Learning Collision Detection for a Six Degrees of Freedom Robot Manipulator with a Neural Network

Alexandre Coulombe
Electrical and Computer Engineering Department
McGill University
Montreal, Canada
alexandre.coulombe@mail.mcgill.ca

***Abstract*—Industrial robots currently on the market are reactive, that is, they stop when a collision occurs. However, robots could possibly detect and prevent these collisions in real time. The goal of this project is therefore to design and implement a software solution capable of detecting that the robot to enter in a collision. A neural network classifier is used to learn the model of the robot using data from a point cloud collision detection algorithm and classifies, using the joint angles of the robot, whether there is a collision imminent or not. The results show that the neural network classifier achieves high accuracy, performs quickly and is almost as precise as the point cloud algorithm used to train it. This makes it reliable to be used in environments where the robot is running in real time.**

***Index Terms*—neural networks, collision detection, robot manipulator, point cloud**

I. Introduction

The lack of real time collision detection can lead to many undesired situations. In some cases, collisions can wear the product and objects in the environment. In worse cases, the collision can leave permanent damage or even injure a person working with the robot.

The goal of the project is to design and implement a collision detection system that is able to detect positions that will lead to a collision and stop the collision from occuring, all while running in real time. The scope of the project deals with self-collisions between the robot with itself. The project deliverable is a neural network that will detect positions leading to self-collisions along the robot's trajectory.

A. Related Work

Collision detection systems have been employed to make collaborative robots for safe human and robot interactions. Many approaches use geometric volumes to model the robot and detect collisions between object [2]. Models such as Bounding Spheres, Axis-Aligned Bounding Boxes (AABB), Oriented Bounding Boxes (OBB) and Bounding Cylinders are used as an abstraction of the robot's true model and provide easier computations for the detection of collisions. However, even though the computations are simple, the techniques don't scale well when you have to compare a large number of these simple geometric volumes. These techniques have been applied to robot manipulators and have performed well. These approaches become too slow for large environments with numerous objects that need to be tested for collisions.

Other approaches taken to make collision detection and collision avoidance systems use neural networks. These neural networks were trained to detect collisions faster than more analytical methods. One solution uses a neural network to detect collisions reactively by using the joint angle signals and joint torque signals to classify collision after a contact has occurred [3]. This approach classifies collisions well after being trained with samples of collisions with input-output pairs of data containing the joint angles, joint torque and the external torque as the inputs and the target classification as an output. Another neural network was trained to recognize patterns in the robot position in order to determine if positions lead to self-collision [4]. This approach manages to reduce the computation overhead by learning the positions that are in collision and those that are not. This makes it quick to determine if a self-collision occurs. This neural network is partnered with another algorithm to deal with the set of positions that the neural network cannot classify, due to the neural network making false positive and false negative classifications on the subset.

B. Neural Networks

Artificial neural networks are systems that are inspired by the neural structure of brains [1]. They are used for learning a classifier or regressor. In our case, we use them as a classifier. Artificial neural networks use a structure composed of neurons, or nodes, that form numerous layers. They learn patterns and relationships from examples and use its training to predict the output of similar data. The structure of a neural network is an input layer where feature values are inserted in the network, hidden layers, and an output layer which gives the output values of the neural network's prediction, as seen in Figure 1.

Neural networks use forward propagation, which is passing the input values, $\vec{a}_{(0)}$ through the layers, to make a prediction at the output layers. In forward propagation, the input of a neuron in layer l, $\vec{z}_{(l)}$, is the sum of the outputs of the neurons of the previous layers, $\vec{a}_{(l-1)}$, each multiplied by a set of weights, $\mathbf{W}_{(l)}$, and added to an activation bias, $\vec{b}_{(l)}$. Then, the input of the neuron goes through an activation function, $g(.)$, which adds a non-linear aspect to the neural networks. This allows neural networks to learn non-linear relationships. For all the neurons of a layer, the computations that each layer performs are as follows:

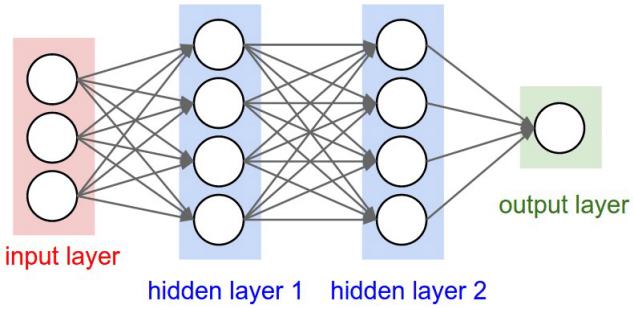

Fig. 1. Neural network structure with two hidden layers

$$\vec{z}_{(l)} = \mathbf{W}_{(l)}\vec{a}_{(l-1)} + \vec{b}_{(l)} \tag{1}$$

$$\vec{a}_{(l)} = g(\vec{z}_{(l)}) \tag{2}$$

The error on the output of the neural network is computed through a loss function, $\mathbf{J}(\mathbf{W}, \vec{b})$. The loss of the neural network is related to the accuracy of its prediction. In order to reduce the loss, the neural network must be trained with examples. The training updates the weights and biases using gradient descent techniques, with steps $\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \mathbf{W}}$ and $\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \vec{b}}$, to reach local minimums of the loss function. In order to find the derivatives, the chain rule is used to find $\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \vec{a}}\frac{\partial \vec{a}}{\partial \vec{z}}\frac{\partial \vec{z}}{\partial \mathbf{W}}$ for the weights and $\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \vec{a}}\frac{\partial \vec{a}}{\partial \vec{z}}\frac{\partial \vec{z}}{\partial \vec{b}}$ for the biases, where $\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \vec{a}}$ is the error of the next layer, $\vec{\delta}_{(l+1)}$, $\frac{\partial \vec{a}}{\partial \vec{z}}$ is the derivative of the activation function, $g'(\vec{z}_{(l)})$, $\frac{\partial \vec{z}}{\partial \mathbf{W}}$ is the output of the layer, $\vec{a}_{(l)}$, and $\frac{\partial \vec{z}}{\partial \mathbf{b}}$ is just $\vec{1}$.

To train a neural networks, back propagation is used to propagate the error of the output through the structure. The loss is propagated back to the each of the neurons of a layer by using the error of the next layer, $\vec{\delta}_{(l+1)}$, multiplied by the weights connected the neurons, $\mathbf{W}_{(l)}$ as follows:

$$\vec{\delta}_{(l)} = \mathbf{W}_{(l)}^T\vec{\delta}_{(l+1)} .* g'(\vec{z}_{(l)}) \tag{3}$$

$$\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \mathbf{W}_{(l)}} = \vec{\delta}_{(l)}\vec{a}_{(l-1)} \tag{4}$$

$$\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \vec{b}_{(l)}} = \vec{\delta}_{(l)} \tag{5}$$

$$\mathbf{W}_{(l)} = \mathbf{W}_{(l)} - \alpha\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \mathbf{W}_{(l)}} \tag{6}$$

$$\vec{b}_{(l)} = \vec{b}_{(l)} - \alpha\frac{\partial \mathbf{J}(\mathbf{W},\vec{b})}{\partial \vec{b}_{(l)}} \tag{7}$$

where, in (3), the $.*$ is the element-wise multiplication operator.

The weights and biases are updated at the end using (6) and (7), which are the gradient descent equations, and a learning factor, $0 < \alpha < 1$, which controls the size of the steps on the descent. One that is too large will make the neural networks diverge from the local minimum, while one that is too small will required more time to learn.

II. Method

In my thesis, an approach to collision detection explored was using point clouds of components. The entire robot and components in its environment are modeled as point clouds as in Figure 2. For any pairs of points $\mathbf{p}_i$ and $\mathbf{p}_j$ on two different components, the distance d_{ij} can be computed using the Euclidean distance of the points:

$$d_{ij} = \sqrt{\sum(\mathbf{p}_i - \mathbf{p}_j)^2} \tag{8}$$

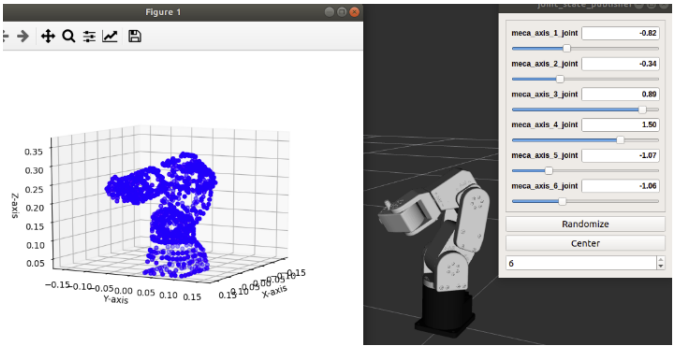

Fig. 2. Point Cloud Model of a Robot Manipulator

If the Euclidean distance of the two points is smaller than a certain threshold, ϵ, which is given by the precision desired, we can signal the robot that a collision can occur. However, the larger the point cloud becomes, the more computation needs to be performed. The algorithm complexity is $\mathbf{O}(n^2)$ for n points, which means that every point needs to be compared to every other point in the point cloud to determine if the distance between two points is under the threshold value. By training a neural network to learn the point cloud algorithm, we can achieve a solution as precise as it, while having a more reliable computation time $\mathbf{O}(1)$.

A. Neural Network Implementation

Neural networks have some very useful properties, such as being capable of learning non-linear functions, are tolerant to noise and can be used for regression and classification. For this project, a neural network was trained as a classifier with the two outputs: no collision detected, and collision detected. The output values can be interpreted as the confidence it has for a certain answer. If it has a higher confidence for a collision detected, it signals that the robot is in a potential collision. The features used by the neural network are the joint angles of the robot. The joint angle values are propagated through the network to generate the output prediction, which is used to detect a collision.

The first step in integrating the neural network to the problem was to implement the forward and back propagation algorithms, described by (1)-(7). This was done from scratch in Python 3.7 as a way to better understand the theory behind neural networks. With the two algorithms, a neural network generator was made where the user would need to input the hyperparameters of the neural network, such as the number of

nodes in each layer, the number of layers, and the learning rate. The loss function used is the mean-square error (MSE). The activation function options are the sigmoid function and rectified Linear Unit, but only the sigmoid function was used in the project. The weights and biases are randomly initialized with the correct dimensions. After training, the neural network parameters can be saved to a file that can be imported by the neural network Python module to regenerate the network.

B. Training

Training the network required a data set that was split into three subsets: the training set, the validation set, and the test set. The data set used was a collection of samples taken from the robot's configuration space. These samples are composed of input-output pairs where the input is the joint angles of the robot and the target output is whether there is an imminent collision. The target outputs were produced by the point cloud collision detection algorithm. The training set uses samples from the data set. Half the samples are positions in collision and the other half is positions that are not in collision. This is important because if the training set is not balanced, the classifier will grow a bias towards the class with the most samples in the training set. This would make it not generalizable and make it not perform well. The training set comprises the only samples the neural network will use to adjust its parameters. The validation set is used to verify how generalizable the neural network is as it is training. It can also be used to perform an early stop condition for the training. The validation set loss will decrease with the training set loss as the neural network is learning. At some point, the validation set loss will start increasing. At this point, the neural network is beginning to overfit the training set, since this is a way of reducing the training loss. So we can stop training at this point and keep the neural network generalizable. At the end of training, we use the test set to see how well the neural network performs to samples it has never encountered, and demonstrates how well it generalizes.

C. Structures

Using the neural network generator, multiple structures have been built and went through training using the data set sampled for the project. The first structure used comprised an input layer with the same number of nodes as there are joints on the robot, three hidden layers, and an output layer with two nodes. The hidden layers had four nodes each. The second structure used three hidden layers of six nodes, six nodes and four nodes respectively. This was to determine how the number of nodes in layers impact the accuracy of the classifier. A third structure was made and used two hidden layers of eight nodes each. This was to see the accuracy difference of having a shallower neural network by moving the nodes on the last hidden layer to the previous layers. A fourth structure was also generated using two hidden layers of ten nodes each to see the difference in accuracy due to having more nodes on a shallower structure. A fifth neural network with the same structure as the fourth went through training with a smaller training set and validation set to view the impact of the training set on the accuracy. Results and performance for each of these models are found in Section III.

The training utilized epochs, which is going over the training set multiple times, and stochastic gradient descent. The weights and bias parameters would be updated after every sample. The training used the early stop training condition from the increase in the loss function of the validation set. The first four neural networks were trained with the same training set and validation set, 17000 and 3000 samples respectively, and around 4 million samples for the test set. The fifth used 1700 and 300 samples for the training set and validation set respectively, but used the same test set as the rest. We use such a small number of samples compared to the data set to examine generalization and difference in training. The difference in training is apparent in the loss of the neural networks during training as seen in Figures 3 and 4.

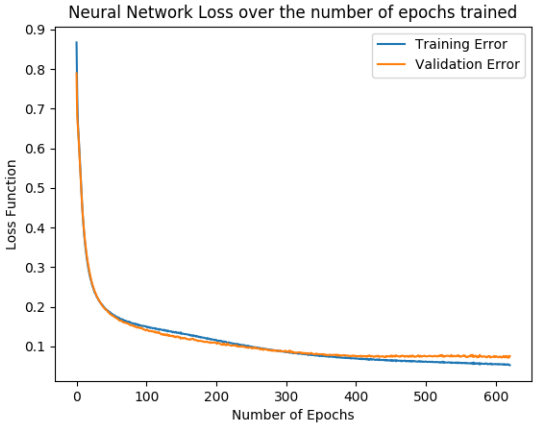

Fig. 3. Loss of the neural network over epochs using 1700 and 300 samples for the training set and validation set respectively

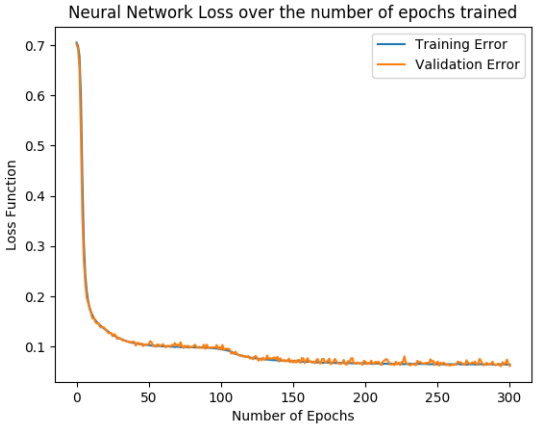

Fig. 4. Loss of the neural network over epochs using 17000 and 3000 samples for the training set and validation set respectively

The neural network that had a smaller training set had to go through more epochs to converge to a good solution, it

is about double between Figures 3 and 4. Also to note, the validation set loss seems to diverge from the training set loss on the smaller sets, while for bigger sets they remain similar. This is possibly due to the neural networks with more samples in their training set and validation set are able to learn from more diverse samples leading to better generalization.

III. Results

For the experimental results, the Euclidean distance point cloud collision detection system utilizing (8) is taken as a baseline for the comparison of all the approaches. The point cloud approach is has a high precision and serves as a good baseline for the accuracy criterion. It is assumed that it identifies imminent collisions with 100% accuracy. The accuracy is the percentage of agreement the approaches have with each other and shows how well the approach identifies imminent collisions correctly. We also use the percentage of false positives and false negatives as criteria. False positives are interpreted as saying that a collision is imminent when there is none. False negatives are interpreted as having a imminent collision and declaring that there is none. Even though we want the highest accuracy possible, if the approach does not identify the collision state at a position, it is preferable to have a false positive over a false negative to avoid jeopardizing safety. The results are summarized in Table I.

Model	Accuracy	FP	FN	FPF	Time (μs)
Point Cloud	1.0	-	-	-	4622.81
NN [5,4,4,4,2]	0.9528	0.0468	0.0004	0.9915	9.795
NN [5,6,6,4,2]	0.9740	0.0257	0.0003	0.9984	10.134
NN [5,8,8,2]	0.9794	0.0200	0.0005	0.9756	10.134
NN [5,10,10,2]	0.9811	0.0185	0.0004	0.9788	10.473
NN [5,10,10,2] (less samples)	0.9674	0.0323	0.0002	0.9938	10.473

TABLE I
PERFORMANCE OF EACH MODEL COMPARED TO THE BASELINE BASED ON THE ACCURACY, THE RATIO OF FALSE POSITIVES (FP), THE RATIO OF FALSE NEGATIVES (FN), AND THE RATIO OF FALSE POSITIVE WHEN FALSE (FPF)

The first thing to notice is the significant increase in speed. The point cloud approach is on average 4622.81 μs, with the maximum time being 7778 μs when no collision is imminent and the minimum time being 62 μs when it finds an imminent collision immediately. All the neural networks achieve around 10 μs, which is over 450 times faster than the point cloud approach's average time.

We can see that the NN[5,10,10,2] structure achieved the highest accuracy with 98.11% in agreement with the point cloud algorithm used to train it. It has also made false positives 97.88% of the time it is wrong, making it conserve safety when it is wrong. Something to notice is that the higher the number of nodes in the neural network, the better its performance. The first model contains 12 nodes through all its hidden layers, while the second and third models both have 16 nodes and the fourth has a total of 20 nodes. It is also worth noting that all the networks have a false positive percentage when the neural network produces the wrong output, all over 97%. This makes it more cautious than the point cloud collision detection and make it not lose too much safety when operating.

When it comes to the depths of the networks, the depth does not seem to be beneficial for this application. The shallower networks perform better than the deeper networks. This may be due to the lack of interconnection between nodes. There are fewer nodes used to produce the input of a node in the next layer, making it for the network to propagate hidden features found in the neural network.

Another observation can be done for the size of the training and validation sets. The neural network with the same structure as the highest-performing structure for the larger training and validation sets does worse when the sets are smaller. This can be because, when the training set and validation set are larger, the neural network is able to learn from a data set that is more representative of the real configuration space of the robot. This would allow it to learn more about the configuration space and be able to be more accurate when generalizing to the real configuration space.

IV. Conclusion

In conclusion, the proposed NN architecture(s) is able to learn the point cloud collision detection algorithm with a very high accuracy, making a computationally faster solution without compromising the precision and safety of the solution. When it produces the wrong output, it is generally a false positive, which makes safety reliable. When training a neural network, it is noticed that there is an increase in accuracy when doing more epochs on the training set. Also, when increasing the sample size for the training set and validation set, there is an increase in accuracy. When it comes to the structure of a neural network, higher accuracy is achieved when using more nodes in the hidden layers. Improvements that can be done is optimizing the neural network structure and parameters in order to achieve the highest accuracy possible. Some of the limitations of this work is the entire analysis being done in simulations without physics engines. Using a physics engine, we would be able to have a cross-validation detection with the physics engine collision detection systems and have more accurate samples than the samples produced by the point cloud algorithm that are assumed to be correct. Future work includes discovering if neural networks are able to learn other more accurate collision detection algorithms.

References

[1] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," Prentice Hall, 2002.

[2] Y. Shen, Q. Jia, G. Chen, Y. Wang and H. Sun, "Study of rapid collision detection algorithm for manipulator," 2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA), Auckland, 2015, pp. 934-938.

[3] S. Abdel-Nasser and A. Nikos, "Human-robot collision detection based on neural networks," International Journal of Mechanical Engineering and Robotics Research, vol. 7, pp 150-157, 2018.

[4] J. Son, H. Kwak and G. Park, "Back propagation neural network based real-time self-collision detection method for humanoid robot," 2011 11th International Conference on Control, Automation and Systems, Gyeonggi-do, 2011, pp. 1505-1508.