RFID and Machine Learning: Elevating Asset Management to New Heights

Brianna Hinds[1], Mosa Alsalih[1], Edwin Nwokeabia[1], Alicia Martinez[1], Jervany Moncherry[1], Divine Odeh[1], Christian Charley[1], Elizabeth Hinton[1], William Aguirre[1]

[1]Department of Engineering and Computing, Houston Christian University, Houston, TX 77074, USA
Robotics and Engineering Organization (REO)
Houston, United States

***Abstract*—Using PyTorch and other Python libraries, we created an artificial neural network that takes as input RFID antenna data and outputs the x and y coordinates of an item in inventory space. Our vision is to make the inventory tracking system faster and more efficient. With the end goal of making everything processed *within* the spaceship and minimizing as much crew work and groundwork as possible so other eminent tasks can be focused on. Our results yielded high rewards as we used Mean Squared Error (MSE) as our loss function to measure how good or bad our ANN was. MSE represents the average squared difference between the predicted and the actual target values within a dataset. Our MSE was 0.0434, making our ANN model high-performing, in other words, our model is making predictions that are close to the actual values.**

***Index Terms*— Artificial Neural Networks (ANN), Radio Frequency Identification (RFID), Space Application (NASA), PyTorch, Machine Learning (ML)**

I. Introduction

Inventory tracking in space involves meticulous collecting, maintaining, and cataloging of objects with codes, whether that be with barcodes or RFID tags. The use of RFID technology in an inventory tracking context ensures that tools and equipment are properly managed when stations have a limited crew. We want to take this a step further and implement artificial intelligence, resulting in complete automation of the inventory tracking system. Inventory items will be marked with an RFID tag and then RFID antennas are used to read the tag. The data from the antennas correspond to the signal strength of each tag in that x and y location. Our goal was to create our own 3D space through the RFID tags and antennas, but challenges with the RFID technology (later explained in Section V) caused us to use another dataset [1].

II. Data Analysis

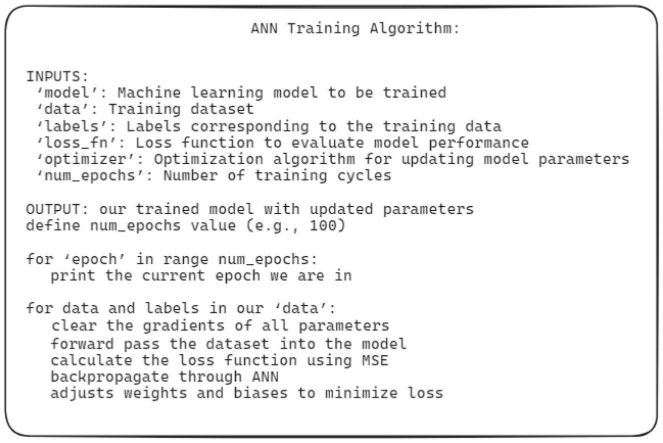
The data we used came from a secondary source which pertains to a Real-Time Locating System where tags are located based on signal strength and other properties taken from the various antennas [1]. The dataset is defined with twelve fields/headers: TagID, rssi_antenna1, rssi_antenna2, rssi_antenna3, rssi_antenna4, rc_antenna1, rc_antenna2, rc_antenna3, rc_antenna4, read, true_x, true_y. Here is a breakdown of each field:

- **TagID**: The unique identifier for the tag (total of 40,000)
- **rssi_antenna1** to **rssi_antenna4**: Received Signal Strength Indicator (RSSI) values from the four antennas, measures in decibel-milliwatts (dBm). Lower values usually indicate weaker signals.
- **rc_antenna1** to **rc_antenna4**: Received Code (RC) values from the four antennas. These might represent signal quality metrics specific to the reader system. In the context of the dataset, they used ThingMagic Mercury 6, a high-performance UHF RFID reader.
- **read**: This value indicates whether the take was successfully read or not.
- **true_x**: The true x-coordinate of the tag's location.
- **true_y**: The true y-coordinate of the tag's location

Before feeding the raw data into our ANN model, it was important for us to standardize the inputs. When the model analyzes the data, each feature it reads (rssi_antenna 1 to 4 and rc_antenna 1 to 4) is in a consistent standard format, improving accuracy later down the road for our model. Data standardization scales the features, so our mean is 0 and our standard deviation is 1, resulting in our data contributing equally during model training. In our program we make a call to sklearn's StandardScaler() function [2]. Equation (1) represents the formula used by StandardScaler(), which uses the standard z-score formula used when standardizing data around the mean and standard deviation.

$$z = (x-\mu) / \sigma \tag{1}$$

With most machine learning algorithms, we split the entire dataset into a training set and a testing set. The training set allows the ANN to learn, influencing the model as we continually pass the data through, teaching it patterns and relationships within the data. The testing set evaluates the performance of the ANN with *unseen* data. Doing this split allows us to avoid our model overfitting the data, meaning that it knows the data so well it cannot generalize and predict new data. The pseudocode for how we wanted our training model to work is shown in Fig.1.



```
ANN Training Algorithm:

INPUTS:
 'model': Machine learning model to be trained
 'data': Training dataset
 'labels': Labels corresponding to the training data
 'loss_fn': Loss function to evaluate model performance
 'optimizer': Optimization algorithm for updating model parameters
 'num_epochs': Number of training cycles

OUTPUT: our trained model with updated parameters
define num_epochs value (e.g., 100)

for 'epoch' in range num_epochs:
   print the current epoch we are in

for data and labels in our 'data':
   clear the gradients of all parameters
   forward pass the dataset into the model
   calculate the loss function using MSE
   backpropagate through ANN
   adjusts weights and biases to minimize loss
```

Figure 1. Pseudocode of our training algorithm.

When the weights and biases adjust, different optimizers use various techniques to minimize loss. In the context of our project, we use PyTorch's Adam optimizer [3]. Short for "Adaptive Moment Estimation", the Adam optimizer is an iterative optimization algorithm with the purpose to of minimizing the loss function by adjusting the learning rate as it gets closer to its loss function [4]. The lower the loss (also known as the cost), the higher the accuracy of our model. The loss function we use is the Mean Squared Error. It measures how close a regression line is to a set of data points. Equation (2) shows how MSE is applied to a general data set. Where *n* is the number of observations, y_i is the observed value, and $\hat{y}_i$ represents the corresponding predicted value. The Σ indicates that a summation is performed over all values of *i*. In Fig. 2, a regression line is applied to a graph, the closer a data point is to the regression line, the smaller the error, essentially *decreasing* MSE [6].

$$MSE = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 \tag{2}$$

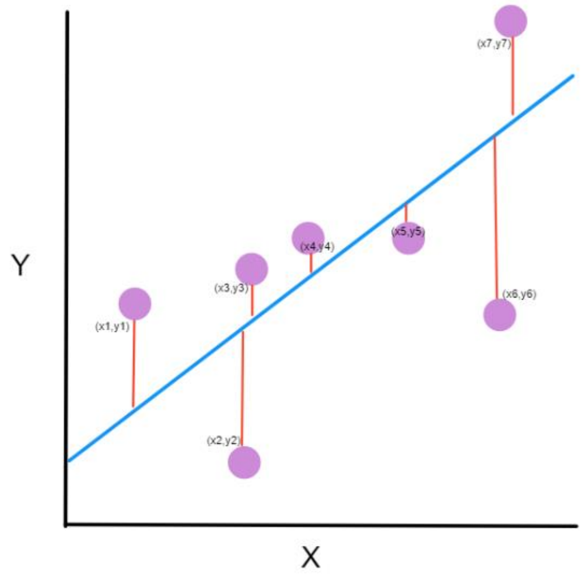
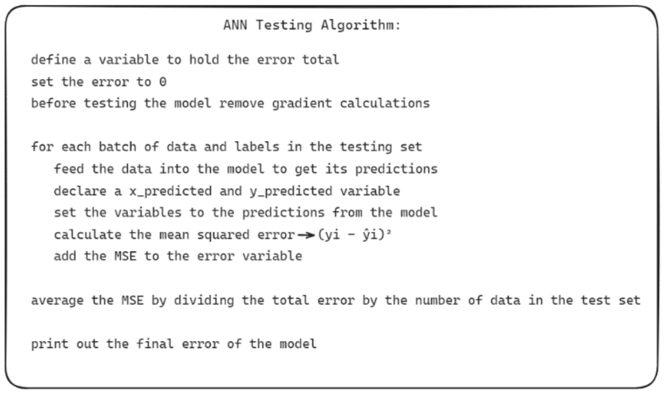


Figure 2. A graph showing MSE

The smaller the MSE, the more precise predictions our model can make. In our case we calculated the MSE to test the accuracy of our model during the testing stage. The pseudocode for our testing algorithm is shown in Fig. 3.



```
ANN Testing Algorithm:

define a variable to hold the error total
set the error to 0
before testing the model remove gradient calculations

for each batch of data and labels in the testing set
   feed the data into the model to get its predictions
   declare a x_predicted and y_predicted variable
   set the variables to the predictions from the model
   calculate the mean squared error → $(y_i - \hat{y}_i)^2$
   add the MSE to the error variable

average the MSE by dividing the total error by the number of data in the test set

print out the final error of the model
```

Figure 3. Pseudocode of our testing algorithm.

At the end of our testing algorithm, we are given the MSE value calculated for our ANN model. In our case our MSE is 0.0434 showing the low loss of our model.

III. Model Design

An ANN is a parallel artificial neural network made of connected nodes, the nodes process and transmit input signals to the next node through activation functions. The ability of ANNs to model and extract previously unknown characteristics and correlations gives us a reliable model to predict our output. For these specific regression tasks, our end goal is to predict the x and y values based on the data from the RFID antennas (such as a specific tag signal strength).

Fig. 4 shows our neural network architecture. The ANN contains eight input neurons, three hidden neurons, and an output layer of two. The weights and biases for each neuron connection are defined by the forward pass method created in our ANN model class.

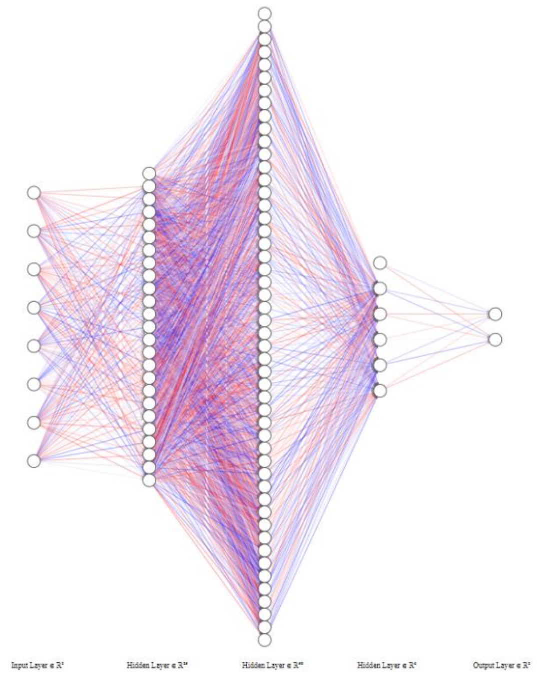


Figure 4. Our ANN Model

Our input layer corresponds to the eight features we are analyzing: rssi_antenna1, rssi_antenna2, rssi_antenna3, rssi_antenna4, rc_antenna1, rc_antenna2, rc_antenna3, and rc_antenna4. Our hidden layers are responsible for learning the complex patterns and relationships between the data. The hidden layers each have a ReLU activation function, the first layer has twenty-five neurons, the second layer has fifty neurons, and the third hidden layer has six neurons. The final layer of the ANN is the output layer. This only has two neurons corresponding to the predicted x-coordinate and the y-coordinate respectively.

When building a neural network, the way each neuron activates for a specific input is important. The activation function we used is what is known as ReLU (or Rectified Linear Unit), which is a mathematical piecewise linear function that outputs the input directly *if* that input is positive, otherwise, it will output zero. See Fig. 5 [7]:

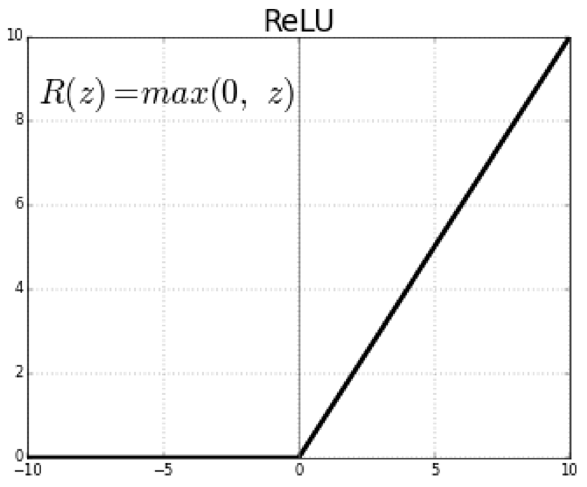
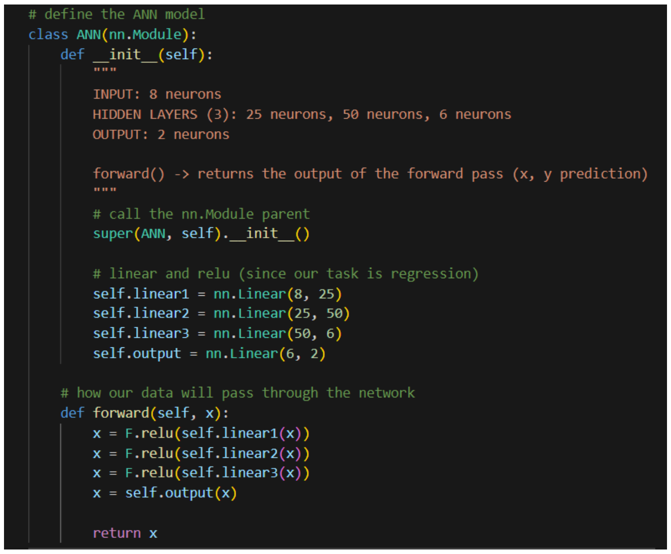


Figure 5. ReLU Graph

Regression tasks often involve capturing complex, non-linear relationships between input features and the target variable, making the ReLU activation function perfect for modeling these kinds of relationships. Another reason we picked ReLU was to avoid any vanishing gradient problems during training. When implementing backpropagation to a neural network, some gradients can become very small or vanish as they flow back through the layers. ReLU helps reduce this problem because gradients are "1" for positive values and "0" for negative values. This facilitates learning as the gradients now propagate more effectively through the network.

The model built through our 'ANN' Class, proposes a multi-layer perceptron with ReLU activations in the hidden layers. Fig. 6 presents how the ANN was built using Python and the PyTorch library.



```
# define the ANN model
class ANN(nn.Module):
    def __init__(self):
        """
        INPUT: 8 neurons
        HIDDEN LAYERS (3): 25 neurons, 50 neurons, 6 neurons
        OUTPUT: 2 neurons

        forward() -> returns the output of the forward pass (x, y prediction)
        """
        # call the nn.Module parent
        super(ANN, self).__init__()

        # linear and relu (since our task is regression)
        self.linear1 = nn.Linear(8, 25)
        self.linear2 = nn.Linear(25, 50)
        self.linear3 = nn.Linear(50, 6)
        self.output = nn.Linear(6, 2)

    # how our data will pass through the network
    def forward(self, x):
        x = F.relu(self.linear1(x))
        x = F.relu(self.linear2(x))
        x = F.relu(self.linear3(x))
        x = self.output(x)

        return x
```

Figure 6. Defining the ANN Class

Having established a well-defined model architecture, the next section explores the practical challenges we encountered during the project, on both the RFID technology side and during the building of our neural network. After training and testing our model, we wanted to visualize the predictions with respect to the ground truth values of the x and y coordinates. Fig. 7 shows the Pyplot mapping of our results [5].

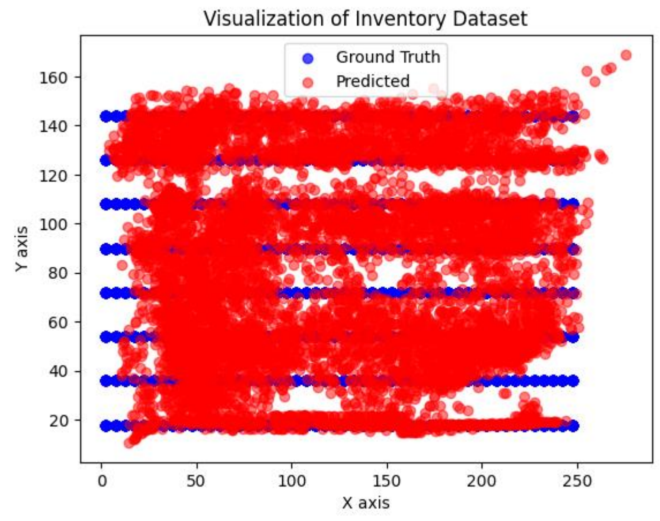


Figure 7. Visualization of our ANN Analysis

IV. Budget

TABLE I

Part Name	Project Budget			
	Part Description	*Qty*	*Unit Price*	*Total Price*
CF-RU5106	5/15M UHF RFID Passive Long Range Card Reader RS232/RS485/Wiegand Parking System	8	$106.09	$848.72
HiLetgo RFID Kit	Mifare RC522 RF IC Card Sensor Module + S50 Blank Card + Key Ring for Arduino Raspberry Pi	6	$ 6.16	$36.96
Raspberry Pi	Raspberry Pi 4 Model B 2019 Quad Core 64 Bit WiFi Bluetooth (4GB)	2	$66.89	$ 133.78
DSD Tech	DSD TECH SH-U09C USB to TTL Serial Adapter with FTDI FT232RL Chip Compatible with Windows 11, 10, 7 and Mac OS	6	$ 12.49	$ 74.94
HiLetgo TTL converter	HiLetgo 5pcs RS232 to TTL Converter Module COM Serial Port Board MAX3232 MAX232CSE	1	$ 7.89	$ 7.89
CD cards	32GB Raspberry Pi Preloaded (RASPBIAN/Raspberry Pi OS) SD Card	2	$ 9.98	$ 19.96
HydraNFC	The HydraNFC is an incredibly powerful and flexible 13.56MHz NFC sniffing/reading/writing/emulating	7	n/a	$0.00
TOTAL:				$1,122.25

a. Project Budget Table

Table 1 above details the materials used to acquire data for our ANN. While we successfully obtained the necessary data for most components, acquiring RFID data on a larger scale presented a significant challenge for our project. We delve deeper into this specific hurdle and the solutions we explored in the next section.

V. Challenges

Our full project was to collect our dataset using RFID readers and tags. Simulating an inventory room, inventory items would be marked with an RFID tag and then we would use RFID readers/antennas to triangulate the location of each item tag. We wanted to use the readers in a way that would allow us to create a 3D space for the data to not only be visualized but also be fed into the ANN. The ANN would then interpret the data and show the crew members an output giving the x, y, and z coordinates of the item they were trying to locate. We were, however, unable to get the RFID technology working, this problem did not bring our project to a halt as we were able to find another data set from a secondary source [1]. In this section, we will break down the problems we had in the RFID part of our project and on the AI side.

RFID CHALLENGES

While working on the RFID section of this project we were able to get a small-scale version of the RFID tracking system working, however, when we decided to create it on a larger scale it did not work. Our challenges were:

Reader Compatibility Issues: Obtaining the appropriate RFID readers proved to be a significant hurdle in scaling up the project. While the small-scale version functioned adequately, transitioning to a larger scale highlighted compatibility issues with the chosen readers. These readers might have lacked the necessary range, frequency range, or other technical specifications required to effectively capture the data from the RFID tags within the expanded system. Despite initial research and testing, finding readers that could seamlessly integrate with the expanded setup posed a considerable challenge.

Data Integrity and Signal Strength: Other than reader compatibility challenges, maintaining data integrity emerged as another obstacle. The code initially utilized for the RFID tracking system resulted in information dumping, meaning the outputting data was just the tag's hex number. This output only led to inefficiencies in data processing and analysis. To address this issue, our solution involved the implementation of a Digital Frequency Signal Tester. This Tester enabled the team to assess the strength and quality of the transmitted signals *between* the RFID tags and readers. By cross-referencing signal strength data with the time taken for transmission, we successfully derived a method to calculate the distance between the RFID tags and the readers.

System Optimization and Integration: Extensive optimization efforts were required to incorporate RFID technology into a larger-scale system. The complexities of synchronizing many RFID scanners and tags became more apparent as the project grew. One of the biggest technological challenges was ensuring that different components could communicate and exchange data without interruption while also keeping the system reliable. Careful attention to detail in the specific hardware configuration and software development was required to fully optimize the system so it could handle the increased volume of RFID tag data efficiently.

While the small-scale RFID tracking system demonstrated initial success, scaling up the project presented multifaceted challenges ranging from reader compatibility and data integrity issues to system optimization and environmental considerations. Addressing these challenges necessitated a combination of technical expertise, innovative problem-solving, and iterative refinement to achieve the desired functionality and reliability in the expanded RFID deployment.

AI CHALLENGES

Machine learning models were a new topic of exploration for us, diving into this new field came with its challenges.

Finding the Hyperparameters of Best Fit: After initially building our model, we did not get the low error we desired, so we took the time to fine-tune all hyperparameters, starting with our learning rate (LR). We initially had our LR equal to 0.0001, as we wanted our model to take small steps and adjustments to the weights and biases at each error it makes during training. Our final LR was 0.001, giving us the most effective optimizer for our model training. The biggest hyperparameter we had trouble finalizing was the number of hidden layers and neurons within those layers. As a group new to neural network building, we knew we needed an input and output layer, and that the hidden layers depend on the task and how *complex* it is. Our first ANN model had three layers, with one hidden layer of 100 neurons. It gave us a decent output, but we knew we needed to do better for the model to predict more accurately. Our final ANN model (as seen in Section III) has five layers, with three hidden layers. These hidden layers allow the activation to capture more patterns and abstract features of the inputs given. Starting simple and gradually increasing our architecture complexity gave us a benchmark to compare performance at each change, resulting in the ideal ANN model we wanted that minimized loss.

VI. Summary

This paper mapped out our journey through our project of merging RFID and AI, with the end goal of making inventory tracking on the spaceship more efficient and automated for the space crew. We investigated the effectiveness of our ANN for inventory tracking in 2D space. The model trained on a secondary source dataset that provided the RFID signal strengths and achieved a Mean Squared Error of 0.0434. We analyzed the model's performance and observed that our MSE tells us that our predicted values are going to be around 0.2 cm away from its *actual* position in the room. These results demonstrate the potential of this AI approach for inventory tracking.

VII. Conclusion

We learned a lot through this process of working with the RFID technology and building an ANN model. Our end results showed a promise of using an Artificial Neural Network for inventory tracking. However, there are opportunities for further improvements. Further work could explore outputting the x, y, and z coordinates of an item, or even explore other neural network architecture such as Graph Neural Networks. Exploring the limitations of the model, such as handling unseen data with different characteristics, could provide valuable insights for future research. Even though we had troubles and difficulties with the RFID technologies, our model testing with the secondary dataset tells us that we are extremely confident in taking this project into a 3D scale.

Acknowledgment

The authors gratefully acknowledge the contributions of J. McCarthy and G. Hinton to the artificial intelligence and machine learning industry.

We would also like to acknowledge Houston Christian University for their facility and equipment use, which were instrumental in conducting our research.

We are also deeply grateful to our mentor, Dr. Ricardo Ramirez, for his invaluable guidance and support throughout this project. His expertise in electrical engineering and machine learning was crucial in helping us to complete this project. His encouragement and insights were instrumental in shaping the success of this project.

References

Datasets:

Eduardo Luis Gomes, Mauro Pereira Sérgio Fonsecaand André Eugenio Lazzaretti, "Passive RFID indoor location dataset". Zenodo, Apr. 28, 2022. doi: 10.5281/zenodo.6502312.

Documentation:

"Documentation scikit-learn: machine learning in Python — scikit-learn 0.21.3 documentation," scikit-learn.org. https://scikit-learn.org/0.21/documentation.html

"PyTorch documentation — PyTorch 2.1 documentation," pytorch.org. https://pytorch.org/docs/2.1/

D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv.org, Dec. 22, 2014. https://arxiv.org/abs/1412.6980

Matplotlib, "Matplotlib: Python plotting — Matplotlib 3.3.4 documentation," matplotlib.org. https://matplotlib.org/stable/index.html

Figures:

S. SHARMA, "Activation Functions in Neural Networks," Medium, Sep. 06, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

M. Binieli, "Machine learning: an introduction to mean squared error and regression lines," freeCodeCamp.org, Oct. 16, 2018. https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/