2022

A NON-DETERMINISTIC DEEP LEARNING BASED SURROGATE FOR ICE SHEET MODELING

Hannah Jordan

A NON-DETERMINISTIC DEEP LEARNING BASED SURROGATE FOR ICE SHEET MODELING

By

HANNAH MICHELLE JORDAN

Thesis presented in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

The University of Montana

Missoula, MT

July 2022

Approved By:

Scott Whittenburg, Dean of The Graduate School

Graduate School

Douglas Brinkerhoff, Chair

Dept. of Computer Science

Jesse Johnson

Dept. of Computer Science

Andrew Ware

Dept. of Physics

Jordan, Hannah, M.A., Summer 2022 *Computer Science*
Abstract Title Chairperson: Douglas Brinkerhoff

Surrogate modeling is a new and expanding field in the world of deep learning, providing a computationally inexpensive way to approximate results from computationally demanding high-fidelity simulations. Ice sheet modeling is one of these computationally expensive models, the model used in this study currently requires between 10 and 20 minutes to complete one simulation. While this process is adequate for certain applications, the ability to use sampling approaches to perform statistical inference becomes infeasible. This issue can be overcome by using a surrogate model to approximate the ice sheet model, bringing the time to produce output down to a tenth of a second or less. In this paper, we introduce the use of a conditional variational autoencoder as a surrogate model for approximating an ice sheet model producing surface velocity predictions. For a fair comparison, we test both deterministic and stochastic approaches and discuss the drawbacks and benefits to both model types. We train a standard vanilla neural network architecture, a neural network architecture using dropout and normalization, and a neural network with added dimensionality reduction using principal component analysis. These surrogate models produce output that is representative of the high-fidelity data, but there is variability between the surrogate and high-fidelity model. This divergence cannot be determined for a deterministic model without further analysis such as model ensembling. The use of a stochastic network, such as the conditional variational autoencoder, provides a solution to this problem. This network provides us with a method to quantify the uncertainty within the surrogate using the model's natural stochasticity. This implementation has the potential to be applied across multiple fields because of the black box nature of the architecture.

1 Introduction

Ice sheet modeling provides important insight on the life cycle, physics and future changes to glaciers around the world. Glaciers and ice sheets slide as a result of the conversion of potential energy in the form of accumulated ice at high elevations. This occurs either through viscous dissipation within the ice itself, or by frictional dissipation at the interface between the ice and underlying bedrock or sediment. This is referred to as 'sliding' and is responsible for a majority of the observed velocity over much of the Greenland ice sheet (Maier et al., 2019). Variations in ice flow dynamics make up >50% of contemporary ice loss in Greenland (Mouginot et al., 2019), because of this it is crucial to develop a model that can produce sliding predictions with few errors. If the basal sliding predictions are off in a simulation, the resulting sea level rise projections will be uncertain as well. Modeling basal sliding remains one of the most significant open problems in glacier dynamics and a variety of approximations have been proposed. This problem stems from the difficulty associated with validating the models of sliding and hydrology due to lack of sufficient observational constraints on model parameters.

For this study, we work with an ice sheet model that is a spatially explicit and fully coupled system of partial differential equations. While this model provides us with a powerful tool for modeling glacial ice flow, it is computationally expensive, making downstream applications and parameter analysis infeasible. As an attempt to address the problem associated with the variability in the model parameters, downstream analysis can be used for parameter inference. One such strategy is to perform Markov Chain Monte Carlo (MCMC) to make an inference of the distributions of model parameters given observations. This method uses sequential sampling to construct a distribution, which requires sampled parameters to be passed through a model hundreds to thousands of times in order to construct an accurate representation of the parameter distribution. This distribution is superior to parameter values because we are able to quantify the variance in our results, providing insight into the certainty of the model. Without an efficient way to run the model, this application is computationally infeasible. One solution to this problem is the use of a surrogate model, allowing us to improve our runtime significantly when sampling sequentially.

Surrogate modeling is the process of using statistical modeling to approximate the solution to complex simulations. For a high-fidelity model $F(x)$, a surrogate is a function $G(x)$ such that $G(x) \approx F(x)$. This function is determined by training a machine learning model to produce output that is representative of the high-fidelity model. The surrogate is computationally less intensive than the original model, allowing for tractable downstream applications. A surrogate model applied in this application moves the computational expense out of the sampling and into the generation of training data, which is easily parallelized. This solu-

tion also provides easy access to approximate gradients and higher derivatives of model output with respect to input parameters. These powerful models can be used alongside high-fidelity models to assist in computations, used for parameter estimation and optimization, risk analysis and more. Surrogate models have been used in the field of biology, physics, geology and more and is an ever expanding field of deep learning. The use of artificial neural networks, Gaussian process regression, random forests and other architectures have been used to emulate high-fidelity models ranging from molecule generation to climate modeling.

One implementation uses a deep neural network to emulate the shortwave and longwave radiation parameterization in a climate model, the Super-Parameterized Energy Exascale Earth System Model (SP-E3SM) (Pal et al., 2019). This model is a super-parameterized version of the Energy Exascale Earth System (E3SM) climate model. The super parameterized version replaces the deep convective parameterization in E3SM which is a source of model biases. The SP-E3SM reduces these model biases, but is significantly more expensive than the original model, therefore a surrogate has been applied to model the radiation calculations to improve on the computational expense. These calculations were predicted with an accuracy of 90-95% and at 8-10 times faster than the original parameterization. Two dense, fully connected, feed-forward deep neural networks with three hidden layers containing 32 neurons per layer were used as a surrogate, one for the longwave calculation and the other for the shortwave.

Another application uses a two level hybrid neural network approach to create a surrogate for geological carbon sequestration modeling (Xiao et al., 2022). This study uses a deep convolutional autoencoder for nonlinear dimensionality reduction, then a linear transition unit is applied between the encoder and decoder to evolve the latent space from one time step to the next. The use of dimensionality reduction in combination with a neural network was used to create an ensemble surrogate model for MCMC analysis of the probability distribution of eight parameters used to model log-speed (Brinkerhoff et al., 2021). In order to assess the non-uniqueness within the model parameter solutions, a Bayesian approach is used to characterize the joint posterior probability distribution over the parameters. This distribution is not analytically tractable, so an MCMC approach is used with a surrogate model. A 5000 member ensemble of neural networks with added dimensionality reduction to reduce the free parameter space was used as the surrogate.

Another approach is through the use of physics-informed neural networks (PINNs). Raissi et al. (2019) introduce these methods, experimenting with using PINNs to model the Schrödinger, Allen-Cahn, Navier-Stokes and Korteweg-de Vries equations. These models are useful for solving high dimensional inverse problems and are able to retain high predictive accuracy even when the temporal gap between data points are large. Lütjens et al. (2021) use a PINN in combination with Polynomial Chaos Expansion (PCE) to

simulate a climate model for ocean modeling. These combined methods are used for uncertainty propagation of known parameter uncertainties. This model uses PINNs to aid in the computational speed up, while adding stochasticity by using PCE.

While the surrogate is much faster than the high-fidelity model, it is an approximation to the solution and we do not necessarily know how accurate the surrogate is when compared to the original model. If we are operating under the assumption that the high-fidelity model is a good approximation of reality, we would like to be able to account for the degree to which the surrogate deviates, as well as account for the extra variability in any downstream analyses using the surrogate. For our purposes, we are interested in producing error metrics alongside our surrogate estimations. Generative modeling, specifically the use of a variational autoencoder provides a solution to this problem. The variational autoencoder is a deep neural network that is trained to map input data to a latent space distribution which is then sampled from to produce output data, a reconstructed form of the input. The sampling process is randomized, providing us with a non deterministic network. This allows us to produce an infinite number of solutions from the prior distribution for the same input which can provide an insight into the performance of the network. Variational autoencoders (VAEs) have been used to generate images, text and audio and are a computationally inexpensive model that allow for a successful approximation of the probability distribution of a data set, but they do have multiple trade-offs including variance loss leading to image blurriness, disentanglement, the balancing loss between the KL Divergence and reconstruction loss and more (Singh and Ogunfunmi, 2021). These shortcomings are primarily based on VAE applications to images and applications to physical models may lead to different issues, but the assumptions of the VAE are likely to be more well suited towards physical applications. VAEs are a useful deep learning architecture, but they do not allow you to constrain what output will be produced. In the application to handwritten digits, a trained VAE will produce random handwritten digits when sampled from, but conditions cannot be placed on this output. To limit the control the output of the VAE, we can inform the model with added conditions. With the addition of conditions when both training and during generation, we can limit our sampling to the confined part of the latent space providing a means to only produce digits associated with a label. This form of a VAE is called a conditional variational autoencoder (cVAE) and has been useful in the field of surrogate modeling because of its ability to limit data generation.

cVAE's are recent in the surrogate modelling world and have been applied to a few new applications. KilonovaNet uses a cVAE to emulate radiative transfer simulations for the study of kilonovae spectra, the result of an astronomical event occurring upon the merging of two neutron stars or black holes (Lukošiūtė

et al., 2022). This implementation was successful because of its ability to be trained on spectra that has not been pre-processed and reduces the time required for parameter inference. The authors point out possible errors that stem from the data itself, which require further study of kilonavaes to quantify, as well as an unavoidable error that comes from using a cVAE due to data-compression. Despite these drawbacks, this application proved to be a successful surrogate, allowing for the quick analysis of kilonova candidates. Gabbard et al. Gabbard et al. (2021) used a cVAE for Bayesian parameter estimation for gravitational-wave astronomy. The cVAE was trained on binary black hole signals to produce Bayesian posterior probability estimates while avoiding computationally intensive Bayesian inference approaches. Physically appropriate decoder distributions were used for the output parameters and they utilized deep convolutional neural networks as the encoder, decoder and a 'recognition' encoder. Accuracy of the model is illustrated by plotting the posterior distribution of their model, VItamin, and the current standard. VItamin produces samples at a rate of $\sim$6 orders of magnitude faster than the authors benchmark studies. A problem arises from the difference between the noise the cVAE is trained on and the actual noise of the gravitational wave signals. Jiang et al. (2021) use a cVAE to create a crustal model that relates the Rayleigh surface wave velocity and the structure of the crust and upper mantle. The cVAE provides a good estimate of the non-linear relationship between the group velocity and crustal model. This study uses the cVAE to estimate model uncertainty without having to apply regularization. When compared to the benchmark crustal model, the cVAE model reveals more detail in the features and in general provides more subtle feature information. This model did run into problems with overfitting in the network.

In this work, we develop a cVAE surrogate to approximate an ice sheet model and compare with three deterministic neural network surrogates. We compare a total of four different deep neural network architectures in two categories: deterministic and non-deterministic. Three deterministic neural networks were tested with varying levels of complexity including the use of principal component analysis for dimensionality reduction. These three models have varying levels of accuracy, but all perform well as a surrogate. Our cVAE surrogate is stochastic and is the main focus of this study. This architecture serves as an adequate surrogate and provides a means for error quantification due to its non-deterministic generative nature. We begin by defining the architectures used, their implementations, our results and then discuss the implications of these results.

2 Methodology

A surrogate model is designed to approximate the function $Y = f(X)$ such that $Y_{predicted} \approx f(X)$. The function $f(X)$ is computed with a high-fidelity model that is computationally intensive. Figure 1 displays the goal of a surrogate model. We have a set of known points used as training examples, with the surrogate we are aiming to interpolate between these points in order to approximate our predictions. As seen in figure 1, this distribution is not exact, displaying the surrogate's error. For this application, we are using the surrogate to estimate the function computed with an ice sheet model, which requires $10 - 20$ minutes to generate surface velocity predictions given a set of eight physical parameters. The goal of our experiment is to perform a comparison of four neural network architectures used to emulate this high-fidelity model. The surrogate models are trained to output ice sheet model predictions given physical parameters as input. The surrogate applies nonlinear transformations to the physical parameters and maps to an output used to approximate the ice sheet model predictions. The goal of the model is to match the model predictions as closely as possible to the ice sheet model predictions. The derivation of each network is described below. $\mathbf{Y}$ is a vector of the generated ice sheet model log speed predictions and $\mathbf{X}$ is a vector of physical parameters used to generate these predictions. We abbreviate the ice sheet model as ISM and surrogate model as SM.

2.1 Dataset

The dataset used here is the same as that in Brinkerhoff et al. (2021); see that work for a more detailed description. In particular, we seek to emulate the mapping from 8 model parameters to a spatially distributed but temporally averaged field of glacier surface speeds via a fully coupled model of glacier flow and channelizing subglacial hydrology subject to a time-varying basal meltwater input discretized with the finite element method. To generate the dataset, we identified plausible upper and lower bounds for each of the 8 parameters (which represent physical quantities controlling dynamic friction between ice and bedrock, the characteristic scales of the subglacial hydrologic system, and the porosity of ice), and uniformly drew 5000 samples within these bounds using Latin Hypercube Sampling (Loh, 1996) to generate $\mathbf{X}$. We then ran the ice sheet model for 20 years, which was sufficient for non-physical transients related to incompatibilities in initial conditions to disperse, and averaged the model's predicted speed over the last five simulation years. The surface speeds for each of these simulations were collected into an array and the logarithm taken to form $\mathbf{Y}$. We work with the log of the velocities due to the large variability in the magnitude of the fields produced by the ice sheet model. We will henceforth refer to the surface velocity as the log-speed.

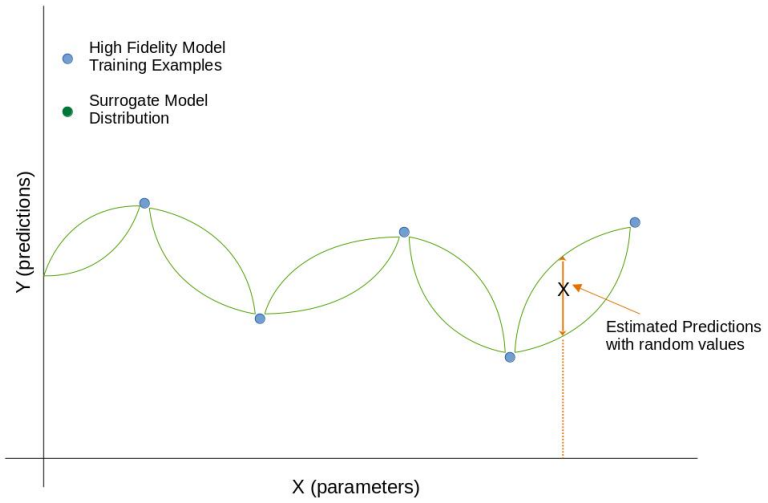


Figure 1: Surrogate model visualization shows the surrogate model distribution around the high-fidelity model training examples. This is a conceptual diagram, not visualization of actual data. The trained surrogate can then be sampled from using random values and the conditioned parameters. These predictions may or may not be accurate, therefore the ability to quantify the variability in these predictions is needed. This variability is displayed as the surrogate distribution at any value of X.

2.2 Deterministic Neural Network as a Surrogate

In order to compare our stochastic surrogate implementation we implemented three separate deterministic neural networks with varying levels of complexity for comparison against our main contribution. These networks are deterministic, and therefore will produce the same output for a set of parameters once trained. For downstream surrogate analysis, this requires further work, such as aggregation with bootstrap in order to determine the model's performance. These three SMs are fully connected neural networks, comprised of multiple layers of artificial neurons each with an associated weight and bias vector. Input is passed through and a linear transformation is performed at each layer. While training, these neurons are fine tuned so that the model outputs data that is as close to the training predictions as possible. There are many variations of the neural network and are the foundation of deep learning. We decided to test and implement three with varying levels of complexity for comparison against the cVAE SM, below we describe the derivation and implementation of these networks.

2.2.1 Network Architecture

The goal of a multi-layer perceptron (MLP) is to approximate a function f where $Y = f(X)$, X is input and Y is data. For our ISM, Y are ISM log-speed predictions and X are physical parameters. We use an MLP to define the mapping $Y = f^*(X;\theta)$ where f^* is the approximating function and θ are trained network parameters.

$$A^{(m)} = h^{(m-1)}W^{(m)} + b^{(m)} \tag{1}$$

$$h^{(m)} = \sigma(A^{(m)}) \tag{2}$$

where $h^{(m-1)}$ is the input, $W^{(m)}$ is layer weight matrix, $b^{(m)}$ is the layer bias vector, σ is the activation function and m is the current layer. These functions are applied for each layer of the perceptron and the transformed data is used as input for the next layer. These layers are comprised of nodes or neurons that are connected to the next layer. Every node within a layer is connected to all the nodes in the layer before and after it. These layers perform a set of sequentially applied functions on the input in order to output data that is representative of the training data.

This general structure follows for all neural networks. Each layer's weight matrix and bias vector are tuned during each step in the training process to accurately transform the input parameters to the desired output. We used a five layer neural net. For the output layer activation we used the identity, while all other layers used the Rectified Linear Unit (ReLU) activation, where

$$ReLU(X) = MAX(0, X). \tag{3}$$

2.2.2 Implementation

For the multi-layer perceptron surrogate, we tested several different configurations: a vanilla network, a neural network with dropout and normalization, and another that used a form of principle component analysis (PCA) to perform dimensionality reduction which reduces the number of learnable parameters. All networks were implemented in Pytorch (Paszke et al., 2019).

Figure 2 displays the general architecture of our neural networks without PCA. The SMs consist of five linear transformation layers whose output is denoted by $h^{(m)}$. The first nodes, $h^{(0)}$, are the input parameters with dimension $n = 8$. The first layer maps from this input dimension to 128 nodes. The results from these

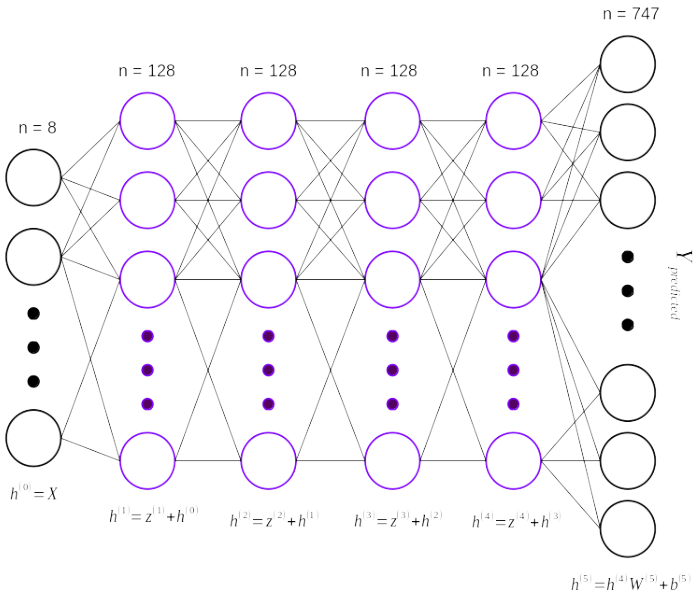


Figure 2: General deep neural network architecture used as a surrogate in this study. Our vanilla and dropout networks follow this architecture with variation within each hidden layer.

transformations are then used as input for the next layer. This process is repeated for layers two, three and four, which do not change dimensions. The final linear layer maps from 128 to 747 nodes, the dimension of our ISM predictions. The final layer transformation is not passed through the activation function and directly returned as $Y_{predicted}$. This basic schema follows for all networks. Below we outline the differences between the three models.

Vanilla Neural Network. Our vanilla network is a simple neural network implementation. Referring to figure 3, we see the internal architecture of each linear layer for our network. This is a standard set of transformations for any fully connected neural network. We first pass input through a linear transformation and get our activation vector, $a^{(m)}$. This value is then passed through our activation function, ReLU, outputting the transformed data that will be useful for the next layer, $z^{(m)} = ReLU(a^{(m)})$. The final step is to add the output from our last transformation to our current layer in order to relate these layers, $h^{(m)} = z^{(m)} + h^{(m-1)}$. These steps are repeated for layers one through four. The last layer does not pass the transformed product through the activation function, but instead returns $h_{(5)} = h^{(4)}W^{(5)} + b^{(5)}$. This is Y_{pred}, the predicted output. This last layer must learn the free parameters directly from the data, this requires determining the key points among 747 data points.

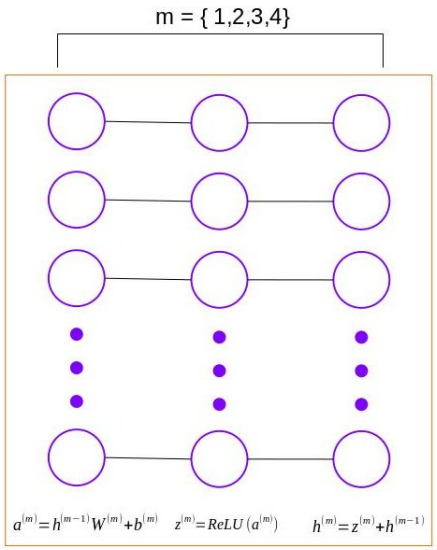


Figure 3: Hidden layer architecture for the vanilla neural network surrogate. This consists of a simple set of transformations standard in most neural networks: linear transformation and activation.

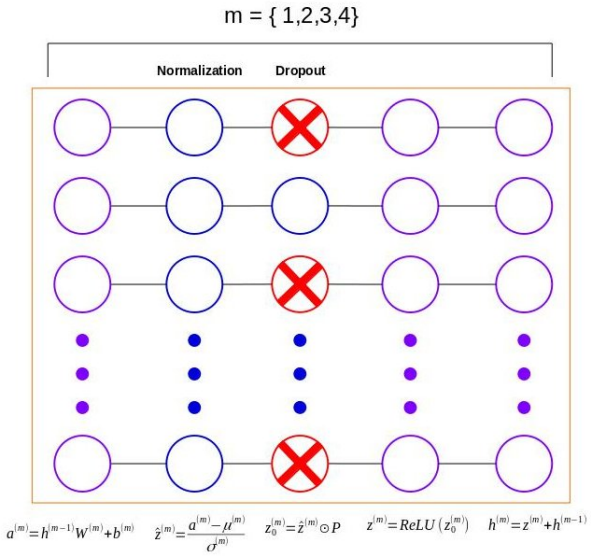


Figure 4: Hidden layer architecture for the dropout neural network surrogate. This is is similar to the vanilla surrogate, with added layer normalization and dropout.

Neural Network with Dropout and Normalization. The next neural network has the same layout as the vanilla neural network, but includes dropout and normalization. We initialize four normalization functions for layers one through four. The layer architecture is visualized in figure 4. The LayerNorm class from PyTorch applies layer normalization over the input. This is done by computing the mean and standard-deviation and returning the normalized input. This class uses Layer Normalization (Ba et al., 2016), which reduces the time to train and is independent of mini-batch size like Batch Normalization (Ioffe and Szegedy, 2015). Batch normalization has little to no effect when the batch size is small, this problem is avoided in layer normalization by applying the normalization on the neuron for a single instance across all features rather than applying normalization on batches. Layer normalization fixes the problem that arises when the changes of one layer causes correlated changes to the next layer by fixing the mean and variance to $\mathcal{N}(0, 1)$ of the summed inputs within each layer.

$$\hat{z}^{(m)} = \frac{a^{(m)} - \mu^{(m)}}{\sigma^{(m)}} \tag{4}$$

Each layer will be normalized after being passed through the linear transformation, excluding layer 5. Once the data is normalized, we apply dropout to the activation vector

$$z_0^{(m)} = \hat{z}^{(m)} \odot P \tag{5}$$

where $\hat{z}^{(m)}$ is the normalized activation vector, $\odot$ is the element-wise product between matrices, and P is a vector of normally distributed random variables with a mean of p. $p = \{0.0, 0.2, 0.5, 0.5\}$ for layers $\{1, 2, 3, 4\}$ respectively. During training, dropout randomly zeroes some of the input tensor elements. In our case, we pass in our $a_{(m)}$ values and the same output is returned, but with some of the nodes deactivated. This is to prevent nodes from becoming dependent on each other, which can lead to overfitting. We use PyTorch's Dropout layer which performs random dropout, removing a percentage of the features which has been shown to greatly reduce overfitting (Hinton et al., 2012) by randomly setting a specified percentage of neurons within a hidden layer to zero. This prevents neurons from becoming dependent on one another and swaying the produced output to be fitted to the training data. The final two transformations are consistent with the vanilla net, but rather than passing the activation vector into ReLU, we pass in our transformed activation vector.

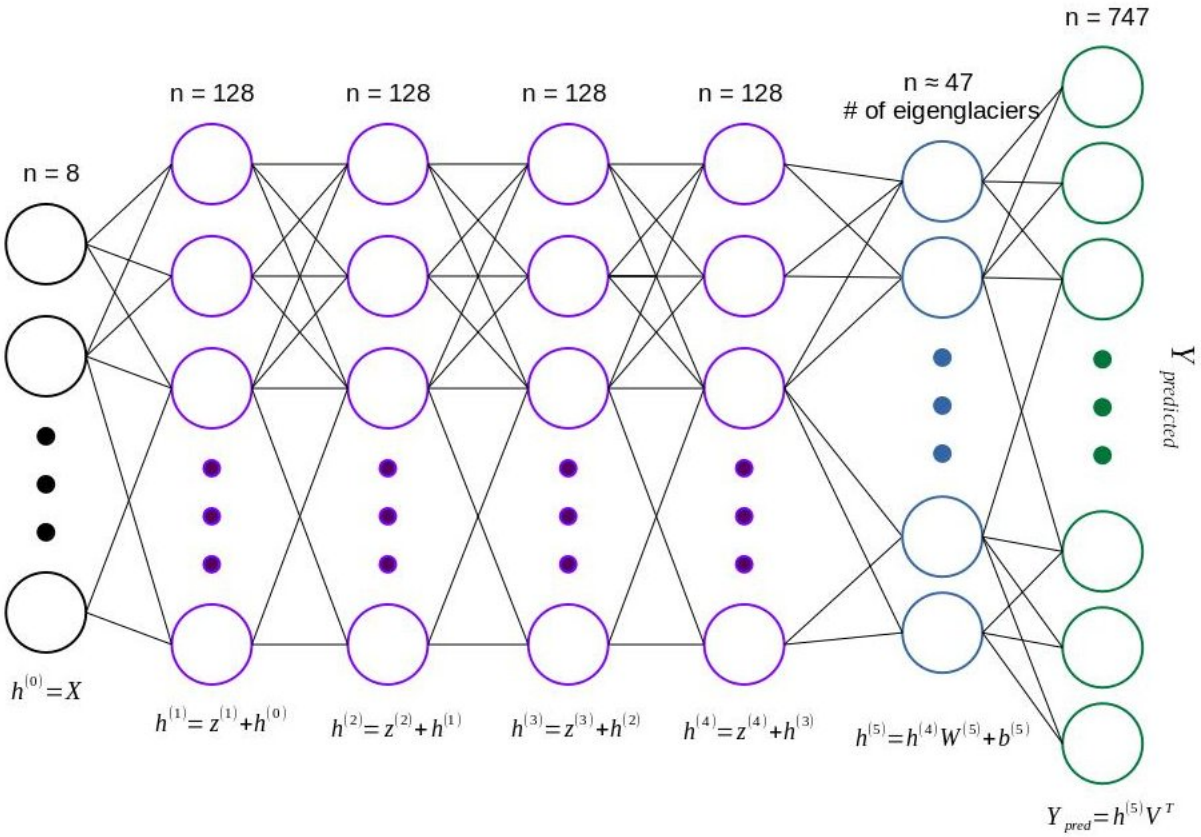


Figure 5: Eigenglacier neural network surrogate architecture consisting of hidden layers matching figure 4 with added dimensionality reduction. There is an extra linear transformation added to project to the ice velocity space using PCA.

Neural Network with Eigenglaciers. Our final neural network uses PCA for dimensionality reduction and is based on Brinkerhoff et al. (2021). This process of dimensionality reduction determines the optimal basis for mapping from learned coefficients within the last hidden layer to the spatially explicit ice velocity space. The last hidden layer in the network learns the mapping from some coefficients back to the ice velocity space, meaning that this final layer contains 747 free parameters, consistent with our output dimensions. With PCA we are able to come up with the optimal basis for mapping to that space without any free parameters because this is determined outside the scope of our SM. This should allow our model to learn faster and be less susceptible to overfitting. These eigenglaciers are a reduced set of the data that represent the full set of data. This is done by computing the eigendecomposition:

$$\mathcal{S} = V \Lambda V^T \tag{6}$$

where Λ is a diagonal matrix of eigenvalues whose diagonal entries represent the variance in the data. The columns of V are the eigenvectors of the covariance matrix S where

$$\mathcal{S} = \sum_{i=1}^{X} \omega_{d,i} \left[\log_{10} \mathcal{F}(\mathbf{X}_i) - \log_{10}(\bar{\mathcal{F}}) \right]^2 \tag{7}$$

where ω_d is a vector of randomly sampled weights that sum to 1 and

$$\log_{10}(\bar{\mathcal{F}}) = \sum_{i=1}^{X} \omega_{d,i} \log_{10} \mathcal{F}(\mathbf{X}_i). \tag{8}$$

We are able to simplify the data by assessing the fraction of the variance in the data still unexplained after representing it with the number of eigenglaciers, j.

$$f(j) = 1 - \frac{\sum_{i=1}^{j} \Lambda_{ii}}{\sum_{i=1}^{m} \Lambda_{ii}} \tag{9}$$

We determine how many eigenglaciers to retain by computing a cutoff threshold $c = max_j \in \{1, ..., m\} : f(j) > s$ where $s = 10^{-4}$. For this SM, $c = 51$, but ranges $47 < c < 51$. We can then approximate any log-speed field as

$$\mathcal{F}(\mathbf{X}) \approx \sum_{j=1}^{c} \lambda_j(\mathbf{X}) V_j \tag{10}$$

where V_j is the j-th eigenglacier and λ_j is its coefficient.

This model has the same architecture as our Dropout NN, but the last layer maps from 128 nodes to the number of eigenglaciers and an extra step is employed after the last linear transformation, see figure 5. Our final transformation is as follows

$$Y_{SM} = z^{(5)}\hat{V}^T \tag{11}$$

where Y_{SM} is the predicted log-speed output from our SM. This maps the data from the number of eigenglaciers back to the dimension of our ISM log-speed predictions.

2.2.3 Training

Now, we need to have a way to train our SMs. In this case, we seek to find the function $f^*(X)$ such that Y_{SM} and Y_{ISM} are as similar as possible. This is done by computing the loss, or $\mathcal{L}(\theta)$, $\theta = \{W_1, \ldots, W_5, b_1, \ldots, b_5\}$, then performing gradient descent. There are many different ways to compute the loss, but you must be able to take the gradient in order to perform gradient descent. Our implementation uses mean square error loss, the sum of the difference between predicted and observed data squared.

$$MSE = \frac{1}{n}\sum_{i-1}^{n}(Y_i - Y_{pred_i})^2 \tag{12}$$

The parameter and log-speed vectors from our high-fidelity model are passed through each model and the predicted log-speeds are output. We calculate the loss between the ISM prediction and our SM prediction and increment our optimizer for a total of 3000 epochs with a batch size of 128. An epoch is complete once all training data has been passed through the SM once, and the output evaluated and the SM adjusted. We split our training data into batches of size 128 for performance improvements, rather than storing values for the full set, we work with batches then update weights and biases after all data has been seen. We use the Adam optimizer (Kingma and Ba, 2014) to optimize the objective.

2.3 Stochastic Neural Network as a Surrogate

The last surrogate tested and the primary focus of this work was a cVAE. The cVAE is a VAE that is conditioned on a set of parameters. This allows for more controlled sampling, as well as downstream applications. cVAEs are a part of the variational inference family and are a generative model, producing samples from $P(Y|X)$ rather than a deterministic $Y = f(X)$ as with our previous models. The key difference between a VAE and cVAE is that a VAE samples from $P(Y)$ where the cVAE samples from $P(Y|X)$, taking into

account the conditions X. With this stochastic nature, we are able to continuously sample from the latent space to determine the divergence of the surrogate from the high-fidelity model.

The basic architecture of a cVAE includes an encoder and decoder neural network, see figure 6. Data is passed through the encoder which compresses the data into the latent space. The latent space is then decompressed through the decoder, outputting predicted data values. These values are passed through a loss function and the encoder and decoder neural networks are optimized so that the produced values match closely with observed.

2.3.1 Derivation of Conditional VAE

The goal of our model is to determine the probability of the output ISM predictions occurring given the physical parameters, or $P(Y|X)$. The ISM is deterministic as it is computing the solution to a function, $Y = f(X)$ where Y are model predictions, such as log-speed in this case, and X are the physical parameters. For every value of X, the produced Y values are constant. A SM is an approximation of $Y = f(X)$. This model does not perfectly approximate the original function, and we account for this by allowing the surrogate to produce random output from the distribution $P(Y|X)$. How do we go about finding $P(Y|X)$? The cVAE computes an approximate sample from $P(Y|X)$ and if enough samples are taken, the model eventually converges to the distribution, which is characterized in the sense of a histogram by looking at a large number of samples. In order to compute these samples, we need to find a way to approximate the unknown distribution. We need to ensure that our model is representative of the dataset, this is to say that for every data point, Y, there is at least one setting of the latent space that will cause the model to generate data similar to Y. We are aiming to maximize the probability of each Y given some condition X:

$$P(Y|X) = \int P(Y, z|X)\, dz \tag{13}$$

Using the product rule

$$P(Y|X) = \int P(Y|z, X)P(z|X)dz \tag{14}$$

where $P(Y|z, X)$ is the distribution of the ISM predictions given the latent space and physical parameters, represented by the Decoder. $P(z|X)$ is the latent space prior distribution given the physical parameters. The Encoder is used to efficiently train the model, and maps the data to the latent space. When samples are generated, we sample directly from $P(z|X)$ and run the resulting latent vector along with the conditions

through the encoder.

We are aiming to find a way to sample from $P(Y|X)$, as it stands now this is intractable. The first problem we overcome is how to choose the latent space such that it is representative of the latent information. All VAEs make the assumption that the latent space can be represented by sampling from a simple distribution, in most cases $\mathcal{N}(0, I)$. This is done by training an encoder that maps from n-dimensional data to a set dimension latent space, which we assume to be normally distributed a priori. Sampling from this latent space thus produces an encoded set of data that is representative of the original data. Now we need to sample from $P(Y|z, X)$, but need to do so in a way that is not computationally intensive. The distribution can be directly sampled from, but in high dimensional spaces, the number of samples needed to represent the distribution could be large. To avoid this problem, we apply the knowledge that for most values of z, $P(Y|z, X)$ will be nearly zero. We want the cVAE to sample values of z that are likely to have produced Y given the conditions X.

Rather than computing $P(Y|z, X)$ for z, we will instead propose a new function $Q(z|Y, \theta)$ where θ are learned parameters. We seek to maximize $P(Y|X)$ with respect to the parameters of this approximating distribution. This can be equivalently accomplished by minimizing the Kullback-Leibler (KL) divergence between $P(z|Y)$ and $Q(z|Y, \theta)$), or $\mathcal{D}[Q(z)||P(z|Y)]$. The KL-divergence (Joyce, 2011) measures the similarity between two distributions, and was used to train our model to match $P(Y|X)$ as closely as possible. This allows us to compute $E_{z\sim Q}P(Y|z, X)$ without oversampling. We will train this proposed distribution in order to maximize the similarity between $E_{z\sim Q}P(Y|z, X)$ and $P(Y|X)$.

To compute the relationship between $E_{z\sim Q}P(Y|z, X)$ and $P(Y|X)$, we first look at the KL-divergence, or $\mathcal{D}$, between $P(z|Y)$ and $Q(z)$. We will ignore the conditions X for the time being.

$$\mathcal{D}[Q(z)||P(z|Y)] = E_{z\sim Q}[\log \frac{Q(z)}{P(z|Y)}] \tag{15}$$

As it stands, we are unable to compute the KL-divergence and we must expand this equation to bring it to a computable form, which is called the evidence lower bound (ELBO). Expanding the expectation out we get

$$\mathcal{D}[Q(z)||P(z|Y)] = \int Q(z)\ \log \frac{Q(z)}{P(z|Y)} dz \tag{16}$$

We take all conditionals with respect to $Q(z)$.

$$\mathcal{D}[Q(z)||P(z|Y)] = \int Q(z)\ \log \frac{Q(z)P(Y)}{P(z,Y)} dz \tag{17}$$

Which then expands to

$$\mathcal{D}[Q(z)||P(z|Y)] = \int Q(z) \left(\log P(Y) + \log \frac{Q(z)}{P(z,Y)} \right) dz \tag{18}$$

Removing terms not dependent on $Q(z)$, we reduce the equation to

$$\mathcal{D}[Q(z)||P(z|Y)] = \log P(Y) - \int Q(z)\ \log \frac{P(z,Y)}{Q(z)} dz \tag{19}$$

Now, we have an equation with an integral with terms that we know, $P(z,Y) = P(z)P(Y|z)$, and because we are minimizing the KL-divergence over $Q(z)$, $\log P(Y)$ is a constant with respect to $Q(z)$. This gives us the ELBO

$$\mathcal{D}[Q(z)||P(z|Y)] = \log P(Y) - ELBO. \tag{20}$$

This displays the lower bound on the log of the evidence, $\log P(Y) > ELBO(Q)$

$$ELBO(Q) = \int Q(z)\ \log \frac{P(z,Y)}{Q(z)} dz = E[\log P(Y,z) - \log Q(z)], \tag{21}$$

with expectation taken with respect to $Q(z)$.

The ELBO is the core of variational inference. This equation is the general basis of any VAE model and allows us to compute a loss in order to train the model. To apply it to our conditional application of the VAE, we will simply impose our physical parameters on the model.

$$ELBO(Q) = E[\log P(Y,z|X) - \log Q(z|X)] \tag{22}$$

Applying the product rule on $\log P(Y,z|X)$

$$logP(Y,z|X) = \log P(Y|z,X)P(z|X) = \log P(Y|z,X) + \log P(z|X) \tag{23}$$

$$ELBO(Q) = E[\log P(Y|z,X)] + E[\log P(z|X) - \log Q(z|X)] \tag{24}$$

where data loss $= E[\log P(Y|z,X)]$ and prior loss $= E[\log P(z|X) - \log Q(z|X)] = \mathcal{D}[Q(z|X)||P(z|X)]$.

This reduces the ELBO to a sum of the data loss and prior loss.

$$ELBO(Q) = \text{dataloss} + \text{priorloss} = E[\log P(Y|z,X)] + KL[Q(z|X)||P(z|X)] \tag{25}$$

To compute the dataloss we can avoid sampling through the entire prior space by taking one sample of z and treating the corresponding value of $P(Y|z,X)$ as an approximation for the expectation. This is a standard in variational inference (Blei et al., 2017).

We propose a distribution $Q(z|Y,X)$ such that we can perform stochastic gradient descent in order to train the model. This is typically done by saying that $Q(z|Y,X) = \mathcal{N}(z|\mu(Y;\theta), \Sigma(Y;\theta), X)$ where μ and Σ are arbitrary deterministic functions with parameters θ, for example: neural networks. These functions are designed to encompass the mean, μ and standard deviation, Σ, of the distribution based on the data and physical parameters. We then assume $P(z|X) \approx \mathcal{N}(0, I)$, therefore reducing the KL-divergence to the comparison between two multivariate Gaussian distributions, both of which being easily tractable.

$$\mathcal{D}[\mathcal{N}(z|\mu(Y;\theta), \Sigma(Y;\theta), X||\mathcal{N}(0, I)] \tag{26}$$

2.3.2 Implementation

The cVAE is composed of a series of simple neural networks that take data and conditions as input and outputs predicted data. A cVAE has three main components: Encoder, Latent Space, and Decoder.

The Encoder takes a vector of ISM predictions and the corresponding vector of physical parameters. These two vectors are concatenated together then passed through the encoder. The encoder contains four linear transformation layers. The first layer maps from $N = n_{datapoints} + n_{parameters} = 755$ to 512 features. Next we map down to 256 features. The output from layer one and layer two are passed through the ReLU activation function. Now, we want to have a way to sample from the latent space and in order to do so, the last two layers of our encoder neural network will both map from 256 features to our latent dimension. This can be set to any value, our model mapped to 40 latent variables. We will call these two layers μ and Σ and they will be trained to encapsulate the mean and standard deviation of our proposed distribution. Our encoder outputs μ and Σ which will be used to sample from the latent space.

$$\begin{pmatrix} \mu_z \\ \Sigma_z \end{pmatrix} = Encoder(Y, X) \tag{27}$$

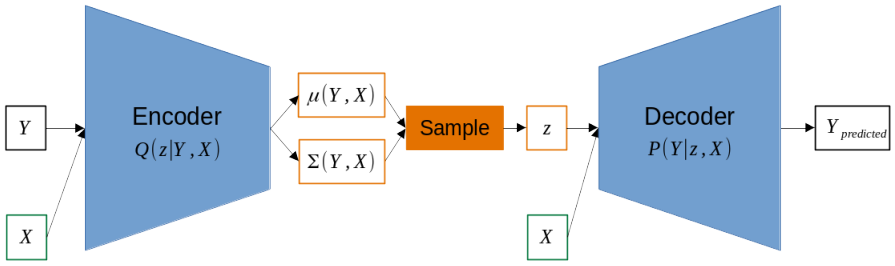


Figure 6: Conditional variational autoencoder network architecture which consists of an encoder which outputs μ and Σ distributions that characterize the data distribution. These distributions are used to sample from the latent space, which is then passed through the decoder along with the conditions X. The decoder outputs a set of data predictions.

Now that we've mapped our ISM predictions and parameters, Y and X, into an encoded latent space with reduced dimensions, we need to generate new predictions with our SM, $Y_{predict}$. This is where the decoder comes in. The cVAE Decoder accepts z, a sample from the latent space, and X the physical parameters, or conditions. We compute z using the following equation:

$$z = \mu_z + \epsilon\Sigma_z^{\frac{1}{2}} \tag{28}$$

where $\epsilon \sim \mathcal{N}(0, I)$. This reparameterization trick is performed in order to back-propagate through the layer that samples z. Without ϵ, $z = \mu + \Sigma$ is just a random sampling and is therefore a discrete operation, with ϵ this becomes a continuous operation.

The decoder is again made up of a series of linear transformation layers, and is similar in structure to the encoder, but rather than condensing the data it decompresses. There are three layers in the decoder. The first maps from the latent dimension, 40, to 256 features. The second maps from 256 to 512 features, and the last layer maps from 512 features back to $n_{data-points}$, 747. Again, the first and second layer outputs are passed through the ReLU activation function. The output of the last linear layer is returned as $Y_{predict}$.

$$Y_{predict} = Decoder(z, X) \tag{29}$$

So far we have approximated $P(Y|X)$ using an encoder, decoder and by sampling from the latent space. With our generated model predictions, we need to have a way of determining how well our model performed. This is done by computing the ELBO 25.

The prior loss, $KL[Q(z|X)||P(z|X)]$ is computed as follows using the results from our encoder network.

$$KL[Q(z|X)||P(z|X)] \approx -\frac{1}{2}\sum_{i=1}^{n}[1 + \Sigma_i - \mu_i^2 - e^{\Sigma_i}] \tag{30}$$

The data loss, $E[\log P(Y|z, X)]$ is computed as a standard data loss, for this application we used L2 normalization to determine the loss between our SM and ISM predictions.

$$E[\log P(Y|z, X)] = \sum_{i=1}^{n}[(Y - Y_{predict})^2] \tag{31}$$

2.3.3 Training

The cVAE model was trained on 80% of the 4,133 ice sheet model velocity predictions, the other 20% were reserved for testing. The model maps to a latent dimension of 40 and was trained over 3000 epochs with a batch size of 128. For each epoch, the model is trained on the training dataset. The ISM log-speed predictions and physical parameters are passed to the cVAE. The SM predictions, μ and Σ are output. The prior loss 30 is computed with μ and Σ and the data loss 31 computed then the two are summed together. Gradient descent is performed using Adam (Kingma and Ba, 2014).

3 Results

To validate the performance of our SMs we use the coefficient of determination, or R-squared, to measure the quantitative similarity between the SM and ISM log-speed predictions. This provides a mode of determining how close the model's predictions are to the desired values. R-squared is defined as the explained variation over the total variation. A score of 1.0 represents a perfect match between the predictions and the desired value. The value can be negative, which means that the model is arbitrarily worse. A score of 0.0 represents a situation where the model is always predicting the average output without taking the input into account. The coefficient of determination is computed as follows, and was implemented using SciKit Learn's r2_score

Model	Runtime (s)
Vanilla	120.1924
Dropout	164.1810
Eigen	182.7690
cVAE	169.4615

Table 1: Time to train each surrogate network over 3,000 epochs.

function (Pedregosa et al., 2011)

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{32}$$

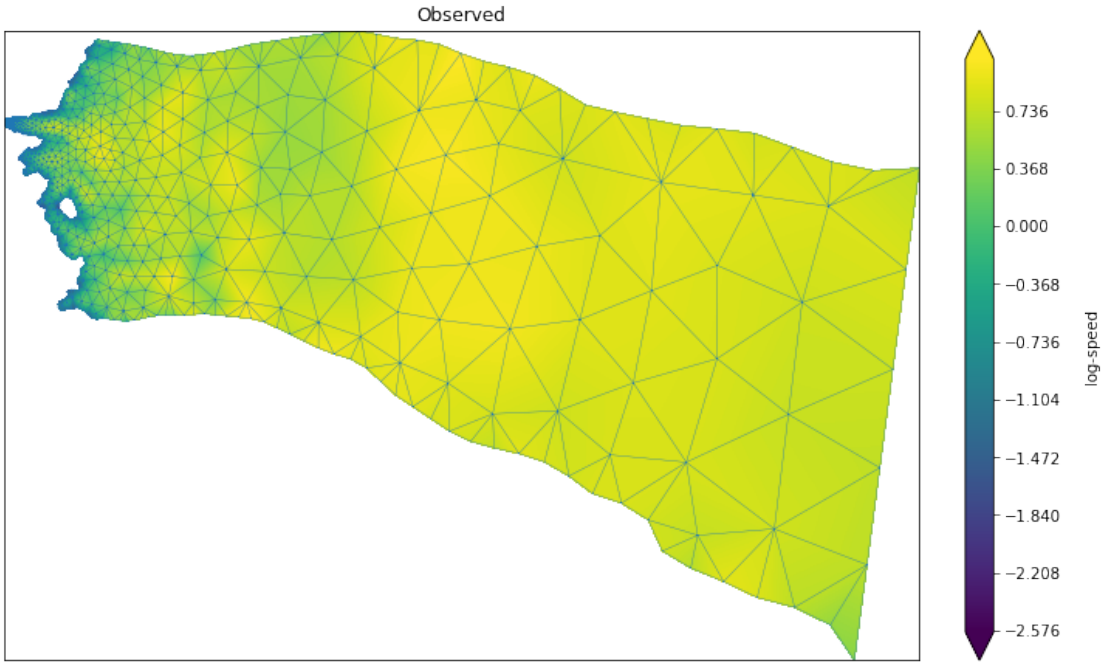
where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \epsilon_i^2$.

All models were tested with the same test data set. A total of 827 simulations were used for testing, out of the 4,133 total ice sheet model predictions. To draw from the deterministic surrogate, parameters were input and the log-speed predictions were output. To draw from the stochastic surrogate, parameters were input alongside a random sampling from the latent space, $z \in \mathcal{N}(0, 1)$ to generate log-speed predictions. Within the training and test set of predictions, there are simulations that are not physically realistic which produced log-speed values that are too high relative to reality by several orders of magnitude. We will discuss the implications of this.

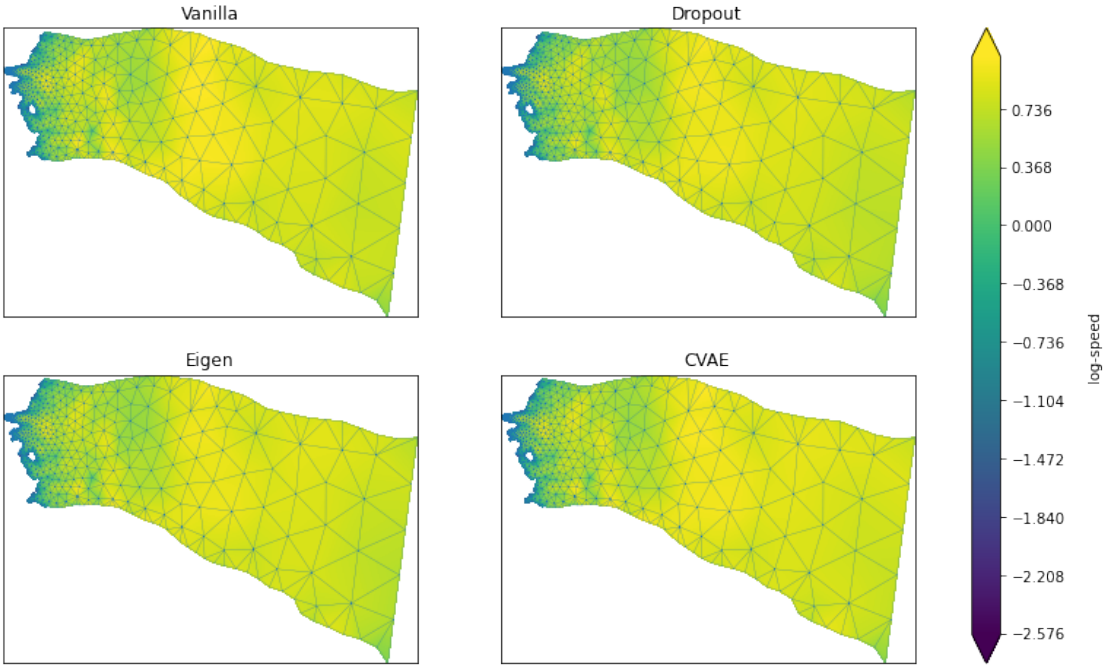
The times required to train the SMs were recorded and are displayed in table 1. Once the SMs were trained, this test data was passed through our SM and the predicted output was returned. The data is output in a vector of size 747, each index mapping to a coordinate on the glacier. Initial analysis consisted of producing plots, see figure 7, and comparing the predicted display to the ISM output.

All of our models performed well on a majority of the testing data. Our Eigen and cVAE models appear to better represent more of the subtlety in the dataset, but this is dependent on the input parameters. To get a good idea of the dataset and where the models performed the best and where they performed poorly, we inspected a random sampling from our results. Figures like figure 7 were created and evaluated. Looking at figure 7, you can see the output of all four models and the observed log-speeds, we refer to the ISM predictions as the observed for simplicity. Each point represents a coordinate on the glacier, and the color denotes the log-speed at that point.

The following figures display the R-Squared values for a subset of 784 of our test set results. For each SM, we compute the R-squared between the SM and ISM predictions. Figure 9 shows the R-squared distribution

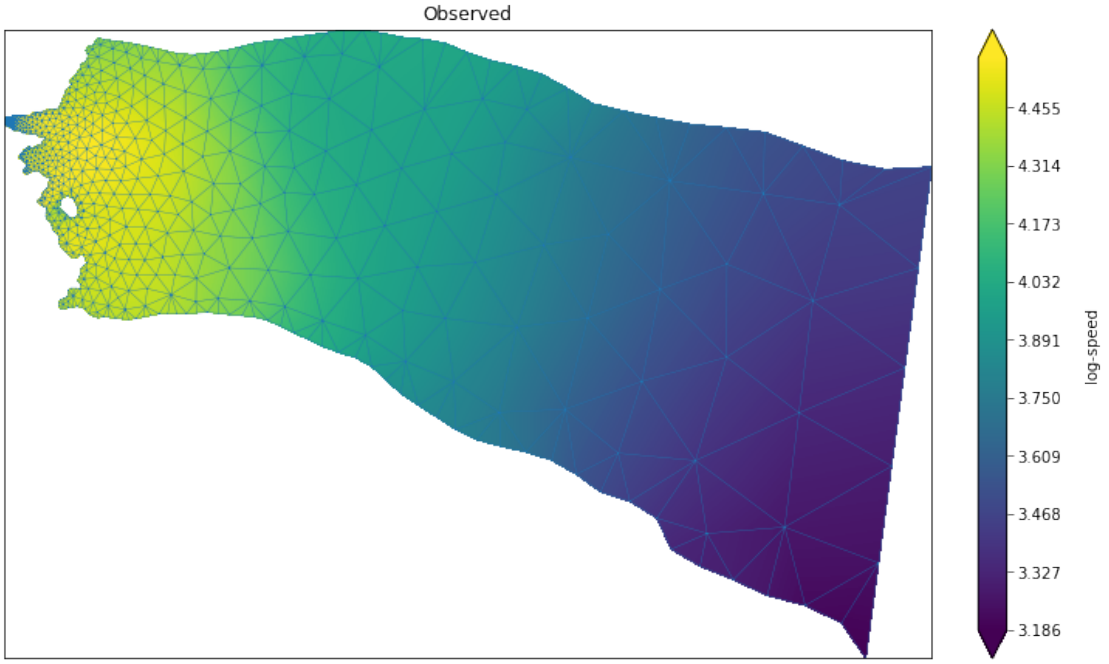


(a) High-fidelity log-speed output.

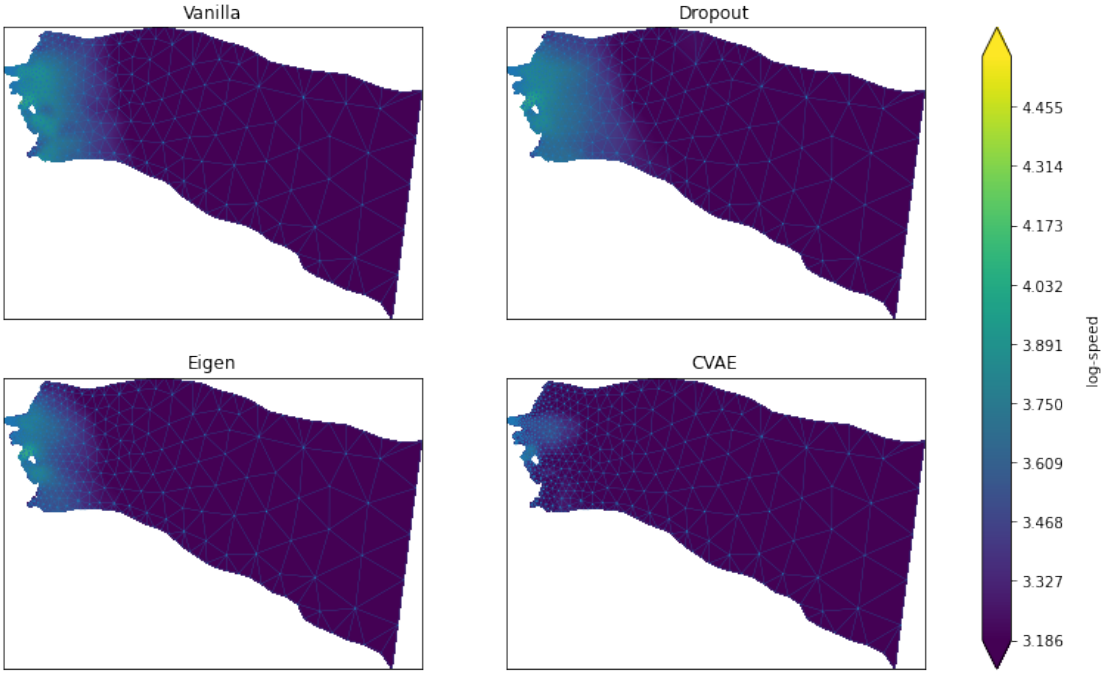


(b) Surrogate model output for Vanilla NN, Dropout NN, EigenGlacier NN and cVAE

Figure 7: Log-speed comparison for the following set of physical parameters: [-0.5762, -0.7871, -0.1270, -0.4546, 7.3049, -0.8897, -0.9761, -3.6475]. All models performed the best for this simulation. This set of predictions is a fairly physically realistic simulation as it has physically reasonable estimates of velocities. The R-squared values for Vanilla, Dropout, Eigen and cVAE are as follows: [0.9990, 0.9985, 0.9986,0.9994].



(a) High-fidelity log-speed output.



(b) Surrogate model output for Vanilla NN, Dropout NN, EigenGlacier NN and cVAE

Figure 8: Log-speed comparison for the following set of physical parameters: [-1.1392, -3.2271, -0.7134, 0.7272, 4.7162, -0.2786, -0.9831, -2.0378]. This set of predictions is an example of unrealistic simulation results, as you can see, the velocities are an order of magnitude greater than the other simulations. All models performed the worst for this simulation, as expected, with R-squared values for Vanilla, Dropout, Eigen and cVAE are as follows: [-6.7868, -7.0688, -11.2620,-21.1166].

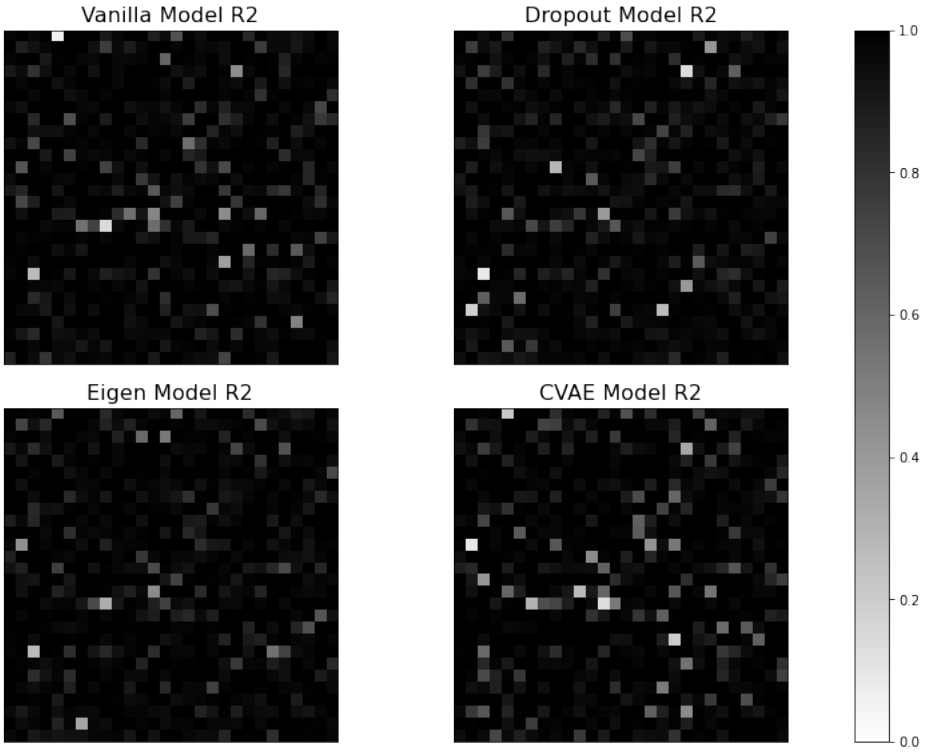


Figure 9: R-Squared values for the simulations produced by the four SM's. The figure displays a random sampling of 784 of the 827 tested simulations. R-squared values less than 0 are not included. This displays the distribution of R-squared values for the test data set, demonstrating that most simulations are approximated with accuracy by all four SM's.

Model	Average R^2
Vanilla	0.8590
Dropout	0.9040
Eigen	0.9090
cVAE	0.8110

Table 2: R2-score value for each surrogate averaged over the entire test set.

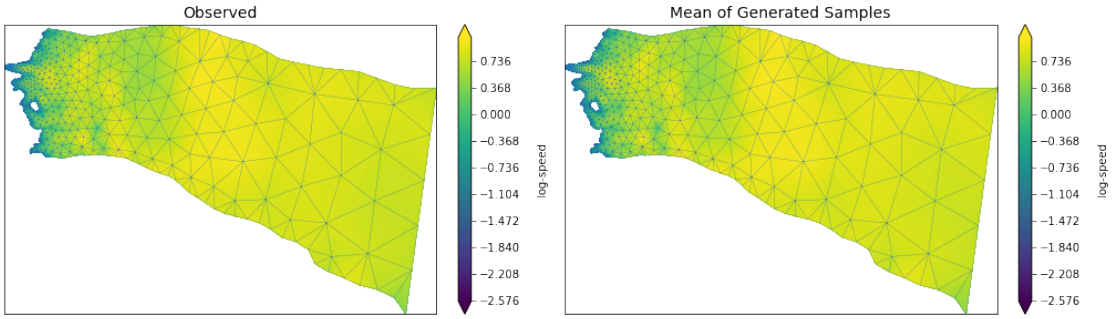
for our test set. The SMs produce output that is representative of the ISM log-speeds given the parameters for a majority of the test data. All four SMs appear to perform with similar accuracy. From this, we pose the questions: How do we know the SM is doing well? and Can we quantify the surrogates uncertainty?

This query leads us to the cVAE. With the autoencoder, we are able to generate new data and determine the variance of the SM outputs when we randomly sample from the latent space. To quantify our cVAE surrogate performance, we take a vector of parameters, a random sampling from our latent space, and pass these values through the trained decoder. The output is stored, the process repeated 100 times and the mean and variance of the data computed. Figure 10 displays the results of this experiment for the same parameters input to produce the results viewed in figure 7b, demonstrating one of the SMs highest scoring predictions. Figure 11 displays the results for the same parameters as 8b, one of the SMs poorest scoring predictions.

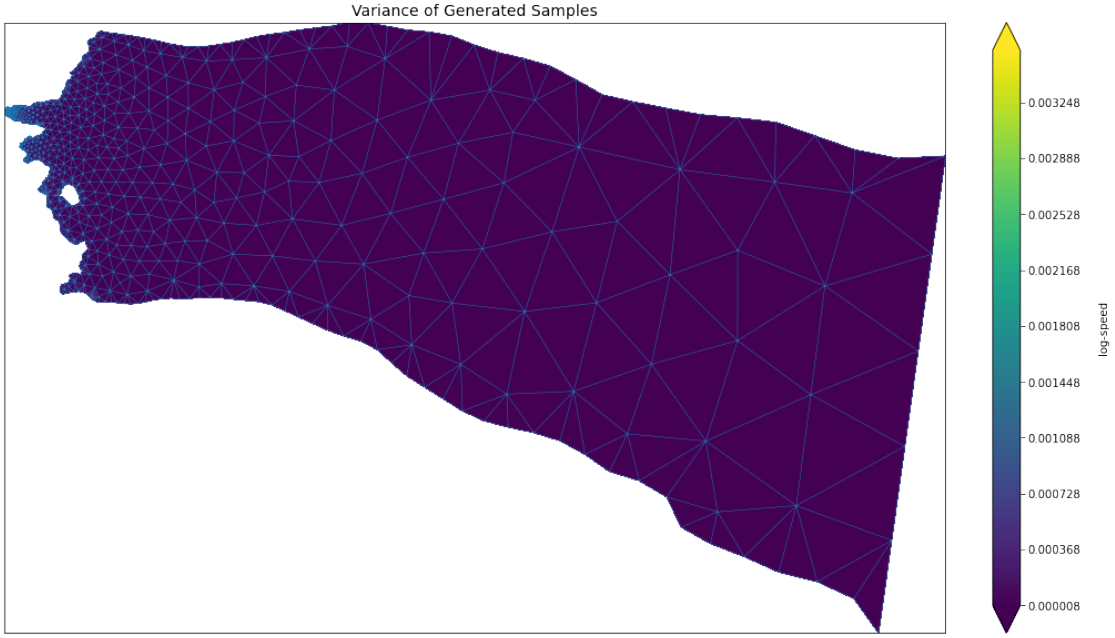
4 Discussion

This study provides useful insight on using a surrogate model to predict the output of a high-fidelity ice sheet model. All four of the SMs that were trained performed well. On average, the SM matched the ISM predictions with an accuracy of 80% and greater. We define the accuracy as the R-squared score between the generated and the high-fidelity predictions, an average R2 score of 0.8 therefore will be referred to as having an accuracy of 80%. This average includes all predictions from the reserved test set, some of which are not physically feasible. All of these SMs train in a minimal amount of time, and produce output at a rate of 4 orders of magnitude faster than the ISM with the surrogate requiring 0.01 seconds on average and around 20 minutes for the high-fidelity model. With a less computationally intensive way to make these predictions, any of these models can be used in a downstream application requiring consecutive, non parallel predictions. These results allow us to conclude that any of these models would be an appropriate surrogate to use to predict ISM output and with this training data we were able to develop multiple successful SMs, but the cVAE surrogate provides us with more insight into its divergence from the high-fidelity model.

From our research, there is not a clear surrogate architecture that is more accurate with its predictions, but each model did have its own benefits. The vanilla neural network was the easiest to implement. This network took little time to develop and did not require fine tuning, it almost immediately produced viable predictions. The vanilla model produced log-speed predictions with an average accuracy of 85.9% on the test set. This was the second worst average, but still good when considering the lack of overhead for this

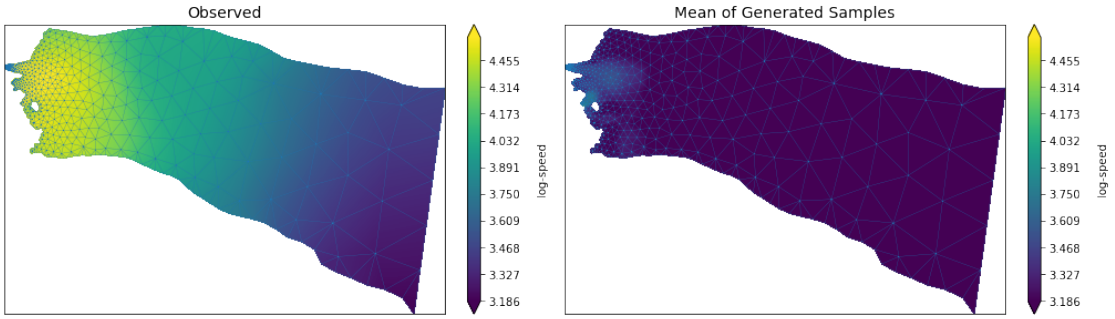


(a) High-fidelity predictions compared to the mean of the generated surrogate predictions. The mean of the predicted values have an R-squared score of 0.9995.

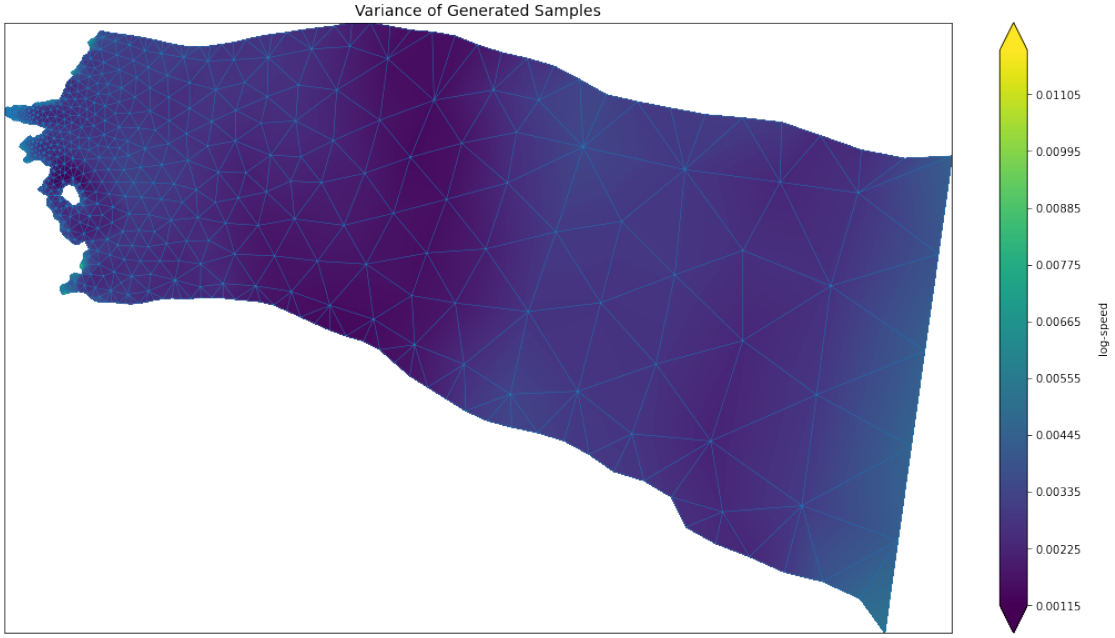


(b) Variance of generated predictions.

Figure 10: Mean and variance comparison for 100 generated log-speed predictions using the set of physical parameters used to display figure 7.

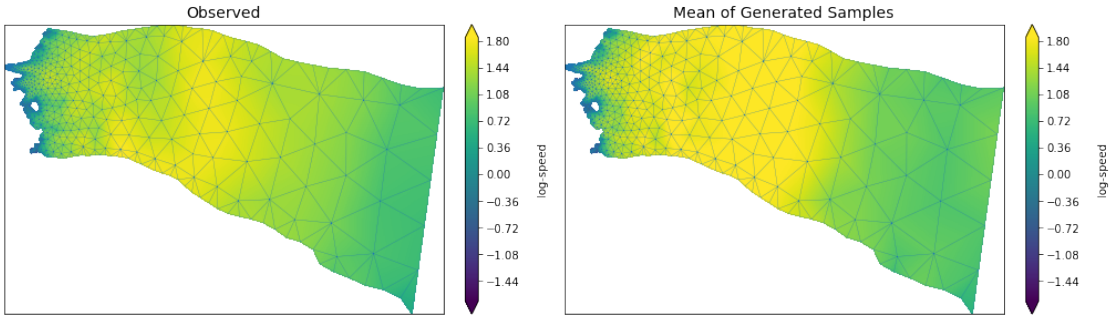


(a) High-fidelity predictions compared to the mean of the generated surrogate predictions. The mean of the predicted values have an R-squared score of -19.5792.

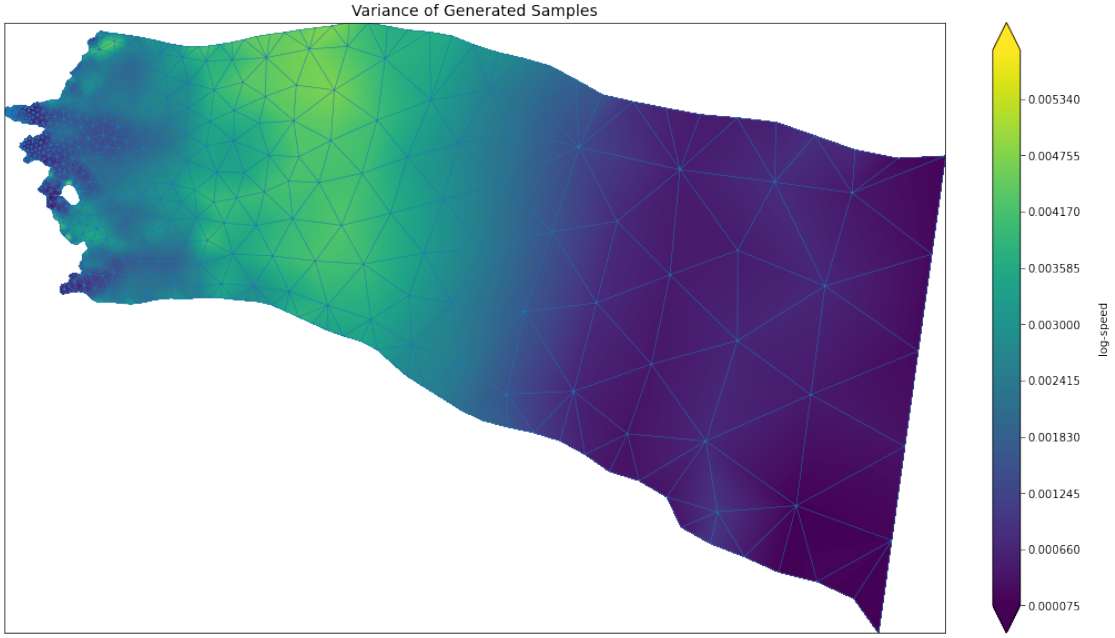


(b) Variance of generated predictions.

Figure 11: Mean and variance comparison for 100 generated log-speed predictions using the set of physical parameters used to display figure 8.

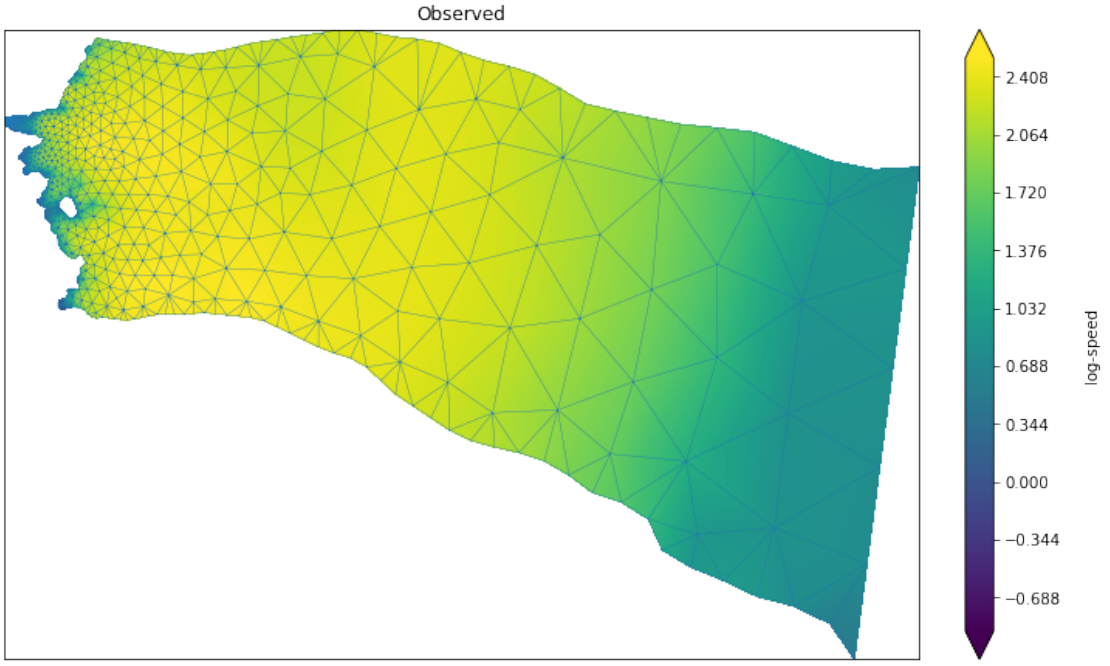


(a) High-fidelity predictions compared to the mean of the generated surrogate predictions. The mean of the predicted values have an R-squared score of 0.9372.

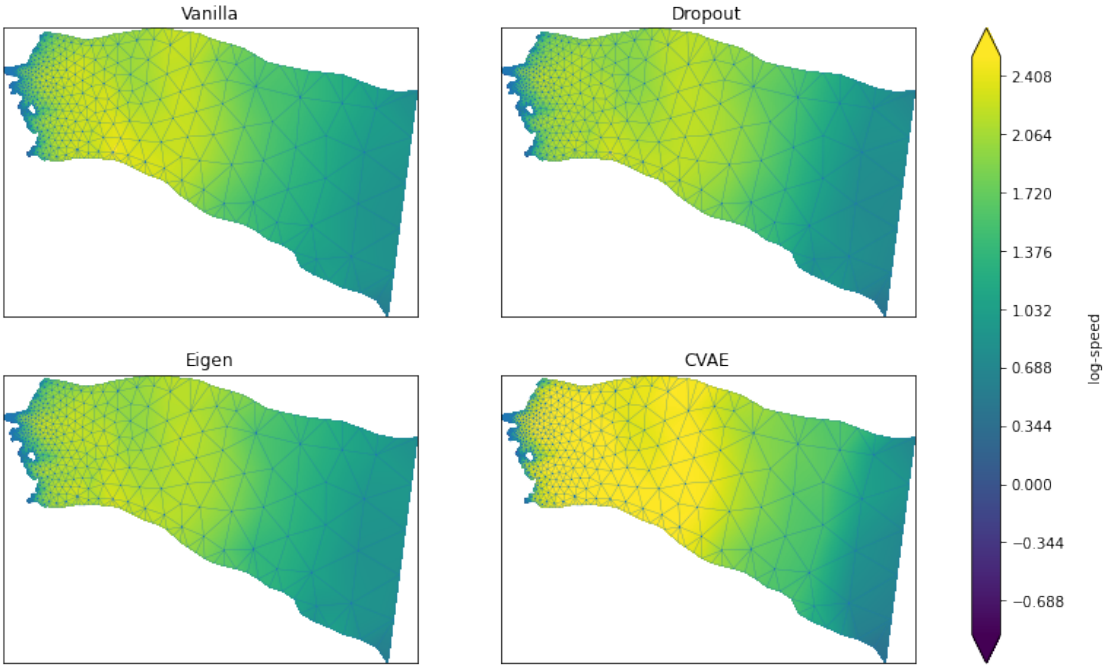


(b) Variance of generated predictions.

Figure 12: Mean and variance comparison for 100 generated log-speed predictions using the following physical parameters: [-3.7314, -2.5107, -1.4014, -0.0745, 7.4946, -0.1002, -0.5004, -3.3198].

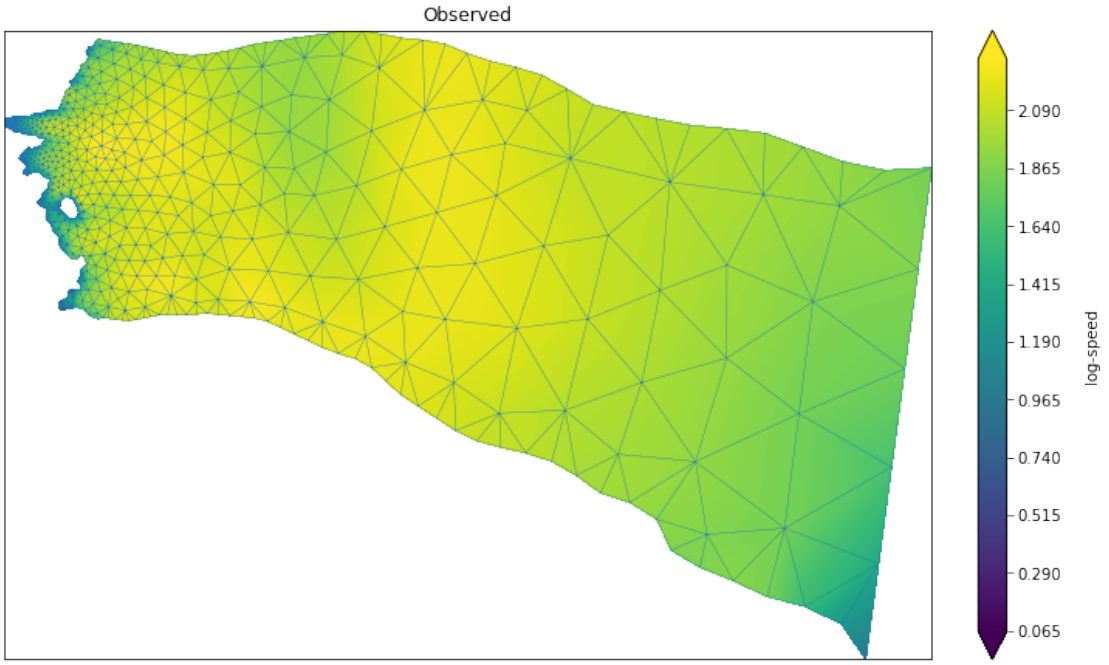


(a) High-fidelity log-speed predictions.

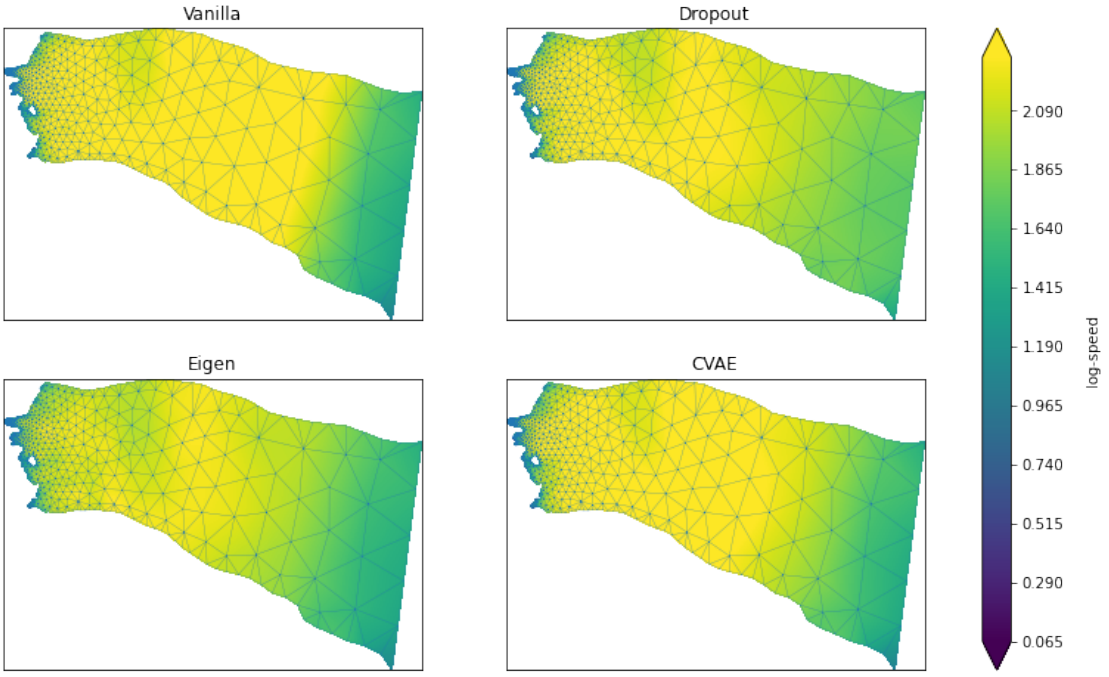


(b) Surrogate model output for Vanilla NN, Dropout NN, EigenGlacier NN and cVAE.

Figure 13: Log-speed comparison for the following physical parameters: [-2.1816, -3.0566, -2.1465, -0.5513, 6.2063, -0.3293, -0.7574, -3.9971]. The R-squared values for Vanilla, Dropout, Eigen and cVAE are as follows: [0.8925, 0.8745, 0.8949,0.9390].



(a) High-fidelity log-speed predictions.



(b) Surrogate model output for Vanilla NN, Dropout NN, EigenGlacier NN and cVAE.

Figure 14: Log-speed comparison for the following physical parameters: [-3.8066, -3.4316, -1.7715, -1.3950, 5.1750, -0.8825, -0.3995, -3.0596]. The R-squared values for Vanilla, Dropout, Eigen and cVAE are as follows: [0.8833, 0.9387, 0.9738,0.9442].

model. While the vanilla network performed well on a majority of the test data set, it had a difficult time encapsulating the subtle variation in some of the more diverse log-speed predictions. Figure 13 is a great example of this. As you can see, the surface velocities produced by this set of parameters display a different pattern than the majority of our training predictions. The vanilla network is able to capture the general change in velocity as you move further down the glacier (from bottom right to top left), but does not encompass the magnitude of the log-speed change. This is most likely caused by overfitting to the training set where most of the training predictions look like figure 7 where log-speed peaks at 1 with a gradual decrease in speed as we move down the glacier. All of our SMs perform best with these predictions, any predictions that differ from this general pattern are not as accurately reproduced. This makes sense as there are fewer predictions outside of the general pattern of figure 7 which leads the vanilla model to reward predictions more closely resembling the majority of the training data, over-fitting the model to the training data.

The next surrogate tested was our dropout model. Like the vanilla network, this architecture was easy to implement, but did require more fine tuning to get working most accurately. The dropout model performs a little better than the vanilla model as it utilizes both normalization and dropout in an attempt to prevent the model from over-fitting during training. This attempt improves the average accuracy to 90%, but doesn't completely solve the problem. Looking at figure 13, we can see that there is not a clear improvement. The dropout model does better at capturing the transition from slow to fast at the start of the glacier, which can be seen more clearly in the predictions seen in figure 14. We see a clear improvement from the vanilla to dropout surrogate, with the dropout generating smoother transitions with a more physically feasible set of predictions. This demonstrates the power of dropout where instead of overfitting to our training data, the model is forced to not depend on neighboring pixels.

Building on this, the eigen model improves upon the vanilla model even more. This is clearly demonstrated in figure 14, as you can see the progression between the three SMs. The eigen surrogate uses the same architecture as our dropout model with the added use of dimensionality reduction. This appears to help the model focus on the key elements when making a prediction. On average, this surrogate performed with close to the same accuracy as the dropout model. The eigen model performs well, but does require more time to implement and train. While more time is required to add in the dimensionality reduction of this model when compared to our vanilla network, this is still a simple model to implement and was effective at capturing the subtleties of the data.

The cVAE surrogate did not perform as well as our other SMs, but it fluctuates in performance less

and is able to generate the subtleties in the velocity predictions more consistently. The cVAE was also simple to implement and has the added benefit of using a form of dimensionality reduction like the eigen model. This surrogate trained quickly and can quickly generate new predictions. Referring back to figure 13, it is apparent that the cVAE is the only surrogate that generates a prediction that visually matches the high-fidelity prediction. The other models encompass the pattern, but not the intensity of the velocities. This is not always the case and results vary between the SM's compared. In certain cases the vanilla model performs better, other times the cVAE performs better, and so on. While the results are not conclusive to rule out one architecture as the most effective surrogate for this application, the cVAE lends itself to simple error quantification.

4.0.1 Error Quantification

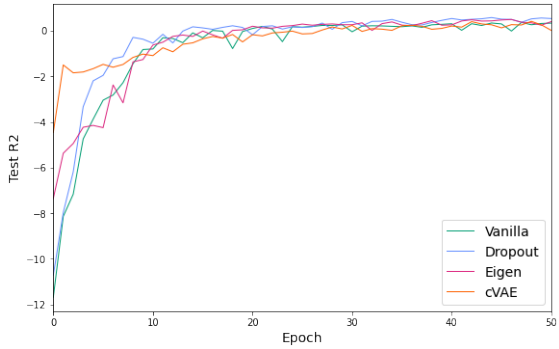
While these models are able to make accurate predictions, they do not lend themselves to a simple method of quantifying uncertainty in the surrogate. This is because for every set of parameters, there exists only one output. In other words, when you input the parameters X_a into F_{eigen}, the same results will be produced every time. During training, the neural network is fine tuning the linear transformations applied to the input that produces output closest to the training data. With the cVAE, we are adding stochasticity because we are sampling from the latent space to make our predictions, therefore no two computations will be exactly the same. This encapsulates the variability in the surrogate model, and provides insight into the fact that it is an approximation of our high-fidelity model because we are able to sample multiple times with the same set of parameters and evaluate the mean and variance of the output. Although the cVAE surrogate performs worse on average with an accuracy of 81%, the downstream application power of this model provides us with a powerful tool.

The cVAE does not necessarily outperform either of the other SMs, but it does have a tendency to be more robust to parameters that produce lots of variation. See figure 13 where the cVAE outperforms the other three models. This is possibly because the model is not relying solely on the parameters as input, but also utilizing the information contained within the log-speed fields when training. This builds a latent space that better represents the relationship between the parameters and velocity predictions. While this model does not perform as well as our other neural network models, it is the only model that we can easily evaluate to quantify the error metrics of the surrogate without aggregation for analysis of the variance between SM predictions.

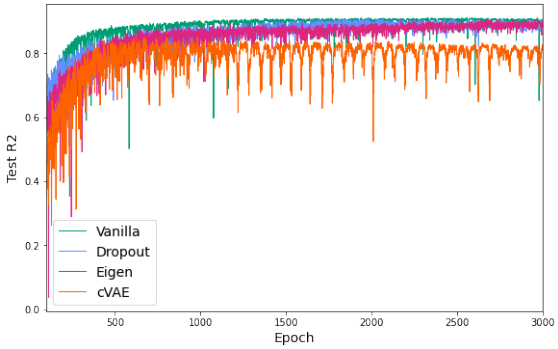
As we have discussed, a surrogate model is an approximation to a high-fidelity model. This means that

the surrogate may not make predictions with 100% accuracy. Our cVAE surrogate provides us with a means to understand how well our model is performing by assessing multiple predictions for the same parameter input. We performed this process for many parameters, and within our test set the physically reasonable predictions were matched well with little variation, see figure 10. This informs us that the cVAE surrogate produces consistent predictions when randomly sampled from the latent space with the same conditions. Looking at figure 11 we can see that although the model does not predict the results of the input parameters accurately, it does do so with little variance. This implies that even though the model fails to predict the correct values, it does so consistently so this is not a fault of our model but is indicative that this parameter set is an outlier. Knowing that the magnitude of the velocities are not physically appropriate, we can disregard this test.

The results for figures 10 and figure 11 indicate that our cVAE surrogate performs as expected without any abnormal sections of variation. Referring to figure 12 we can see the mean and variance of 100 outputs of the cVAE surrogate. This displays where the surrogate performs poorly. The variance between the different predictions increases by a factor of ten as you move into the middle of the glacier showing that that model is not as certain what to do for these parameters, leading to the variance amongst the different predictions. We expect this variance, as it is where our surrogate performs poorly. While this shows that our surrogate performs poorly, it indicates that our model is behaving as expected. This analysis cannot be performed with the other surrogate models we tested without training multiple models and aggregating them using bootstrap aggregation (Breiman, 1996).

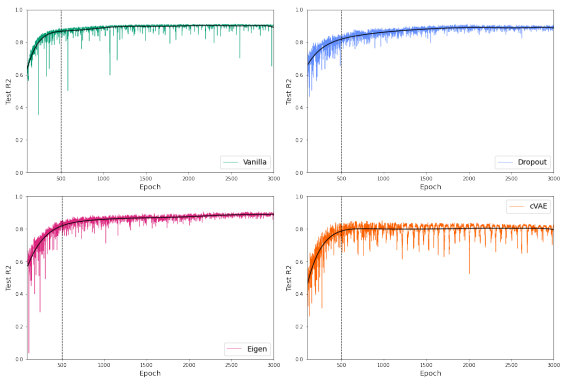


(a) Average R-Squared score for training epochs 1 to 50.

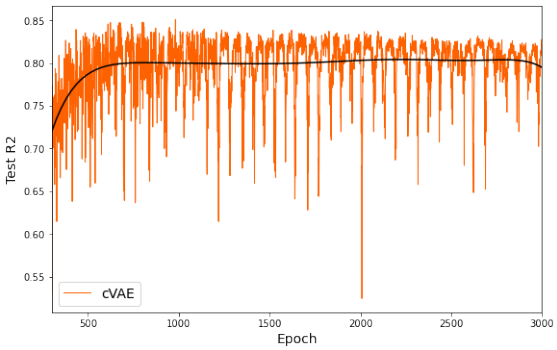


(b) Average R-Squared score for training epochs 100 to 3000.

Figure 15



(a) Individual surrogate average R2-score from epoch 100 to 3000.



(b) cVAE average R2-score from epoch 100 to 3000.

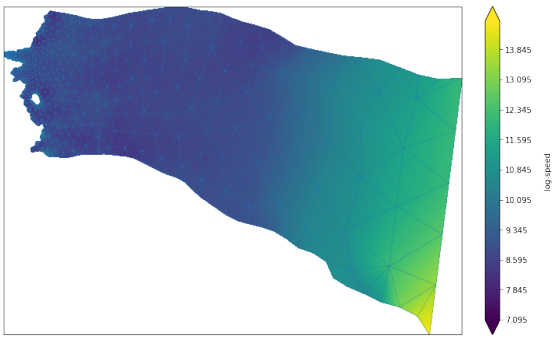
Figure 16

4.0.2 Training

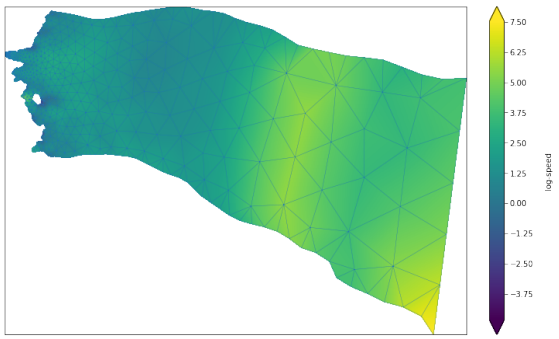
During the training process all of the surrogate's were trained for 3000 epochs for experiment control, but the question arose whether all four of the models needed to be trained for that long. To explore this, we tracked the average R-squared score for the entire training and test set over the course of the training cycle, see figure 15 and figure 16. These figures display the average R2 score of the SM predicted values and ISM predictions computed after every epoch. Figure 15 displays the first 50 epochs, which show our SMs performing as expected during the first stages of training. We begin with SM predictions on the data test set that do not correlate with our ISM, but as training continues the surrogates begin to improve. Both the vanilla and dropout models are the least accurate to start, then the eigen with the cVAE being the most accurate. After around 10 epochs all of our models begin to perform with approximately the same accuracy. This changes after 100 epochs, see figure 15b, where our models begin to diverge. We see our deterministic neural network models climbing until after 2500 epochs where they reach a more consistent state with an accuracy of 90% while the cVAE peaks after 500 epochs with an accuracy of 80%. We observe more variability in the cVAE and vanilla surrogate's accuracy overall, while the dropout and eigen models are more variable initially, but reach a more consistent state than the other surrogates. This is more clearly displayed in figure 16a. Looking more closely at the cVAE training results, figure 16b, we can see that our average performance plateaus after just over 500 training epochs. For our other models, they continue to improve in accuracy over this time. I argue that it is enough to train our cVAE surrogate for 1000 epochs or less and we will still obtain an adequate surrogate. This speaks to the immediate effectiveness of this surrogate.

4.0.3 Sampling outside scope of training data

What happens when parameters that are not within the scope of the training data are input into our models? The answer to this depends on how much the novel input diverges from the training input. If the input is within a reasonable range of the training parameters, the output produced will be consistent with previous predictions. The problem arises when the input is much different than any previously seen input. The model will produce output that is not representative of the model. This implies that in order to develop a surrogate that can effectively be used to approximate a high-fidelity model, the surrogate must be trained on a representative data set containing a diverse set of input that will be used with the surrogate. Any input that is outside the scope of the training data will result in output that may or not not be physically feasible.



(a) Predictions generated by the vanilla surrogate with a vector of zeroes passed as the input parameters.

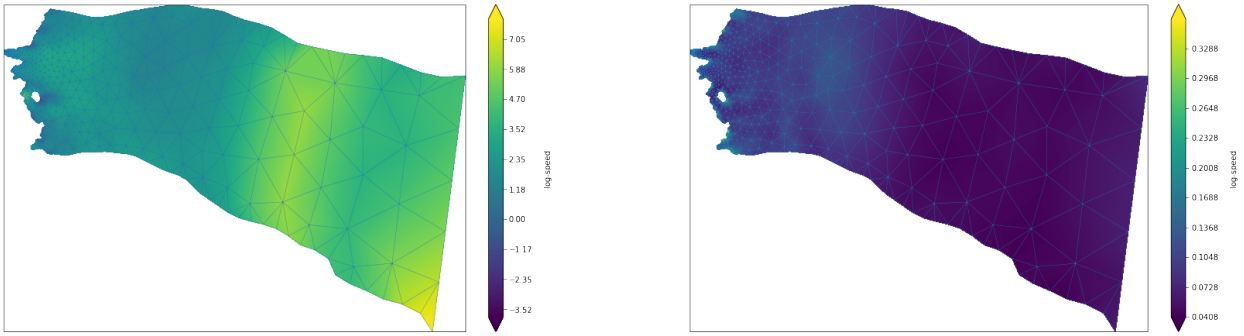


(b) Predictions generated by the cVAE surrogate with a vector of zeroes passed as the input parameters.

Figure 17

A generative model, such as a cVAE, is designed to produce new output by sampling from the latent space. Deterministic neural networks do not generate new data in the same sense, as a sample of a trained network will produce the same output every time the model is ran. This output will differ from the training output slightly, so in this sense new output is produced, where with a generative model, an infinite number of solutions exists for the same set of parameters. Figure 17 displays the results of passing a set of parameters consisting of all zeroes through the cVAE and vanilla surrogates. It is not immediately clear that this is not a viable set of parameters as both models produce results that appear to be obtainable to the untrained eye. With the cVAE we are able to generate these results multiple times, as displayed in figure 18. From this process it becomes more clear that the model is unable to produce a consistent set of predictions. The variance in the predictions from these parameters ranges from less than 0.205 to greater than 0.637. This is a significantly higher variation than all other predictions. Seeing that there is significant variance among

the generated predictions shows us that the surrogate does not have a constrained value in the latent space for this input. This allows us to disregard any results with too much variance when testing on stochastic parameters. With the deterministic surrogates, we have no way to quantify the discrepancy and rely on visually determining whether the solution is viable.



(a) Mean

(b) Variance

Figure 18: Mean and variance comparison for 100 cVAE generated log-speed predictions using a vector of zeroes as parameter input. This demonstrates the results of passing input outside of training data scope into the surrogate.

4.0.4 Future Improvements

While the surrogate models developed were all able to successfully emulate the high-fidelity ice sheet model, but struggle to make accurate predictions when the variance in the log-speed predictions is large. If the log-speed value spikes to a value larger than the average test set prediction, all of the surrogates have difficulty emulating this accurately. Another problem arises when the surrogate produces a prediction that is physically impossible. These models need to have a way of improving on these predictions.

These models perform well for this application, but can still be improved. A way to do so would be to incorporate a Physics Informed Neural Network (PINN) into our surrogate architecture to help further inform the model. This would help the model to make more informed decisions based on physical equations and limitations. In order to maintain the features provided by the cVAE, we could apply ideas used by PINNs to our existing architecture by adding a penalty to our data loss corresponding to physical limitations. An analysis of the physical parameters used in our models could be performed in order to better understand what parameters cause the ice sheet model to produce unrealistic velocity predictions. Using these results, we could penalizes these predictions using the physical equations associated with the parameters deemed most variable.

5 Conclusion

We developed a deep learning based surrogate model to approximate an ice sheet model using a conditional variational autoencoder. Three neural network based surrogate's were used to compare our novel architecture, a vanilla neural network, a neural network with dropout and normalization, and a neural network with dimensionality reduction to reduce the number of free parameters. These models were trained and tested using physical parameters and the corresponding log-speed predictions output by the ice sheet model. All of networks take parameters as input and output velocity predictions in an order of magnitude faster than the original high-fidelity model. All of these models produced output with an average R2-score of 80% and greater. While our conditional variational autoencoder (cVAE) surrogate did not outperform their benchmark models, we were able to infer the divergence between the cVAE surrogate and ice sheet model without aggregation which is crucial for quantifying the variability in applications using the surrogate. Our cVAE surrogate allows us to generate new predictions for the same set of parameters because of the architecture's stochasticity, providing a simple approach for quantifying the distribution the surrogate produces. This study introduces the use of a cVAE as a surrogate for an ice sheet model, allowing us to encompass the variability introduced by a surrogate model in order to quantify this error when used in downstream applications.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Brinkerhoff, D., Aschwanden, A., and Fahnestock, M. (2021). Constraining subglacial processes from surface velocity observations using surrogate-based bayesian inference. *Journal of Glaciology*, 67(263):385–403.

Gabbard, H., Messenger, C., Heng, I. S., Tonolini, F., and Murray-Smith, R. (2021). Bayesian parameter estimation using conditional variational autoencoders for gravitational-wave astronomy. *Nature Physics*, 18(1):112–117.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Jiang, P., Meinert, N., Jordão, H., Weisser, C., Holgate, S., Lavin, A., Lütjens, B., Newman, D., Wainwright, H., Walker, C., and Barnard, P. (2021). Digital twin earth – coasts: Developing a fast and physics-informed surrogate model for coastal floods via neural operators.

Joyce, J. M. (2011). *Kullback-Leibler Divergence*, pages 720–722. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Loh, W.-L. (1996). On latin hypercube sampling. *The annals of statistics*, 24(5):2058–2080.

Lukošiūtė, K., Raaijmakers, G., Doctor, Z., Soares-Santos, M., and Nord, B. (2022). Kilonovanet: Surrogate models of kilonova spectra with conditional variational autoencoders.

Lütjens, B., Crawford, C. H., Veillette, M., and Newman, D. (2021). Pce-pinns: Physics-informed neural networks for uncertainty propagation in ocean modeling.

Maier, N., Humphrey, N., Harper, J., and Meierbachtol, T. (2019). Sliding dominates slow-flowing margin regions, greenland ice sheet. *Science Advances*, 5(7):eaaw5406.

Mouginot, J., Rignot, E., Bjørk, A. A., van den Broeke, M., Millan, R., Morlighem, M., Noël, B., Scheuchl, B., and Wood, M. (2019). Forty-six years of greenland ice sheet mass balance from 1972 to 2018. *Proceedings of the National Academy of Sciences*, 116(19):9239–9244.

Pal, A., Mahajan, S., and Norman, M. R. (2019). Using deep neural networks as cost-effective surrogate models for super-parameterized e3sm radiative transfer. *Geophysical Research Letters*, 46(11):6069–6079.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

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. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Raissi, M., Perdikaris, P., and Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

Singh, A. and Ogunfunmi, T. (2021). An overview of variational autoencoders for source separation, finance, and bio-signal applications. *Entropy (Basel, Switzerland)*, 24(1):55–.

Xiao, C., Zhang, S., Ma, X., Jin, J., and Zhou, T. (2022). Model-reduced adjoint-based inversion using deep-learning: Example of geological carbon sequestration modeling. *Water Resources Research*, 58(1):e2021WR031041. e2021WR031041 2021WR031041.