Estimating Trigonometric Functions Using ML – Midterm Report

Alexandra Walker and Mattie Fuller

University of Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
awalker17@uccs.edu and mfuller@uccs.edu

Abstract

This paper proposes training an RL agent to be able to estimate the solutions to triangles containing missing angles and sides. These estimates will be evaluated via mean squared error (MSE) and other similar metrics; should the margin of error be small, the hidden functions that the agent is approximating can be found. That is, to show that an RL agent can 'discover' trigonometric functions such as sin, cos or tan.

Introduction

Machine Learning (ML) is a method of computational learning utilizing large data sets to analyze algorithms for the purpose of finding trends in data. This proposal will outline a method of analyzing right triangles to estimate sine, cosine, tangent, and possibly more like Pythagorean theorem, Pythagorean identity and many more. Right triangles have many unique and intricate properties that can be expressed by elementary functions. To that end, this paper proposes an experiment to teach trigonometry to a ML algorithm, such as a reinforcement learning agent.

Background

Machine Learning was imagined as soon as 1959 and since then has evolved to detect medical conditions (Verma and Peyada, 2020) such as breast cancer, or even predicting natural disasters (Gracia et al., 2021). In addition, papers around solving math problems, have been using RL to better break down algebraic equations for many purposes (Dabelow and Ueda, 2025). The process of finding functions to best fit the data even has its own field as summarized by Kumar and Bhatnagar (2022) for linear regression. For non-linear regression, Gauss-Newton, a popular method as proposed in Korbit and Zanon (2024) is a better regression technique for non-linear applications. Verma and Peyada mentioned earlier used Gauss-Newton to demonstrate a new method of breast cancer classification.

Literature Review

Reinforcement Learning (RL) has been used successfully for similar problems to this. Specifically Q-Learning has been used solve algebraic problems in an exact manner. (Dabelow and Ueda,2025). This was achieved by having the world model consist of 3 mutable stacks, one for the right hand side of the equation, one for the left hand side of the equation and the last effectively as scratch paper for the RL agent to solve the problem. The agent's actions were then defined as moving elements between the stacks. This approach yielded great results; a similar approach likely could be used for solving trigonometry problems.

In Liu et al.'s work from 2022, the team uses a reinforcement learning model to prove trigonometric identities. This is quite a different problem than what we are attempting to do in this paper, but it demonstrates various techniques adjacent to this problem.

In Dietterich's paper from 1999, the paper proposes a method for the q-learning update function based on maximum hypothetical future score denoted by maxQ, this is the method that is implemented in our current RL model.

Problem Statement

In its most simple terms: can a reinforcement learning algorithm learn to solve incomplete triangles therein learning to estimate trigonometric functions.

Dataset

Triangles, are particularly easy to generate with high accuracy. With this in mind, a custom dataset can be easily generated with thousands if not hundreds of thousands of samples in only a few minutes, so finding data to train on with solid data is easy to create. However, as discussed later, more information is necessary to allow reinforcement learning to go more smoothly. Including, missing data, the type of triangle the agent is trying to solve, and having the correct answer for the missing information is vital to train on. Our methodology section will go into how these will be used, but during training having the answer and the triangle type included into the dataset information simplifies the process, as we are using RL and Q-Learning, if we do not separate the dataset by triangle type this will result in the q-learning table trying to learn to solve all types of triangles within the same domain, which in practice heavily reduced the model's capability to solve triangles. The triangle type is one of (potentially) 7 different types of triangles that have unique ways of solving,

in a basic geometry class these would likely be taught in relation to missing data, so a triangle that has 2 sides and and angle in between would be classified as "Side, Angle, Side" abbreviated as SAS. Given this the following types of triangle emerge: AAA, AAS, ASA, SAS, SSA, SSS, and lastly UNSOLVABLE, which gives us 7 distinct types that a triangle can be classified as, the methedology used for classifying these triangles into these distinct types will be discussed in a later section. In context of our data set this adds 1 more column with 7 possible discrete values.

Triangle Type Classifier

Triangles are classified into 7 distinct types: AAA, AAS, ASA, SAS, SSA, SSS, and UNSOLVABLE, this is done based on what sides/angles the incomplete triangles have. To determine what type a triangle falls into we first convert the array of angles and sides into a array of 1 or 0 for each element, representing if the value for it was given or not. Then to classify triangles into the 7 categories, we perform bit wise and and xor operations against the angle inclusion array -¿ the side inclusion array, if an angle is between two known sides an and operation will yield three 1 values in the resulting array, likewise if an angle is beside 2 sides a xor operation will yield 3 ones in the resulting array. From there the count of sides and angles can be checked to determine what specific type of triangle we have. Thus allowing us to classify each triangle we generate into one of the above 7 classes. It warrants mentioning that AAA and UNSOLVABLE are both unsolvable as 1 side must be known to solve a triangle, and the unsolvable class is a catch all for any triangle that did not get classified any other way.

Methodology

Initially focusing on using Q-Learning, a Reinforcement Learning method, we seek to estimate sin, tan, and cos by solving training an RL agent to solve triangles with up to 3 values missing. As such, the key elements to consider are the following: States, Actions, Reward Policies, and the Environment model (Ghasemi and Ebrahimi, 2024).

Environment Model And State

Reinforcement Learning (RL) functions by allowing an autonomous agent to interact with a representation of the world via applying actions to a state; then rewarding the agent for 'good' choices and punishing for 'bad' choices. States are specific configurations of the environment, for instance, a chess board with black king h8, white pawn g7 and white king e7, is a state representing black in checkmate to white's pawn protected by white's queen.

The RL agent can only interface with and interpret the world via states. Given this, only information the RL agent needs to learn a task should be included. States can be expressed as N-dimensional vectors where N is the number of significant data types (Harmon and Harmon, 1996). For solving triangles with missing sides or angles, we propose 7 significant features.

| A | B | C | a | b | c | Triangle Type |

Table 1: A table depicting our 7 key features for representing state. Where capital letters are angles and lowercase letters are sides. And triangle type is the aforementioned way we classify triangles into their types

Sides a-c represent the side lengths of their corresponding sides, likewise angles A-C represent the angle measurements of each angle. These 6 features are likely the minimum required to train the RL agent to solve triangles. And the agent is proven to be capable of solving triangles with moderate accuracy when only provided with these 6 features later on. Even though these 6 features are the bare minimum, there is one more feature that is almost mandatory to include: "Triangle Type." Without adding the triangle type feature, the q-learning model, that is the q-table ends up training to solve different types of triangles within the same state spaces, thus confusing the model directly proportional to how many different types of triangles it was trained on. By adding in this 7th feature we eliminate a large quantity of the confusion that was previously created by breaking the q-table's action probabilities into distinct state spaces for each type of triangle.

Actions

An RL agent views the world through states; it interacts with the world through actions. Any manipulation of a state must be represented as an action that the agent can preform. In this way, we must define all the possible ways that the agent is permitted to interact with the state.

We identified 3 distinct strategies to implement and test for what action sets the agent has assess too. We have currently implemented the first and the framework to implement the others.

- Action Set 1 – The Increment Method. In this action set we provide the agent 2 actions that can be performed upon each of the 6 sides/angles, these actions are to either add a small quantity or subtract a small quantity. This results in 12 total actions as there are 6 features the agent could choose to modify and 2 actions it may perform to any of them thus $6 * 2 = 12$. This method has many drawbacks one major draw back is that it can never perfectly solve for values, and how close it gets to the answer is directly based on how small of an increment value is chosen, a modification that can be made to improve this model is to provide two additional actions to the model: one action to increase the size of the increment value and one action to decrease the size of the increment value, this at the very least would allow the agent the capability to get closer to the answer more easily and in fewer steps. But another significant drawback to this method arises, if we are to assume that a maximum of 3 values are unknown to the agent then that means the other 3-5 are known values and thus the agent should not modify. This means that at best (where 3 values are unknown) the agent should not modify 50% of its features and thus consequently it should not perform 50% of its actions. This

can be accounted for by penalizing the agent whenever it attempts to modify a given value, which has proven to be a decent way to prevent this from causing the agent to take prohibited actions.

- Action Set 2 – The Stack Method. In this method, similar to the work done by Dabelow and Ueda, we propose the following structure for actions: We give the RL agent a "internal memory" or a "working memory" which we will represent as a stack, we then give the RL agent the ability to push and pop entries from the stack, giving way to the first 7 actions: moving any angle or side onto the stack, and popping the top item off the stack. Then let the agent also move mathematical symbols onto the stack, yielding 4 more actions, pushing "+,-,*,/" onto the stack. From here the concept behind this method should start to become clear, the intent is to allow the agent to construct its own formulas and then allow it to pass the stack it created to a mathematical evaluation function and then return the result back to the agent. The last mandatory action that this method requires are the 6 actions of solving the stack the agent has created and pushing the resulting value into any of the 6 features that it could be solving for (the 6 angles/sides). This method provides the most dynamic and modular approach, allowing the agent a lot of freedom, but this method also produces many drawbacks, firstly we must ensure the agent does not create invalid mathematical equations, likely this can be achieved by implementing a heavy punishment for any attempt to solve its stack that fails. For instance the stack can be converted to a string, consider the stack ["2","/","4"] this can convert to "2/4" and then can be passed to the python eval() function. A simple way to disincentivize invalid equations is to introduce a heavy penalty for any attempt to solve its stack that causes the eval() function to throw an error. The other drawback to this method is shear amount of actions that can be added. The bare minimum actions for this method are listed above, but stack transformations such as shifting, shuffling, rearranging and allowing more types of mathmatical symbols to be added are all types of actions that could expand upon this method though it is unclean if they would help or hinder it as of yet.
- Action Set 3 – The Limited Math Method. In this action set we expand upon action set 1, but rather than providing nearly limitless freedom as in Action Set 2, in this action set we provide very limited actions as to how the agent can multiply, divide, add, subtract, ect. We propose a small set of actions below:
  - To add or subtract a small value K from any of the 6 sides/angles, where k is a small quantity. This is the exact same as action set 1, included in this action set. This sums to 12 actions, as for each feature we may add or subtract from it, thus 6*2=12.
  - Subtract any two angles, A and B from each other, provided that A $\neq$ B and store to a temp variable Z. That is A-B, for any two angles A and B; A $\neq$ B. This sums to 6 more actions.
  - Multiply any 2 Angles or Sides x and y to each other and store to a temp variable Z, provided x $\neq$ y. This results in 6*5=20 more actions, since for each of the 6 angles/sides the agent can choose any of the remaining 5 to multiply by, resulting in 20 actions.
  - Divide any 2 angles or sides x and y from each other and store to a temp variable Z. This also results in 20 possible actions. Note that
  - Store the value in the temp variable Z into any of the 6 angles/sides in the state, this reduces the number of actions as each of the above actions do not have to have 6 sets of them where each different set stores to a different state index. Rather they all store to Z and then the agent can choose where to put Z. It warrants mentioning for this to be effective Z likely need to be an additional state feature.

  This restricted set of actions allows most operations needed to roughly solve each type of triangle. But this method might not provide enough freedom to properly estimate the trigonometric functions as most ways of calculating trig functions require intermediate steps, for instance the CORDIC method for approximating trigonometric functions would not be possible within this limited action set (Volder 1959), and thus the agent must find a different way to approximate these values or alternatively we must add new actions to allow better methods.

Reward Policies

In the initial work that has been completed we have implemented Q-learning based on the maxQ learning formula described by Dietterich where the learning update rule is defined as:

$q = (1 - a) * c + (a * r + (L * maxQ)$

Where a is the learning rate, c is the current q value from the choice being evaluated, r is the reward from the action that was performed, L is the discount factor and maxQ is the estimate of the optimal future value. Defined as:

$maxQ = max(Qtable[nextstate])$

IE: the maximum value in the row of the Qtable for the resulting state of the action being assessed. (Dietterich 1999)

Our reward policies in context to the currently implemented RL agent with action set 1, our policies are as follows:

- The agent receives a penalty of -0.1 for each step taken
- The agent receives a reward of +0.01 for the step that removed a null value, that is the first step that sets a unknown value (denoted as null) to a value receives a small reward to incentivize quickly filling in the null unknown values.
- The agent receives a reward of +0.2 if it lowers the error compared to the known value it is trying to solve for. For instance if the agent is trying to solve a triangle with angles: 60, 60, 60 and sides: 0, 1, 0, if it were to move either of the 0 values closer to 1 it will get a reward. This is done by checking the state before an action and after an action and comparing which has a smaller percent error from the ground truth, then providing a reward if the agent improved the answer, if it does not, no reward is provided nor penalty added.

- The agent receives a large reward of +10 if it solves the triangle, defined as getting the percent error of the triangle to be less than 0.1%
- The agent receives a moderate penalty, -1.0 for attempting to take an invalid action. Invalid actions are defined as: modifying any value that the agent was given, since given values should not be changed as they are already known to be correct.

As shown in Jin et al. (2024) demonstrated a new RL reward algorithm designed for Q-Learning that showed "near-optimal policy with satisfactory convergence, and performs well across MDPs". Seeking to achieve similar results, their proposed algorithm will be the experimental reward policy of the RL agent.

Timeline

The project as it is completed will loosely follow the timeline depicted below, dates in bold are strict and cannot be changed:

2/19/25	Project Proposal Report
2/19/25-2/24/25	Generate Dataset
2/24/25-3/15/25	Build and debug RL 'model'
3/15/25-4/1/25	Train and debug RL 'model'
4/1/25	Midterm Report
4/1/25-5/8/25	Finalize Models and Report Paper
5/8/25	Final Demo

Evaluation

The success or failure of this project will be determined by two factors. Firstly, whether the RL agent is able to accurately estimate the correct answers to the trig problems. Secondly, if we can reverse engineer the trigonometric functions from the RL agent's learned estimations; that is can the RL agent 'discover' trigonometric functions such as sin(x), cos(x), and tan(x). Furthermore metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Squared Error (MSE) can be used to provide numerical representations of how well the RL agent estimates correct answers.

Results

In the midterm section of this course we implemented the a class to generate triangles and identify the triangle's type, this serves as our dataset, creating incomplete triangles with up to 3 angles/sides missing and classifying them into their respective types. Furthermore we have implemented the framework for q-learning and reinforcement learning and tested it with Action Set 1 achieving moderately good results with this.

First we attempted to implement Action Set 1, the increment/decrement method with solely the 6 angles and sides as its state. This resulted in the model often solving poorly when trained against multiple types of triangles (as discussed prior, this can be fixed with the addition of the 7th feature – triangle type). Upon training and testing on various data sets with exploration rate = 0.1, learning rate = 0.1 and discount factor of 0.9, an a increment size of 0.05 with a maximum step count of 1000, the following results were achieved. The charts below demonstrate different MSE values based on what data set the model was trained and tested on, it performed the best when trained and tested on sets comprised of exclusively the same type of triangle, performing worse when trained on random triangles and tested on the same type, and performing the worst when trained and tested on sets containing random mixtures of triangle types:

Tested On	Trained On	Mean Squared Error
Isosceles (ASA)	Isosceles (ASA)	0.00494
Isosceles (ASA)	Random	0.00544
Random	Random	200,694,944.0
SAS	Random	18175.92
SAS	SAS	17948.80

Table 2: Mean Squared Error for various training and testing configurations (100 epochs each, with with exploration rate = 0.1, learning rate = 0.1 and discount factor of 0.9, an a increment size of 0.05 with a maximum step count of 1000). 100 examples for test, 100 examples for train.

When maximum step sizes are increased to 10,000 training takes substantially longer especially on any dataset involving random triangle types, while when training on datasets of the same type it takes no additional time when max step count is increased (as the model is solving them substantially faster). The following results were achieved with all config the same except for max step count increased to 10,000 during training.

Tested On	Trained On	Mean Squared Error
Isosceles (ASA)	Isosceles (ASA)	0.00520
Isosceles (ASA)	Random	0.00545
Random	Random	64,820
SAS	Random	1261.9416
SAS	SAS	1274.6653

Table 3: Mean Squared Error for various training and testing configurations (100 epochs each, with with exploration rate = 0.1, learning rate = 0.1 and discount factor of 0.9, an a increment size of 0.05 with a maximum step count of 1000 for testing and 10,000 for training).

Notice that the set of random triangles trained on random triangle's MSE improved drastically with the training step count increased, although it is still a horrible fit. Additionally note the MSE of SAS against random improved by a similar factor to random to random. This behavior is consistent with what we theorized earlier, that without the 7th feature of triangle type, the model will struggle to differentiate the types internally and thus require more training to achieve decent results, in this case we choose to increase only the max steps allowed rather than the training epochs as to ensure while the model is training it actually finds a good answer, as with 1000 max steps it was unable to do so on most of the training involving random types.

As we can see from these results, Action Set 1 is substantially better at solving isosceles triangles, that is solving for sides rather than angles, and when it is tasked with solving for angles in the case of SAS vs SAS or any of the random ones, it takes substantially more time to train the q-learning table to get decent results.

Finally we tested the reinforcement learning agent again training against 100 random incomplete triangles for a max of 100,000 steps and tested against 20 random incomplete triangles.

Tested On	Trained On	Mean Squared Error
Random	Random	1625187.875

Table 4: Mean Squared Error for trained on random, tested on random, for 100 training epochs with max 100,000 steps, otherwise unaltered configuration.

This result is somewhat confusing, as increasing the max steps for training improved the random vs random MSE substantially when increased from 1,000 to 10,000 but caused the MSE to get substantially worse when increased to 100,000. This is likely due to one of two reasons, either this is causing overfitting to each individual triangle example since so many steps are allowed, or since there is no triangle type feature yet when a different type appears effectively the solution of a different type is attempted, fails and then overwritten.

With these results in mind, the very first thing we can do to improve these results is to add the 7th feature of triangle type into the state, this should hopefully result in all categories of triangle having MSEs closer to that of our results for ASA triangles. From there once the other action sets are implemented we will be able to know which methodology for action set performs the best for solving these incomplete triangle problems.

Conclusion

As such, we hope to show that an RL agent utilizing Q-Learning can estimate solutions to triangles with missing sides or angles. Then using that learned ability to solve triangles to hopefully derive/discover the actual function the agent is approximating.

References

Dabelow, L.; and Ueda, M. 2025. Symbolic equation solving via reinforcement learning. *Neurocomputing*, 613: 128732.

Dietterich, T. G. 1999. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. arXiv:cs/9905014.

Ghasemi, M.; and Ebrahimi, D. 2024. Introduction to Reinforcement Learning. arXiv:2408.07712.

Gracia, S.; Olivito, J.; Resano, J.; del Brio, B. M.; de Alfonso, M.; and Álvarez, E. 2021. Improving accuracy on wave height estimation through machine learning techniques. *Ocean Engineering*, 236: 108699.

Harmon, M. E.; and Harmon, S. S. 1996. Reinforcement learning: A tutorial. *WL/AAFC, WPAFB Ohio*, 45433: 237–285.

Jin, Y.; Gummadi, R.; Zhou, Z.; and Blanchet, J. 2024. Feasible Q-Learning for Average Reward Reinforcement Learning. In Dasgupta, S.; Mandt, S.; and Li, Y., eds., *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, 1630–1638. PMLR.

Korbit, M.; and Zanon, M. 2024. Incremental Gauss-Newton Descent for Machine Learning. *arXiv preprint arXiv:2408.05560*. Accessed: 2025-02-18.

Kumar, S.; and Bhatnagar, V. 2022. A Review of Regression Models in Machine Learning. *JOURNAL OF INTELLIGENT SYSTEMS AND COMPUTING*, 3(1): 40–47.

Liu, Z.; Li, Y.; Liu, Z.; Li, L.; and Li, Z. 2022. Learning to Prove Trigonometric Identities. arXiv:2207.06679.

Verma, H. O.; and Peyada, N. K. 2020. Parameter estimation of aircraft using extreme learning machine and Gauss-Newton algorithm. *The Aeronautical Journal*, 124(1272): 271–295.

Volder, J. E. 1959. The CORDIC Trigonometric Computing Technique. *IRE Transactions on Electronic Computers*, EC-8(3): 330–334.