Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Research Project

Neural Networks as Error Estimators

Robin Sasse, Philipp Schmid, Tobias Weinschenk

Course of Study:	Informatik B.Sc.
Examiner:	Prof. Dr. Miriam Schulte
Supervisor:	Felix Huber M.Sc.
Commenced:	December 22, 2020
Completed:	June 22, 2021

Abstract

Numerical simulations are used in a variety of computational problems. The error of the simulation is often of interest here. However, the error is usually not known and must be calculated using expensive methods such as adjoint error equations. We investigate the use of multiple Convulational Neural Networks (CNNs) architectures as efficient and reliable error estimators for muscle fiber simulations by opendihu.

Contents

1 Introduction

Numerical simulations are used in a variety of computational problems. These methods usually depend on one or multiple temporal and spacial step size parameters. When step sizes are small, the simulation is more exact but takes more time to compute. This yields a secondary problem which is the task of determining the error made by the numerical simulation. In this work we research the possibility of using neural networks as estimators for an adjoint error equation. The objective is to develop an estimator that is faster than the error equation and produces close enough estimates for that error. To understand the context of this error equation, we firstly have to introduce the setting.

The specific problem in our setting is the calculation of muscular impulses with the software opendihu [MEKM19]. Opendihu can calculate the propagation of electrical and chemical waves through muscle fibres. The software takes a current state of the electrical wave and 3 chemical waves as an input and simulates the propagation of these waves for a specified period of time. The electrical channel is our primary object of interest for determining the error made by opendihu; meaning that we use the final state of only the electrical channel as a basis for defining the error. Nonetheless, the chemical channels are not to be ignored as they are responsible for the wave-like propagation of the electrical impulse. Therefore, they need to be included in our estimator.

At specified points in time, opendihu outputs snapshots of the current state of the system and thus, effectively creates a trajectory of the waves over time. The snapshot of most interest to us is the final state of the simulation. Here we want to know how close the final result of the electrical channel is to that of the true solution. The true solution solely depends on the initial values, that is, the initial state of the 4 channels and is obtained by solving the underlying PDE describing the problem. The discrete solution, calculated by opendihu, additionally depends on 2 time step parameters n_0 and n_1 which determine how many time steps are performed in a certain period in time. n_0 is the reaction term of the equation and corresponds to an explicit time step (Heun). n_1 is the diffusion term of the equation and corresponds to an implicit time step (Crank-Nicolson). Essentially, n_0 and n_1 can be seen as parameters for dividing the time steps. Therefore, these parameters also influence the final result. The difference between the true solution and the discrete solution is the error ε that we are interested in. The error can be split into 2 major components $\varepsilon(n_0, n_1) := \varepsilon_0(n_0) + \varepsilon_1(n_1) + r(n_0, n_1)$, where $r(n_0, n_1)$ is small enough to be neglected. Each component only depends on its respective time step division $\varepsilon_0 := \varepsilon_0(n_0)$, $\varepsilon_1 := \varepsilon_1(n_1)$. This will be of importance for our data generation later on. Larger values for n_0 and n_1 imply a higher numerical accuracy and thus, smaller numerical errors.

The error terms ε_0 and ε_1 can be estimated with an adjoint error equation that uses the spacial and temporal differences of the trajectory [EGR+08]. Even though this equation has the advantage that we only need the trajectory in order to determine the discretization error, it is nonetheless expensive because it repeatedly applies a stencil operation to the trajectory image to solve the adjoint problem. Therefore, we want to investigate if neural networks can be used as good enough estimators for this equation.

2 Convolutional Neural Networks as Error Estimators

Convolutional Neural Networks (CNN) are used in a variety of fields related to pattern recognition; such as image processing and voice recognition [AMA17]. To understand the CNNs used in this work we firstly have to discuss their basic elements and features.

Layers: Like classical neural networks, CNNs consist of multiple layers. These must include the input and output layer and may include arbitrarily many hidden layers. The inputs are processed passing from the input layer, through the hidden layers, to the output layer.

Dimensions: We model our CNNs based on basic designs for image processing. These take 3 dimensional inputs which can be thought of as 2D images with multiple channels. Thus, most layers (except for the dense layers) of the CNN are made up of rank 3 tensors ($X \times Y \times \text{RGB}$) or rank 4 tensors when dealing with batches ($N \times X \times Y \times \text{RGB}$).

Convolution Layers: Any CNN has at least one convolution layer, but usually more. Convolutions are used to extract some type of feature from the image. In these layers a convolution kernel is applied to the image, mixing information across all the available channels. The convolution kernel applies a learned function to each pixel and its surrounding pixels. Which pixels are included in this function depends on their proximity to the central pixel and the size of the kernel. To avoid information loss at the image boundaries, a padding can be added to the image. Furthermore, the architect of the CNN can specify how many output channels are created by this kernel. Intuitively, this can be thought of as extracting a variety of different features from the same image.

Pooling Layers: A pooling layer is used to downsample the image at a defined point in the net. The main objective is to lower the complexity of the following operations. Various forms of pooling, such as max pooling and average pooling, exist and are applied for different purposes.

Dense Layers: The dense layer (also called fully connected layer) is the typical layer of the classical neural network. It connects all nodes of the previous layer with all nodes of the following layer. The dense layers are usually applied in the last layer or last couple of layers. Their job is to combine the features previously extracted by the convolutional layers and derive some final result from them.

Even though they are manly used in computer vision, CNNs have already been used for the purpose of solving PDEs in the past [TSSP17]. Therefore, we focus on these networks for the design of our error estimator. As an analogy to the image-processing CNN we view the trajectory generated by opendihu as an image. In this analogy the different channels (electrical and chemical) correspond to the RGB channels of an image and one of the image's spacial dimensions corresponds to the temporal dimension of our trajectories ($X \times Y \times RGB = \text{Space} \times \text{Time} \times \text{channels}$). The objective for the CNN is to find a function that takes a trajectory as an input and approximates the errors ε_0

and ε_1 by its output $(\hat{\varepsilon_0}\ \hat{\varepsilon_1})^{\mathrm{T}}$. We decided not to include the parameters n_0 and n_1 as inputs of the CNNs in order to avoid that they learn a function solely based on these parameters. The adjoint error equation, we want to approximate, does not take these as inputs either.

Our hope is that the convolution kernels of the CNN can learn to estimate the errors by having its convolution kernels approximate the stencil of the underlying adjoint equation. Furthermore, we expect it to find a more efficient method in doing so (e.g. using less convolutions for a good enough approximation or even combing certain extracted features well in the CNN's dense layer).

Architecture of the CNN

For our experiments we have to select an architecture first. As our objective is to learn the stencil operation of the adjoint error equation, we first chose an architecture with many convolution layers as shown in Figure 2.1. We include 2 max-pooling layers to lower the complexity and reduce the number of dense parameters in the fully connected layer. We thus improve the balance between convolutional and dense parameters. Our assumption is that this will potentially give us a closer fit to the error equation which we want to estimate. Whereas a net, consisting mostly of dense parameters, would most likely learn some other function with a tendency of being overfitted.

Before potentially losing important information we added some extra convolution layers at the beginning of the net, giving us a total of 4 convolution layers before the first pooling. In the middle we added 2 more convolution layers, followed by another pooling. Both pooling layers pool stronger in the spacial dimension than in the temporal. The reasoning being that we have more spacial data points than those in the temporal direction.

Afterwards we flatten the resulting tensor and map the output to the vector $(\hat{\varepsilon_0}\ \hat{\varepsilon_1})^{\mathrm{T}}$ via the fully connected layer. We henceforth refer to this architecture as CNN0. It has a total of just over 300 thousand trainable parameters of which half of these parameters are to be found in the dense layer.

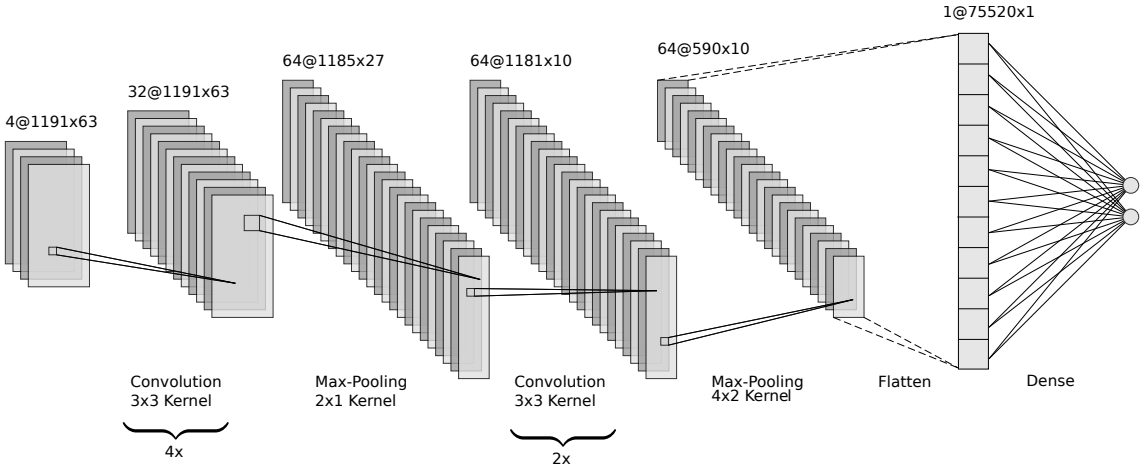


Figure 2.1: Architecture of CNN0 (horizontal = space, vertical = time, layers = features)

After running some tests on smaller training samples, we decided to experiment with a second architectural design shown in Figure 2.2. In this approach we test the possibility of including more convolution layers where each convolution layer produces more features. In order to achieve this

without unnecessarily increasing complexity, we decided to include more pooling layers and also have them pool stronger in the temporal dimension. This architecture will henceforth be referred to as CNN1. It has just over 450 thousand trainable parameters of which only 110 thousand are in the dense layer.

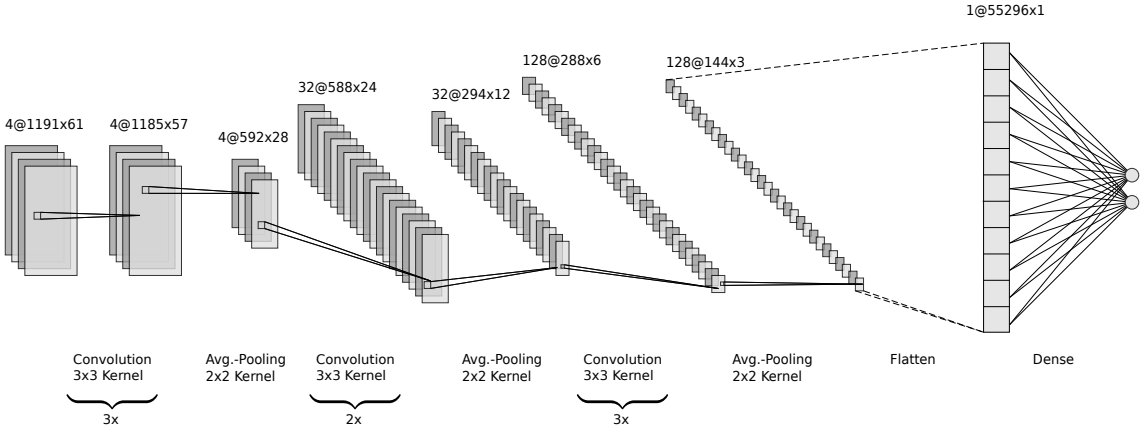


Figure 2.2: Architecture of CNN1 (horizontal = space, vertical = time, layers = features)

We further tested architectures with larger kernels (5×5, 7×7 or unsymmetrics kernels such as 1×3 or 3×9), more or less dense layer parameters (between 10 thousand and 5 million), and even exotic structures that work without any pooling nor any dense layers by letting the entire image collapse through unpadded convolutions. Whereas the nets with more parameters in the fully connected layer tended to overfit to the training data, the nets with too much focus on convolutions tended to result in constant predictors. As both of these results are not desirable, we decided to stay with CNN0 and CNN1 as our baseline architectures for experimenting with larger data sets and modify these based on our results in the following chapters.

The modifications on those nets include the reduction of convolution layers (these get the title "reduced") and then a second reduction focusing on the reduction of features produced by a convolution layer (these get the title "reduced2"). Because the "reduced" nets potentially get rid of paddingless convolution layers, the number of dense parameters in the last layer may increase. This is a direct result of not losing the boundaries in that operation. The final number of parameters for each net is shown in Table 4.1.

3 Data Generation

For the purpose of training the chosen neural networks, we created a total of 10,020 data sets, split 80% for training and 20% for validation. Furthermore, we created 1,000 trajectories for the final testing of our CNNs. Each trajectory is a potential input for a net and each error associated with that trajectory is the respective output. The nets then learn a function that estimates an error for a given trajectory.

In order to generate these trajectories we defined an initial state of the 4 channels as shown in Figure 3.1. This state is representative of an impulse propagation 70ms after the initial impulse has been received. We chose this point based on where we can expect numerically stable results for the rest of the simulation.

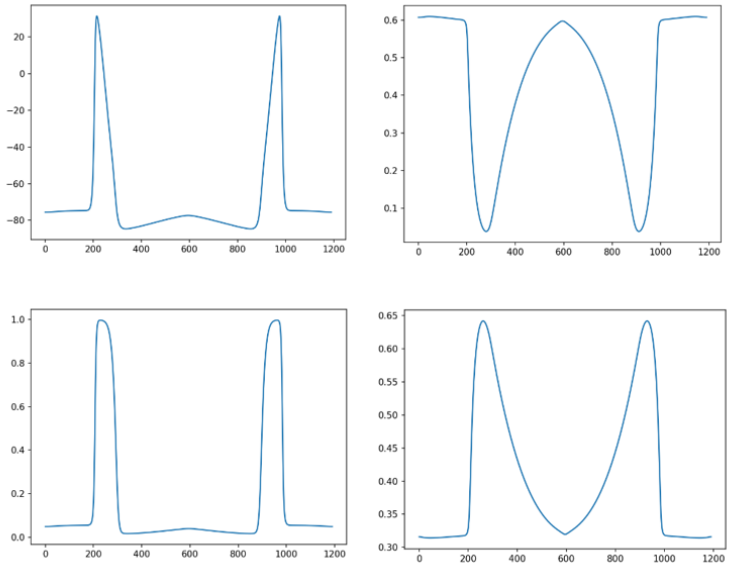


Figure 3.1: Initial state of the electrical (upper left) and 3 chemical channels

In the next step we added random noise to this state in order to generate 500 additional initial states for the simulation. This noise consisted of:

- moderate shifts of all 4 channels, equally far to the left or right, with a constant padding at the opposite side
- scaling of the amplitude between 0.9 and 1.1, uniformly distributed, each channel with a separate randomized factor
- adding of up to 2 sinus waves with amplitudes between -1.5 and 1.5 and frequencies between $\frac{1}{2\pi}$ and $\frac{100}{\pi}$, separately randomized for each channel

We simulated the impulse propagation of each of these 501 initial configurations for 6ms giving us the full trajectory, shown exemplary for the noiseless configuration in Figure 3.2. We repeated this simulation 30 times for each of the initial configurations, each time with different settings for n_0 and n_1, where $n_0 \in \{2, 4, 8, 16, 32\}$, $n_1 \in \{1, 2, 4, 8, 16, 32\}$. All the trajectories with $n_0 = 32$ or $n_1 = 32$ were used as a basis for calculating the error. Since both components of the error are only dependent on their respective step division, the error can be calculated as follows (FS_e denotes the final state of the electrical channel and $\varepsilon_i(n_i), i \in 1, 2$ denotes the partial error of a specific simulation with time division n_i):

$$\varepsilon_0(n_0) = ||FS_e(x, y) - FS_e(32, y)||_2 + \varepsilon_0(32), \text{with } x, y \in \{1, 2, 4, 8, 16\}$$

$$\varepsilon_1(n_1) = ||FS_e(x, y) - FS_e(x, 32)||_2 + \varepsilon_1(32), \text{with } x, y \in \{1, 2, 4, 8, 16\}$$

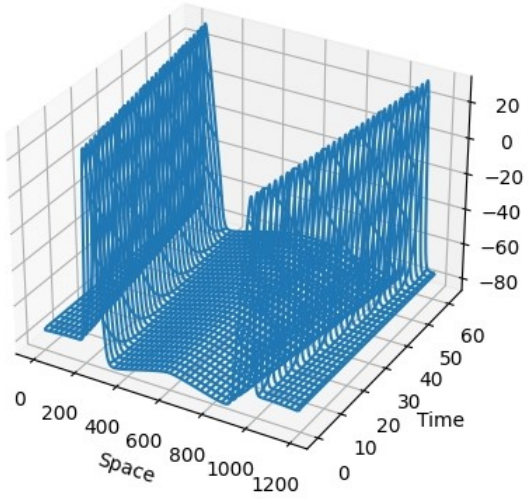


Figure 3.2: Example of a trajectory with 63 temporal and 1191 spacial data points (only the electrical channel)

Since doubling n_i yields an error decreased by factor 4 (second order method), the error $\varepsilon_i(32)$ made by our most precise setting can be neglected for any trajectory generated with time division settings of 16 or lower. Therefore, we obtain 20 useful trajectories for each of the 501 initial problems, giving us a total of 10,020 data points for training and validation. Similarly we generated another 50 trajectories to produce the remaining 1,000 test data for our final analysis of the performance of each net.

To train our nets we had to choose a loss function between the nets' outputs $(\hat{\varepsilon_0}\ \hat{\varepsilon_1})^{\mathrm{T}}$ and the errors ε_0 and ε_1. Since ε_0 and ε_1 change by an order of magnitude depending on the value of n_0 or n_1 respectively, we are interested in some type of percentage error. Because the mean absolute percentage error (MAPE) lead to numerical instability during training, we decided to use the mean squared percentage error (MSPE) instead. We chose Adam for optimization [KB14].

4 Results

Using the settings defined above, our first trainings of CNN0 and CNN1 on the entire data set yielded a constant predictor no matter for how long we trained (up to 10,000 epochs). While more exotic architectures showed this behaviour on smaller data sets, we could not obtain any useful result on the full data regardless of the architecture we chose.

Therefore, we opted for a different approach. Instead of trying to predict the error, we now wanted to predict the logarithm of the error. The reason behind this decision was that while the differences in the trajectories were rather subtle, the differences in the desired outputs were extremely high, with magnitudes varying from 10^{-4} to 10^{-2}. Using the logarithm of the error brings the outputs closer together thus, making it easier for the CNNs to learn their function.

This improvement proofed to be effective. Since this way our output values are already shown by their order of magnitude rather than their true value, we decided to use mean squared error (MSE) instead of MSPE. Changing to MSE further improved the quality of the optimization procedure. This is an interesting phenomenon and deserves more attention of researchers in the future.

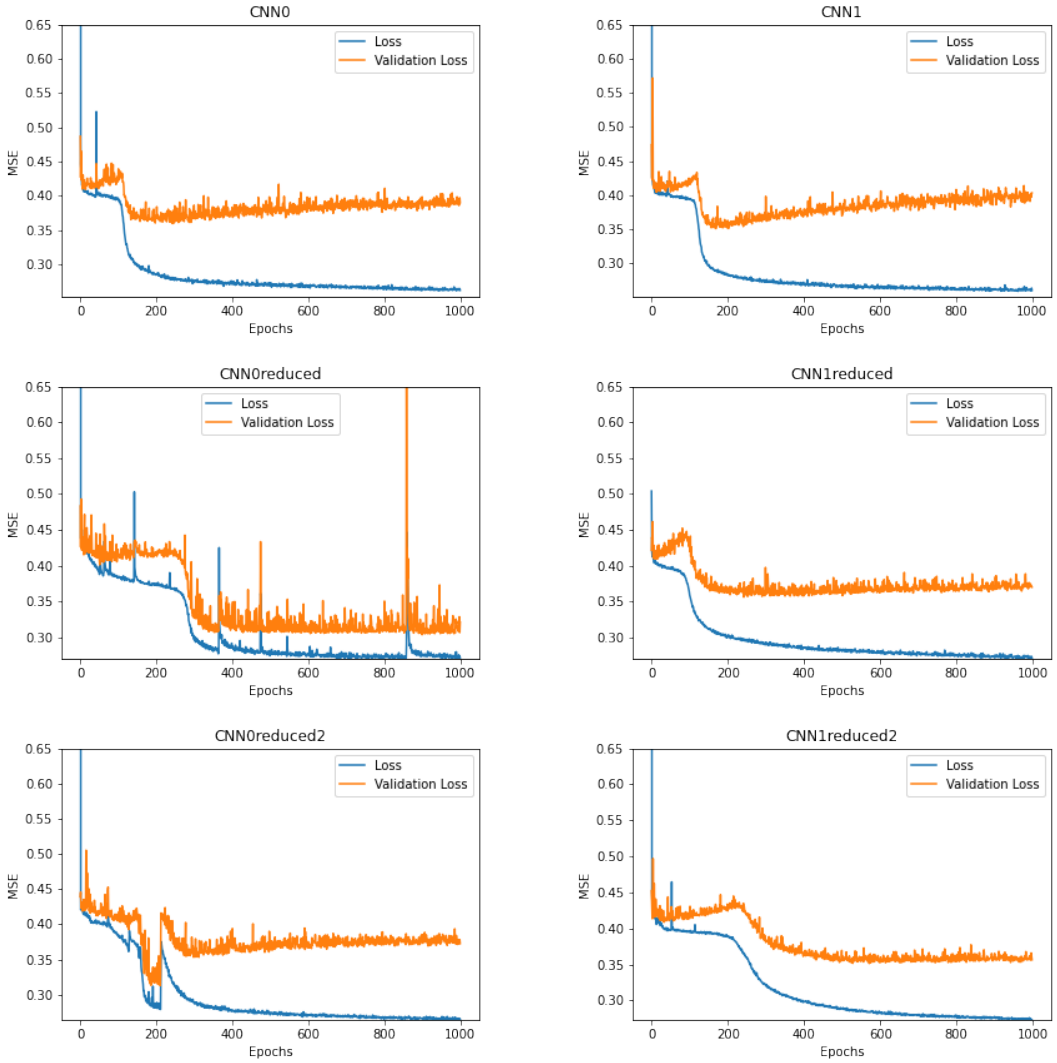
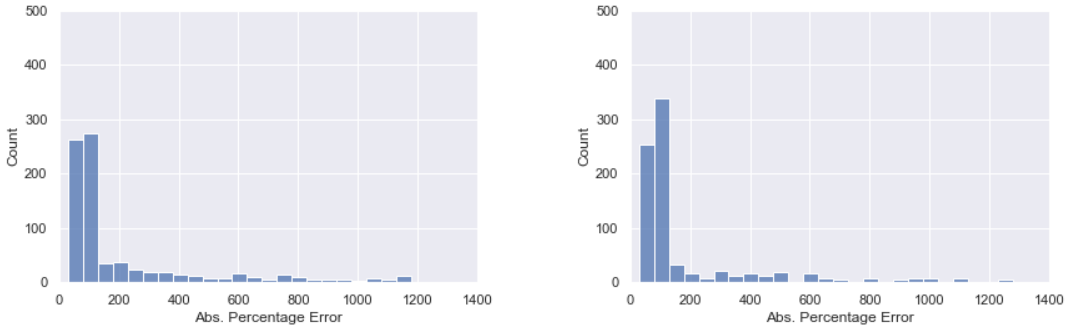


Figure 4.1: MSE over 1,000 epochs of training for different CNNs on the training and validation data

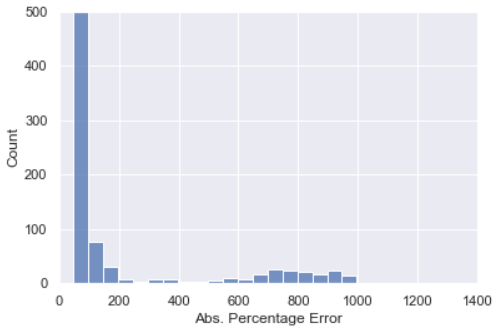
In Figure 4.1 the training of our 2 base nets and 2 modifications of each of those nets, all of them with less parameters than the original, are displayed. Our experiments show that for each architecture an optimum was usually reached within the first 500 epochs of training. After that no major improvements were made. Most architectures even tended to overfit (increasing validation loss) when training longer which is a phenomenon observed quite frequently and lead to the regularization technique of early stopping [GBC16].

For CNN0 the reduction of convolution parameters leads to a stronger performance on the validation set. This happens even as the performance on the training data worsens, indicating overfitting and that our initial design had too many parameters in relation to our sample sizes. CNN0reduced seems to yield results that are more generalizable. For CNN1 the reduction of parameters does not have a significant effect on the training. Nonetheless, an interesting observation that can be made is that CNN1 seems to be prone to overfitting (indicated by a decrease in training loss with a simultaneous increase in validation loss) when early stopping is not applied. This further indicates that our initial designs had too many parameters rather than too few. Further reduction of extracted features per convolution layer does not seem to have any significant effect on the nets' performance.

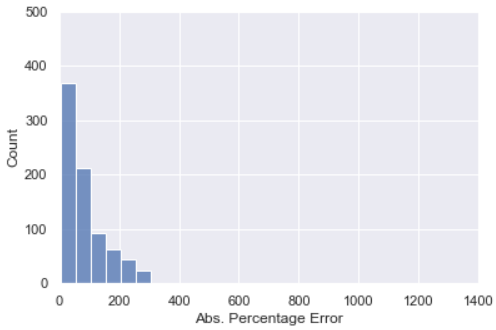


(a) CNN0 ε_0 (mean = 405%)

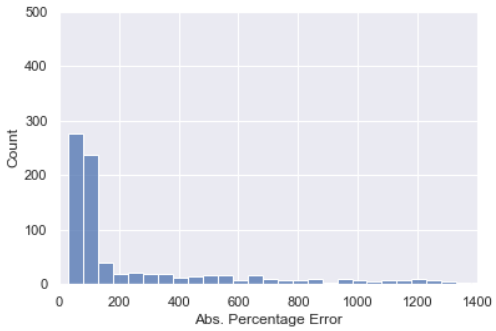
(b) CNN0 ε_1 (mean = 3,420%)



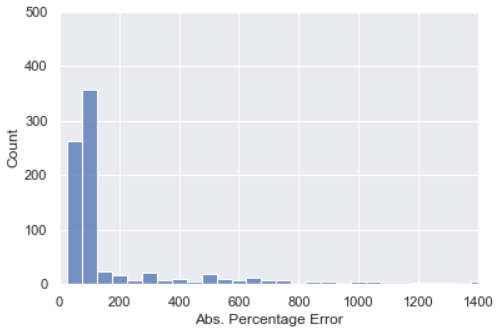
(c) CNN0reduced ε_0 (mean = 456%)



(d) CNN0reduced ε_1 (mean = 245%)

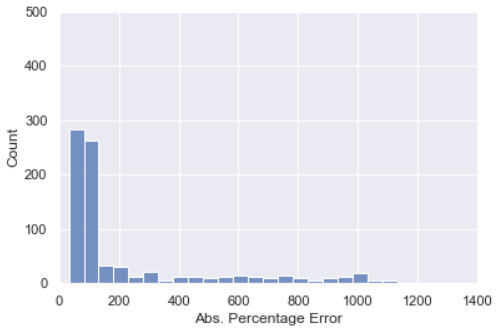


(e) CNN0reduced2 ε_0 (mean = 545%)

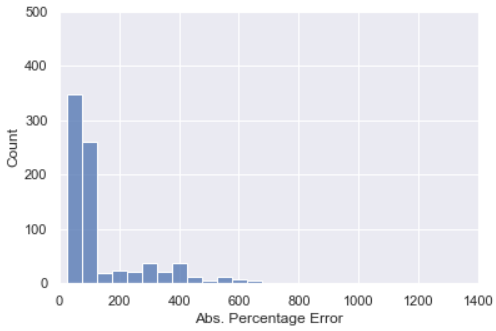


(f) CNN0reduced2 ε_1 (mean = 997%)

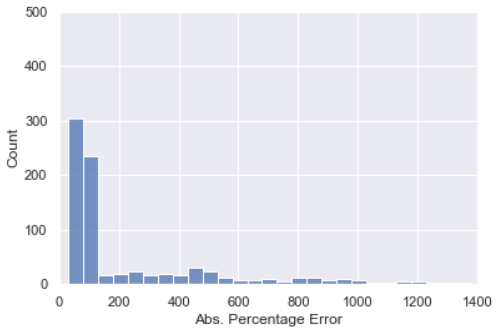
Figure 4.2: Percentage-wise absolute divergence of the estimated output from the actual error for each net with base design CNN0



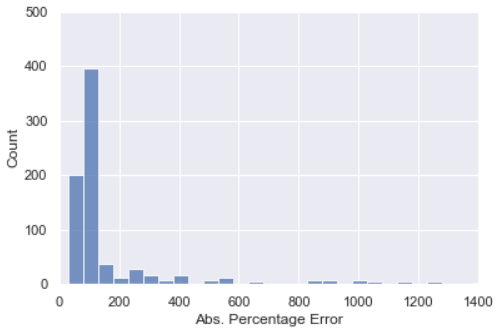
(a) CNN1 ε_0 (mean = 424%)



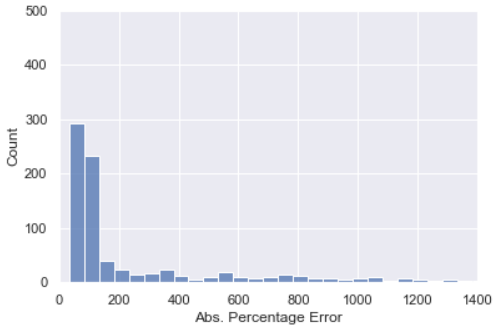
(b) CNN1 ε_1 (mean = 257%)



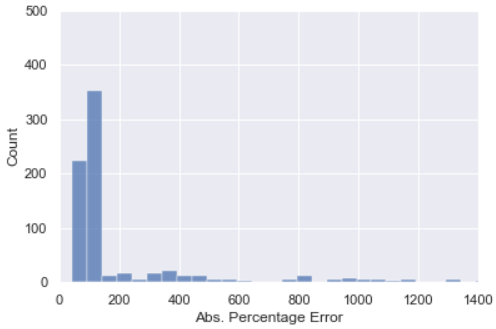
(c) CNN1reduced ε_0 (mean = 423%)



(d) CNN1reduced ε_1 (mean = 2,322%)



(e) CNN1reduced2 ε_0 (mean = 517%)



(f) CNN1reduced2 ε_1 (mean = 12,683%)

Figure 4.3: Percentage-wise absolute divergence of the estimated output from the actual error for each net with base design CNN1

A look at the performance on the test data in Figure 4.2, confirms the assessment. Our initial design for CNN0 had too many parameters and was thus, prone to overfitting. For further researchers we suggest smaller nets when using a similar design. Furthermore, we want to point out that CNN0reduced had the most dense parameters in this design group and the best performance on new data. Looking at classical neural networks is therefore encouraged. These only consist of fully connected layers.

Net	Conv. Param.	Dense Param.	Total Param.	MAPE ε_0	MAPE ε_1
CNN0	158,208	151,042	309,250	405%	3,420%
CNN0reduced	65,856	228,098	293,954	456%	245%
CNN0reduced2	16,800	114,050	130,850	545%	997%
CNN1	343,036	110,594	453,630	424%	257%
CNN1reduced	186,056	148,482	334,538	423%	2,322%
CNN1reduced2	47,096	74,242	121,338	517%	12,683%

Table 4.1: Trainable parameters and performance of each net

For CNN1 and the derived architectures we observe a different behaviour; one that also diverges from our earlier analysis based on the test and validation data plots in Figure 4.1. We can see that the generalization qualities of this architecture decline as the number of parameters is reduced, as shown in Figure 4.3. A clear statement about the correct number of trainable parameters is thus, not possible.

From our results, we can conclude that CNNs can make some approximation to the logarithm of the error made by opendihu's muscle fibre simulation. That said, the quality of our results is not sufficiently high for any real-world applications. Further improvements on the architectural design of CNNs for this specific task are necessary in order to develop any viable error estimator.

Regarding the topic of choosing a good size for a CNN in this particular context, we cannot give any clear directions. The results seem to depend too much on the underlying base architecture. We refer to Table 4.1 for a detailed view on the individual performance of each net.

Lastly, we want to point out the similar performance of each net on ε_0. We do not claim to fully understand this phenomenon but suppose that this might be an indicator which signals that we have not found a design that is capable of actually emulating the stencil operations of the adjoint error equation. If so, we would have expected a result which was much better than the rest of the nets. It might also indicate that the term ε_1, which depends on an implicit method (Crank-Nicolson), is more sensitive to a net's architecture.

For a final cross-validation of the quality of our results we refer to A.1 and A.2.

5 Conclusion and Outlook

To conclude, our results show that CNNs can extract some useful information from a muscle impulse trajectory. This might be due to an emulation of the stencil operation of the adjoint error equation. Nonetheless, it might just arise as a result of a less convincing function learned by the CNNs. Further research of what these nets actually extract from the data is definitely necessary.

The performance of our nets on new data is inconsistent at best and does not suffice for error estimation in practice. Nonetheless, we have shown that at least some approximations are not too far off. Therefore, we believe that further research on custom architectures for error estimating neural networks can lead to better results. Such research may be aimed at, but should not be limited to, CNNs. Especially the optimized performance of CNN0reduced, which has more dense parameters than its original base design, suggests the possibility of using classical neural networks as error estimators.

Furthermore, we suggest to try out Long Short-Term Memory (LSTM) networks. We expect these to be very promising candidates for estimating complex numerical equations. Sirignano and Spiliopoulos [SS18] have shown that LSTMs can be used successfully in this context. The advantage of LSTMs lays in their strong performance on sequenced data. A trajectory can be disassembled along the time axis into a sequence of states of the muscle fibre. Therefore, we consider their use as a valuable approach to solving this problem.

Lastly, we want to highlight another field of research we came across. The difference in performance of our neural nets on ε_0, which depends on an explicit method, and ε_1, which depends on an implicit method, might be related to a greater problem. This might also be of interest for future researchers working on neural networks in the context of approximating temporal numerical equations.

Bibliography

[AMA17] S. Albawi, T. A. Mohammed, S. Al-Zawi. “Understanding of a convolutional neural network”. In: *2017 International Conference on Engineering and Technology (ICET)*. Ieee. 2017, pp. 1–6 (cit. on p. 5).

[EGR+08] D. Estep, V. Ginting, D. Ropp, J. N. Shadid, S. Tavener. “An a posteriori–a priori analysis of multiscale operator splitting”. In: *SIAM Journal on Numerical Analysis* 46.3 (2008), pp. 1116–1146. DOI: https://doi.org/10.1137/07068237X (cit. on p. 4).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 7, pp. 241–249 (cit. on p. 10).

[KB14] D. P. Kingma, J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 9).

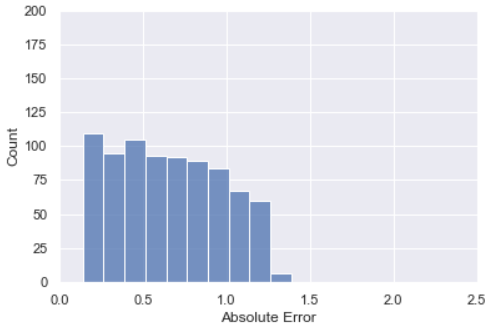
[MEKM19] B. Maier, N. Emamy, A. Krämer, M. Mehl. “Highly parallel multi-physics simulation of muscular activation and EMG”. In: *COUPLED VIII: proceedings of the VIII International Conference on Computational Methods for Coupled Problems in Science and Engineering*. CIMNE. 2019, pp. 610–621 (cit. on p. 4).

[SS18] J. Sirignano, K. Spiliopoulos. “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of computational physics* 375 (2018), pp. 1339–1364 (cit. on p. 14).

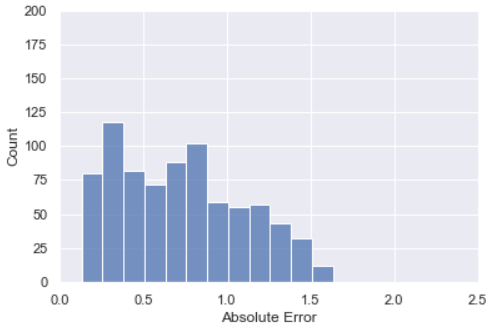
[TSSP17] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin. “Accelerating eulerian fluid simulation with convolutional networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3424–3433 (cit. on p. 5).

All links were last followed on June 20, 2021.

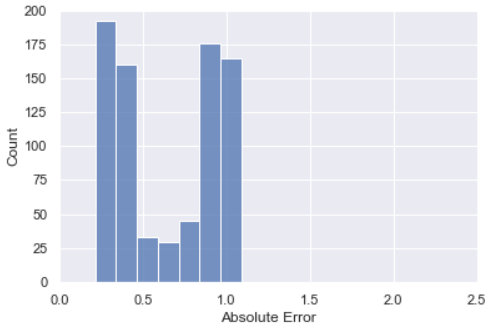
A Appendix



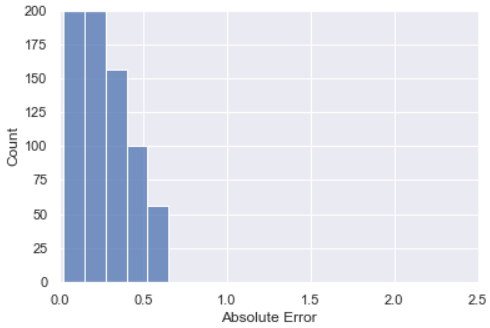
(a) CNN0 ε_0 (mean = 0.676)



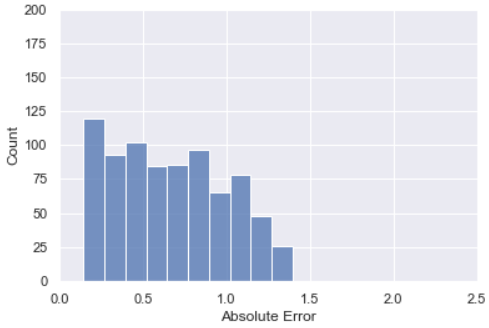
(b) CNN0 ε_1 (mean = 0.805)



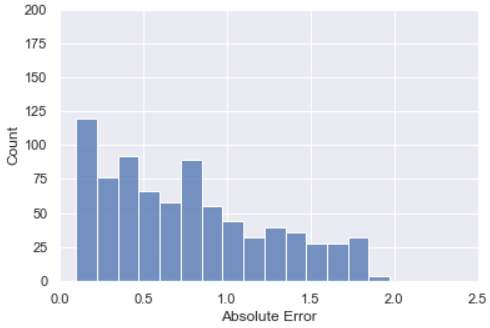
(c) CNN0reduced ε_0 (mean = 0.661)



(d) CNN0reduced ε_1 (mean = 0.297)

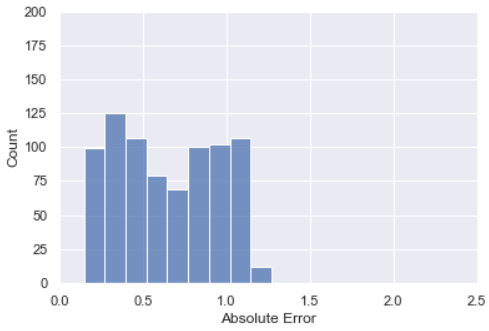


(e) CNN0reduced2 ε_0 (mean = 0.697)

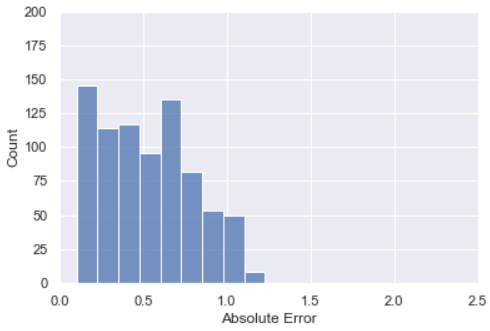


(f) CNN0reduced2 ε_1 (mean = 0.850)

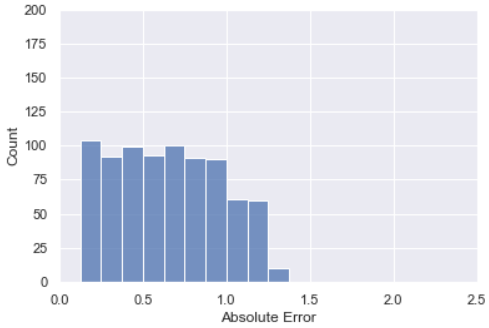
Figure A.1: Absolute divergence of the logarithm of estimated output from the logarithm of the actual error for each net with base design CNN0



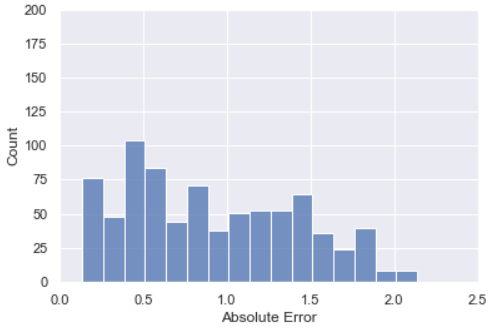
(a) CNN1 ε_0 (mean = 0.661)



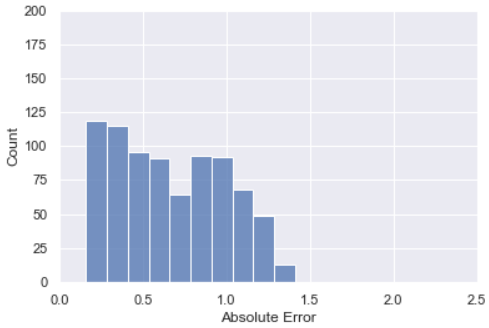
(b) CNN1 ε_1 (mean = 0.565)



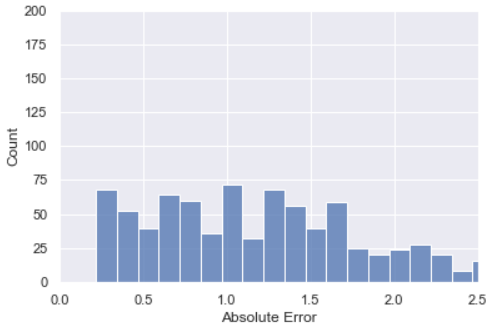
(c) CNN1reduced ε_0 (mean = 0.668)



(d) CNN1reduced ε_1 (mean = 0.990)



(e) CNN1reduced2 ε_0 (mean = 0.693)



(f) CNN1reduced2 ε_1 (mean = 1.300)

Figure A.2: Absolute divergence of the logarithm of estimated output from the logarithm of the actual error for each net with base design CNN1

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature