimization in biology. We hope this approach can speed up scientific research and discovery in biology and beyond.

1. INTRODUCTION

Solving PDE models is a very computationally intensive task. One of the major tasks when working with PDE simulations is to find parameters that allow a given PDE model to give outputs that closely match experimental data. That is usually done through random search which involves running millions of PDE simulations. Assuming each one takes about a second the computational costs quickly add up to days and weeks of waiting for results to come back. That is a major inefficiency in building accurate PDE models. To remedy this we propose to use neural network proxies instead of PDE simulations for parameter search. A neural network proxy can give results that are very close to those of a PDE model while providing significant speedups from the order of seconds to milliseconds.

Here we focus on a specific one-dimensional PDE system of reaction-diffusion that models zebrafish development. Such models help us understand how complex patterns emerge in organisms and are an important tool in developmental biology.

Our contribution is to train a neural network to emulate a PDE model of reaction-diffusion in zebrafish development. We show it offers significant advantages for random parameter search and multi-objective optimization.

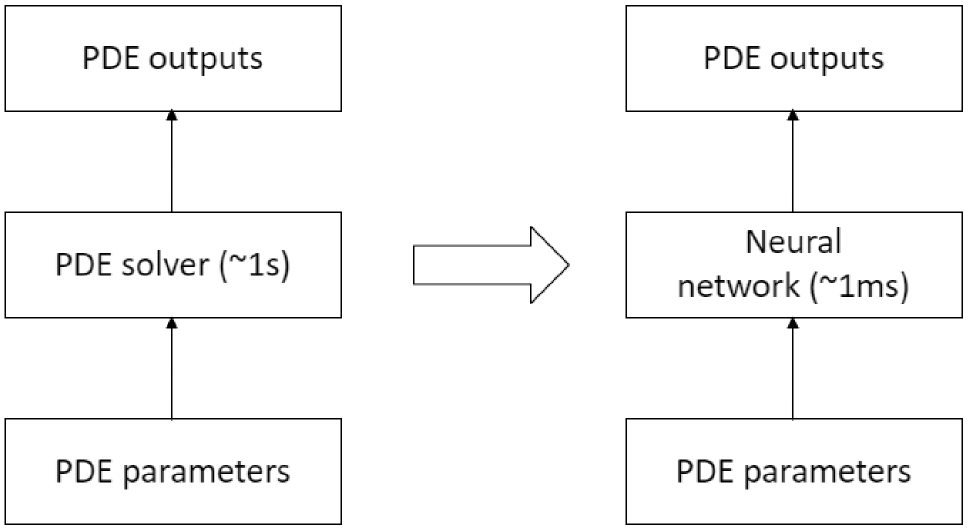


Fig. 1.1. Schematic of proposed changes

2. BACKGROUND

2.1 Mathematical modeling in developmental biology

One of the fundamental problems in developmental biology is how complex patterns in organisms emerge from a group of nearly identical cells. A major tool in understanding such complex pattern emergence is to use reaction-diffusion mathematical models which model how molecular organization changes over space and time [1]. Three major components of reaction-diffusion models are molecular transport, production and clearance. The reaction-diffusion models involve many variable parameters, for example diffusion rate, production rate and decay rate of each protein. Those depend on a system being analyzed and are determined by matching simulations to experimental data. Frequently we want to find parameters that optimize the system for multiple different species or mutations in which case multi-objective or Pareto optimization is used.

One of very important proteins that accounts for pattern formation in tissues is Bone morphogenetic protein (BMP) [2]. To understand its impact we look at its concentration gradient that forms between the ventral (frontal) part of the cell and dorsal (back) part. Changes in BMP gradient can be described as:

$$Change\ in\ BMP\ gradient\ over\ time = Diffusion + Advection + Reaction + Production$$

Diffusion accounts for movement of BMP through extracellular space. Advection refers to transport of BMP through bulk motion, an example could be cell movement due to expanding tissue. Reaction term involves interaction of BMP with extracellular regulators. Production is simply rate of BMP production. The impact of regulators on BMP reaction-diffusion can be more clearly presented in mathematical terms as:

$$\frac{\partial B}{\partial t} = D\frac{\partial^2 B}{\partial x^2} - k_{binding}B \times R + k_{unbinding}BR$$

$$\frac{\partial BR}{\partial t} = k_{binding} B \times R - k_{unbinding} BR$$

where B is BMP concentration, R is regulator concentration, BR is BMP-regulator compound concentration, D is the BMP diffusion coefficient and k represents the binding and unbinding rates of BMP and regulator.

2.2 Machine learning

Machine learning is a class of algorithms where a given task can be learned through implicit pattern recognition rather than by relying on explicit instructions. It can be split into subcategories like supervised learning, unsupervised learning and reinforcement learning. Supervised learning involves fitting a function between inputs and outputs where the outputs have clearly defined labels that are to be exactly predicted by a machine learning model. It can be done either in form of classification or regression. Some popular examples of methods used in supervised learning are Support Vector Machines (SVM), Naive Bayes classifiers, Gaussian Processes and Neural Networks. Unsupervised learning involves finding patterns in data that does not have any labels. It can be done with k-means clustering, Gaussian Mixture Models (GMM) or also with Neural Networks. Reinforcement learning (RL) involves an agent exploring an environment and attempting to find a sequence of actions that leads it to obtaining a highest reward based on reward function that was crafted by a human. Popular approaches to RL include policy gradient [3] and Q-learning [4] and usually use Neural Networks.

2.3 Deep learning

In recent years a subfield of machine learning called deep learning [5] has been gaining popularity due to advances in big data and massively parallel computer hardware [6]. Deep learning involves training neural networks with many layers. The networks are trained through backpropagation [7] and stochastic gradient descent

(SGD) optimization. Some common SGD algorithms which implement an adaptive learning rate include Adagrad [8], RMSprop [9] and Adam [10]. The simplest neural network architecture is a Multilayer Perceptron (MLP) where all nodes are fully connected. If the data involves sequences then usually Recurrent Neural Networks (RNN) [11] are used. A generic RNN diagram is shown in figure 2.1. As seem on the diagram input x(t) is combined with previous hidden state h(t-1) to give current hidden state h(t). Hidden state h(t) can optionally be passed through another linear layer to give output y(t). In form of an equation RNN can be summarized as:

$$h_t = tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh})$$

where W_{ih} and b_{ih} are weights and biases of a linear layer that processes input x_t and W_{hh} and b_{hh} are weights and biases of a linear layer that processes previous hidden state h_{t-1}. Training RNNs is a bit more complex since it involves backpropagation through time where gradients are summed up for all time steps. Standard RNNs struggle with long sequences as gradients start to vanish if they are backpropagated though a long graph. To deal with Long Short-Term Memory Networks (LSTM) [12] were proposed where some gradients are allowed to be passed almost undisturbed. The LSTM equations are:

$$i_t = sigmoid(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$

$$f_t = sigmoid(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$

$$g_t = tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$$

$$o_t = sigmoid(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t$$

$$h_t = o_t \circ tanh(c_t)$$

where i_t is the input gate, f_t the forget gate, g_t the cell gate, o_t the output gate, c_t the cell state, h_t the hidden state and $\circ$ represents element-wise multiplication. Cell state and hidden state are the outputs of the LSTM that get passed to the next layer in the neural network and/or to the next time step. By inspecting the equations we can see there are 8 different linear layers in one LSTM module which makes them pretty computationally expensive. Gated Recurrent Units (GRU) [13] were proposed to alleviate the computational costs as they include 6 linear layers in one module. If the data involves 1D or 2D correlations to be explored like in speech signal or images then Convolutional Neural Networks (CNN) [14] are used. Unlike in MLP, nodes of a CNN are sparsely connected to focus on spatial interdependencies.

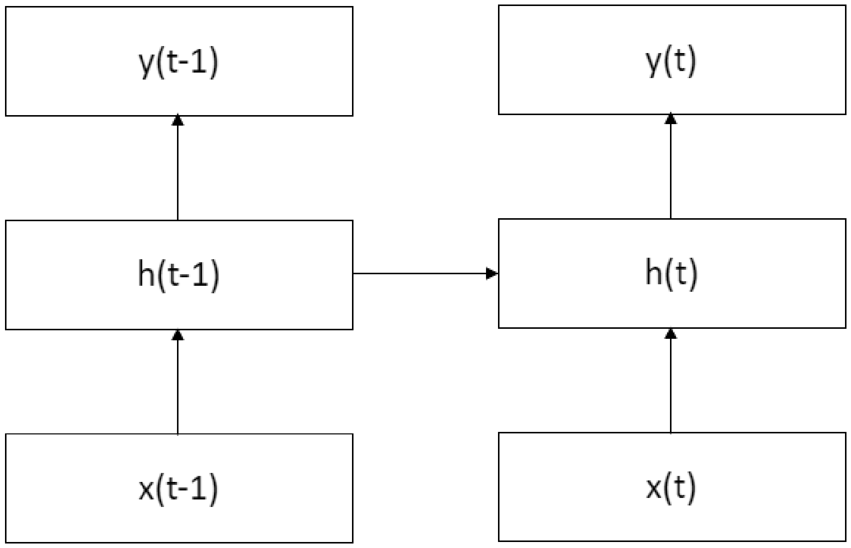


Fig. 2.1. Example RNN diagram

2.4 Applications of machine learning to mathematical modeling

A major advantage of neural network models is that they are built mostly on basic linear algebra operations like matrix multiplications and convolutions which are highly parallelizable and run on the order of milliseconds unlike many scientific simulation which take seconds or minutes. Hence if a neural network could emulate a given mathematical model running on a computer then it could offer major advantages in terms of speed. Such use of neural networks has been seen in materials science, chemistry, physics, robotics and recently biology. We summarize it next.

Schrodinger Equation is a fundamental tool in quantum mechanics. Yet exact solutions are only possible for the smallest systems and otherwise expensive numerical approximations have to be used. To deal with it, [15] uses a sum of weighted Gaussians to predict molecular energy based on distance between molecules. The regression coefficients are found through kernel ridge regression. They generate data using Density Functional Theory (DFT) approximations. The efficiency of the machine learning approach paves the way for large scale exploration of chemical compounds and their energies.

Simulators are a common way to conduct robotics research and development. However the rigid body dynamics models they employ very time consuming due to complex nature of physics they are trying to capture. In [16] authors use experimental data to train MLP and RNN to predict motion and sensory outputs of a robot based on its current positioning and kinematics. That increases effectiveness of building robust control system for robots.

In [17] an MLP is used to predict a potential energy of a molecular system based on DFT calculations. What's notable is the authors carefully crafted a molecular representation as an input to MLP. They took into account that the representation needs to be compact while maximizing resolution of local atomic environment and covering all the relevant space molecule takes. They termed the resulting input as Atomic Environment Vector (AEV). That is in line with other work in deep learning that shows that input representation matters. For example [18] shows they are able to narrow the gap in 3D object detection between stereo and LIDAR vision data by generating a pseudo-LIDAR representation from stereo data.

One of major problems within materials discovery is to be able to identify stable compositions of chemical compounds. It is mostly done through expensive DFT calculations. In [19] they train an MLP to predict formation energy of a crystal based on Pauling electronegativity and ionic radius of species. They achieve very accurate results on garnets and perovskites.

Another crucial yet very computationally costly problem in science is a three-body problem. In [20] the authors train an MLP to predict location of particles 1 and 2 given input of time t and initial location of particle 2. Location of particle 3 follows from problem symmetry. To acquire data they use Brutus numerical integrator. The time saved by using a neural network is at the order of 100 million. Such fast and accurate three-body solver has major implications for research into areas like black-hole systems and core collapse in star clusters.

In [21] machine learning is applied to find exciton dynamics normally computed with costly time-dependent DFT calculations. The particular molecules studied are bacteriochlorophylls in Fenna-Matthews-Olsen complex. MLP is trained to predict their excited state energies from Coulomb matrices which describe electrostatic interaction between nuclei. The neural network is able to reproduce quantum mechanical results with less than 1% error.

Most related to our project is the work of [22] that applies neural networks to a mechanistic PDE model of pattern formation in bacteria. They train an LSTM to predict spatial distribution of a molecule based on PDE parameters like cell growth rate, cell motility and kinetics of gene expression. The data is generated by a PDE solver. Their neural network achieves R^2 of around 0.99 and provides speedup of about 30,000x. That carries great potential for efficiently exploring the parameters space of the PDE model and finding spatial distributions not easily seen before.

Most of the above work uses deep learning which scales better with increasing data sizes than more traditional machine learning approach like SVMs and Gaussian Processes. That is good choice considering the data here comes from simulation and is automatically labeled. Since it does not require strenuous manual annotation like vision, speech and text data, there is little limit on how much data can be generated from those simulations and deep learning is a good choice of framework to be applied to it. In some cases where data is sequential we can see authors use RNNs to solve the problem. That is conceptually correct but we need to also consider that RNNs operate in sequences and involve multiple linear layers in a single module which considerably slows them down compared to standard MLPs. Since inference latency is a crucial concern when emulating scientific simulations with neural networks, this disparity between MLP and RNN has to be taken into account.

3. METHODS

3.1 PDE model of reaction-diffusion

We consider a PDE model of reaction-diffusion in zebrafish development that is represented by equations below. There are six proteins: BMP, Chordin, Noggin, BMP-Chordin, BMP-Noggin and Sizzled that interact with each other.

$$\frac{\partial B}{\partial t} = D_B \frac{\partial^2 B}{\partial x^2} + \phi_B(x) + \lambda_{tBC} \frac{1}{1 + \frac{S}{kit} + \frac{C+BC}{kmt}} BC + \lambda_{aBC} \frac{1}{1 + \frac{S}{kia} + \frac{C+BC}{kma}} BC$$
$$- k_{onC} B \times C + k_{offC} BC - k_{onC} B \times N + k_{offN} BN - dec_B B$$

$$\frac{\partial C}{\partial t} = D_C \frac{\partial^2 C}{\partial x^2} + \phi_C(x) + \lambda_{tC} \frac{1}{1 + \frac{S}{kit} + \frac{C+BC}{kmt}} C + \lambda_{aC} \frac{1}{1 + \frac{S}{kia} + \frac{C+BC}{kma}} C$$
$$- k_{onC} B \times C + k_{offN} BC - dec_C C$$

$$\frac{\partial N}{\partial t} = D_N \frac{\partial^2 N}{\partial x^2} + \phi_N(x) - k_{onN} B \times N + k_{offN} BN - dec_N N$$

$$\frac{\partial BC}{\partial t} = D_{BC} \frac{\partial^2 BC}{\partial x^2} + \lambda_{tBC} \frac{1}{1 + \frac{S}{kit} + \frac{C+BC}{kmt}} BC + \lambda_{aBC} \frac{1}{1 + \frac{S}{kia} + \frac{C+BC}{kma}} BC$$
$$+ k_{onC} B \times C - k_{offN} BC - dec_{BC} BC$$

$$\frac{\partial BN}{\partial t} = D_{BN} \frac{\partial^2 BN}{\partial x^2} + k_{onN} B \times N - k_{offN} BN - dec_{BN} BN$$

$$\frac{\partial S}{\partial t} = D_S \frac{\partial^2 S}{\partial x^2} + \frac{V_s \times B^n}{K^n + B^n} - dec_S S$$

The model has variable parameters that are describing in Table 3.1. For each set of parameters there are seven mutations described in Table 3.2, each involving solving a separate set of PDEs. Most of the variable parameters have to be randomly searched according to their ranges so that the outputs of the simulation match the experimental data. The fitness of the parameters is determined by Normalized Root Mean Square Error (NRMSE) between the final BMP distribution from the simulation and the experimental values. The values of S_{max} that are used to find appropriate k_{it} and k_{ia} ranges are determined based on other simulation not discussed here. The value of k is determined from maximum value of BMP coming out of corresponding Chordin loss function simulation for which k is not needed yet. You can think of k as being indirectly derived from the other parameters. The value of V_s is 100 except for Sizzled loss function simulation where it's 0.

3.2 PDE acceleration through neural networks

The inputs to our neural network are the PDE model parameters that are generated randomly according to ranges in Table 3.1. The output is the final distribution of BMP concentration over 36 points in 1D space. Both inputs and outputs are normalized by taking a logarithm of base 10 of all the values and then dividing the resulting values by 10. Also any value less than 10^{-8}, including 0, is approximated to 10^{-8}. This way we ignore concentrations too small to be significant and avoid taking a logarithm of zero. To collect data for neural network training we run 100,000 simulations, each consisting of 7 different mutations for a total of 700,000 unique simulation data points. 90% of data is used for validation and 10% is used for validation.

We consider two neural network architectures: MLP and LSTM. In MLP model, shown in Figure 3.1, the PDE parameters are passed through a sequence of linear layers, each followed by Rectified Linear Unit (ReLU) activation function. The output layer gives the whole distribution of BMP concentrations at once. That is in contrast to an LSTM model, shown in Figure 3.2, where the BMP concentrations are output in a sequence, one by one. Here the PDE parameters are passed first through a linear layer that gives a higher dimensional parameter embedding. Then the parameter

Table 3.1.
Variable parameters in the PDE model.

Parameter	**Symbol**	**Min**	**Max**
BMP Production Rate	ϕ_B	10^{-3}	10^{-1}
Chordin Production Rate	ϕ_C	10^{-2}	10^{1}
Noggin Production Rate	ϕ_N	10^{-2}	10^{2}
Noggin Decay Rate	dec_N	10^{-5}	10^{-1}
Sizzled Decay Rate	dec_S	10^{-5}	10^{-1}
BMP-Chordin Decay Rate	dec_{BC}	10^{-5}	10^{-3}
BMP-Noggin Decay Rate	dec_{BN}	10^{-5}	10^{-3}
Chordin Diffusivity	D_C	0.5	50
Noggin Diffusivity	D_N	10^{-2}	10^{2}
BMP-Chordin Diffusivity	D_{BC}	10^{-2}	10^{2}
BMP-Noggin Diffusivity	D_{BN}	10^{-2}	10^{2}
Binding Rate for BMP and Chordin	k_{onC}	10^{-4}	10^{0}
Binding Rate for BMP and Noggin	k_{onN}	10^{-4}	10^{0}
BMP-Chordin Degradation by Tolloid	λ_{tBC}	10^{-4}	10^{0}
Chordin Degradation by Tolloid	λ_{tC}	10^{-4}	10^{0}
BMP-Chordin Degradation by Bmp1a	λ_{aBC}	10^{-4}	10^{0}
Chordin Degradation by Bmp1a	λ_{aC}	10^{-4}	10^{0}
Michaelis Constant of Tolloid	k_{mt}	10^{0}	10^{2}
Michaelis Constant of Bmp1a	k_{ma}	10^{0}	10^{2}
Sizzled Inhibitor Constant with Tolloid	k_{it}	$0.1 \times S_{max}$	$10 \times S_{max}$
Sizzled Inhibitor Constant with Bmp1a	k_{ia}	$0.1 \times S_{max}$	$10 \times S_{max}$
Hill Function Parameter	k	N/A	N/A
Max of Sizzled expression	V_s	100	100

Table 3.2.
Mutations

Mutation	**Change in parameters**
Wild type (WT, no mutation)	None
Chordin loss of function (CLF)	$\phi_C = 0$
Noggin loss of function (NLF)	$\phi_N = 0$
Bmp1a loss of function (ALF)	$\lambda_{aBC} = 0$, $\lambda_{aC} = 0$
Tolloid loss of function (TLF)	$\lambda_{tBC} = 0$, $\lambda_{tC} = 0$
Bmp1a and Tolloid loss of function (TALF)	$\lambda_{aBC} = 0$, $\lambda_{aC} = 0$, $\lambda_{tBC} = 0$, $\lambda_{tC} = 0$
Sizzled loss of function (SLF)	$V_s = 0$

embedding is concatenated with an LSTM output at a previous step in sequence and passed to an LSTM module which outputs the BMP concentration at the current point in sequence. A BMP concentration of 0 (-0.8 after normalization) is given as a dummy input at the first step of a sequence. The sequence length of LSTM is 36 since there are 36 points in space for the PDE model.

To match neural network outputs with actual PDE outputs an L1 loss is calculated between the two. It is then backpropagated through the neural network to calculate gradient at each weight of the neural network. Then the weights are optimized through Adam algorithm with an initial learning rate of 0.001. The training is run for 100 epochs.

Except for L1 loss we consider two other metrics to evaluate how well our neural network metamodel is doing. First is Coefficient of Determination R^2 calculated from normalized values. The other is relative error between NRMSE of PDE system and NRMSE of the neural network model calculated as shown below.

$$Error = 100\% \times \frac{|NRMSE_{NN} - NRMSE_{PDE}|}{NRMSE_{PDE}}$$

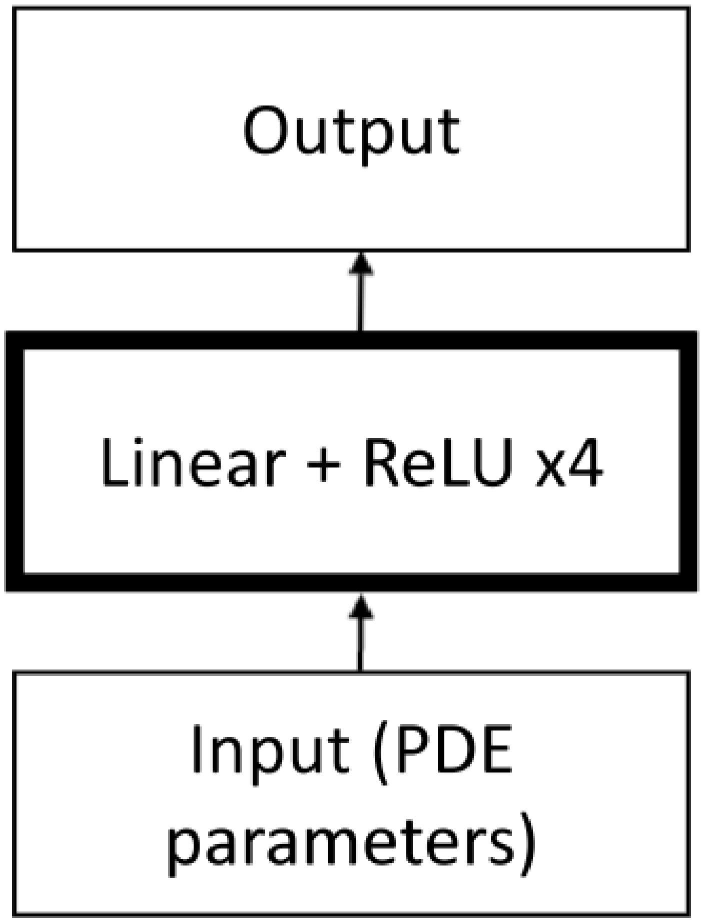


Fig. 3.1. MLP model architecture

This metric allows us to directly evaluate how a neural network would do during parameter search as compared to the original PDE model.

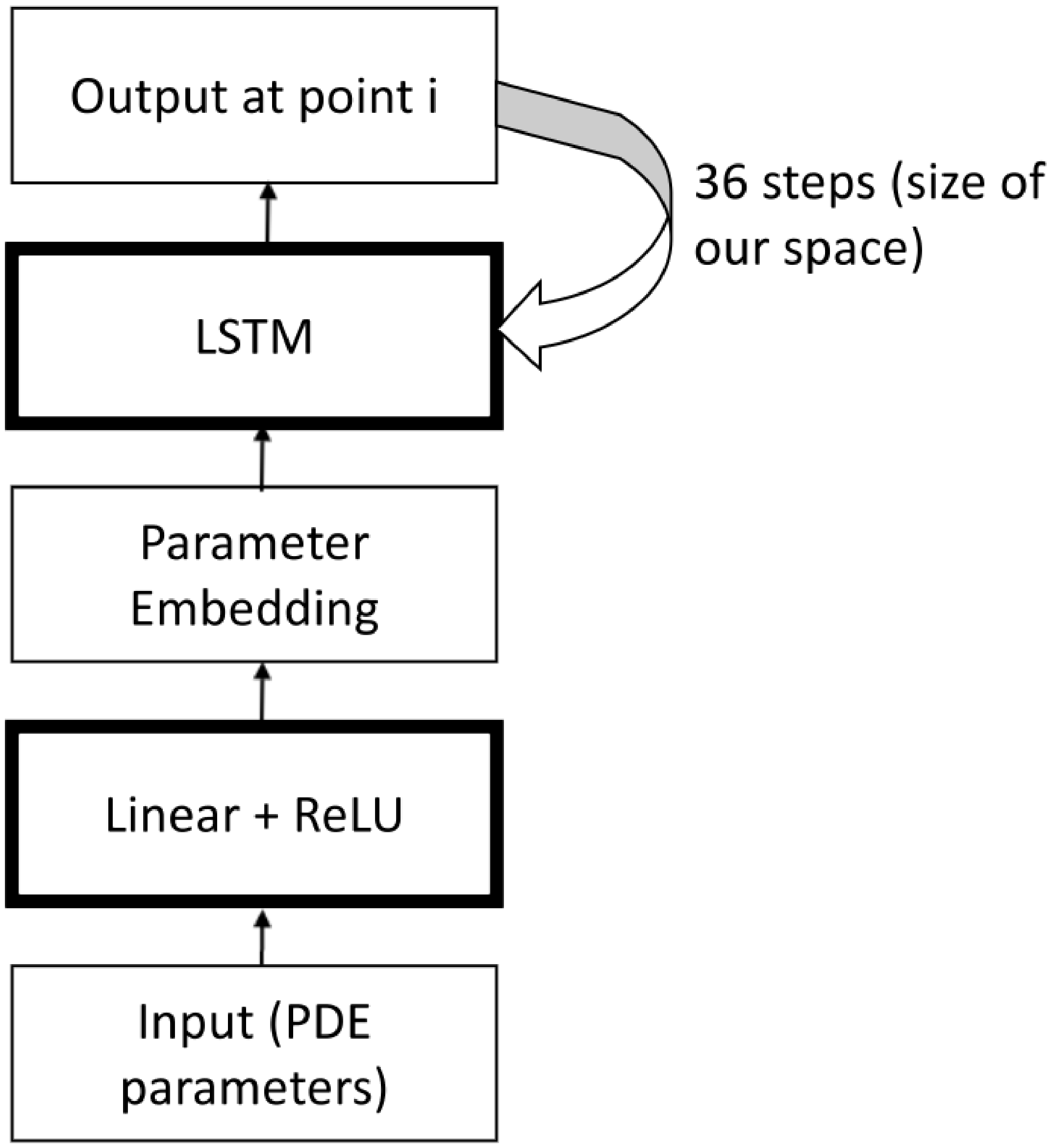


Fig. 3.2. LSTM model architecture

4. RESULTS

4.1 Evaluation of neural network metamodels

Tables 4.1 and 4.2 show the validation set results of training different MLP and LSTM models. The MLPs are grouped in format MLP-number of layers-number of units. For example MLP-3-256 means the model has 3 layers, first with 256 outputs, second with 256 outputs and third with 36 outputs. MLP-4-256 would have one more layer with 256 outputs. For LSTMs we consider modules with output sizes of 256 and 512. In addition to accuracy metrics like R^2 and relative error we also consider number parameters and computational cost metrics like number of floating point operations (FLOPs), latency on a standard Intel CPU and latency on Titan X Pascal GPU. Among MLP models, MLP-4-1024 has the best accuracy while among LSTM models it is LSTM-512. We can also see that LSTMs models slightly outpeform MLP models in accuracy. For example LSTM-512 has the same number of learnable parameters as MLP-4-1024 with a relative error lower by over 1%. That is due to LSTM's ability to understand sequences. However that also comes with a bigger computational cost. All of FLOPs, CPU latency and GPU latency are more than 10x larger for LSTM-512 than for MLP-4-1024. From here on we only consider the best MLP and LSTM models hence we refer to the MLP-4-1024 model as MLP and to the LSTM-512 model as LSTM.

Next we investigate how the neural network model responds to the amount of data it is fed and how its error changes as training progresses. We only consider the MLP model here since the LSTM model is expected to give very similar trends. Table 4.3 shows how results of MLP model vary depending on different data sizes used. As expected the accuracy improves as more samples are used. Using 100,000 samples gives satisfactory results. Results are acceptable for 10,000 samples and not good for

Table 4.1.
Comparing MLP models

	MLP-3-256	**MLP-3-1024**	**MLP-4-256**	**MLP-4-1024**
Parameters	0.0812M	1.11M	0.147M	2.16M
FLOPs	0.162M	2.22M	0.294M	4.32M
CPU latency	0.092ms	0.120ms	0.125ms	0.319ms
GPU latency	0.187ms	0.189ms	0.237ms	0.237ms
R^2	0.9969	0.9982	0.9984	0.9989
Rel. error	12.19%	9.57%	8.34%	6.99%

Table 4.2.
Comparing LSTM models

	LSTM-256	**LSTM-512**
Parameters	0.533M	2.11M
FLOPs	37.9M	151M
CPU latency	5.96ms	12.4ms
GPU latency	5.51ms	5.67ms
R^2	0.9994	0.9996
Rel. error	7.67%	5.74%

1000 samples. Such differences in accuracy based on number of samples are quite standard compared to other deep learning application where usually 10,000 training samples are needed. This table also shows importance of our relative error metric. For 1000 samples we get a high relative error while still a respectable R^2. That is because R^2 is calculated on normalized log values since calculating it on actual values would make those of larger magnitude dominate which we do not want. However just using log values might not tell the whole story and that is why we find the relative error metric very useful. It uses actual, not normalized, outputs and gives a deviation from the result we actually want to reproduce for PDE parameter optimization which is the NRMSE between experimental data and simulation.

Table 4.3.
Comparing results by number of data samples used during training process

	1k	10k	100k
R^2	0.9675	0.9955	0.9989
Rel. error	68.17%	16.70%	6.99%

Figure 4.1 shows how validation error changes as training progresses on 100,000 samples. We can clearly see that neural network is learning and error decreases from above 50% at the start of training to under 10% at the end of training.

Table 4.4 shows the relative error by mutation for the best MLP model, MLP-4-1024, and the best LSTM model, LSTM-512. We can see that the results are generally consistent across all mutations.

Table 4.4.
Relative error by mutation

	MLP	**LSTM**
Mean error	6.99%	5.74%
WT error	6.58%	5.57%
CLF error	7.38%	5.96%
NFL error	7.64%	7.02%
ALF error	8.14%	5.50%
TLF error	5.28%	4.21%
TALF error	6.54%	4.37%
SLF error	7.46%	7.74%

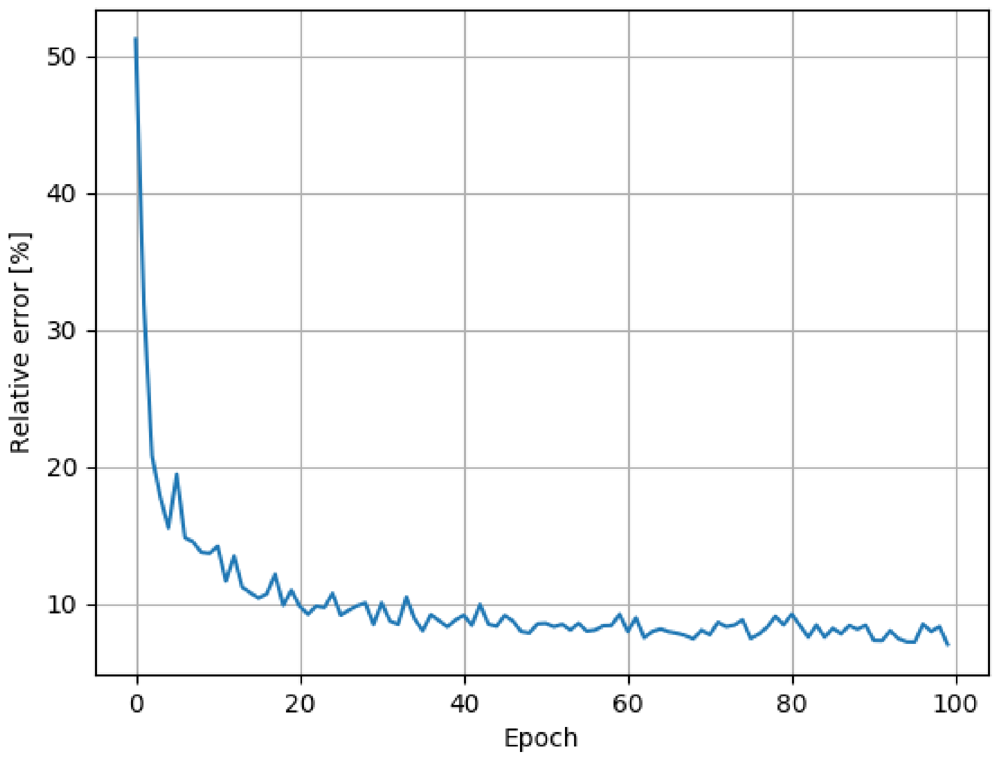
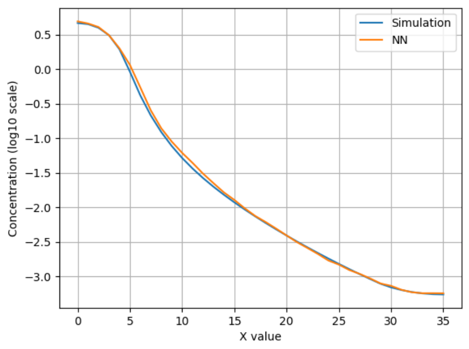
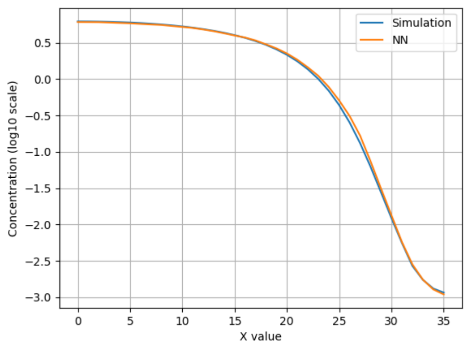


Fig. 4.1. Training plot

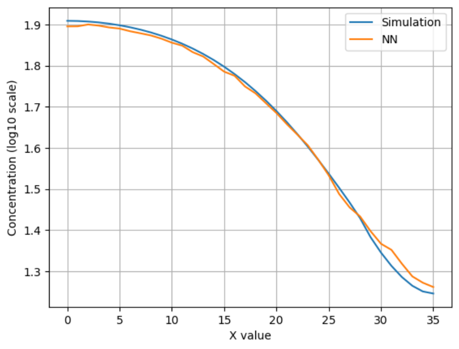
Figures 4.2 and 4.3 show how MLP and LSTM model respectively reproduce the PDE outputs on seven randomly chosen samples, one for each mutation. The mutation plots are not of the same wild-type sample so that we can see a greater diversity of results. We can see that generally the neural networks give final BMP distribution that is very similar to the one given by a PDE simulation. Since LSTM gives outputs in form of a sequence, one by one, we would expect that its plots would generally be smoother than those of MLP. That is generally true in the plots we see. Figures 4.2(c), 4.2(e) and 4.2(f) have some irregularities that disappear or smooth out when looking at Figures 4.3(c), 4.3(e) and 4.3(f).



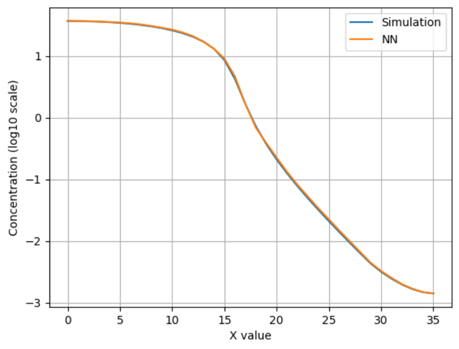
(a) Example WT distribution



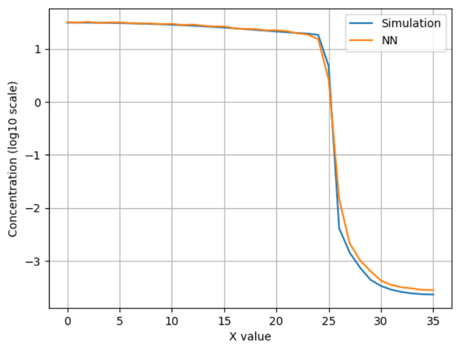
(b) Example CLF distribution



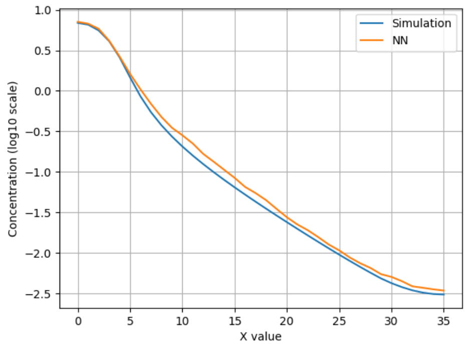
(c) Example NLF distribution



(d) Example ALF distribution

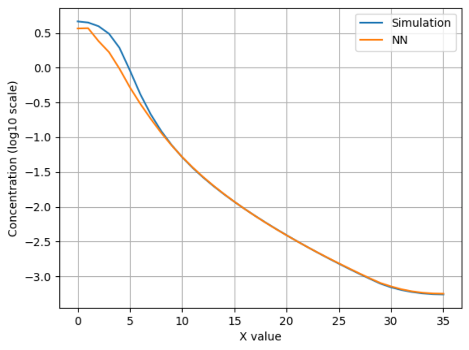


(e) Example TLF distribution

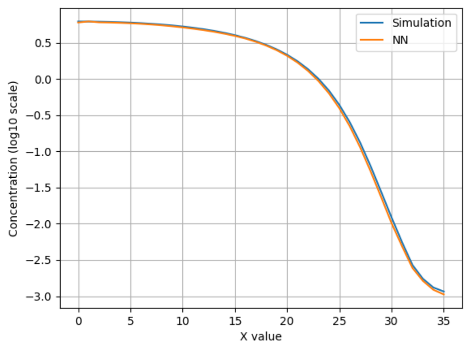


(f) Example TALF distribution

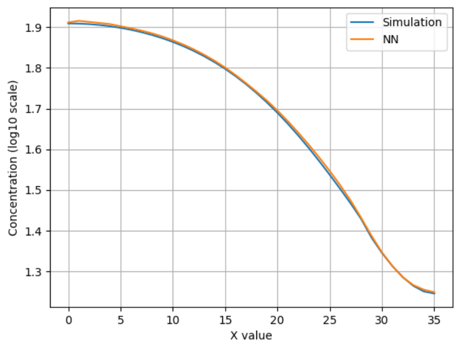
Fig. 4.2. Comparison of protein distributions obtained by PDE simulation and MLP metamodel



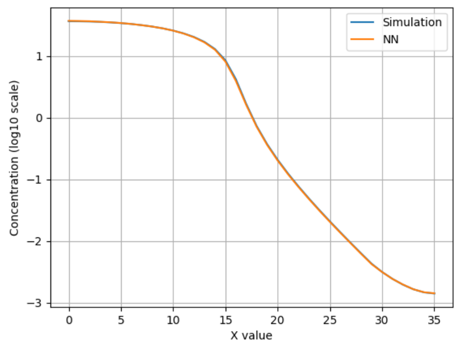
(a) Example WT distribution



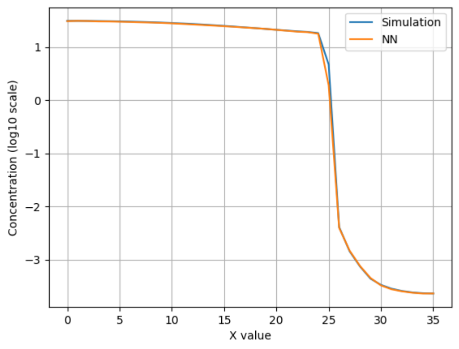
(b) Example CLF distribution



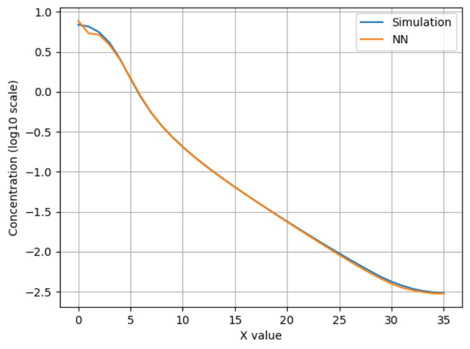
(c) Example NLF distribution



(d) Example ALF distribution



(e) Example TLF distribution



(f) Example TALF distribution

Fig. 4.3. Comparison of protein distributions obtained by PDE simulation and LSTM metamodel

Figures 4.4 and 4.5 are density plots that show how the NRMSE results compare between a neural network metamodel and a PDE simulation on validation data. The black line symbolizes the best fit line calculated from the points. On WT data the best fit line is $y = 0.981x + 0.005$ for MLP and $y = 0.996x + 0.0009$ for LSTM. On CLF data the best file line is $y = 0.984x + 0.003$ for MLP and $1.005x + 0.002$ for LSTM. A perfect match would be a straight line $y = x$. In case of both WT and CLF data the LSTM slightly outperforms the MLP in giving slopes closer to 1 and y-axis intercepts closer to 0. Visually, LSTM model also seems to give less outliers on CLF data than MLP model. Since these are density plots we can see some circular yellow hotspots on the plots. These are irrelevant, they just show where most points fall coming out of PDE simulations. More important is the yellow to light blue line that forms and shows most points indeed fall on y=x.

Both quantitative and qualitative evaluations slightly favor the LSTM model. It gives higher R^2, lower relative error, produces smoother BMP distributions and reproduces the mutation data with less variation. However those improvement come at significantly higher computational cost of at least 20x. Since we would like to use the neural network metamodel for rapid exploration of PDE parameter space that is considerable. Hence for practical purposes we consider the MLP model to be superior than LSTM model.

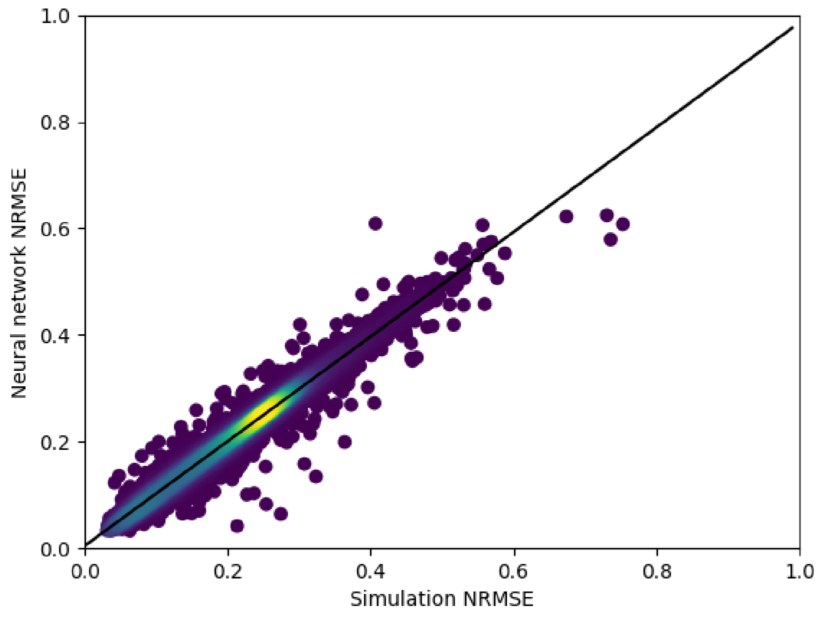
4.2 Evaluation in context of multi-objective optimization

As already mentioned, multi-objective optimization is an important tool for quatitative biology. Here we consider multi-objective optimization in terms of balancing error obtained between simulation and experiment for different mutations. We show how accurately can the MLP model reproduce results obtained from PDE simulations.

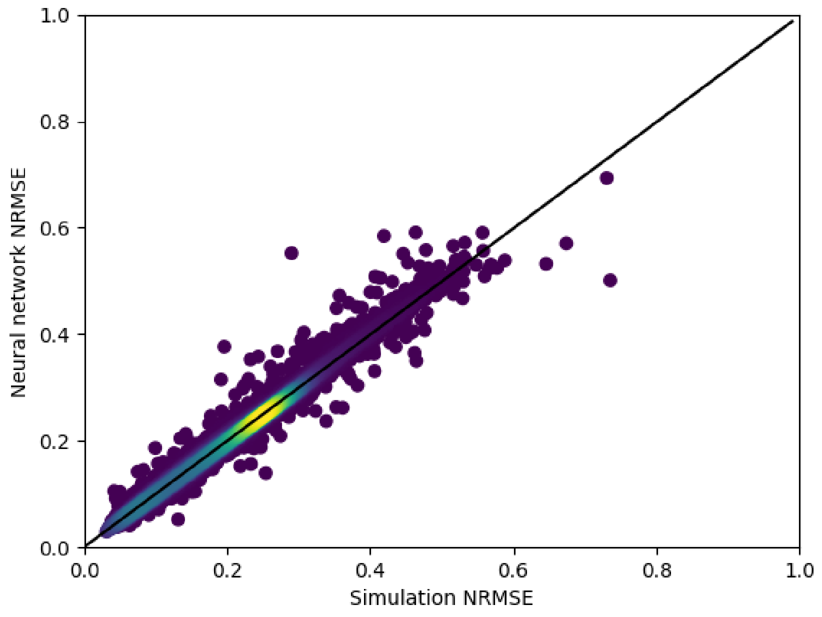
Figures 4.6(a) and 4.6(b) show how a multi-objective plot of WT NRMSE vs CLF NRMSE compares between a neural network and a PDE simulation. We can see the regions with highest density are almost the same for both PDE model and a neural network. Similarly figures 4.7(a) and 4.7(b) compare the two while assuming all the points present have NRMSE less 0.2 for every other mutation, so for NLF, ALF, TLF, TALF and SLF. The comparison between PDE model and neural network also looks promising, points on both plots occupy the same region.

Figure 4.8 shows the Pareto frontiers found based on data in figure 4.7. Pareto frontier is generated by finding all points that are not dominated by any other points. This means there is no other point that has both WT NRMSE and CLF NRMSE lower than any of the points on the plot. We can see simulation and neural network produce similar frontiers. The largest deviation is observed for the points in lower right part which have the lowest CLF NRMSE. However after looking at their exact values we can see the difference is about 10% which is consistent with relative error results from Table 4.4. We acknowledge this Pareto frontier occupies a relatively small area from figure 4.7 and is hence quite simpler that Pareto frontiers usually encountered. This is more of a demonstration of how neural networks could be applied to Pareto optimization in biology. A more interesting example could be a frontier between two different species like zebrafish and fruit fly.

To generate discussed pareto frontier we used 70000 points, since our validation set has 10000 samples and there are 7 mutations per sample. Running 70000 simulations using our PDE system would take about 20 hours assuming each simulation takes about 1 second. On the other hand the neural network metamodel would only need about 1 minute to get the results back. That is why using neural networks for multi-objective optimization has great potential.

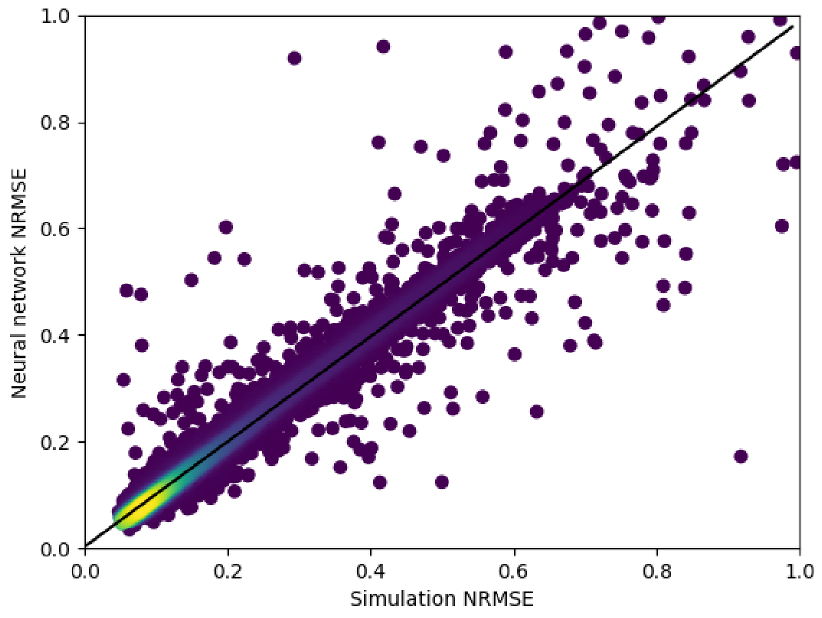


(a) MLP

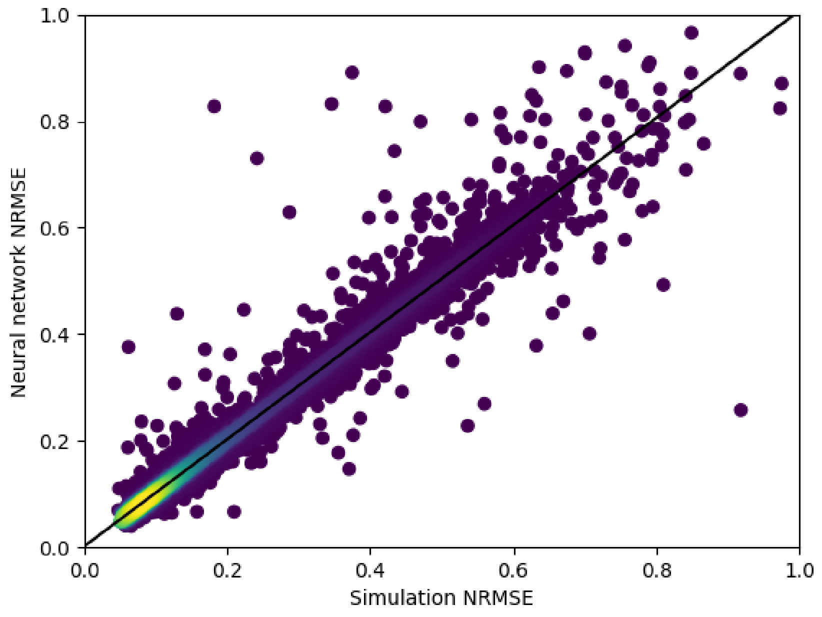


(b) LSTM

Fig. 4.4. Comparison of NRMSE values obtained by simulation and neural network metamodel for WT data

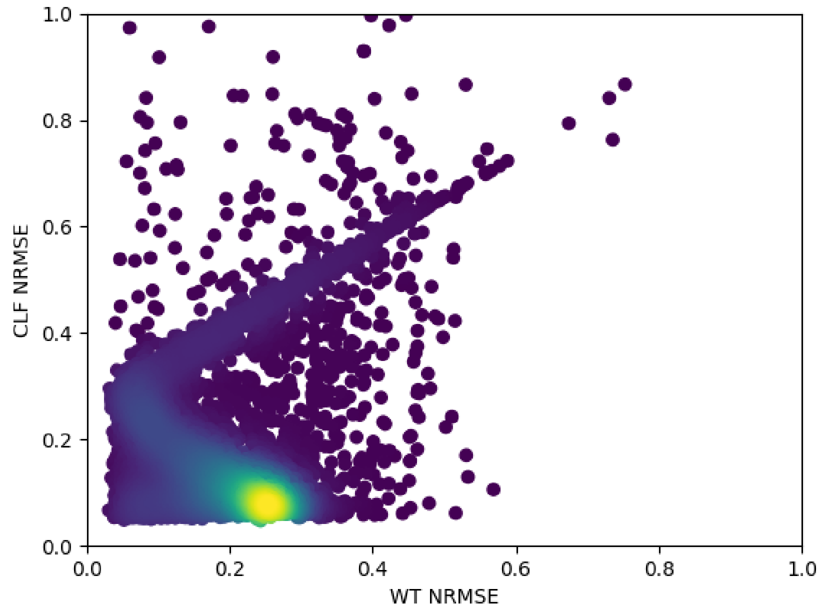


(a) MLP

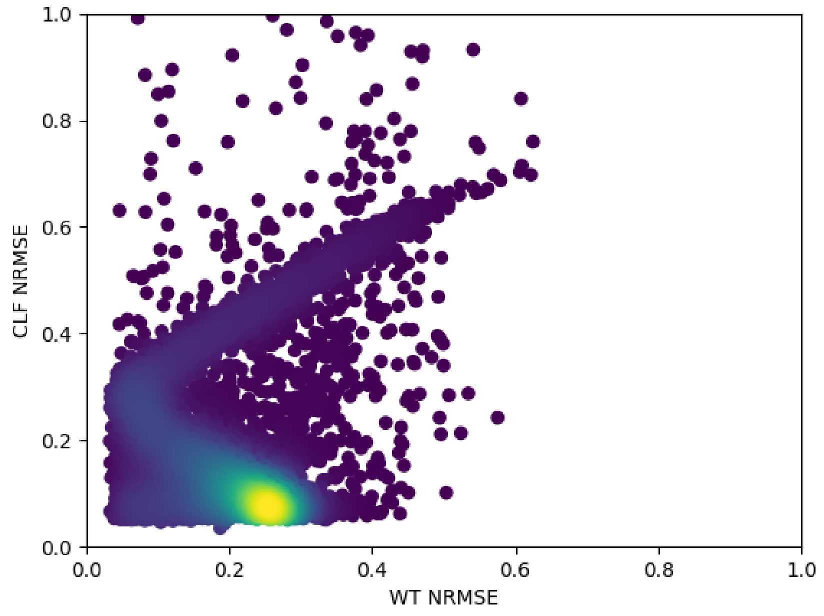


(b) LSTM

Fig. 4.5. Comparison of NRMSE values obtained by simulation and neural network metamodel for CLF data

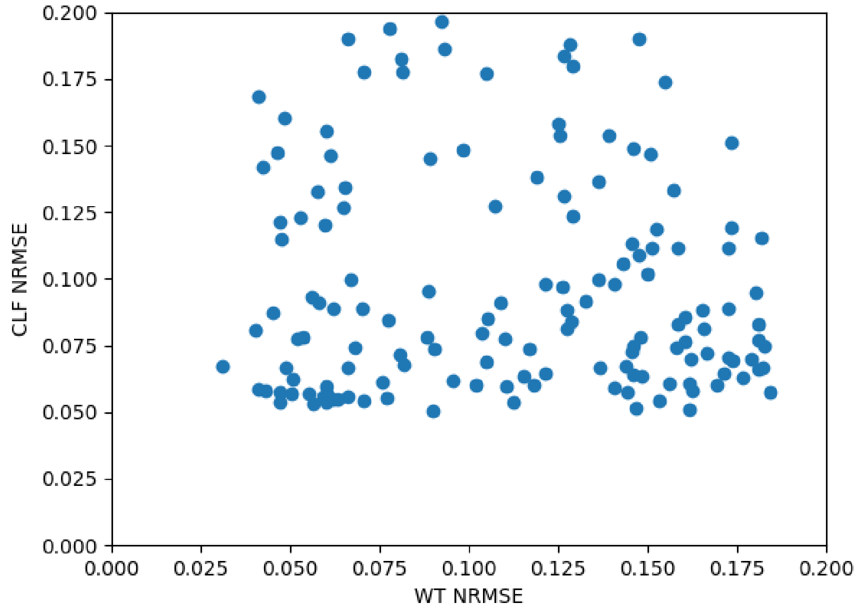


(a) Simulation

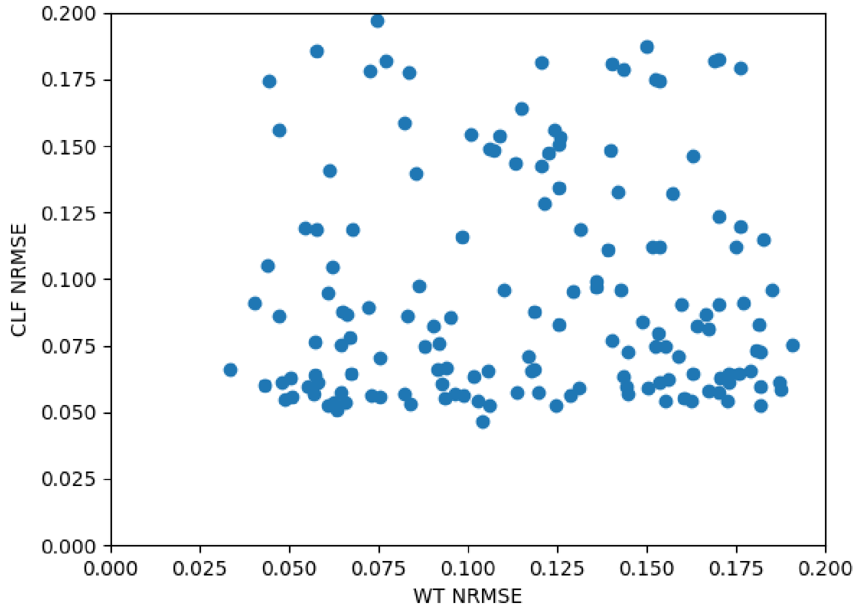


(b) Neural network

Fig. 4.6. Comparison of multi-objective plots between simulation and NN considering only WT and CLF mutation data



(a) Simulation



(b) Neural network

Fig. 4.7. Comparison of multi-objective plots between simulation and NN if all mutation NRMSE values are less than 0.2

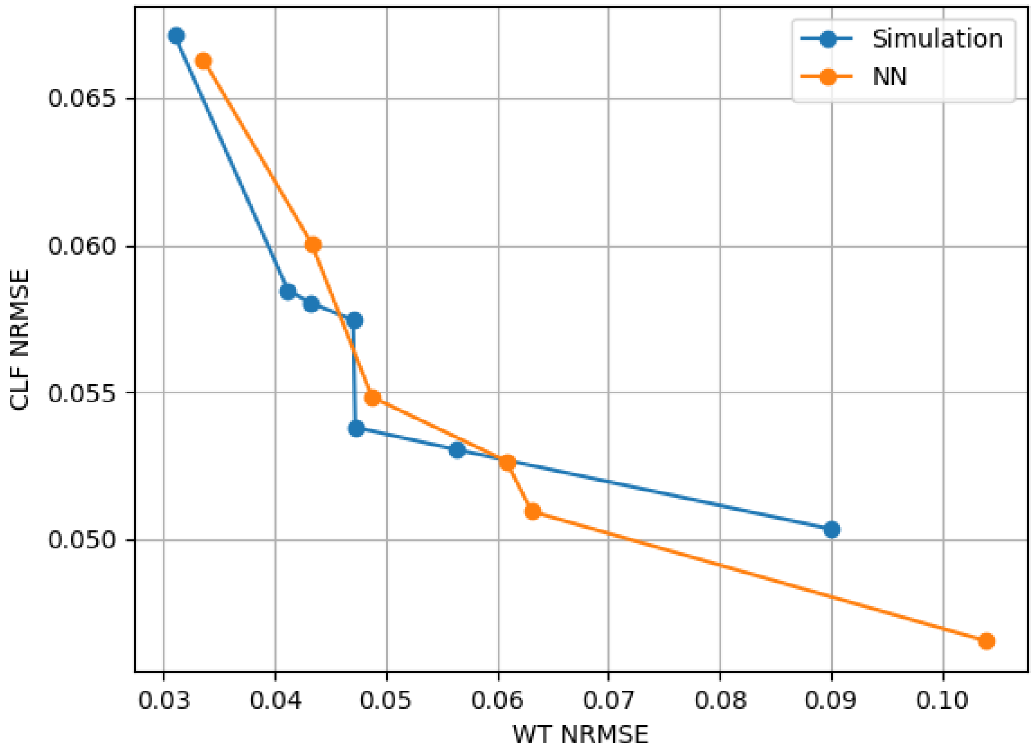


Fig. 4.8. Comparison of Pareto frontiers

5. CONCLUSION

We show that neural network metamodels are an effective tool to accelerate biological simulations. They offer speedups of about 1000x while preserving accuracy with R^2 above 0.99 and relative NRMSE error under 10%. We consider a specific PDE model from zebrafish development but the neural network models discussed here can be applied to any other PDE system. Compared to [22] we train both LSTM and MLP model and show that MLP offers advantages in speed without sacrificing much accuracy. We also show the potential of using neural network metamodels for multi-objective optimization in biology.

Future work could involve applying the neural network approach to more complex PDE system that model 2D or 3D dynamics. The multi-objective optimization described here for mutations could also be extended for multiple species, for example zebrafish and fruit fly. Another important area of contribution to PDE modeling could be in solving an inverse problem. In the inverse problem the goal is to find parameters that match the PDE outputs to experimental data. This could be simply done by replacing PDE solvers with neural network and conducting an accelerated random search as implied by this work. However neural networks also offer a unique advantage for gradient descent approaches to parameter search as they are fully differentiable. This means gradient descent could use faster and more accurate backpropagation instead of numerical gradient approximations. Finally neural network and machine learning could also contribute to inverse problem by employing reinforcement learning to find optimal parameters. The actions from RL could be the parameters searched and reward could be the inverse of error between PDE simulation or NN metamodel and experiment. In a similar fashion reinforcement learning has already been used in Neural Architecture Search (NAS) [23].

We hope our work serves as a useful tool for scientists and engineers who work with complex physical simulations and enhances scientific research and discovery in biology and beyond.

REFERENCES

REFERENCES

[1] M. Thompson, H. Othmer, and D. Umulis, “A primer on reaction-diffusion models in embryonic development,” *Encyclopedia of Life Sciences*, 2018.

[2] J. Zinski, Y. Bu, X. Wang, W. Dou, D. Umulis, and M. C. Mullins, “Systems biology derived source-sink mechanism of bmp gradient formation,” *eLife*, 2017.

[3] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, 1992.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, and et al, “Human-level control through deep reinforcement learning,” *Nature*, 2015.

[5] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, 2015.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *NIPS*, 2012.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, 1986.

[8] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *JMLR*, 2011.

[9] G. Hinton, “Overview of mini-batch gradient descent,” 2012.

[10] D. P. Kingma and J. L. Ba, “Adam: a method for stochastic optimization,” *ICLR*, 2015.

[11] A. Karpathy, “The unreasonable effectiveness of recurrent neural networks,” 2015.

[12] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, 1997.

[13] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *EMNLP*, 2014.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, 1998.

[15] M. Rupp, A. Tkatchenko, K.-R. Muller, and O. A. von Lilienfeld, “Fast and accurate modeling of molecular atomization energies with machine learning,” *Physical Review Letters*, 2012.

[16] C. J. Pretorius, M. C. du Plessis, and C. B. Cilliers, "Simulating robots without conventional physics: A neural network approach," *Journal of Intelligent & Robotic Systems*, 2013.

[17] J. S. Smith, O. Isayev, and A. E. Roitberg, "Ani-1: an extensible neural network potential with dft accuracy at force field computational cost," *Chemical Science*, 2017.

[18] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving," *CVPR*, 2019.

[19] W. Ye, C. Chen, Z. Wang, I.-H. Chu, and S. P. Ong, "Deep neural networks for accurate predictions of crystal stability," *Nature Communications*, 2018.

[20] P. G. Breen, C. N. Foley, T. Boekholt, and S. P. Zwart, "Newton vs the machine: solving the chaotic three-body problem using deep neural networks," *MNRAS*, 2019.

[21] F. Hase, S. Valleau, E. Pyzer-Knapp, and A. Aspuru-Guzik, "Machine learning exciton dynamics," *Chemical Science*, 2016.

[22] S. Wang, K. Fan, N. Luo, Y. Cao, F. Wu, C. Zhang, K. A. Heller, and L. You, "Massive computational acceleration by using neural networks to emulate mechanism-based biological models," *Nature Communications*, 2019.

[23] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *CVPR*, 2018.