CTCR Deep Learning - Feedforward Neural Network Analysis

1st Michael Marinaccio
Clemson Department of Electrical and Computer Engineering
Clemson University
Clemson, SC, United States
mmarina@g.clemson.edu

***Abstract*—In the world of continuum robotics, physics-based modeling has developed significantly. Physics-based models are able to accurately define how each section of the robot should position and orient itself to achieve a desired end effector position and orientation. With physics-based models, they serve as highly advanced estimators of position and orientation. There is a point at which it becomes increasingly more difficult to improve upon the accuracy of these models, paving the way for new approaches. With the rise of deep learning accessibility in recent years, it is intriguing to see how well deep learning models could be applied to continuum robotics. This paper provides an analysis of Feedforward Neural Networks based on the first publicly available concentric tube continuum robotic (CTCR) dataset released by the Continuum Robotics Laboratory in Toronto, Canada. The work provides a comprehensive overview, through the analysis of FNN's, for how different preprocessing techniques, loss functions, activation functions, batch size changes, and overall architecture behave with this dataset. Methods for comparison include measurements of translational and rotational error, as well as how well the given model's Cartesian and Quaternion representative predictions stay within a given margin of error. The paper aims to provide the CTCR and the rest of the continuum robotic community with not only what types of FNN architecture and metrics do and don't work well with the dataset, but also prove that deep learning has a promising future in continuum robotics.**

***Index Terms*—Neural Network, Feedforward Neural Network (FNN), Loss Function, Activation Function, Hidden Layers, Concentric Tube Continuum Robot (CTCR)**

I. Introduction

A concentric tube continuum robot (CTCR) is one which is formed by a series of flexible, superelastic tubes nested inside one another [8]. They are different from tendon driven continuum robots [12] in that they are controlled using motors which control the lengths and bending of the tubes rather than actuating tendons with motors. CTCR's are used widely in many fields, such as the medical industry for minimally invasive surgery. There is great interest in modeling forward and inverse kinematics with CTCR's. One way in which this is approached is using Cosserat Rods [12] [13].This is an example of a physics-based approach to modeling a continuum robot. In recent years, there has been interest in the possibilities of deep learning being applied to CTCR's, in an effort to avoid physics-based modeling and focus solely on a neural network learning the weights and parameters of a given CTCR dataset [1] [2] [3] [4]. Thanks to the Continuum Robotics Laboratory in Toronto, Canada [1] [2] [3], the first publicly available CTCR dataset was released in 2022. From this dataset and [3], a benchmark for deep learning approaches to CTCR-based prediction methods was given. This sparked interest for this paper's research on how Feedforward Neural Networks (FNN), could be applied to forward kinematics. From this, the interest lies in end effector prediction based on base fram to end effector-1 frame position and orientation information, without the use of any physics-based modeling. The dataset [1] is relatively new in the CTCR field, so that is why this work presents an analysis of FNN's. This paper serves to provide as an overview on what does and doesn't work well in the context of deep learning with the Continuum Robotics Laboratory CTCR dataset [1]. Also, after reviewing [3], the results were accurate for what was being predicted. However, other than the number of activations in each hidden layer and training process, there was a lack of specifics on the architecture being used. The number of epochs trained was 1,000, which after analysis for this research, it is claimed that this level of training may be overfitting the model to the CTCR dataset specifically [1]. It is also because of this, that this research was aimed at looking into overfitting with this dataset and to make a more robust and generalizeable model.

A. Contribution

In this paper, there are four main contributions:

- FNN feature splitting, loss functions, activation functions, and batch sizing were analyzed to provide a more holistic view of the given CTCR dataset
- Developed FNN model to accurately predict end effector position and orientation using the CTCR dataset
- Discovered use of dataset without joint space and previous joint space configuration to predict end effector

Special thanks to Dr. Ian Walker and the Clemson ECE Department for supporting learning and research throughout the course of this project. Special thanks as well to the Continuum Robotics Laboratory in Toronto, Canada for making their CTCR dataset publicly available for use and improvement of CTCR learning-based research.

- Help develop knowledge as to how deep learning in the CTCR field can progress

B. Feedforward Neural Network Background

Feedforward Neural Networks (FNN) are categorized by the direction of data flow in one direction from the input layer to hidden layers to output layer. In this paper, the architecture used for all testing was fully-connected FNN, meaning each neuron of one layer has an output with associated weight connected as an input individually to each neuron in the following layer. Another name for this type of network is Multilayer Perceptron (MLP) [10]. In an FNN, the input layer is categorized as the layer in which raw input data is fed into the network, each neuron representing a feature of the data [10]. The next stage contains the hidden layers, which can be an arbitrary number of layers by design choice. In the hidden layers, the actual learning takes place. As mentioned previously, for a layer, all neurons are connected individually to each neuron in the next layer. Each connection has its own associated weight. During training, for each neuron, the weighted sum of is calculated and a bias is added.

$$z_j^l = \sum_{i=1}^{N}(w_{ij}^l \cdot a_i^{l-1}) + b_j^l \tag{1}$$

In the Equation (1), j corresponds to each neuron in layer l. The variable z is the weighted sum of each neuron in layer l. The a term is corresponds the activation output value of neuron i from the previous l layer. The b term is the bias of each neuron in layer l. The summation occurs from i to N number of neurons in the previous layer.

This weighted sum is then passed through an activation function, which introduces non-linearity to the model and helps it learn. This process continues through all hidden layers of the model until the output layer. Once the output layer is reached, the network has its predictions and compares these to the ground truth value. This comparison is done through the use of a loss function, giving a metric to measure how accurately the model made its predictions. Once the loss is computed, the network does what is known as back-propagation, computing the gradients of the loss with respect to the weights and biases. Once this is done, an optimization process (optimizer) is deployed to iteratively adjust the weights and biases with the goal of minimizing the loss (these adjustments help the model learn).

II. Dataset Overview

The dataset used in this paper's work is title, "A Dataset and Benchmark for Learning the Kinematics of Concentric Tube Continuum Robots", created and published by the Continuum Robotics Laboratory in Toronto, Canada [1] [3]. It can be found on GitHub HERE. The dataset is based on a three-tube CTCR. [3] [4]. For dataset collection, 6 DOF sensors were placed at the distal ends of each tube [3]. The Continuum Robotics Laboratory utilized an electromagnetic tracking system known as Aurora [3]. The data measured, defined by the Continuum Robotics Laboratory [1] contains the following:

- six absolute joint values
- six relative joint values
- pose of the base
- pose of the proximal sensor attached to the outermost tube
- pose of the sensor attached to the middle tube
- pose of the distal sensor attached to the most inner tube

As per [1], the dataset consists of 100,000 joint configurations, collected in eight sequences of 12,500 points. The size of the dataset is 56.5 MB in the format of a CSV file. In the CSV file, there are 40 columns, described below:

- Joint Configuration q_k
  - Column 1 - α_1
  - Column 2 - β_1
  - Column 3 - α_2
  - Column 4 - β_2
  - Column 5 - α_3
  - Column 6 - β_3
- Joint Configuration Difference Δ_{qk} = q_k - q_{k-1}; Δ_{α_i} rad; Δ_β mm;
  - Column 7 - Δ_{α_1}
  - Column 8 - Δ_{β_1}
  - Column 9 - Δ_{α_2}
  - Column 10 - Δ_{α_2}
  - Column 11 - Δ_{α_3}
  - Column 12 - Δ_{α_3}
- Columns 13-19 (repeat four times up to column 40 for i=0,1,2,3)
  - Position t = [x_i, y_i, z_i] mm
    - Column 13 - x_i
    - Column 14 - y_i
    - Column 15 - z_i
  - Orientation = $[n_i,\ \epsilon_{i,1},\ \epsilon_{i,2},\ \epsilon_{i,3}]^T unitless$
    - Column 16 - n_i
    - Column 17 - $\epsilon_i, 1$
    - Column 18 - $\epsilon_i, 2$
    - Column 19 - $\epsilon_i, 3$

For the purposes of this research, the focus was columns 13-40, position and orientation information. The data was split into training, testing, and validation data, as described below:

- Training: 80,000 points
- Validation: 10,000 points
- Test: 10,000 points

The way in which data was sampled is described in the section labeled "Feature Splitting".

III. Prediction Interest

A. Goal - Forward Kinematics

According to [11], Forward Kinematics in robotics refers to calculating tip position and orientation of the end effector with known position and orientation of base to end effector frame - 1. In [3], it is mentioned that there is interest in how machine or deep learning can be applied to CTCR's without any physics-based modeling. For this paper, this raises the question, "Can position and orientation of a CTCR end effector be accurately predicted by a deep learning model, solely relying on the dataset at hand and utilizing no physics-based modeling?". In [3], it is mentioned there is interest in four different types of prediction. These are x, x_{pose}, $x_{position}$, and $x_{three-points}$ [3]. For the purposes of this research, the one focused on was x_{pose}, defined as:

$$x_{\text{pose}} = [x3, y3, z3, \eta, \varepsilon_1, \varepsilon_2, \varepsilon_3]^T \tag{2}$$

Therefore, the predictions for this research were based on the end effector position and orientation variables. This corresponded to all network architectures having and output of seven for the seven predictions that needed to be made for x_{pose} in 2.

B. Error Calculations

In order to measure model performance, three different metrics were used, two of which defined by [3], translational and rotational error. In the following equations 3-5, a variable with a hat corresponds to a predicted value and a variable without one corresponds to a ground truth value. The Translational Error (Euclidean Distance), denoted $_t$, is defined as:

$$e_t = \sqrt{(x - \hat{x})^2 + (y - \hat{y})^2 + (z - \hat{z})^2} \tag{3}$$

The Rotational Error, measuring error of a predicted Quaternion representation, is defined as:

$$e_\theta = \min\Big\{ 2 \arccos\left(\eta\hat{\eta} + \varepsilon_1\hat{\varepsilon}_1 + \varepsilon_2\hat{\varepsilon}_2 + \varepsilon_3\hat{\varepsilon}_3\right), \\ 2 \arccos\left(\eta\hat{\eta} - \varepsilon_1\hat{\varepsilon}_1 - \varepsilon_2\hat{\varepsilon}_2 - \varepsilon_3\hat{\varepsilon}_3\right) \Big\} \tag{4}$$

The last measurement of accuracy was a margin of error tested to make sure the predicted value is within a given plus or minus tolerance, defined as:

$$|x - \hat{x}| \leq T \tag{5}$$

T is the tolerance value and x can correspond to any variable in the network prediction. In the continuum robotics field, 1-3% of the total continuum robot length is typically used as an acceptable margin of error. This was calculated based on Table I, given in [3], which details the lengths of the inner, middle, and outer tubes for each section of the CTCR being studied. The base frame section length is given as well. The calculations for an acceptable tolerance are seen below:

- i=0; 16mm for base frame
- i=1; 210+165+110mm = 485mm
- i=2; 210+165+110mm = 485mm
- i=3; 210+165+110mm = 485mm
- Total Length; 1471 mm approximately 57.91 inches
- 1471mm * 0.01 = 14.71 mm, 1% of total length

Therefore, 14.71 mm plus or minus is acceptable. Any accuracy more precise is deemed good and anything over is deemed worse. The tolerances chosen for measurement in the "Results and Discussion" section are 1.0, 2.5, 5.0, 14.71, and 50.0 to give a good idea of how the model is performing with respect to the target tolerance accuracy measure of 14.71 mm.

IV. Hardware

All models were created and trained using PyTorch on a Windows 10 system using the following hardware:

- AMD Ryzen 5 3600 (3.6GhZ) 6 core CPU
- Ripjaws 16GB DDR4-3600 RAM
- NVIDIA GeForce GTX 1660 TI Graphics Card

It is important to note that the capabilities of the hardware led to some design limitations in terms of the complexity of the models that were able to be trained (and for how long). These limitations guided design for this paper. The primary focus, was to learn more about the patterns in the dataset itself, rather than try to develop the most complex architecture. Also, to learn which CTCR movements generalize best with which FNN concepts.

V. Results And Discussion

In this section, the methodology, analysis, results, and overall approach to developing the FNN architecture will be discussed. The different areas of exploration will be discussed to give the reader a view on what types of design decisions were made and why they were made. All models for were trained with a learning rate of 0.01. The following topics will be explored:

- Feature Splitting
- Loss Functions
- Activation Functions
- Batch Sizing

A. Feature Splitting

Given that the dataset being used was collected in eight sequences of 12,500 datapoints for an overall 100,000 data points, the question of whether there would be an impact on accuracy if the data was sampled in a specific way was raised. When doing a training/testing-validation split, the data is sampled randomly. That is, if a dataset has 10,000 points, and you do an 80-20 split, the training data will have 8,000 points, the validation 1,000 points, and the testing 1,000 points. The points are taken randomly from the dataset. This is great if you have no organization, for example sections of data, that are similar, because randomly sampling will represent the entire dataset well by capturing all different types of possibilities for the input. However, for this paper, the dataset is organized by section. For example in this case, eight sequences of 12,500 data points. where each section can be categorized by a time of data collection where variables/conditions are similar and the movement of the CTCR is "within the same class" of movement. Therefore, the interest lied in whether accuracy would be affected if data was sampled traditionally with no account for organization of data in the CSV file, or if the data was proportionately sample from each 12,500 section. That is, when referring to "splitting" in this section, a proportionate amount of random samples were taken for training, validation, and testing data as follows. For each of the eight 12,500 point sections:

- Training - 10,000 points

- Validation - 1,250 points
- Testing - 1,250 points

These points were combined resulting in 80,000 training, 10,000 validation, and 10,000 testing. When referring to ”no splitting”, the split is simply 80,000 random for training, 10,000 for validation, and 10,000 testing that could be come from anywhere in the CSV file. The next area of exploration in this section that was also explored was the result on accuracy for using ”All” or ”Some” of the features for input, defined below:

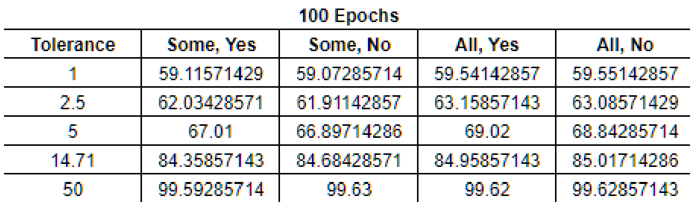
- All - 33 Features
  - Joint Configuration variables (all)
  - Joint Configuration Difference variables (all)
  - Position and Orientation (i=0 to end effector frame - 1) variables (all)
- Some - 21 Features
  - Position and Orientation (i=0 to end effector frame - 1) variables (all)

This allows for four possible combinations of using ”All” or ”Some” features, and ”Splitting” or ”No Splitting”. All possible combinations were trained for 100, 500, and 1000 epochs, and to verify accuracy, [EQUATION] was used for tolerances of 1.0, 2.5, 5.0, 14.71, and 50.0. The FNN architecture used for this training is as follows:

- 21 or 33 input features
- Six Hidden Layers (numbers are sizes): h1=64, h2=96, h3=50, h4=25, h5=18, h6=15
- Seven prediction outputs corresponding to x_{pose}
- Leaky ReLU Activation
- No Kaiming Normal Weight Initialization
- ADAM Optimizer
- No batching

As seen in Figure 4, it is clear that utilizing all 33 features provides the better/lowest loss. The training loss is close to the validation loss, so the model can be deemed not over or underfitting itself too much. Given interest in staying true to forward kinematics, it was chosen to move forward using only some (21) features (just the base to end effector - 1 frame position and orientation information). This analysis, did however, provide good insight into which amount of features would provide higher accuracy. In Figures 1, 2, and 3, when looking at the same category ”Some” or ”All” and just comparing whether splitting was used or not, it can be seen that there is an overall increase of about 1% accuracy. For example, Figure 2, shows this well, as most all accuracies are about 1-2% higher when compared to the no splitting counterpart. It is because of this, that it was chosen to stick with feature splitting throughout the rest of the analysis, and to only use 21 features as input as described previously. It was proven that proportionately sampling from the eight sequences of data collection in the dataset provided a more holistic view / better representation of the CTCR movement, allowing for higher accuracy.

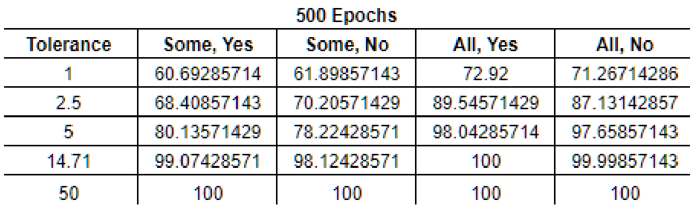
In 5, an example of the training vs. validation loss is plotted. This shows that doing feature splitting did not affect learning



100 Epochs

Tolerance	Some, Yes	Some, No	All, Yes	All, No
1	59.11571429	59.07285714	59.54142857	59.55142857
2.5	62.03428571	61.91142857	63.15857143	63.08571429
5	67.01	66.89714286	69.02	68.84285714
14.71	84.35857143	84.68428571	84.95857143	85.01714286
50	99.59285714	99.63	99.62	99.62857143

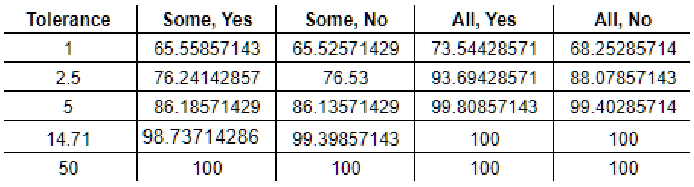
Fig. 1. Feature Splitting - 100 Epochs



500 Epochs

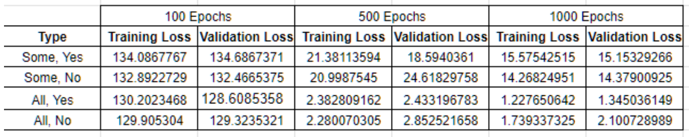
Tolerance	Some, Yes	Some, No	All, Yes	All, No
1	60.69285714	61.89857143	72.92	71.26714286
2.5	68.40857143	70.20571429	89.54571429	87.13142857
5	80.13571429	78.22428571	98.04285714	97.65857143
14.71	99.07428571	98.12428571	100	99.99857143
50	100	100	100	100

Fig. 2. Feature Splitting - 500 Epochs



Tolerance	Some, Yes	Some, No	All, Yes	All, No
1	65.55857143	65.52571429	73.54428571	68.25285714
2.5	76.24142857	76.53	93.69428571	88.07857143
5	86.18571429	86.13571429	99.80857143	99.40285714
14.71	98.73714286	99.39857143	100	100
50	100	100	100	100

Fig. 3. Feature Splitting - 1000 Epochs



	100 Epochs		500 Epochs		1000 Epochs	
Type	Training Loss	Validation Loss	Training Loss	Validation Loss	Training Loss	Validation Loss
Some, Yes	134.0867767	134.6867371	21.38113594	18.5940361	15.57542515	15.15329266
Some, No	132.8922729	132.4665375	20.9987545	24.61829758	14.26824951	14.37900925
All, Yes	130.2023468	128.6085358	2.382809162	2.433196783	1.227650642	1.345036149
All, No	129.905304	129.3235321	2.280070305	2.852521658	1.739337325	2.100728989

Fig. 4. Feature Splitting - Losses Comparison

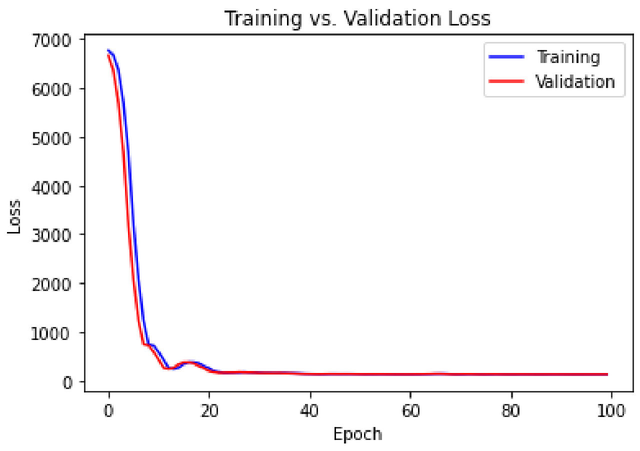


Fig. 5. Example Loss Graph - Some Features, Splitting, 100 Epochs

	100 Epochs			
Tolerance	MSE	Smooth L1	Huber Loss	MAE
1	12.88857143	42.53571429	44.37142857	17.11285714
2.5	30.53571429	61.18142857	61.91	57.18714286
5	53.85142857	65.23857143	66.79	64.51714286
14.71	87.68857143	79.06857143	82.64428571	78.68285714
50	99.80714286	99.59	99.64714286	99.61142857

Fig. 6. Loss Functions - 100 Epochs

	500 Epochs			
Tolerance	MSE	Smooth L1	Huber Loss	MAE
1	15.62714286	61.72571429	64.15285714	63.34
2.5	36.06571429	67.85857143	74.52	69.71428571
5	60.92	75.74857143	85.05714286	76.14857143
14.71	94.42428571	93.78714286	99.11857143	90.19285714
50	100	100	100	99.62571429

Fig. 7. Loss Functions - 500 Epochs

in a negative way, as the training and validation loss steadily decrease and are close to one another in value. This indicates the model is not overfitting. On the topic of overfitting, through this analysis, it started to become clear that training for more than a few hundred epochs was not needed, and could lead to overfitting if done.

B. Loss Function Analysis

Computing loss through the use of a loss function is a critical part of FNN's, or any neural network architecture. It allows the model to measure its performance during training and verify that performance with validation data. The FNN can then back propagate and adjust its weights and biases through the optimization process. Becuase loss functions allow the model to learn over each training epoch, choosing the one that works best for the dataset being used is critical. It is important to choose a loss function that captures the patterns and all variations in the dataset. This section focuses on the loss functions that were tested, and why they were chosen. The loss functions chosen to be tested were:

- Mean-Squared Error (MSE)
- Mean Absolute Error (MAE)
- Huber Loss
- Smooth L1

All loss functions are common functions well suited for regression prediction. For more information, they are defined in the PyTorch Documentation, located HERE. For the loss function analysis, all loss functions were used individually and independently in the FNN. They were all trained and tested for 100 and 500 epochs each. The architecture used for this is below:

- 21 input features
- Six Hidden Layers (numbers are sizes): h1=64, h2=96, h3=50, h4=25, h5=18, h6=15
- Seven prediction outputs corresponding to x_{pose}
- Leaky ReLU Activation
- Kaiming Normal Weight Initialization
- ADAM Optimizer
- No batching

Figures 6 and 7 showcase the accuracies for each tolerance value for each loss function over 100 and 500 epochs respectively. Based on these figures, the highest accuracies were seen with Smooth L1 and Huber Loss. These functions are very similar to one another, so there similar performance makes sense. This can also be seen in the Training vs. Validation

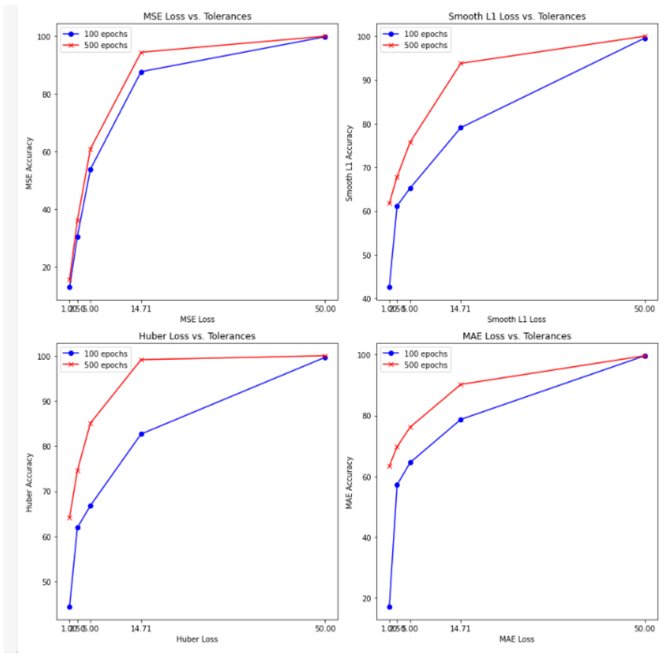


Fig. 8. Loss Functions Comparison

Loss Figure 9. It is important to note that these functions also performed similarly better compared to MAE and MSE on average for all tolerances. This shows that these functions are more precise at helping the model learn to predict x_{pose}. For example, what is meant by this is that for 100 epochs, Smooth L1 ranges from 42.54% to 99.59% as the tolerance goes from 1.0 to 50.0. Comparing this to MSE loss which ranges from 12.89% to 99.8% and MAE loss which ranges from 17.11% to 99.61%. The accuracies with more precise margin of error is much lower for MSE and MAE compared to Huber and Smooth L1. This is confirmed in Figure 8. Moving forward, it was elected to use Smooth L1, although Huber Loss could have been chosen as well providing similar performance.

	100 Epochs		500 Epochs	
Loss Function	Training Loss	Validation Loss	Training Loss	Validation Loss
MSE	115.3067551	116.4167175	41.22105408	47.85066986
MAE	6.786943436	7.873418808	4.461691856	4.470654964
Huber Loss	5.9140625	6.084542751	1.842700243	1.774027228
Smooth L1	6.82937336	7.237857819	3.21807766	3.199614525

Fig. 9. Loss Functions - Training vs. Validation Loss Comparison

	Batch Size		
Tolerance	32	64	128
1	60.16	58.70714286	61.79857143
2.5	65.23428571	61.19285714	68.23857143
5	72.26857143	65.40714286	76.64714286
14.71	89.06285714	84.91857143	91.63714286
50	100	99.95285714	100

Fig. 10. Batch Accuracies - 100 Epochs

C. Batch Size Analysis

In a neural network, it is possible to send in batches of data. For example, if the batch size was 32, for this research, the input would be 32 samples of 21 input features from the dataset training Dataloader. Changing the batch size can impact computational efficiency during training. It can also help with stabilization in the learning process to help create a more generalized prediction model. This section outlines how the following batch sizes were used during training to see which performed the best. The batch sizes used are below:

- 32
- 64
- 128

The model was trained for 100 epochs using each of the batch sizes. The architecture of this model is:

- 21 input features
- Six Hidden Layers (numbers are sizes): h1=batch_size, h2=96, h3=50, h4=25, h5=18, h6=15
- Seven prediction outputs corresponding to x_{pose}
- Leaky ReLU Activation function after each hidden layer
- Kaiming Normal Weight Initialization
- ADAM Optimizer
- MSE Loss
- Batch size varied
- Batch Normalization Layer after h1
- Dropout Layers of 0.2 rate after h3 and h5

This architecture is a bit different than previous ones described due to the batch normalization layer, input batch sizing, and the dropout layers. The batch sizing and batch normalization layer are used to stabilize and accelerate the training process. Dropout layers randomly drop a percentage of neurons at that point in the network. Given loss functions and feature splitting had been explored, at this point, the goal was to start making the model more robust, generalize able, and aim to avoid over or underfitting. Dropout layers help to introduce an element of randomness into the network and make the network's learning stronger overall.

Based on Figures 10 and 11, it can be seen that a batch size of 128 performed the best overall. This was chosen moving forward in the design process. It is important to note, that larger batch sizes could have been tested, but were chosen not to due to hardware limitations.

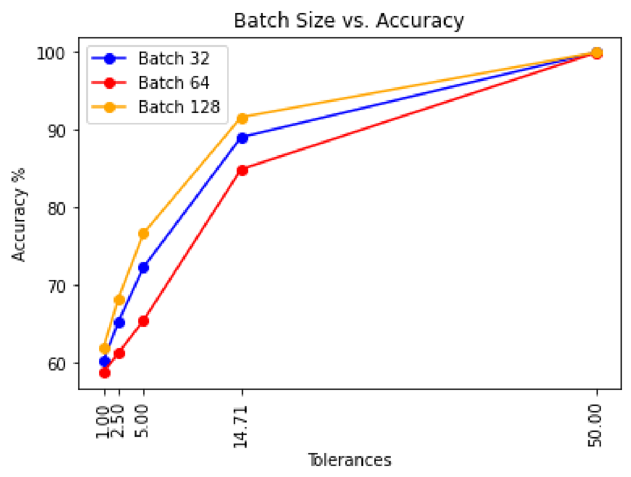


Fig. 11. Batch Accuracies Visualization

D. Activation Function Analysis

Activation functions allow a model to learn complex patterns/behavior of the dataset by introducing non-linearity to the learning process. Choice of activation function is critical in the design of a FNN. The activation function also enables back-propagation to work effectively. For the purposes of this research, the goal was to study the effect of the activation function on the learning process with the CTCR dataset. Therefore, a given activation function was always used after each hidden layer and multiple different activation functions were not used in the same model. This is possible, but was not analyzed for this research. Four activation functions tested are below:

- ReLU
- Leaky ReLU
- Parametric ReLU (PReLU)
- TanH

All activation functions are commonly used in neural networks. For more information, they are defined in the PyTorch Documentation, located HERE. The activation functions above can all be applied to regression based prediction tasks. ReLU is commonly used and allows positive values to pass through while squashing negative values. PReLU, slightly modified ReLU, has a small negative slope that allows the model to learn slope during training. This is good for tasks with varying degrees of linearity. Leaky ReLU is also a modification of the ReLU function in which a small degree of negative values are allowed to pass through the network (due to small negative slope). It is useful if negative gradients affecting the network is a concern. Lastly, TanH keeps values in the range [-1,1], and is useful for problems that center around a zero gradient. For the activation function analysis, they were all trained and tested for 100 epochs each. The architecture used for this is below:

- 21 input features

	Activation Function			
Tolerance	ReLU	Leaky ReLU	PReLU	TanH
1	60.08714286	60.16	61.14571429	59.02
2.5	64.43	65.23428571	67.21285714	61.62428571
5	70.95	72.26857143	76.54857143	65.96428571
14.71	88.95714286	89.06285714	94.55857143	81.48428571
50	100	100	100	99.61285714

Fig. 12. Activation Accuracies - 100 Epochs

Activation Function	Epochs	Training Loss	Validation Loss
ReLU	100	18.53024435	63.95670203
LeakyReLU	100	83.27457832	71.66801003
PReLU	100	12.75964283	37.26969125
Tanh	100	147.2628821	147.8721449

Fig. 13. Activation Losses

- Six Hidden Layers (numbers are sizes): h1=batch_size, h2=96, h3=50, h4=25, h5=18, h6=15
- Seven prediction outputs corresponding to x_{pose}
- Varying Activation function after each hidden layer
- Kaiming Normal Weight Initialization
- ADAM Optimizer
- MSE Loss
- Batch size of 32
- Batch Normalization Layer after h1
- Dropout Layers of 0.2 rate after h3 and h5

The results of the model for each activation function can be seen in Figures 12 and 13

The model was only trained for 100 epochs for sake of simplicity and analysis. Therefore, the losses in 13 are not meant to be viewed as meaning the overall model is trained well. Rather, they help to identify which activation function helped the model learn best. It can be seen that the lowest training / validation loss was with the PReLU activation function. In 12, it is seen that the PReLU activation function

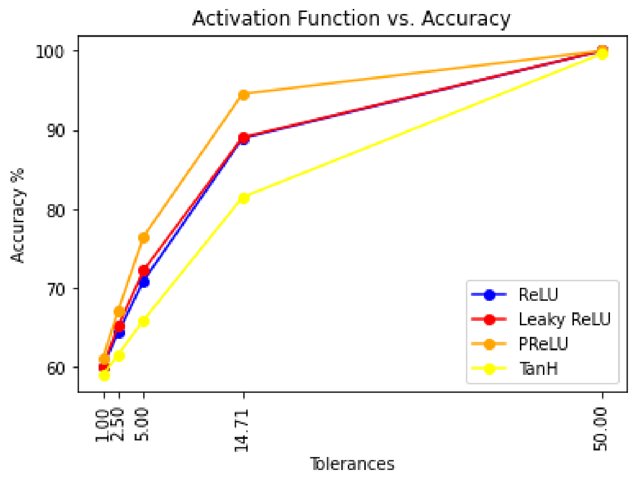


Fig. 14. Activation Functions Accuracy Visualization

150 Epochs			
Design	Loss Function	Training Loss	Validation Loss
1	MSE	8.055999936	34.88022638
1	Smooth L1	1.241999273	2.388759631
2	MSE	11.83105258	73.06991992
2	Smooth L1	1.583335121	3.070143401

Fig. 15. Final Loss - 150 Epochs

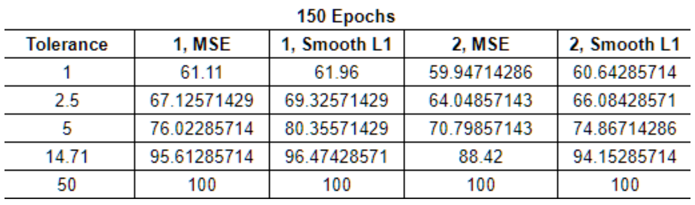
use provided the highest accuracy on average throughout all tolerances tested. This is visualized in 14 The PReLU activation function was chosen moving forward in the model design process.

VI. Final Design

After analyzing the effects of the previous metrics, the goal was to design a final model combining what performed well to develop a stronger model. Two different designs were tested. Design 1 was trained for and Design 2 were trained for 150 epochs using MSE loss. Design 1 and 2 were then trained for 150 epochs using Smooth L1 loss. The MSE loss function was changed from a Smooth L1 to a lower performing MSE for confirmation of performance. The architecture similarities for the designs is seen below:

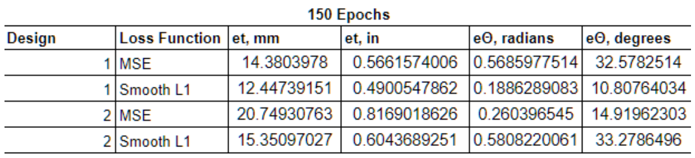
- Design 1:
  - 21 input features
  - Seven Hidden Layers (numbers are sizes): h1=batch_size, h2=96, h3=50, h4=30, h5=25, h6=18, h7=15
  - Seven prediction outputs corresponding to x_{pose}
  - PReLU function after each hidden layer, 0.25 weight
  - Kaiming Normal Weight Initialization
  - ADAM Optimizer
  - MSE and Smooth L1 Loss tested
  - Batch size of 128
  - Batch Normalization Layer after h1
  - Dropout Layers of 0.2 rate after h3 and h5
- Design 2:
  - 21 input features
  - Thirteen Hidden Layers (numbers are sizes): h1=batch_size, h2=96, h3=50, h4=30, h5=25, h6=18, h7=15, h8=96, h9=50, h10=30, h11=25, h12=18, h13=15
  - Seven prediction outputs corresponding to x_{pose}
  - PReLU function after each hidden layer, 0.25 weight
  - Kaiming Normal Weight Initialization
  - ADAM Optimizer
  - MSE and Smooth L1 Loss tested
  - Batch size of 128
  - Batch Normalization Layer after h1
  - Dropout Layers of 0.2 rate after h3, h5, h7, h9, h11

The following will detail and analyze the results of these tests on Designs 1 and 2.



		150 Epochs		
Tolerance	1, MSE	1, Smooth L1	2, MSE	2, Smooth L1
1	61.11	61.96	59.94714286	60.64285714
2.5	67.12571429	69.32571429	64.04857143	66.08428571
5	76.02285714	80.35571429	70.79857143	74.86714286
14.71	95.61285714	96.47428571	88.42	94.15285714
50	100	100	100	100

Fig. 16. Final Accuracy



		150 Epochs			
Design	Loss Function	et, mm	et, in	eΘ, radians	eΘ, degrees
1	MSE	14.3803978	0.5661574006	0.5685977514	32.5782514
1	Smooth L1	12.44739151	0.4900547862	0.1886289083	10.80764034
2	MSE	20.74930763	0.8169018626	0.260396545	14.91962303
2	Smooth L1	15.35097027	0.6043689251	0.5808220061	33.2786496

Fig. 17. Final Translational and Rotational Error

In Figure 15, the training and validation loss is seen. It is clear the performance is better for Smooth L1 loss. The performance is also much better for Design 1 using Smooth L1 loss. The training and validation loss are reasonably close to one another, which helps to confirm that the model is not overfit and was learning the patterns in the dataset properly.

In Figure 16, the margin of error accuracy is seen. It is clear that the best performing model was Design 1 with Smooth L1 loss. This is further confirmed in Figure 17, where e_t in mm and inches, as well as e_θ in radians and degrees, is seen. Design 1 with Smooth L1 Loss was able to achieve a sub-12.5 mm accuracy for position prediction, which is below 1% of the CTCR's total length. Design 1 with Smooth L1 loss was also able to achieve a sub-11 degree accuracy for orientation prediction. A breakdown of the number of correct and incorrect predictions, by variable, for the 10,000 point test set (70,000 individual variable predictions), with a tolerance of 14.71 mm is seen below:

- Total: 70000; Correct: 67532; Incorrect: 2468; Accuracy: 96.47428571428571%
  - Correct Counts
    - x_3: 9997- 14.281428571428572%
    - y_3: 8801 - 12.572857142857144%
    - z_3: 8734 - 12.477142857142857%
    - n_3: 10000 - 14.285714285714285%
    - $\varepsilon_{3,1}$: 10000 - 14.285714285714285%
    - $\varepsilon_{2,3}$: 10000 - 14.285714285714285%
    - $\varepsilon_{3,3}$: 10000 - 14.285714285714285%
  - Incorrect Counts
    - x_3: 3 - 0.004285714285714286%
    - y_3: 1199 - 1.7128571428571426%
    - z_3: 1266 - 1.8085714285714285%
    - n_3: 0 - 0.0%
    - $\varepsilon_{3,1}$: 0 - 0.0%
    - $\varepsilon_{2,3}$: 0 - 0.0%
    - $\varepsilon_{3,3}$: 0 - 0.0%

The percentages in the correct and incorrect counts above correspond to the percentage of total points in the 70,000

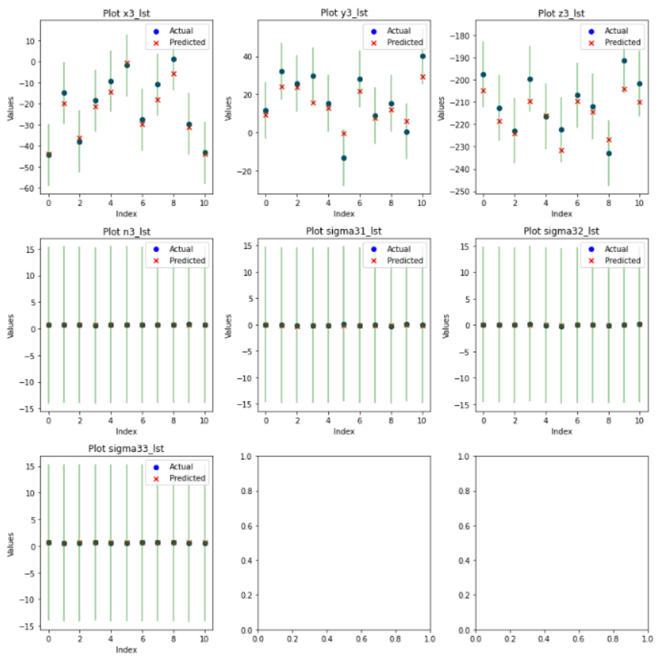


Fig. 18. Design 1 - Smooth L1 Loss - Random Sample Predictions - 14.71 Tolerance

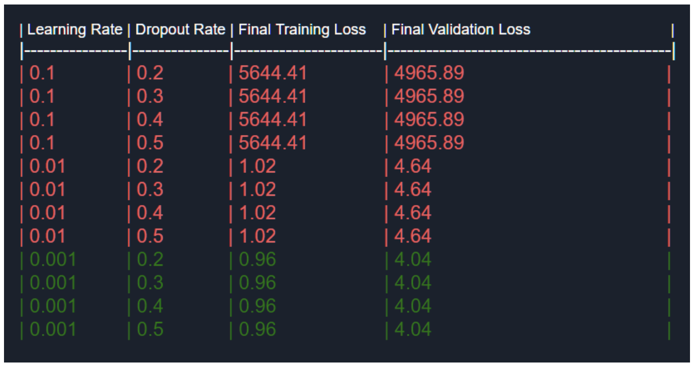
dataset. Therefore, a percentage of 14.286 for a variable is considered 100% of that specific variable being correct. It is seen that orientation predictions were 100% accurate. On the other hand, the model had slight difficulty in predicting y and z positions. Even through this difficulty, it is shown based on 16, that the predictions that were incorrect must lie somewhere between the tolerance of 14.72 and 50.0.

A random sample of testing points (10 points) were taken and graphed for each individual prediction variable in x_{pose}. These are visualized in Figure 18 , where the green bar represents a 14.71 tolerance above and below the ground truth point seen in the blue circle. The predicted point is represented by the red 'x'. In Figure 18, it is shown that the model is extremely accurate when prediction orientation variables. It is also accurate for position variables, but less precise. Nonetheless, the 18 showcases that the model is capable of predicting position and orientation within acceptable accuracy.

A. Hyperparameter Tuning

After it was determined that Design 1 with Smooth L1 loss was the best performing model, the next step was to tune the hyperparameters. This is a process in which you are able to iteratively test all combinations of certain parameters to see which model performs the best. The design was tuned for learning rate and dropout rate. The learning rates that were tested were [0.1, 0.01, and 0.001]. The dropout rates that were tested were [0.2, 0.3, 0.4, and 0.5]. The results of this can be seen in 19.

In 19, the text highlighted in green represents the best performance for a learning rate of 0.001. The dropout rate did not seem to affect the performance, so the final confirmed



```
| Learning Rate | Dropout Rate | Final Training Loss | Final Validation Loss |
|---------------|--------------|---------------------|-----------------------|
| 0.1           | 0.2          | 5644.41             | 4965.89               |
| 0.1           | 0.3          | 5644.41             | 4965.89               |
| 0.1           | 0.4          | 5644.41             | 4965.89               |
| 0.1           | 0.5          | 5644.41             | 4965.89               |
| 0.01          | 0.2          | 1.02                | 4.64                  |
| 0.01          | 0.3          | 1.02                | 4.64                  |
| 0.01          | 0.4          | 1.02                | 4.64                  |
| 0.01          | 0.5          | 1.02                | 4.64                  |
| 0.001         | 0.2          | 0.96                | 4.04                  |
| 0.001         | 0.3          | 0.96                | 4.04                  |
| 0.001         | 0.4          | 0.96                | 4.04                  |
| 0.001         | 0.5          | 0.96                | 4.04                  |
```

Fig. 19. Design 1 - Smooth L1 Loss - Hyperparameter Tuning

design was Design 1 with Smooth L1 Loss, a learning rate of 0.001, and a dropout rate of 0.2.

VII. Conclusion

To conclude, multiple Feedforward Neural Networks (FNNs) were trained and tested on the Continuum Robotics Laboratory CTCR public dataset. Through this process, a final design was able to be created, capable of predicting end effector position and orientation. The final translational accuracy was < 12.5mm or < 0.5in and the final rotational accuracy was < 0.19 radians < 11 degrees on average. This paper hopes to provide insight into what does and doesn't work well in terms of FNN architecture with CTCR datasets, with a main aim of progressing the field of CTCR.

References

[1] ContinuumRoboticsLab, "ContinuumRoboticsLab/CRL-dataset-CTCR-pose," GitHub, https://github.com/ContinuumRoboticsLab/CRL-Dataset-CTCR-Pose (accessed Dec. 5, 2023).

[2] A Dataset and Benchmark for Learning the Kinematics of Concentric Tube Continuum Robots. YouTube, 2022.

[3] R. M. Grassmann, R. Z. Chen, N. Liang and J. Burgner-Kahrs, "A Dataset and Benchmark for Learning the Kinematics of Concentric Tube Continuum Robots," 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan, 2022, pp. 9550-9557, doi: 10.1109/IROS47612.2022.9981719.

[4] R. Grassmann, V. Modes and J. Burgner-Kahrs, "Learning the Forward and Inverse Kinematics of a 6-DOF Concentric Tube Continuum Robot in SE(3)," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 2018, pp. 5125-5132, doi: 10.1109/IROS.2018.8594451.

[5] A. Chawla, C. Frazelle and I. Walker, "A Comparison of Constant Curvature Forward Kinematics for Multisection Continuum Manipulators," 2018 Second IEEE International Conference on Robotic Computing (IRC), Laguna Hills, CA, USA, 2018, pp. 217-223, doi: 10.1109/IRC.2018.00046.

[6] P. E. Dupont, J. Lock, B. Itkowitz, and E. Butler. Design and control of concentric-tube robots. IEEE Transactions on Robotics, 26(2):209–225, 2010.

[7] D. C. Rucker, B. A. Jones, and R. J. Webster III. A geometrically exact model for externally loaded concentric-tube continuum robots. IEEE Transactions on Robotics, 26(5):769–780, 2010.

[8] Bergeles C, Gosline AH, Vasilyev NV, Codd PJ, Del Nido PJ, Dupont PE. Concentric Tube Robot Design and Optimization Based on Task and Anatomical Constraints. IEEE Trans Robot. 2015 Feb 3;31(1):67-84. doi: 10.1109/TRO.2014.2378431. PMID: 26380575; PMCID: PMC4569019.

[9] S. Lek, Y.S. Park, Multilayer Perceptron, Editor(s): Sven Erik Jørgensen, Brian D. Fath, Encyclopedia of Ecology, Academic Press, 2008,Pages 2455-2462, ISBN 9780080454054, https:/doi.org/10.1016/B978-008045405-4.0012-2. (https://www.sciencedirect.com/science/article/pii/B9780080454054001622)

[10] Alex, "Feedforward Neural Networks and multilayer perceptrons," Boostedml, https://boostedml.com/2020/04/feedforward-neural-networks-and-multilayer-perceptrons.html (accessed Dec. 8, 2023).

[11] J. Burgner-Kahrs,"Introduction to Modeling," opencontinuum-robotics.com, https://www.opencontinuumrobotics.com/101/2022/11/25/intro-modeling.html: :text=The%20forward%20kinematics%20of%20a,tip%20from%20its%2 (accessed Dec. 8, 2023).

[12] F. Janabi-Sharifi, A. Jalali and I. D. Walker, "Cosserat Rod-Based Dynamic Modeling of Tendon-Driven Continuum Robots: A Tutorial," in IEEE Access, vol. 9, pp. 68703-68719, 2021, doi: 10.1109/ACCESS.2021.3077186.

[13] D. C. Rucker, B. A. Jones, and R. J. Webster III, "A geometrically exact model for externally loaded concentric-tube continuum robots," IEEE Transactions on Robotics, vol. 26, no. 5, pp. 769–780, 2010.