Felix Schürmann Lab

Learning Ion-Channel Dynamics with Physics-Informed Neural Networks

Student: Thibault Niederhauser
Supervisor: Omar Awile
Professor: Prof. Felix Schürrman

École Polytechnique Fédérale de Lausanne
Semester Project
06/2021

Contents

1. Introduction

Deep-learning has been gaining increasing attention over the recent years and seen numerous application in widely different scientific domains ranging from image recognition [1] to genomics [2]. However, a significant drawback of deep-learning approaches and artificial neural networks is that they are often exclusively data-based. This implies that a large amount of data is required and that potential physical or mathematical knowledge of the system is not taken into account. In the case of an application such as the modeling of a physical system, a model exclusively trained on data might predict nonphysical behaviors when tested on unseen data.

To tackle those issues, Raissi et al. introduce Physically Inspired Neural Networks (PINNs) [3][4]. PINNs aim at solving supervised learning task while being guided by the physical knowledge of the system, expressed in the form of a non-linear partial differential equation (PDE). Such framework can be used for data-driven discovery of PDEs, a problem type where a PDE is known up to a set of parameters λ , a set of (potentially noisy) training data reflecting the PDE is available and the aim is to discover the parameter set λ that best describes the data.

The implementation of data-driven discovery of PDEs is closely related to automatic differentiation [5], a computational differentiation method that makes use of computational graphs. In automatic differentiation, the derivative of a variable can efficiently be computed by applying the chain-rule to the computational graph. Automatic differentiation underlies the processes of back-propagation in deep-learning and, in the case of PINNs, can be implemented to compute the partial derivatives present in the PDEs. Thus, automatic differentiation allows data-driven discovery of PDE parameters through PINNs to be efficiently implemented; particularly since frameworks such as PyTorch and TensorFlow enable straight-forward user-friendly implementation of automatic differentiation.

Data-driven discovery of PDE is a typical problematic encountered in modeling of ion-channels. Indeed, ion-channel conductance dynamics can be described by the Hodgkin-Huxley model, involving a set of ordinary differential equations (ODEs) and some open parameters. The aim of ion-channel modeling is to find the parameters that characterize best the dynamics of a specific ion-channel. Typically, a set of experiments is designed to gather experimental data about a specific ion-channel and, based on the data and the physical equations, the optimal parameters are computed. Different model fitting approaches exist to discover the model parameters, such as the method described in [6]. Raissi et al. work suggest that PINNS could also be used to address this problem.

Henceforth, the aim of this work is to explore whether PINNs can be implemented to model ion-channel dynamics by conducting a data-driven discovery of the Hodgkin-Huxley equations' parameters. In a first step, a replication of the work of Raissi et al. on discovery of the Burger's equation is performed. Then, the PINN-model is adapted for ion-channel dynamics modeling. Eventually, a set of computational experiments is be performed to evaluate the ion-channel mode and to compare it to a model developed according to [6].

2. Implementing Physically Inspired Neural Networks

Raissi et al. showed that PINNs could lead to data-driven solutions of non-linear PDEs [3] and data-driven discovery of non-linear PDEs parameters [4].
The aim of this section is to implement PINNs in `python` and confirm that such architecture is able to learn PDE parameters. This is done by replicating part the work of Raissi et al. on data-driven discovery of PDEs [4].

2.1 Replication

The replication deals with the problem of data-driven discovery of partial differential equations. Given a small set of scattered and potentially noisy observations of the hidden state $u(t, x)$ of a system and partial differential equation known up to some parameters λ, the aim is to find the parameters λ that best describe the observed data.
The replication focuses on the one dimensional Burgers' equation; a second order partial differential equation widely used in applied mathematics including fluid mechanics and traffic flow. The Burger's equation is defined as :

$$u_t + \lambda_1 u u_x - \lambda_2 u u_{xx} = 0 \tag{2.1}$$

where $u(x, t)$ is the fluid velocity, u_t is the first order derivative of u with respect to time t and u_x and u_{xx} are respectively the first and second order derivatives of u with respect to the position x. λ_1 and λ_2 are the equations' parameters.
Here, a relatively small data-set of experimentally known data-points $\{x, t, u(x, t)\}$ is available and the parameters λ_1 and λ_2 are unknown and need to be discovered.
To begin with, a neural network meant to learn the outputs $u(x, t)$ from the inputs x and t is designed. This so-called u-net is a feed-forward neural network of 10 layers of 20 neurons. The neurons of the hidden layers have *tanh* as activation function as in [4] (see Fig. 2.1). To facilitate learning, the weights of the neural net are initialized with xavier-initialization, the input data x and t is normalized between -1 and 1 before being fed into the neural network and the output state $u(x, t)$ is included in $[-1, 1]$.
Nevertheless, the *u-net* is not efficient on its own because the quantity and quality of available data might not be sufficient to enable satisfying learning. Moreover, learning to map $\{x,t\}$ and $u(x, t)$ with such a "black-box" model does not give any information about the unknown parameters λ_1 and λ_2, which are meant to be discovered. Using the physical knowledge of the system (Eq. 2.1) allows to tackle those issues.
In this sense, a second function, *f-net*, is designed. f-net takes x and t as inputs and returns the following output f:

$$f(x, t) := u_t + \lambda_1 u u_x - \lambda_2 u u_{xx} \tag{2.2}$$

To compute f, the terms u are obtained through a feed-forward pass through the u-net, hence f-net and u-net have shared parameters. Moreover, computation of the partial derivatives of u (u_x, u_{xx} and u_t) are required. One key aspect of the PINNS implementation proposed by Raissi et al. is that the partial derivatives are computed using auto-differentiation. The derivatives u_t and u_x are by differentiating the u-net output u, with respect to one of the input, respectively t and x. The second order derivative u_{xx} can be obtained following the same principle, by computing the gradient of u_x with respect to the input variable x.

In a system ruled by Burger's equation, it is known that $f(x, t) = 0$. This way the shared parameters of the u-net and f-net (weights and biases), as well as the equation parameters λ_1 and λ_2 can be learnt by minimizing the following training loss:

$$L = MSE_u + MSE_f \tag{2.3}$$

where

$$MSE_u = \frac{1}{N} \sum_{i=1}^{N} |u(t_u^i, x_u^i) - u^i|^2$$

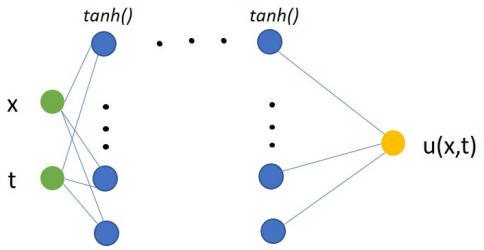


Figure 2.1: u-net: a feed-forward neural net with *tanh* as activation function. It takes x and t as inputs and returns $u(x,t)$

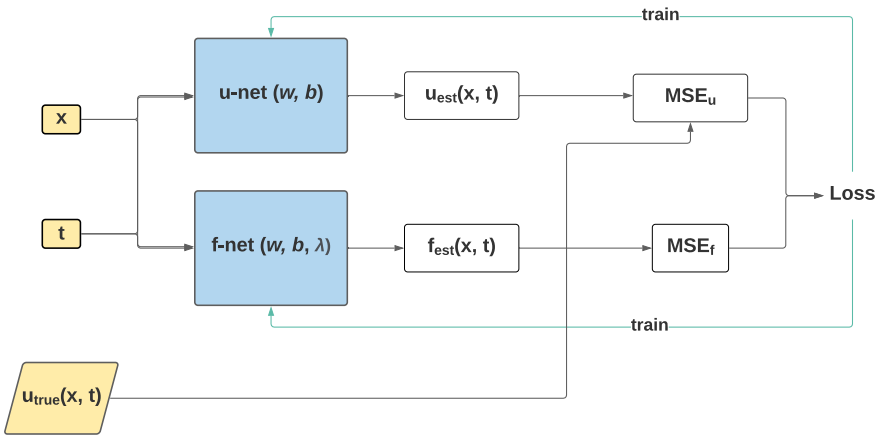


Figure 2.2: Training pipeline: x and t are the inputs of *u-net* and *f-net*. *u-net* outputs an estimate of the state $u(x,t)$. f-net computes the function f as given in Eq. 2.2. This allows to compute the loss as in Eq. 2.3 and learn the parameters w (weights), b (biases), λ_1 and λ_2.

and

$$MSE_f = \frac{1}{N}\sum_{i=1}^{N}|f(t_u^i, x_u^i)|^2 \tag{2.4}$$

Here $u(t_u^i, x_u^i)$ denotes the prediction of the u-net and $\{t_u^i, x_u^i, u^i\}$ is the training data. MSE_u enforces learning based on the data, whereas MSE_f allows to learn the the structure imposed by Eq. 2.1. The whole training pipeline is summarised in Figure 2.2.

Two different optimizers are used to minimize the loss function: ADAM [7] and L-BFGS [8]. As mentioned in [9], for smooth PDEs L-BFGS converges faster and in less iterations than ADAM. This is due to the fact that L-BFGS, relies on both the slope (Jacobian) and curvature (Hessian estimate) of the loss function, whereas ADAM only makes use of the first order derivative. However, L-BFGS is more likely to fall in a sub-optimal local minima than ADAM. Therefore ADAM is used over the first hundreds or thousands training iterations to avoid falling into a highly suboptimal minima; then L-BFGS is implemented to compute the final minima.

2.2 Challenges

The code implemented by [4] for data-driven discovery of the Burger's equation is publicly available at: *https://github.com/maziarraissi/PINNs* . The original code uses the deep-learning library TensorFlow and the aim of this section is to reproduce the same implementation using PyTorch. Therefore, the main obstacles

were linked to the translation from TensorFlow to PyTorch.
A first challenge was to reconstruct the same neural net architecture as in [4]. In the provided TensorFlow code, the neural net architecture is implemented *manually*, i.e. without using built-in TensorFlow layer functions. The neural net feed-forward pass is constructed as a function that applies weights, biases and activation functions iteratively to the function input. The weights and biases are stored as TensorFlow variables and updated during training. The choice of building the neural net *manually* was supposedly made because it simplifies parameters monitoring and debugging.
However, for the replication in PyTorch, the choice was made to implement the *u-net* directly through PyTorch built-in functions, as the framework is optimized to work efficiently with these. Moreover, the model is a simple fully connected network with *tanh()* activation layers, thus the implementation with built-in layers is straight forward and user-friendly.

A more challenging aspect of the replication was to implement automatic differentiation correctly. In this work, automatic differentiation is implemented using the PyTorch automatic differentiation package *torch.autograd*. The challenge emerges from the fact that automatic differentiation (and the package *torch.autograd*) is used for two different means: for the backward propagation of the loss gradient during training and for the derivative computation to obtain u_t, u_x and u_{xx}. In both cases, the automatic differentiation involves the same variables, namely the outputs, inputs, weights and biases of the neural network. Therefore, one needs to ensure that both processes do not interfere.
Moreover, the weights update during training with error backward propagation is given as:

$$\Delta w_i = \frac{dL}{dw_i} \tag{2.5}$$

Since u_x, u_{xx} and u_t are part of the training loss L (see Eq. 2.1 and Eq. 2.4), further order derivatives of those terms (with respect to the weights w_i) are required during training. This implies that automatic differentiation during the backward pass requires the computational graphs of u_x, u_{xx}, u_t, thus the computational graphs of u_x, u_{xx} and u_t need to be stored . Both issues are tackled by the following code implementation for partial derivation:

```
#differentiation
u_t = torch.autograd.grad(u, t, grad_outputs=torch.ones(len(u),1), retain_graph=True,
    ↪ only_inputs=True, create_graph=True)[0]
u_x = torch.autograd.grad(u, x, grad_outputs=torch.ones(len(u), 1), retain_graph=True,
    ↪ only_inputs=True, create_graph=True)[0]
u_xx= torch.autograd.grad(u_x, x, grad_outputs=torch.ones(len(u_x), 1), retain_graph =
    ↪ True, only_inputs=True, create_graph=True)[0]

#Burger's equation
f = u_t + lambda_1 * u * u_x - lambda_2 * u_xx
```

The argument `only_inputs=True` specifies that the gradients with respect to weights and biases (that are computed in the derivation process) are not stored. This way the partial differentiation of u will not interfere with the training process. In addition, the argument `create_graph=True` ensures that the computational graphs of the partial derivatives are stored, which allows further order derivatives to be computed during training (see Eq. 2.5).

2.3 Comparison

The original model from [4] (TensorFlow) and the replication model (PyTorch) are compared. A noiseless training data set $S_{Burgers} = \{x, t, u(x,t)\}$ containing data-points ruled by the Burger's equation is provided on the github repository along with the code. The parameters that were used to generate the dataset $S_{Burgers}$

are:

$$\lambda_1^{true} = 1.00 \qquad and \qquad \lambda_2^{true} = 0.0031831$$

The models aim to discover those values.

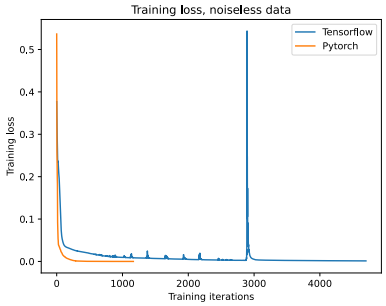
The original and replication models are fitted and evaluated for 3 different cases: once trained on noiseless data and twice trained one noisy data (1% and 10% noise). The noise is added artificially on the noiseless data.
In each case, the models are trained on $N_u = 2000$ data-points randomly sampled from $S_{Burgers}$. In the case of noisy training data, 10000 ADAM iterations are performed before switching to L-BFGS. For noiseless data, L-BFGS is used directly as training is easier and L-BFGS is found to perform well alone.

Figures 2.3, 2.4, 2.5 show the evolution of the loss (see Eq. 2.3) and the identified parameters λ_1 and λ_2 during training. One can observe that the learning dynamics is similar between the original and the replicated models. One can also notice that the LBFGS optimization seems to perform better in the PyTorch implementation than in the original model (the loss decreases slighly faster in the PyTorch implementation under L-BFGS optimization). This difference is due to the fact that different algorithms are used in the replication and original model. Indeed the original model relies on an implementation of the L-BFGS-B [10] algorithm, whereas the replication model uses the built-in pytorch optimizer `torch.optim.LBFGS`, which is inspired from the MATLAB function *minFunc* [1]. In this case, the latter seem to perform slightly better, but this might not be the case for other specific problems. In addition, note that the computational time for one optimization step are very similar in both implementation and for both optimizers, however no extensive comparison was performed.

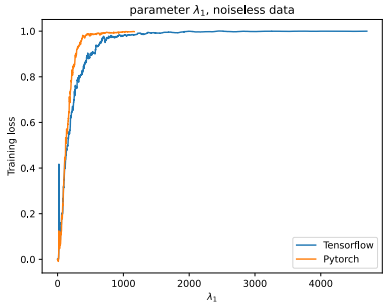
Tables 2.1, 2.2 and 2.3 show the identified parameters λ_1 and λ_2 as well as the errors compared to the true parameters. As one would intuitevely expect, the error on the parameters increases with the noise level. However, both the original and replicated model are able to identify the correct parameters with a low error even for a SNR ratio of 10%. In most cases the parameter error have the same order of magnitude for the TensorFlow and PyTorch models and there is no model that always identifies better parameters than the other. Therefore, one can conclude that the models perform similarly in terms of parameters discover.

All in all, because the learning dynamics, the computational time and the identified parameters of the original and replicated models are highly similar (and this for three different level of noise in the training data), one can conclude that the model proposed by Raissi et al for data-driven discovery of the Burger's equation is successfully replicated.

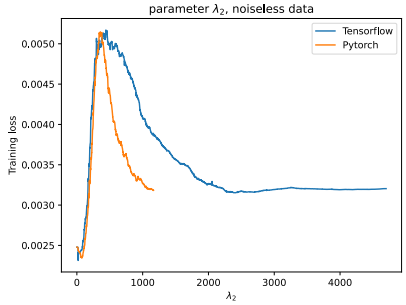
[1] https://www.cs.ubc.ca/ schmidtm/Software/minFunc.html



(a) Loss

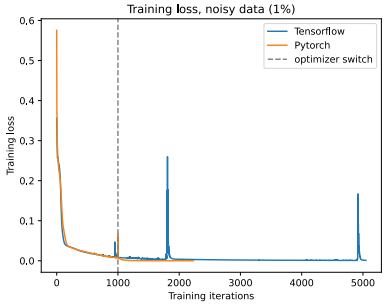


(b) λ_1

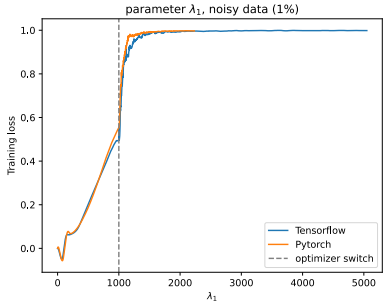


(c) λ_2

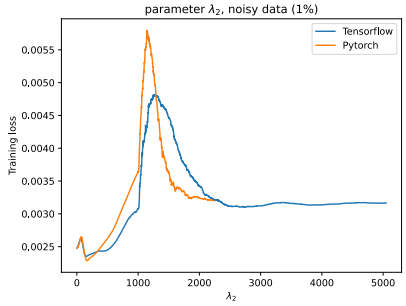
Figure 2.3: Loss and parameters evolution during training: noiseless data



(a) Loss



(b) λ_1



(c) λ_2

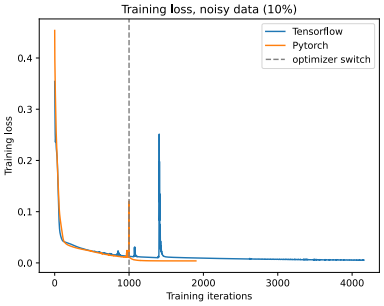
Figure 2.4: Loss and parameters evolution during training: 1% noise

Table 2.1: Learnt parameters: noiseless data

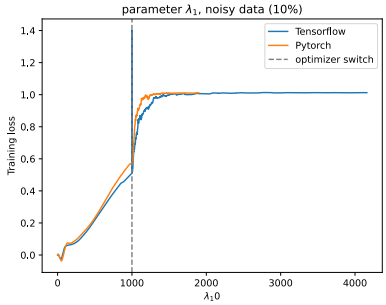
	Tensorflow	Pytorch
λ_1	0.99984	0.99793
λ_2	0.0032041	0.0031823
Error λ_1	0.016%	0.21%
Error λ_2	0.66%	0.025%

Table 2.2: Learnt parameters: noise = 1%

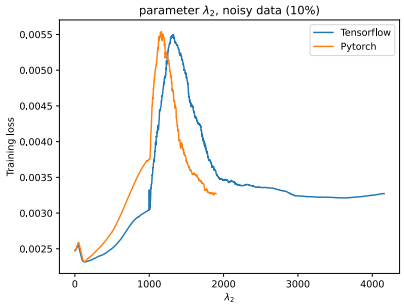
	Tensorflow	Pytorch
λ_1	0.99849	0.99678
λ_2	0.0031642	0.0032027
Error λ_1	0.15%	0.32%
Error λ_2	0.59%	0.62%



(a) Loss



(b) λ_1



(c) λ_2

Figure 2.5: Loss and parameters evolution during training: 10% noise

Table 2.3: Learnt parameters: noise = 10%

	Tensorflow	Pytorch
λ_1	0.98482	1.0113
λ_2	0.0032816	0.0032715
Error λ_1	1.52%	1.13%
Error λ_2	3.09%	2.78%

3. Applying PINNS to ion-channels

The Hodgkin-Huxley equation describes the current flowing through a neuronal cell membrane as:

$$I = C_m \frac{dV_m}{dt} + I_K + I_{Na} + I_{leak} \tag{3.1}$$

where V_m and C_m are respectively the membrane potential and capacity. I_Na and I_K are the currents flowing through the potassium and sodium ion-channels and are expressed as follows:

$$I_{Na} = g_{Na}(V_m - E_{Na}) \tag{3.2}$$

$$I_K = g_K(V_m - E_K) \tag{3.3}$$

where E_k and E_{Na} are the sodium and potassium reversal potentials and g_K and g_{Na} are the channels conductance.
The Hodgkin-Huxley model specifies that ion channel conductance, such as g_k, varies depending on voltage and time:

$$g_K = \bar{g}_K m^i h^j \tag{3.4}$$

$$m_t = \frac{m_{inf} - m}{m_{tau}} \tag{3.5}$$

$$h_t = \frac{h_{inf} - h}{h_{tau}} \tag{3.6}$$

Here, m_t and h_t are the derivatives of m and h with respect to time, $\bar{g}_K$ is the maximal potassium channel conductance (constant) and m_∞, h_∞, m_τ and h_τ are voltage dependent parameters. i and j denote the exponents of the variable m and h. In the original Hodgkin-Huxley equation, they were set to $i = 3$ and $j = 1$, however depending and the ion-channel and the temperature of recording, other values of those exponents were found to enable a better fit of experimental data [6]
The aim of this work is to adapt the implementation of the PINNS framework described in section 2.1 to model ion-channel behaviors. The problem is a data-driven discovery of ODEs parameters: based on experimental (or artificially generated data, see 3.1) and the equations stated above, the aim is to discover the parameter set $\lambda = \{m_\infty, m_\tau, h_\infty, h_\tau\}$ that describes the data best.
Note that this study focuses on the modeling of potassium ion-channels, but could be easily adapted for sodium channels since the governing equations are the same.

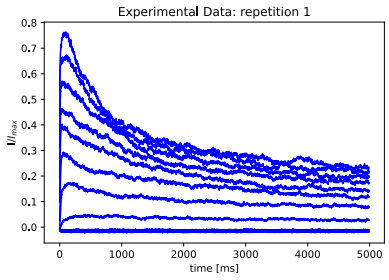
3.1 Data

Two different types of data are used: data recorded experimentally from voltage-clamped recordings and data generated from the Hodgkin-Huxley equations.

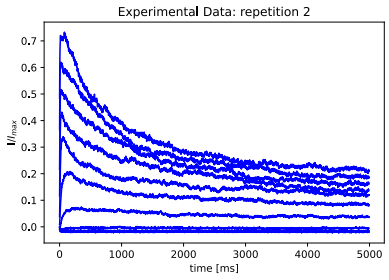
3.1.1 Experimental data

The experimental data comes from the web-based freely accessible platform Channelpedia [1]. The data consists of current traces recorded during voltage-clamped activation experiments. It this experimental setup, the neuronal cells are set under **constant** activation voltage and the current flow is recorded. The experiment is repeated for different activation voltages.
In this work, data from the Kv1.2 potassium ion-channel is used. The experiment temperature is 35°C and the host cell is a Chinese Hamster Ovary cell (CHO). The activation voltages considered are $v_{act} = [-40, -30, -20, -10, 0, +10, +20, +30, +40, +50]mV$. Three different repetitions of the experiment are shown in Figure 3.1.

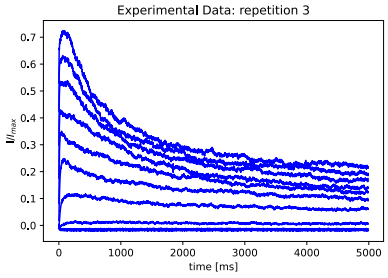
[1] https://channelpedia.epfl.ch/



(a) Repetition 1



(b) Repetition 2



(c) Repetition 3

Figure 3.1: Experimental data: three repetitions of the same experiment. Note that current traces for activation voltages varying between -40 mV and +50 mV were selected. I_{max} correspond to the maximal current value under an activation of +80mV, therefore the normalize data does not reach one in these plots.

Along with the experimental data, a "conventional" (not involving deep-learning) model of the Kv1.2 ion channel is used for comparison means. The conventional model is obtained using the ion-channel modeling techniques presented in [6]. The conventional model is generated based on the three experiment repetitions displayed in Figure 3.1.

3.1.2 Artificially generated data

Besides the experimental one, additional data is artificially generated using Eq. 3.3 to Eq. 3.6. Artificially generated data presents two advantages when it comes to model assessment:

- The level of noise on the data can be controled , thus assessement the model's robustness to noise is made possible.
- The true values of the parameters m_∞, m_τ, h_∞ and h_τ used to generate the data are known, which enables quantification of the error on the discovered parameters.

Artificial data is generated as follows: first, some parameters m_∞, m_τ, h_∞ and h_τ are chosen. Then the ODEs 3.5 and 3.6 are solved for m and h with the chosen parameters. m and h are substituted in Equation 3.4 and the current trace is generated by converting the conductance into current (Equation 3.3).
Note that the parameters m_∞, m_τ, h_∞ and h_τ are voltage dependent. Data is generated for a discrete set of activation voltages: $v_{act} = [-40, -30, -20, -10, 0, 10, 20, 30, 40, 50]mV$. The parameters used to generate the artificial are presented in Table 3.1

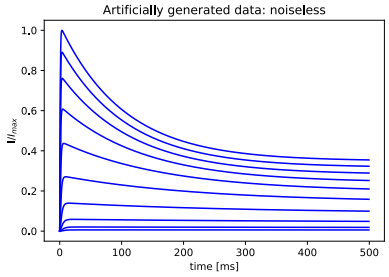
Noise is added to the data according to the following equation:

$$I = I + n \cdot \sigma_I \cdot x_{rnd} \tag{3.7}$$

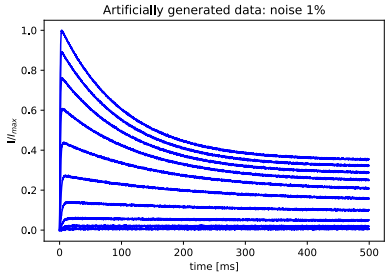
Table 3.1: True parameters

act. [mV]	-40	-30	-20	-10	0	10	20	30	40	50
m_∞	0.1229	0.2062	0.3251	0.4716	0.6233	0.7541	0.8504	0.9133	0.9512	0.9731
h_∞	0.9151	0.8462	0.7460	0.6277	0.5186	0.4385	0.3889	0.3615	0.3472	0.3401
m_τ	4.886	4.221	3.287	2.298	1.519	1.037	0.7800	0.6551	0.5967	0.5699
h_τ	479.1	441.0	380.6	303.3	227.7	170.8	135.9	117.1	107.7	103.1

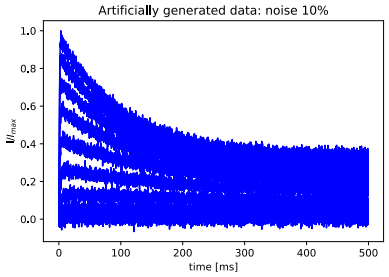
where σ_I is the standard deviation of the current vector I, x_{rnd} is a vector of a random variables uniformly distributed in $[0, 1]$. The noise level parameter n, allows to tune the SNR ratio of the artificial data. In this work, artificial data for three different SNRs is used: 0%, 1% and 10%. The data is shown in Figure 3.2.



(a) $SNR = 0\%$



(b) $SNR = 1\%$



(c) $SNR = 10\%$

Figure 3.2: Artificially generated data

3.2 Single activation voltage

3.2.1 SAV-Model

First of all, a Single-Activation-Voltage model (SAV-model) meant to model ion-channel behaviour under a single activation voltage is designed. In this case, the current trace from only one activation voltage is considered as training data. The activation voltage $v_{act} = +10mV$ is arbitrarily selected. The only input variable of the system is the time variable t.
Here, the problem is slightly different from the Burger's equation discovery presented in section 2.1. In the case of the Burger's equation, the hidden state was a single variable $u(x, t)$, however this time the state is made out of three variables $I_k(t)$, $m(t)$ and $h(t)$. Therefore three neural nets, *i-net*, *m-net* and *h-net* are implemented. The three neural nets have independent parameters, take t as input and output respectively $I_k(t)$, $m(t)$ and $h(t)$. As previously, the networks consist of 10 layers of 20 neurons with activation funtion *tanh()*.

Then, an additional network, *f-net*, that encapsulates the physical knowledge of the system, is designed.*F-net* takes t as input and outputs a set of vectors, f_1 to f_5, that will be used in the training loss to enforce learning of the structure imposed by the physical equations of the system. Based on Eq. 3.3, 3.5 and 3.6, f_1, f_2 and f_3 are defined as follows:

$$f_1 := \bar{g}_K m^2 h(V_m - E_K) - I_K \tag{3.8}$$

$$f_2 := \frac{m_\infty - m}{m_\tau} - m_t \tag{3.9}$$

$$f_3 := \frac{h_\infty - h}{h_\tau} - h_t \tag{3.10}$$

Note that in Eq. 3.8, the exponents of m and h are respectively 2 and 1 (and not 3 and 1 as typically seen in the Hodgkin-Huxley model), as there were found to enable better fit of the data [6].

In the case of voltage-clamped experiments with voltage activation beginning at $t = 0$, the following initial conditions are known:

$$h(t = 0) = 1 \qquad m(t = 0) = 0$$

To enforce this behavior in the PINNS model, the following functions f_4 and f_5 are returned by the $f - net$:

$$f_4 := m(t = 0) \tag{3.11}$$

$$f_5 := h(t = 0) - 1 \tag{3.12}$$

Inside f-net, the variables m, h and I_K are computed by forward passes through the m-net, h-net and i-net respectively, hence f-net has shared parameters with m-net, h-net and i-net. Moreover, m_t and h_t are computed using automatic differentiation.
Finally, the weight and parameters of the neural nets as well as the parameter set λ can be learnt by minimizing the loss function:

$$\begin{aligned} L &= MSE_i + MSE_{f1} + 10 \cdot MSE_{f2} + 10 \cdot MSE_{f3} + MSE_{f4} + MSE_{f5} \\ &= \frac{1}{N}\sum_{i=1}^{N} |I(t^i) - I^i|^2 + \frac{1}{N}\sum_{i=1}^{N} |f_1(t^i)|^2 + 10 \cdot \frac{1}{N}\sum_{i=1}^{N} |f_2(t^i)|^2 + 10 \cdot \frac{1}{N}\sum_{i=1}^{N} |f_3(t^i)|^2 + |f_4|^2 + |f_5|^2 \end{aligned} \tag{3.13}$$

Where N is the size of the training set and $\{t^i, I^i\}$ is the training data. MSE_i enforces learning based on the data, whereas the other terms of the training loss enable learning of the physical properties of the system (Eq. 3.3 to 3.6).Note that the weight factors 10 before MSE_{f2} and MSE_{f3} are chosen heuristically. They are implemented because it was noticed that the model had difficulties to learn the dynamics of $m(t)$ and $h(t)$ (encapsulated in MSE_{f2} and MSE_{f3}). Giving those terms a higher weight minimizes this issue.

As in section 2.1, the training loss is minimized using ADAM and L-BFGS. ADAM is typically run over 1000 to 10000 iterations depending on the problem complexity, before switching to L-BFGS.

3.2.2 Experiments

In order to assess the model's ability to discover the parameters and fit the data, 4 different experiment are designed.

- **Experiment 1: parameters recovery from artificial data**. The model is trained using a subset of $N_t = 2000$ points randomly sampled from the artificially generated data. The learnt parameters are then compared to the known true value used to generate the data. The trained i-net predictions and the discovered physical equation are tested on the complete data-set.
- **Experiment 2: experimental data fitting**. Experimental data can differ significantly from artificially generated data. A first step to evaluate how well the model can fit experimental data is to perform the same experiment as above, but on experimental data.
  Here again, $N_t = 2000$ training data points are sampled from the experimental data and the discovered equations as well as the i-net predictions are tested on the complete dataset.
- **Experiment 3: fitting several repetitions**: Experimental data varies from one recording repetition to another. Therefore, there is interest to see if the model is able to learn a consistent data fit and identify meaningful parameters if trained on several experimental repetitions. To do so, the model is trained on the current traces from the 3 experimental repetitions (see Figure 3.1).

3.3 Multiple activation voltages

3.3.1 MAV-Model

The model for single activation voltage presented in section 3.2.1 can be extended to build a Multiple-Activation-Voltage model (MAV-model) that takes into account the membrane potential and this way make a more complete and useful model of the ion-channel.
The model architecture is fundamentally the same as for the single activation; the only difference is that the input is no longer t, but a multidimensional vector $[t, v]$ and the state consists of the three variables $I(t, v)$, $m(t, v)$ and $h(t, v)$. Hence, for n_t time points and n_v activation voltages, the f-net outputs f_1, f_2 and f_3 take dimensions $n_t \mathrm{X} n_v$, f_4 and f_5 have dimensions n_v and, importantly, the parameter set that is meant to be discovered becomes $\lambda = m_\infty, h_\infty, m_\tau, h_\tau$ where m_∞, h_∞, m_τ and h_τ are multidimensional vectors of size n_v. Instead of 4 parameters, the model aims at discovering $4n_v$ parameters, which increases the complexity of the task. This reflect the fact that the parameters are voltage dependent.

3.3.2 Experiments

In order to test model for multiple activation voltages and asses its ability to discover correct parameters and fit the data, different experiments are designed. Experiment 1 to 3 are repeated, but for the multiple-activation-voltage model:

- **Experiment 4: parameters recovery from artificial data**. As in experiment 1, the model is trained using a subset of $N_t = 2000$ points randomly sampled from the artificially generated data. The learnt parameters are then compared to the known true value used to generate the data. The trained i-net predictions and the discovered physical equation are tested on the complete data-set.
- **Experiment 5: Fitting experimental data**. Experiment 2 is repeated for the multiple-activation-voltage model. $N_t = 2000$ training data points are samples from the experimental data and the discovered equations as well as the i-net predictions are tested on the complete dataset.
- **Experiment 6: Fitting several repetitions and comparison to the conventional model**: The model is trained on the current traces from all 3 experimental repetitions. The prediction form the i-net

as well the identified equation are evaluated. Moreover, the discovered parameters are compared to the parameters of the "conventional" model mentioned in section 3.1.1, which was built using methods from [6].

3.4 Results

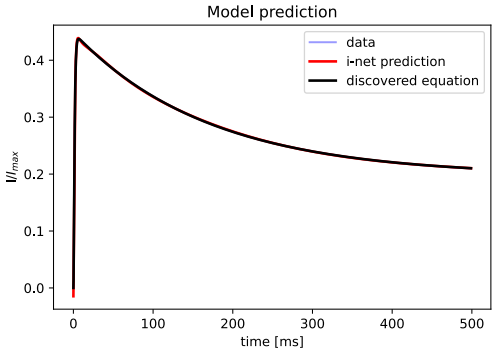
3.4.1 Single activation voltage

Experiment 1

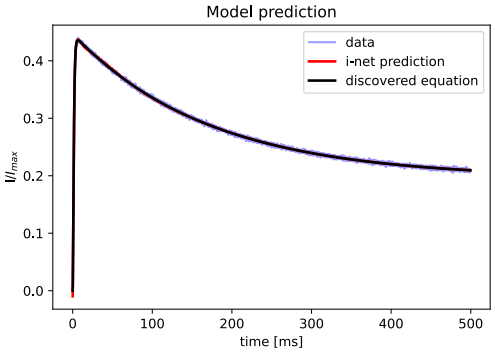
Table 3.2 shows the discovered parameters as well as the error on the discovered parameters relatively to the true parameters presented in Table 3.1. Here, the considered activation voltage is +10 mV, hence the parameters that are meant to be discovered are: $m_\infty = 0.7541$, $h_\infty = 0.4385$, $m_\tau = 1.037$ and $h_\tau = 170.8$. Table 3.2 also shows the MSE of the discovered equation and the i-net prediction.
Figure 3.3 depicts the i-net prediction and the discovered equation for noiseless and noisy data.

Table 3.2: experiment 1

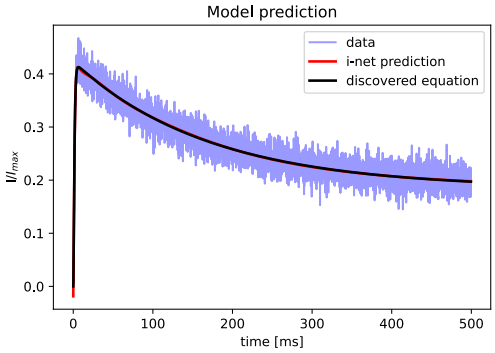
	noiseless	noise = 1%	noise = 10%
m_∞	0.2054	0.2049	0.2054
h_∞	0.4387	0.4377	0.4387
m_τ	1.0308	1.0326	1.0244
h_τ	170.7581	172.2014	171.0204
Error m_∞	72.76%	72.83%	72.76%
Error h_∞	0.05%	0.18%	0.05
Error m_τ	0.60%	0.42%	1.22%
Error h_τ	0.02%	0.82%	0.13%
MSE i-net	2.0378e-07	6.9425e-06	9.1366e-05
MSE discovered equation	7.4482e-09	3.7876e-06	8.9277e-05



(a) oiseless



(b) Noise = 1%

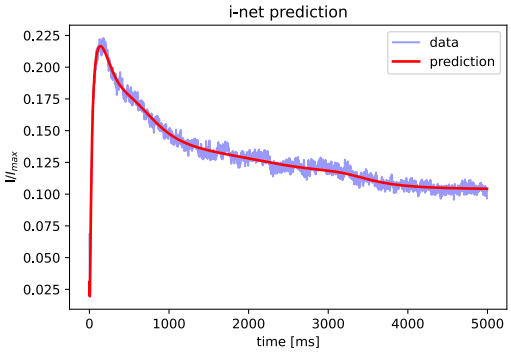


(c) Noise = 10%

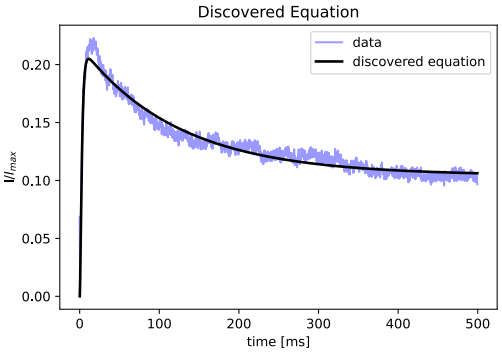
Figure 3.3: Single activation voltage: fitting artificially generated data

Experiment 2

Figure 3.4 shows the discovered equation and the predictions of the trained i-net compared to the experimental data. Table 3.3 shows the mean square errors of the i-net predictions and discovered equation with respect to the data. Note that the model was trained on $N_t = 2000$ training points and the MSEs are computed on the whole current trace.



(a) Predictions from trained i-net



(b) Discovered equation

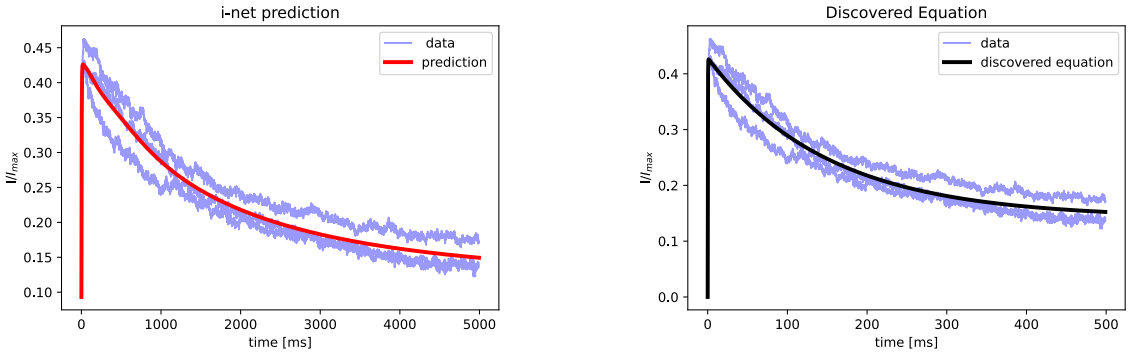
Figure 3.4: Single activation voltage: fitting experimental data

Table 3.3: Single activation voltage: Experimental data MSEs

MSE i-net	1.206e-05
MSE discovered equation	3.165e-05

Experiment 3

This time, the model is trained on three different repetitions (with an activation voltage of +10 mV). The i-net prediction and the discovered equation are presented in Figure 3.5. The MSEs of the predictions can be found in Table 3.4



(a) i-net prediction

(b) Discovered equation

Figure 3.5: Single activation voltage: fitting on several repetitions

Table 3.4: Single activation voltage: MSEs from multiple experimental repetitions

MSE i.net	6.715e-4
MSE discovered equation	6.948e-4

3.4.2 Multiple activation voltages

Experiment 4

Tables 3.5, 3.6 and 3.7 show the discovered parameters and the errors. The true parameters used to generated the training data are displayed in Table 3.1.
Figure 3.6 shows the i-net prediction and discovered equations for the multiple-activation-voltage model and for different noise levels. Table 3.8 contains the MSEs of the i-net prediction and discovered equation.

Table 3.5: Discovered parameters: noiseless data

act. [mV]	-40	-30	-20	-10	0	10	20	30	40	50
m_∞	0.0536	0.0766	0.112	0.147	0.18	0.213	0.234	0.248	0.257	0.2627
h_∞	0.34	0.501	0.541	0.518	0.476	0.435	0.401	0.373	0.350	0.332
m_τ [ms]	7.50	7.67	6.41	1.50	0.29	0.12	0.08	0.07	0.06	0.06
h_τ [ms]	151.00	134.00	127.50	128.00	128.50	126.00	122.00	117.50	113.50	110.50
Error m_∞	56.39%	62.85%	65.55%	68.83%	71.12%	71.75%	72.48%	72.85%	72.98%	73.00%
Error h_∞	62.85%	40.79%	27.48%	17.48%	8.21%	0.80%	3.11%	3.18%	0.81%	2.38%
Error m_τ	53.50%	81.71%	95.01%	34.73%	80.91%	88.43%	89.74%	89.31%	89.94%	89.47%
Error h_τ	68.48%	69.61%	66.50%	57.80%	43.57%	26.23%	10.23%	0.34%	5.39%	7.18%

Table 3.6: Discovered parameters: 1% noise

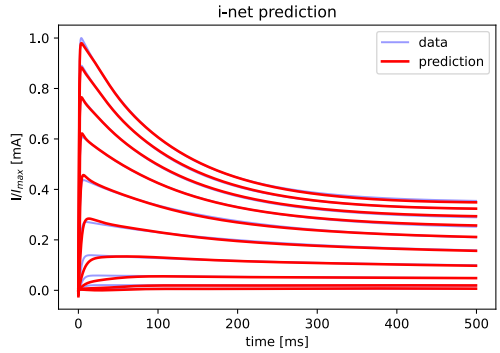
act. [mV]	-40	-30	-20	-10	0	10	20	30	40	50
m_∞	0.0536	0.0766	0.112	0.147	0.183	0.213	0.234	0.248	0.257	0.263
h_∞	0.342	0.501	0.541	0.516	0.476	0.435	0.401	0.3731	0.3502	0.3316
m_τ [ms]	8.160	7.365	3.510	0.520	0.178	0.104	0.078	0.067	0.062	0.060
h_τ [ms]	151.00	134.00	127.50	128.00	128.50	126.50	122.00	117.50	113.50	110.50
Error m_∞	56.39 %	62.85 %	65.55 %	68.83 %	71.12 %	71.76 %	72.48 %	72.85 %	72.98 %	73.00 %
Error h_∞	62.63%	40.79%	27.48%	17.80%	8.21%	0.80%	3.11%	3.21%	0.86%	2.50%
Error m_τ	67.01%	74.48%	6.78%	77.37%	88.28%	89.97%	90.00%	89.77%	89.61%	89.47%
Error h_τ	68.48%	69.61%	66.50%	57.80%	43.57%	25.94%	10.23%	0.34%	5.39%	7.18%

Table 3.7: Discovered parameters: 10% noise

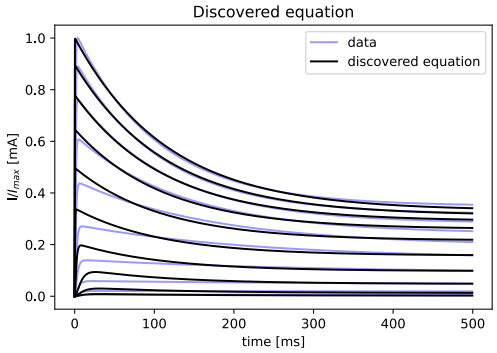
act. [mV]	-40	-30	-20	-10	0	10	20	30	40	50
m_∞	0.0600	0.0773	0.106	0.143	0.178	0.208	0.229	0.244	0.254	0.262
h_∞	0.3257	0.4437	0.5409	0.5636	0.5221	0.4644	0.4147	0.3768	0.3488	0.3280
m_τ [ms]	7.845	7.250	3.985	0.905	0.230	0.111	0.078	0.066	0.061	0.059
h_τ [ms]	137.50	133.50	125.00	119.00	119.00	120.00	118.50	116.00	113.50	110.45
Error m_∞	51.18%	62.51%	67.39%	69.68%	71.44%	72.42%	73.07%	73.28%	73.30%	73.08%
Error h_∞	64.41%	47.57%	27.49%	10.21%	0.67%	5.91%	6.63%	4.23%	0.46%	3.56%
Error m_τ	60.56%	71.76%	21.24%	60.62%	84.86%	89.30%	90.00%	89.93%	89.78%	89.65%
Error h_τ	71.30%	69.73%	67.16%	60.76%	47.74%	29.74%	12.80%	0.94%	5.39%	7.13%

Table 3.8: Multiple activation voltage: MSEs on artificially generated data

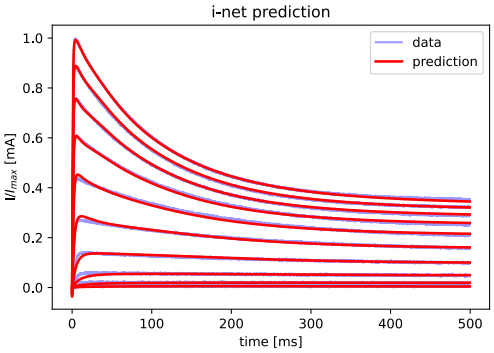
MSE i-net prediction	1.87e-5	1.72e-5	3.75e-4
MSE discovered equation	3.72e-4	3.47e-4	6.99e-4



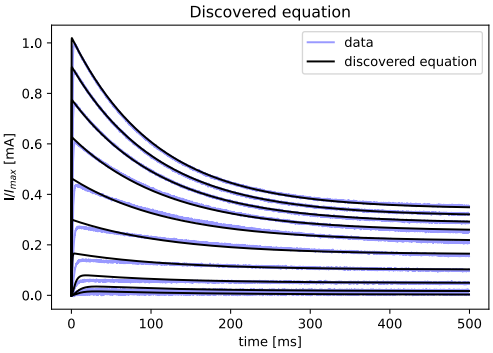
(a) i-net prediction, noiseless



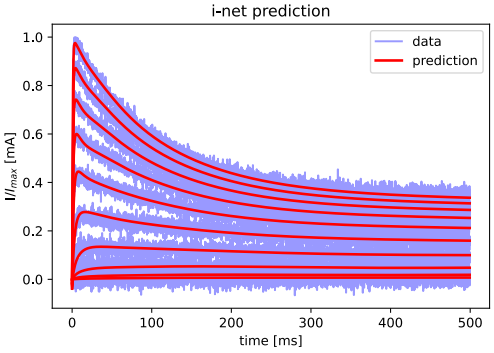
(b) Discovered equation, noiseless



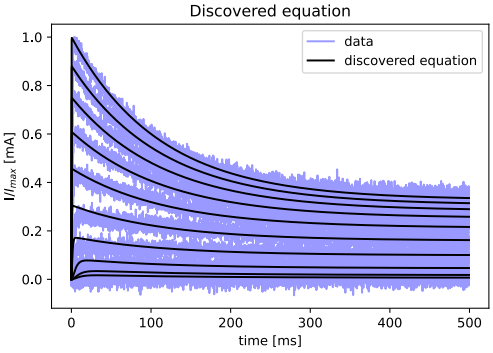
(c) i-net prediction, 1% noise



(d) Discovered equation , 1% noise



(e) i-net prediction, 1% noise



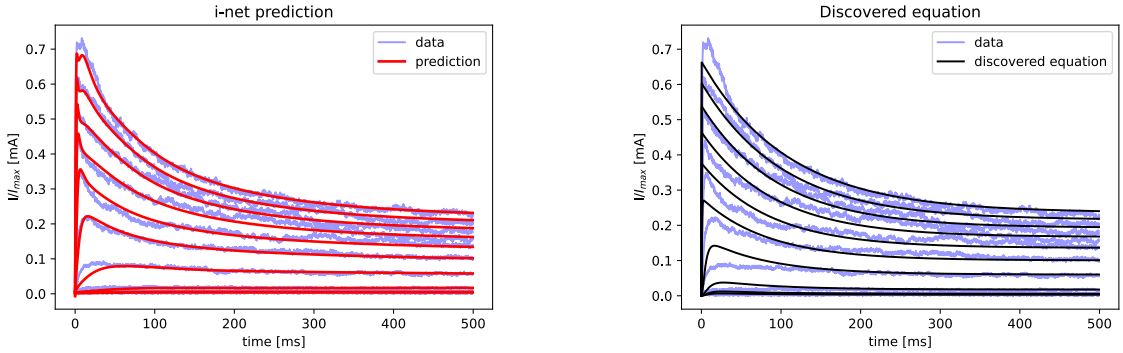
(f) Discovered equation , 10% noise

Figure 3.6: Multiple activation voltages: fitting artificially generated data

Experiment 5

Table 3.9: Multiple activation voltage: MSEs on experimental data

MSE i-net prediction	7.157e-5
MSE discovered equation	3.225e-4



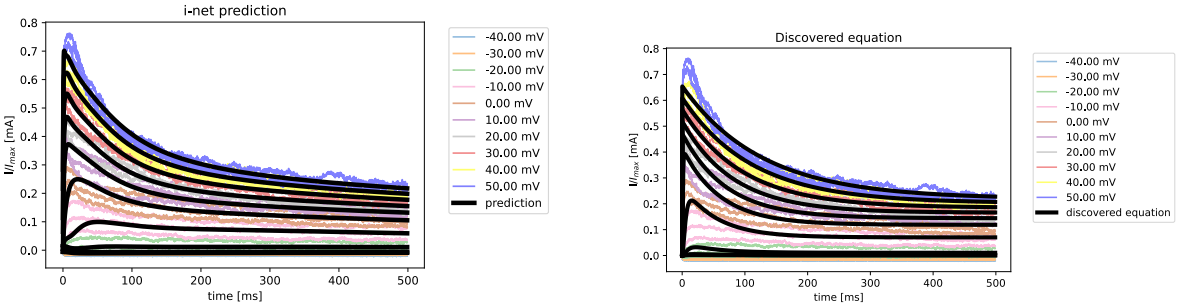
(a) i-net prediction

(b) Discovered equation

Figure 3.7: Multiple activation voltages: fitting experimental data

Experiment 6

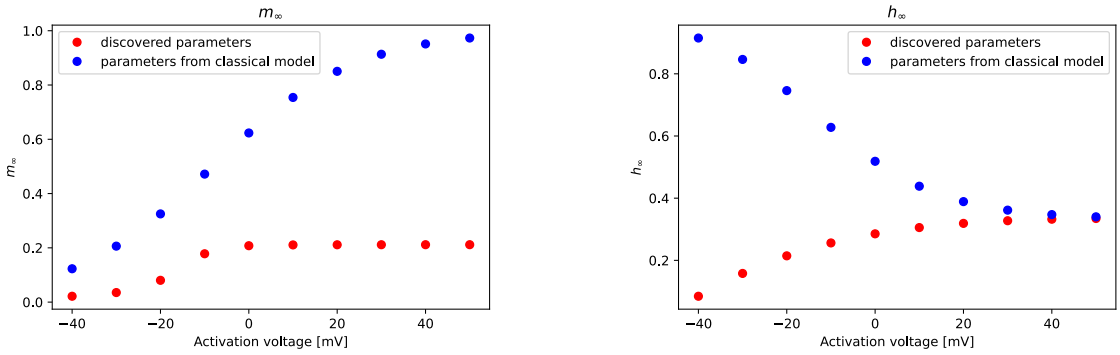
Figure 3.8 shows the i-net predictions and discovered equations when the MAV-model is trained on data from three different repetitions. Figure 3.9 shows a comparisons between the discovered parameters of the MAV-model and the parameters of the conventional Kv1.2 model.



(a) i-net prediction

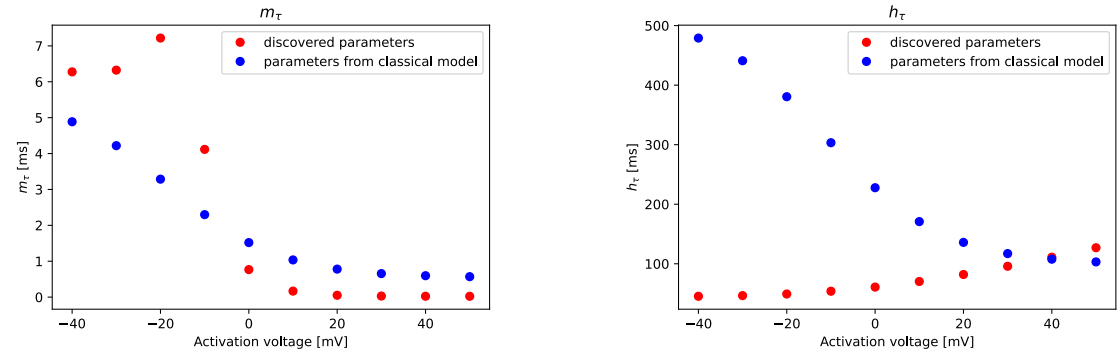
(b) Discovered equation

Figure 3.8: Multiple activation voltages: fitting experimental data



(a) m_∞

(b) h_∞



(c) m_τ

(d) h_τ

Figure 3.9: Parameters: conventional model vs. MAV-model

Table 3.10: Experiment 6

MSE i-net prediction	6.80e-4
MSE discovered equation	8.04e-4

4. Discussion

4.1 Results interpretation

Experiment 1 explores how well the model is able to recover the true parameters of the physical system in the case of a single activation voltage. From Table 3.2, one can observe that the error on the parameters h_∞, m_τ and h_τ is very low. Interestingly, the error on m_∞ is much higher. However, the MSE of the discovered equation is fairly low and Figure 3.3 shows that the discovered equation fits the data very accurately. Thus, the discovered equation describes the system correctly even though the error on the parameter m_∞ is high. This can be explained by the fact that the variable m (and therefore m_∞ and m_τ) is mostly implied in describing the fats initial rise of the current flow. To describe the fast initial rise, it is important that m_τ has a low value. The parameter m_∞, on the other hand, describes the value towards which m tends as time increases; but as the dynamics of the ion-channel is mostly described by $h(t)$ after the short initial rise, m_∞ becomes less relevant. This way the fit is accurate even though the identified parameter m_∞ is incorrect.

Experiments 2 points out that the model is able to discover parameters that elicit a good fit of the experimental data (see Figure 3.4 (b)). Moreover the trained i-net outputs correct predictions, but might over-fit the data (see see Figure 3.4 (a)).

Experiment 3 shows that the SAV-model is able to discover parameters that fit reasonably the experimental data when trained on several experimental repetitions. The fact that the discovered equation displayed in Figure 3.5(b) represents a satisfying fit of the 3 experiment repetitions suggests that the model is able to learn parameters representing an "average" model of the ion-channel. Figure 3.5(a) indicates that the i-net is able to do so as well.

Experiment 4 allows to assess whether the MAV-model can recover the parameters that were used to generate the data. Tables 3.5 to 3.7 show that there is a significant error on the parameters m_∞ and m_τ and that the error on h_∞ and h_τ is high for low activation voltages and decreases as the activation voltages increases.
The impact of the error on parameters m_∞ and m_τ, which are mostly implied in modeling the initial rise dynamics, can be observed in Figure 3.6: the discovered equations do not fit accurately the data for low values of t.
However, given the high errors on almost all parameters, one would expect the fits displayed in Figure 3.6 to be much worse that they actually are. A potential explanation for the relatively good fit despite wrong parameters, is that the variables $h(t)$ and $m(t)$ (which depend on the discovered parameters, see Eq. 3.5 and 3.6) interact with each other in the conductance equation (Eq. 3.4) and can counterbalance each other. This way if the parameters m_∞ and m_τ are incorrect, some other incorrect values of h_∞ and h_τ might counteract their effect and result in an equation that elicits an acceptable data fit.

Experiment 5 and Figure 3.7(b) show that the parameters discovered by the MAV-model fail to accurately capture the rising dynamics of the experimental data. Besides this, the fit is still reasonably good, however worse than in the simpler case of the SAV-model. This points out that the complexity of the problem increases as the activation voltage is taken into account in the model, which makes learning more challenging. In addition, one can observe that the i-net actually predicts the ion-channel dynamics better than the discovered equation, which raises the interrogation whether the neural net could be directly implemented as a "black-box" model of the ion-channel (see section 4.3).

Finally, *Experiment 6* assesses if the parameters learnt by the MAV-Model result in an accurate average model of the Kv1.2 ion-channel and evaluates how it compares to the conventional model built according to [6].
The discovered equations displayed in Figure 3.8 show an acceptable fit of the data, however as in *Experiment 5,* there is an noticeable error on the rising dynamics (for low values of t). Again, as in *Experiment 5*, one can observe from Figure 3.8 and Table 3.10, that the i-net prediction actually model the average ion-channel

dynamics better than the discovered equation.
Finally, Figure 3.9 shows that the parameters learnt by the MAV-model are significantly different from the parameters of the conventional model and do not follow the same voltage dependency dynamics. In the conventional model, the parameter-voltage relationship is imposed by specific functions whose meta-parameters are tuned to find a good data description; this is however not the case for the MAV-model where the voltage-parameter relationship is freely learnt for a set discrete voltages. This explains party why the parameters of both models are different. It is however surprising to observe that two sets of such different parameters can lead to two acceptable data fit. As mentioned above, this is possibly due to the counterbalancing effect of $h(t)$ and $m(t)$ on the conductance.

4.2 Problem specificity

It is important to note the specificity of the data-driven equation discovery implemented in this work. Indeed, only one specific type of ion-channel recorded in one specific type of host cell was considered and the experiment used to generate the experimental data is limited to voltage-clamped activation at a temperature of 35°C. Thus the obtained ion-channel model is cell-specific and temperature specific and might not be accurate for other experiments such as deactivation or recovery from inactivation (see [6]).
The experiment-specificity of the ion-channel model is reflected by the fact that the parameters discovered by the MAV-model are significantly different from the conventional model parameters even though the MAV-model fits the activation experimental data rather accurately. The parameters learnt by the MAV-model supposedly counterbalance each-other to elicit a good fit on activation data, but would potentially very inaccurately model other dynamics such as deactivation or recovery from inactivation. For those reasons, it is believed that the equation discovered by the MAV-model overfits the activation experiment and would generalize poorly on other experiments.
Finally, one of the main difficulty encountered in modeling the ion-channel dynamics based on the activation voltage-clamped experiment is that the initial rise is difficult to be learnt by the i-net. Yet, if the i-net fails to capture the initial rise dynamics, the model is also unable to discover parameters that yield a good fit of the data. Therefore, encapsulating the initial rise dynamics is crucial to obtain a suitable model. To tackle this issue, the the specificity of the activation voltage-clamped experiment is used by enforcing the initial conditions $h(0) = 1$ and $m(0) = 0$ (Eq. 3.11 and 3.12). Adding those initial conditions to the model provides enough guiding for the i-net to learn the initial rise. Nonetheless, as shown by the results, the MAV-model yields a poorer data fit for low values of t than for high values of t. Another way to enforce learning of the initial rise could be to manually weight more the data-points with low values of t in the training set.

4.3 Outlook

To conclude, the MAV-model is able to discover parameters that fit reasonably both artificially generated and experimental data. However, the discovered parameters diverge significantly from the true parameters and the conventional model parameters respectively. This suggest that the developped PINNs-based approach allows to model ion-channel dynamics correctly but, in the specific case of this work, overfits the activation experiment (because the model was exclusively trained on activation experiments). Incorporating more experiments to the training pipeline (such as deactivation or recovery from inactivation) could help tackle this issue. Alternatively, trying to enforce a specific parameter-voltage relationship could also be considered. Indeed, in the case of the conventional model developed according to [6], the parameter-voltage dependencies are described by specific mathematical functions. However, in the case of the PINN-model, the parameters are learnt independently for a set of discrete activation voltages. Adding new terms in the training loss to force the model to learn a specific parameter-voltage relationship could be investigated in further work.
Additionally, the MAV-model designed in this work is experiment-specific, temperature-specific and host-cell specific. An interesting extension to train the model on data from more experimental repetitions, from other host cells or other experimental setups in order to obtain a more general model of the targeted ion-channel.

Another potential extension would be to consider more input parameters. Indeed, the MAV-model uses time t and voltage v as input variables, but more input variables could be added to the model, such as the temperature or the host cell. Furthermore, the implemented neural networks are simple feed-forward networks with *tanh()* activation function. This is a rather a naive structure and exploring more complex neural network architectures would be an interesting model extension.
Finally, the results from *Experiments 4 to 6* show that the i-net predictions tend to fit the data better than the discovered equations. This raises the following interrogation: could the trained i-net directly be used as a "black-box" model of the ion-channel? This questions how well the i-net would perform if tested on other set-ups than activation voltage-clamped experiment and if would be able generalize well on unseen data. Moreover, it is unsure whether a forward pass through the i-net is as computationally efficient as solving the Hodgkin-Huxley equations for some discovered parameters.

A. Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: NIPS'12 (2012), pp. 1097–1105.

[2] Babak Alipanahi et al. "Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning". In: *Nature biotechnology* 33.8 (Aug. 2015), pp. 831–838. ISSN: 1087-0156. DOI: 10.1038/nbt.3300. URL: https://doi.org/10.1038/nbt.3300.

[3] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations". In: (2017). arXiv: 1711.10561 [cs.AI].

[4] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations". In: (2017). arXiv: 1711.10566 [cs.AI].

[5] Atilim Gunes Baydin et al. "Automatic differentiation in machine learning: a survey". In: (2018). arXiv: 1502.05767 [cs.SC].

[6] Rajnish Ranjan et al. "A Kinetic Map of the Homomeric Voltage-Gated Potassium Channel (Kv) Family". In: *Frontiers in Cellular Neuroscience* 13 (2019), p. 358. ISSN: 1662-5102. DOI: 10.3389/fncel.2019.00358. URL: https://www.frontiersin.org/article/10.3389/fncel.2019.00358.

[7] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2017). arXiv: 1412.6980 [cs.LG].

[8] Dong C. Liu and Jorge Nocedal. "On the Limited Memory BFGS Method for Large Scale Optimization". In: *Math. Program.* 45.1–3 (Aug. 1989), pp. 503–528. ISSN: 0025-5610.

[9] Yanan Guo et al. "Solving Partial Differential Equations Using Deep Learning and Physical Constraints". In: *Applied Sciences* 10.17 (2020). ISSN: 2076-3417. DOI: 10.3390/app10175917. URL: https://www.mdpi.com/2076-3417/10/17/5917.

[10] Ciyou Zhu et al. "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization". In: *ACM Trans. Math. Softw.* 23.4 (Dec. 1997), pp. 550–560. ISSN: 0098-3500. DOI: 10.1145/279232.279236. URL: https://doi.org/10.1145/279232.279236.