Teaching Computer Programming Using Mathematics: Examples from Middle-School and Graduate School

Article *in* SN Computer Science · November 2024

DOI: 10.1007/s42979-024-03386-z

4 authors:

Marios S Pattichis
University of New Mexico
425 PUBLICATIONS **5,654** CITATIONS
SEE PROFILE

Hakeoung Hannah Lee
University of Virginia
23 PUBLICATIONS **34** CITATIONS
SEE PROFILE

Sylvia Celedón-Pattichis
University of Texas at Austin
78 PUBLICATIONS **703** CITATIONS
SEE PROFILE

Carlos A. LópezLeiva
University of New Mexico
66 PUBLICATIONS **312** CITATIONS
SEE PROFILE

SN Computer Science (2024) 5:1006
https://doi.org/10.1007/s42979-024-03386-z

ORIGINAL RESEARCH

Teaching Computer Programming Using Mathematics: Examples from Middle-School and Graduate School

Marios S. Pattichis[1] · Hakeoung Hannah Lee[2] · Sylvia Celedón-Pattichis[2] · Carlos LópezLeiva[3]

Received: 3 April 2024 / Accepted: 4 October 2024

Abstract

Our goal is to provide integrated lessons where computer programming concepts are introduced based on mathematics. We consider the development of lessons that would be interesting to our students. At the middle school level, digital video generation is used to introduce coding. At the graduate level, we look at the convergence of machine learning models during training. We introduce middle-school students to computer programming through the use of variables, linear equations, and basic algebraic expressions. We motivate students to create digital images using NumPy arrays by experimenting with number representations and coordinate systems. The students create digital videos by building their video characters and moving them around from frame to frame. At the graduate level, we describe how Real Analysis can be applied in Optimization Theory. The students saw and appreciated the connections between Mathematics and Computer Programming. In the graduate course, the students appreciated the rigorous results on the convergence of neural network models. The approach also produced conditions for guaranteeing that the neural network models are uniformly continuous. We have found that the students strongly appreciated the integration of mathematical concepts into basic and advanced coding courses.

Keywords Mathematics and computer programming · Teaching computer science · Machine learning algorithms · Uniformly continuous neural network models

Introduction

There is a strong need to teach the fundamentals of computer programming to the general population [1]. Unfortunately, often, schools allocate very little to no time for educating students how to code. On the other hand, schools are required to provide coursework in mathematics throughout K-12. Furthermore, many of the skills that are taught in mathematics classes are also essential for understanding computer programming. As an example, both mathematics and computer programming encompass foundational concepts and necessitate logical thinking, problem-solving abilities, and the application of creativity. In this paper, we propose to teach computer programming building on its connections to the underlying mathematics [1–3].

By leveraging students' mathematical knowledge, students, and even teachers, can also save considerable time that would otherwise be spent introducing each coding concept independently from its mathematical counterpart, such as the concept of variables. Learning coding in conjunction with mathematics also enables students to revisit and explore mathematical concepts in greater depth, creating a

Hakeoung Hannah Lee, Sylvia Celedón-Pattichis and Carlos LópezLeiva contributed equally to this work.

✉ Marios S. Pattichis
pattichi@unm.edu

Hakeoung Hannah Lee
hklee@utexas.edu

Sylvia Celedón-Pattichis
sylvia.celedon@austin.utexas.edu

Carlos LópezLeiva
callopez@unm.edu

[1] Department of Electrical and Computer Engineering, University of New Mexico, 211 Terrace Street NE, ECE Building, Albuquerque 87131-0001, NM, USA

[2] Department of Curriculum and Instruction, The University of Texas at Austin, 1912 Speedway, Stop D5700, Austin 78712, TX, USA

[3] Department of Language, Literacy, and Sociocultural Studies (LLSS), University of New Mexico, MSC05 3040, 1, Albuquerque 87131-0001, NM, USA

Published online: 01 November 2024

reciprocal relationship between mathematics and computer programming. This integration facilitates greater accessibility to computer programming for students who may not initially have a natural inclination toward the subject [4–6]. By embedding computer programming within the context of mathematics, it becomes more appealing and approachable, capturing the interest and engagement of a wider range of students. The interconnectedness of mathematics and computer programming creates a reciprocal learning process: from mathematics to computer programming, and back from computer programming to mathematics. This integrated approach empowers students to develop knowledge and a versatile skill set that seamlessly bridges the realms of mathematics and computers, preparing them to thrive in an increasingly digital and technologically-driven world.

We present two examples of our efforts. First, we summarize how the underlying middle-school mathematics was used to introduce advanced NumPy programming concepts in the Advancing Out-of School Learning in Mathematics and Engineering (AOLME) project. The successful learning of fundamental mathematical concepts in the AOLME project has already been documented in [4, 7]. In the current paper, we focus on the coding aspects of the project and how it is introduced from the underlying middle-school mathematics. Second, motivated by the success of the AOLME project, we present how the same ideas can be applied in a graduate course in optimization that uses Real Analysis for selecting an optimal Neural Networks model. For this application, we review how Real Analysis can be used to establish convergence of the validation loss sequence generated during neural network training. We also derive conditions that guarantee that the generated neural network models are uniformly continuous. This effort extends our prior efforts to introduce Linear Algebra methods to understand Neural Networks as detailed in [8].

The rest of the paper is organized into 4 sections. In Sect. "Background", we review prior pedagogical efforts to integrate mathematics and computer programming. In Sect. "Methods: Teaching Computer Programming with Mathematics", we describe our methodology. We provide results from student interviews in Sect. "Results from Student Interviews" and concluding remarks in Sect. "Conclusion".

Background

There was a continuous endeavor to connect computer programming to mathematics [9–16]. Articles delve into the interplay between mathematics and computer programming, each offering unique perspectives and ideas.

The article by Feurzeig, Papert, and Lawler [9] explores the use of programming languages as a conceptual framework for teaching mathematics. This work emphasizes the potential of computer programming to enhance students' mathematical understanding and problem-solving abilities. The authors argue that programming languages provide a unique platform that encourages active engagement, promotes critical thinking, and facilitates the development of mathematical reasoning skills.

Goldenberg and Carter [17] focus on the use of computer programming as a language for young children in elementary grades to explore concepts in the mathematics classroom. They argue that, when young children engage with computer programming, they also connect to mathematical practices. The authors argue that when connected to classroom mathematics, computer programming can be used as a third language that decreases barriers and provides young students with the expressive and creative skills they need. Similarly, Benton and colleagues [18] also designed curriculum materials and professional development to support mathematical learning through computer programming for young children aged between 9 and 11 years. The authors discovered that by implementing the program, key foundational concepts become more accessible to students. Solin and Roanes-Lozano [19] approached computer programming as an effective complement to mathematics education and they also conclude that computer programming actually provided more engaging ways to teach mathematical practice standards to students.

In secondary school level, Kaufmann and Stenseth [11] investigated how computer programming can be integrated in mathematics using Processing (Processing is a Java based tool primarily to learn program visual effects supported and distributed by The Processing Foundation). The analysis illustrates students' reasoning when using Processing to solve mathematical problems. The students showed a growth in their argumentation ability, going from basic to more complicated arguments.

In undergraduate level, Wilensky [20] explored the use of the Logo programming language as a tool to develop undergraduate students' understanding of mathematical concepts. He argues that Logo programming offers a unique opportunity for students to build tangible connections between mathematics and computer programming by engaging in hands-on activities. The article emphasizes the importance of creating meaningful connections between mathematics and computer programming to enhance students' mathematical understanding and problem-solving skills. Sangwin and O'Toole [21] investigated how much computer programming is integrated into the curricula of British undergraduate mathematics majors. The authors found that whereas computer programming is taught to all undergraduate mathematics students in 78% of BSc degree courses, in 11% of mathematics degree programs it is not.

Olteanu [22] suggests several recommended conditions for fostering mathematical reasoning and sense-making through the use of an educational programming tool. These conditions include adequate teacher interventions, the design of rhizomatic tasks, the identification of critical aspects, and the utilization of patterns of variation. By adhering to these conditions, educators can create an environment that nurtures students' mathematical thinking and promotes their ability to make meaningful connections and discoveries while engaging with educational programming tools.

Collectively, this body of literature provides valuable insights into the intricate and ever-evolving interplay between mathematics and computer programming. However, despite the knowledge available, a notable gap remains in the absence of a comprehensive curriculum intentionally designed to connect computer programming with mathematics. While the existing literature offers glimpses into the potential synergy between these disciplines, there is a need for a cohesive and structured educational framework that purposefully integrates the two fields. Such a curriculum would not only bridge the gap but also unlock the full potential of combining mathematics and computer programming in educational settings.

The authors [6, 7] also explored the experiences of bilingual Latinx co-facilitators with the new mathematics and computer programming integrated curriculum. The co-facilitators experienced a shift in their perception of mathematics as they utilized computer programming tools in the new curriculum, resulting in a more relatable and meaningful understanding. Embracing their role as co-facilitators, they effectively taught computer programming practices and fostered a positive learning environment. The authors found increases in enjoyment and self-confidence when middle school students took on the co-facilitator role. The study highlights the potential for middle school students, particularly those who are bilingual, to excel in computer programming and bilingual teaching while assuming new roles and goals. The findings from this study indicate that when middle school students have the opportunity to co-teach mathematics and computer programming concepts, they solidify their understanding of these concepts. In a recent study, the authors [4, 5] explored the relationship Latinx students developed with Computer Programming and Mathematics (CPM) while experiencing CPM curriculum in an afterschool setting. Students had significant increases in their self-reported enjoyment and knowledge in CPM as they engaged in the program and the program prepared students with the foundational knowledge, skills, and practices for future endeavors in STEM fields.

Methods: Teaching Computer Programming with Mathematics

Motivation and Setup

In designing our curriculum, we wanted to follow a few guiding principles to help us design effective coding activities. First, we wanted to build coding activities based on the underlying mathematics. Our goal here is to build a better understanding of coding concepts by building on students' understanding of basic mathematical concepts. At the middle-school level, we wanted the students to understand coding variables and basic algebraic operations through their mathematical equivalents. At the graduate-level, for the optimization theory course, we wanted to review and borrow concepts of Real Analysis, which provides basic definitions of convergence. For the computer vision course, we introduced fundamental concepts in Linear Algebra and vector spaces as outlined in [8]. An advantage of the approach is that the students get to use their mathematical knowledge to understand new concepts in basic coding and advanced coding issues in programming machine learning optimization methods. Furthermore, by connecting coding to middle-school math, the middle-school teachers were able to make the connections to coding in their regular classroom lessons. At the graduate level, the material from Real Analysis allowed the students to understand how they can achieve convergence during the training of neural network models. A disadvantage of the approach comes from the fact that the students need to carefully review and understand the underlying mathematics. Alternatively, teaching computer programming without using the underlying mathematics may lead to a superficial understanding of coding fundamentals. Furthermore, for the middle-school lessons, without the connections to the underlying math, the math teachers would not have been able to make connections in their regular math classrooms. Second, we wanted to introduce activities that the students would find interesting and motivational to support further study. At the middle-school level, the students were very excited at the idea of generating digital videos. At the graduate-level, the students were very interested to learn how to train neural network models. Third, we wanted our activities to be fun, and exploratory and to allow the students to experiment. At the middle-school level, the students experimented with algebraic equations in the number guessing game. Later on, they had fun exploring how image representations were used to generate different videos. At the graduate-level, the students would get to study the convergence of their neural networks models through the training process in the final projects.

Middle-School Mathematics and Computer Programming

We summarize our introduction to coding using Mathematics in Table 1. The table summarizes elements of Level 1 of the AOLME curriculum. In AOLME, the students worked collaboratively in small groups. Each group was led by an undergraduate facilitator and a middle-school student co-facilitator. The goal of the curriculum was to introduce the students to coding by building their understanding based on middle-school mathematics. The students worked in Python on the Raspberry Pi.

The first computer programming assignment was based on the number guessing game. The number guessing game allowed us to follow our guiding principles to design activities based on the underlying math, to make it interesting, and fun, exploratory, allowing the students to experiment. The students are asked to memorize an integer between 1 and 10. They then apply basic linear operations to their number (e.g., multiply and add), and then provide the computer with the result. The computer then guesses their number by using the inverse operations. To understand the code, the students need to review variables, basic algebraic operations, and linear equations. Here, we note that the use of algebra provided an entry point into coding. It enabled the students to understand variables through Algebra.

The students also learned about different number representations during middle-school. This mathematical background allowed us to introduce binary numbers and hexadecimals and make the connections to their mathematics lessons. Similarly, coordinate systems served as an entry point to NumPy arrays and array indexing. Different shapes were constructed by filling rectangular shapes with different colors.

Initially, the students thought about their video characters as continuous shapes. They quickly discovered the need to approximate their characters using rectangular color regions. They then worked on putting together their videos as characters moving through the videos (see Table 1).

Understanding the Convergence of Machine Learning Algorithms Using Real Analysis

The successful integration of middle-school mathematics with computer programming motivated the introduction of more advanced mathematics for understanding machine learning models. The focus of this effort focused on establishing convergence during the training of machine learning models. We begin with a simple model for training machine learning models. We then proceed to apply fundamental theorems from Real Analysis to establish convergence.

We summarize the training process in the following pseudocode:

$i \leftarrow 1$
$j \leftarrow 0$
$BestEpoch \leftarrow 1$
$BestLoss \leftarrow \infty$
$Patience \leftarrow smallNumber$
for $epoch = 1$ **to** $MaxEpochs$ **do**
    Train weights W_i for model F_i
    Compute training loss T_i
    Compute validation loss L_i

    if $L_i < BestLoss$ **then**
        $j \leftarrow j + 1$
        $MinLosses_j \leftarrow L_i$
        $BestLoss \leftarrow L_i$
        $BestEpoch \leftarrow i$
        Save model f_i
    end if

    if $epoch > BestEpoch + Patience$ **then**
        Stop training
    end if
    $i \leftarrow i + 1$
end for

Table 1 The integration of computer programming and middle-school mathematics during Level 1 of the AOLME curriculum

Mathematics	Computer Programming
Algebraic operations and their inverses, variables, linear equations	Number guessing game with linear equations
Binary, decimal, hexadecimal number systems and conversions between them	Digital color pixels using hexadecimals, 3-tuples
Coordinate systems	NumPy Arrays, working with rectangular regions in Python
Coordinate plane grid, shapes using rectangles, hexadecimals	Digital color image representations using NumPy arrays and hexadecimals
Approximate continuous-space shapes using digital rectangles	Design game characters using color image representations
Motion, 2D+time	Design character movements, video frames, Python lists, frames per second

The algorithm generates several sequences of real numbers and model functions. More specifically, we generate the following sequence of loss values:

Training loss values: $T_1, T_2, T_3, \ldots$.
Validation loss values: $L_1, L_2, L_3, \ldots$.
Minimum loss values: $MinLosses_1, MinLosses_2, MinLosses_3, \ldots$.

Here, we note that the $MinLosses_i$ sequence is a *subsequence* of the validation losses L_i.

We next describe the process of *early stopping*. The algorithm saves the current optimal model that is associated with a minimum validation loss. The expectation is that the validation loss will be reduced with every epoch. We use the *BestEpoch* to keep track of the epoch that produced the minimum validation loss. Then, if the current epoch is more than the *Patience* number of epochs, we stop the training. In other words, we stop waiting for the validation loss to be reduced any further.

We further demonstrate the early stopping process in Fig. 1. From Fig. 1, we can see that the training loss is significantly reduced with the number of epochs. On the left plot of Fig. 1, we can see a gap between the validation loss and the training loss. Here, a large validation loss indicates that the model does not generalize very well. To address the issue, a standard approach is to try to use data augmentation to increase the training set. As demonstrated in [23] (see Figs. 5.12 and 5.13), data augmentation may help bring down the validation loss to the training loss as shown in the right plot of Fig. 1.

We demonstrate early stopping using sequences in Fig. 2. In Fig. 2, the red dots are used to mark the sequence of $MinLosses_1, MinLosses_2, MinLosses_3, \ldots$. The effect of the *Patience* value is also shown in the Figure. A small value will terminate execution early. We will return

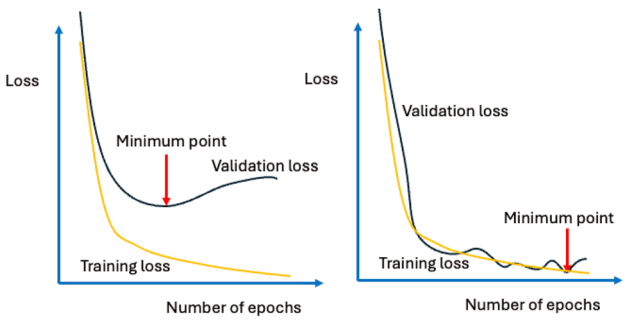

Fig. 1 Early stopping examples. **Left:** High validation loss with gap from training loss indicates bad generalization of the machine learning model. **Right:** Low validation loss that follows training loss indicates good training with data augmentation (see text)

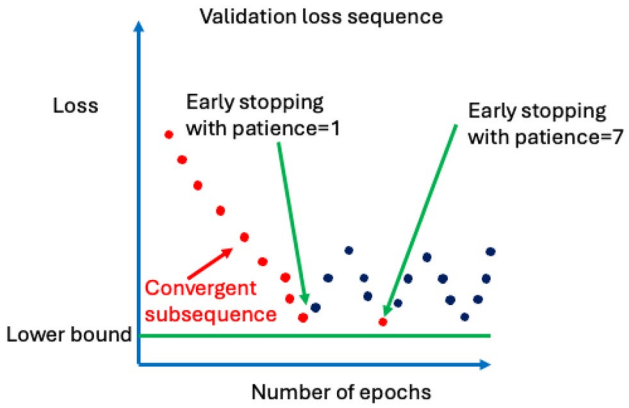

Fig. 2 Convergence of the minimum validation loss. In the example, we assume that the loss function has a lower-bound

to this figure to examine convergence after we introduce some basic terms from Real Analysis.

Let us return to the left plot of Fig. 1. When the validation loss function remains above an earlier minimum value, then early stopping will terminate training. In what follows, we will examine the case when the validation loss function keeps decreasing. This case corresponds to the right plot of Fig. 1 and Fig. 2. In this case, we need to borrow results from Real Analysis to establish convergence.

We are also concerned with the convergence of a sequence of neural network functions:

$$F_1, F_2, F_3, \ldots.$$

where each F_n has a domain in $\mathbf{R}^p$ and a range in $\mathbf{R}^q$. In other words, we allow our neural network functions to map input vectors to output vectors. For our models, we assume that the inputs consist of the weights and input signals. We would like to understand when the sequence F_n converges to F: $F_n \to F$.

In summary, when there is no observed minimum critical point in our validation loss sequence, we ask the following questions:

Q1. What conditions can we impose to guarantee the convergence of the validation loss function?

Q2. What conditions should we impose to guarantee the convergence of large weight vectors?

Q3. What conditions do we need to impose on neural network model functions to guarantee convergence?

Q4. What conditions do we need to impose on neural network model functions to guarantee convergence for a layer with an infinitely large number of connections?

Q5. What conditions do we need to impose on the input images (or videos) and neural network models so as to achieve uniform continuity?

Our questions are motivated by the need to establish convergence properties for large neural network models. We will tackle each question in detail in the next subsections.

Convergence of Sequences and Vectors

We begin with the study of the convergence of a sequence of real numbers. A sequence of numbers may either **converge** or **diverge** (e.g. page 128 in [24]). We say that a sequence converges when it gets arbitrarily close to its limit as $n \to \infty$. Otherwise, the sequence is divergent.

We demonstrate the definition in Fig. 3. On the left, we can see that the loss function oscillates between two values. It clearly does not get arbitrarily close to a limit. The sequence is divergent. On the right, we can see that after a while, the sequence gets very close to it's minimal value. The loss sequence depicted on the right image is convergent.

We can now provide the formal mathematical definition. Let L be the limit. Then the sequence:

$$L_1, L_2, L_3, \ldots$$

converges to L, provided that for any desired $\epsilon > 0$, we can find an $N(\epsilon)$, beyond which, we can get ϵ close to L as given by:

$$|L_{N+1} - L| < \epsilon, |L_{N+2} - L| < \epsilon, |L_{N+3} - L| < \epsilon, \ldots .$$

In other words, if we wait long enough, we get very close to our limit. Going back to Fig. 3, oscillations take us away from any limit because we want to be able to push ϵ to very small values. On the other hand, in the right image, it is clear that we can drive ϵ to arbitrarily small values and we are still close to our limit.

We can also define convergence for vectors (e.g., weight vectors). In this case, we use the standard Euclidean norm to define the length of a vector:

$$||v|| = [v_1^2 + v_2^2 + \cdots + v_m^2]^{1/2}.$$

Similarly, we measure the distance between vectors using:

$$d(w, v) = [(w_1 - v_1)^2 + (w_2 - v_2)^2 + \cdots + (w_m - v_m)^2]^{1/2}.$$

As for real numbers, for vectors, a sequence is defined to be convergent if we can get arbitrarily close to its limit as $n \to \infty$ and measured by $d(.,.)$. We will use this type of convergence for the weight parameters.

Convergence for a Decreasing Validation Loss Sequence

We summarize the basic theorems from Real Analysis in Table 2 (see [24–26]). Here, we note that the most basic requirement is for the loss functions to be bounded. We note that the majority of loss functions are bounded below by zero. When such a bound exists, even for infinite sequences, then the best possible achievable minimum value is given by the **greatest lower bound** (theorem 1). *The greatest lower bound provides the optimal validation loss.*

For a bounded loss function, the convergence of the early stopping algorithm is guaranteed by Theorem 4. Returning to Fig. 2, we can see that the red points representing $MinLosses_i$ correspond to a convergent subsequence that will achieve the optimal validation loss for infinite patience.

Theorem 7 makes it clear that convergence requires beyond a certain epoch, all validation losses will get infinitely close to each other. Here, the emphasis is on *all losses*. We note that the standard practice of examining $|L_{i+1} - L_i| < \epsilon$ to terminate an algorithm is not sufficient. Instead, we require that $|L_n - L_m| < \epsilon$ for some sufficiently large N and $n, m > N$. The theorem makes it clear that oscillating sequences are not convergent unless the amplitude of

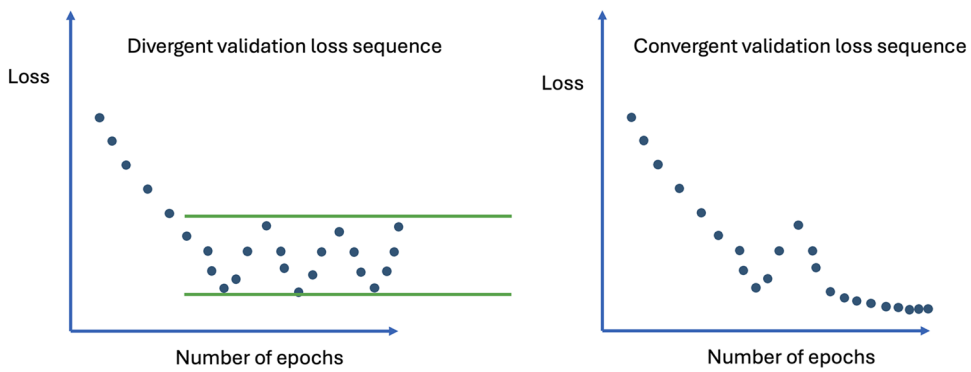

Fig. 3 Every generated loss function is classified as divergent (left) or convergent (right)

Table 2 Mathematical sequence theorem implications for loss function minimization

Real Analysis	Machine Learning
Theorem 1. Every non-empty set that is bounded below will possess a greatest lower bound	Minimization of a loss function bounded below (e.g. zero) will possess a greatest lower bound
Theorem 2. Every real, bounded, infinite set possesses at least one limit point	After a large number of iterations, the minimization of a bounded loss function will make the algorithm produce a limit point
Theorem 3. Every convergent sequence is bounded	Loss function convergence implies it remains bounded during training
Theorem 4. A monotone decreasing sequence that is bounded below will converge to a minimum	The $MinLosses_i$ sequence is clearly a monotone decreasing sequence. If the loss function is also bounded below, then it will converge to a minimum value
Theorem 5. Every sequence has a monotone subsequence	We can always select a decreasing or an increasing subsequence of iterations from an optimization algorithm
Theorem 6. Every bounded sequence has a convergent subsequence	Optimization of bounded loss functions will always converge
Theorem 7. A sequence converges if and only if it is Cauchy	An algorithm converges if and only if all iterations beyond a certain number get close to each other

the oscillation keeps decreasing to zero. Here, we note that reducing the step size will hopefully reduce the oscillation magnitude of the validation loss. If the oscillation magnitude does not tend to zero, we are not converging.

In order to guarantee convergence, without knowing convergence limit, we turn to a simple ratio test. Let the change in validation loss be: $\Delta L_i = |L_i - L_{i-1}|$. Then, based on the triangular inequality, we have that:

$$|L_i| \leq |L_N| + \sum_{j=N+1}^{j=i} \Delta L_j, \quad \text{for} \quad N < i.$$

From the standard ratio test, we can guarantee convergence of L_i provided that $\Delta L_i / \Delta L_{i-1} < 1$ as $n \to \infty$. In practice, we want to develop a metric that the sequence satisfies the convergence criterion for a minimum number of epochs *MinConvCount*. The following code implements the idea:

```
ConvCount ← 0
while ConvCount < MinConvCount do
    ...
    ΔL_i ← |L_i − L_{i−1}|
    r ← ΔL_i / ΔL_{i−1}
    if r < 1 then
        ConvCount ← ConvCount + 1
    else
        ConvCount ← 0
    end if
end while
```

Here, we note that convergence is guaranteed if $ConvCount, MinConvCount \to \infty$.

Overall, convergence may not be associated with achieving an actual minimum. We can get convergence because the loss function is bounded (theorem 2). As long as we are are reducing the loss function, that is also bounded below, we will converge to some minimal value (theorem 4). On the other hand, Real Analysis also makes it clear that we may be able to converge to a minimum by selecting a decreasing subsequence from a bounded function (theorem 5 + theorem 4). Collectively, theorems 1 to 7 provide great insight into the behavior of an infinite sequence of validation loss function values.

Series Convergence and Absolute Summability

In Neural network layers with lots of connections, we are interested in adding up the contributions from each connection. For this to work, we require absolute summability as given by:

$$\sum_k |a_k| < \infty. \tag{1}$$

We cannot relax this requirement. Here, we note a celebrated result by Riemann that showed that if the sum is conditionally convergent but does not satisfy (1), then we can rearrange the terms in the sum to add up to any number (e.g., see Chapter 1 in [27]).

Guaranteeing the Convergence of the Weight Vector Sequence

We now turn to the weight vector sequence. During training, the weight vectors can grow unbounded. This is the well-known problem of *exploding gradients* used to update the weights. We will now derive an approach that avoids this problem while also guaranteeing the convergence of the weight vector sequence.

In Table 2, theorem 6 guarantees that the validation loss will have a convergent subsequence as long as it remains

bounded. The Bolzano-Weierstrass theorem extends the result to vectors. The theorem states that every bounded sequence in $\mathbf{R}^m$ has a convergent subsequence (e.g., page 53 [28]). We will use the Bolzano-Weierstrass theorem to guarantee the convergence of the weight sequence.

Let n represent the number of weight components and M represent the total sum of all the absolute values of all components. We restrict the weights to a bounded set using:

$$W_i = \begin{cases} w_i & \text{if } |w_i| \leq \frac{M}{n} \\ \frac{M}{n} & \text{otherwise.} \end{cases} \tag{2}$$

We then have that:

$$\sum_i |W_i| < \frac{nM}{n} = M < \infty.$$

Given the fact that the weights are bounded, the Bolzano-Weierstrass theorem guarantees that the sequence $W_1, W_2, \ldots$ will contain a convergent subsequence.

Convergence of Neural Network Models

In this section, we study the convergence of neural network functions. We will use one-dimensional plots to introduce the concepts and expand our discussion to vector functions.

We define the **pointwise** convergence of a sequence of functions over a set S by requiring convergence of the values that it generates (e.g., page 261 in [24]):

$$f(x) = \lim_{n\to\infty} f_n(x), \quad x \in S.$$

A sequence of functions $f_1, f_2, f_3, \ldots$ *uniformly converges* to f provided that for any given $\epsilon > 0$, we have that:

$$f(x) - \epsilon < f_n(x) < f(x) + \epsilon,$$

for some $n \geq K$ and for all $x \in S$. Unlike pointwise convergence, the key requirement for uniform convergence is that ϵ remains constant for all values of x. We demonstrate the definition in Fig. 4. In Fig. 4, we can see that f_n lies between $f(x) - \epsilon$ and $f(x) + \epsilon$.

For neural network functions, **uniform convergence** requires that for any $\epsilon > 0$, we can find an N such that (page 170 in [28]):

$$||F(X) - F_n(X)|| < \epsilon, \quad n > N, x \in S.$$

Suppose that F_n maps elements in D to $\mathbf{R}^m$. We have the following (page 173 in [28]):

1. F_n converges *pointwise* to some function F if and only if $F_n(X)$ is a Cauchy sequence for each $X \in D$.
2. F_n converges *uniformly* to some function F if and only if

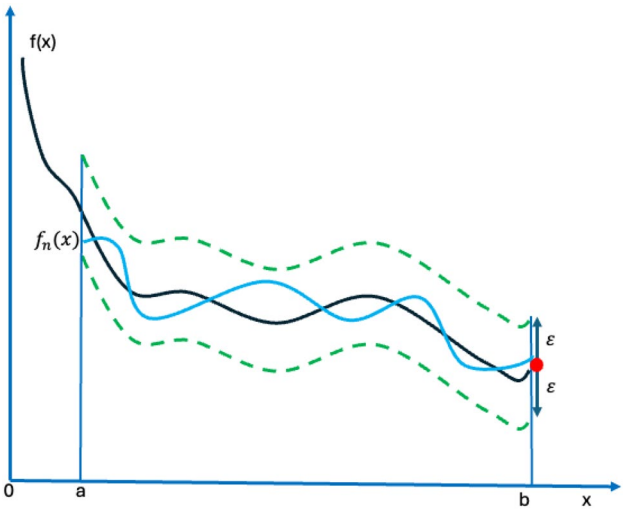

Fig. 4 Uniform convergence for a one-dimensional function

$$\lim_{k,n\to\infty} \sup_{X\in D} ||F_k(X) - F_n(X)|| = 0.$$

As before, testing for Cauchy sequences is computationally expensive. Here, we consider a much simpler approach.

Recall that we know how to guarantee the generation of a converging subsequence by bounding the weight sequence as described in Sect. 3.7. Suppose that W_n represents a converging subsequence. Furthermore, suppose that W_* represents the limit of the subsequence. We write $W_n \to W_*$. Furthermore, let X_0 be any given input signal (e.g., an image or a video). To prove convergence we only require that the neural-network models are **continuous**. In this case, we converge as given by (e.g., see continuity discussion on page 15 of [27]):

$$\lim_{n\to\infty} F_n(X_0) = \lim_{n\to\infty} F(W_n, X_0) \tag{3}$$

$$= F\left(\lim_{n\to\infty} W_n, X_0\right) \tag{4}$$

$$= F(W_*, X_0). \tag{5}$$

We will extend this result even further in section 3.10.

Convergence for a Large Number of Connections

Given the development of large models, we are also interested in the case when a layer includes a large number of connections. In this case, we consider the requirement for convergence for the infinite series given by:

$$\sum_{i=1}^{\infty} w_i f_i(x).$$

In this case, suppose that

$$|w_i f_i(x)| < M_i < \infty.$$

Then by the Weirstrass M-test (page 223 in [29], page 174 in [28]), we have that:

$$\sum_{i=1}^{\infty} w_i f_i(x) \text{ converges if and only if } \sum_{i=1}^{\infty} M_i < \infty.$$

Here, we note that a bounded sum of positive numbers will only converge to a limit.

To satisfy the requirement for convergence we require that the activation functions $f_1, f_2, f_3, \ldots$ remain bounded. Thus, we could use sigmoid or tanh for the activation function. On the other hand, since there are no imposed bounds on ReLU, we should avoid using ReLU activation functions for this layer. Then, as before we clip the weights at M/n where n represents the number of connections as given in (2). In this case, given that the activation functions are bounded above by 1, we have that:

$$\sum_{i=1}^{n} w_i f_i(x) \leq \sum_{i=1}^{\infty} |w_i f_i(x)| \leq n \cdot \frac{M}{n} \cdot 1 = M < \infty.$$

Thus, it is clear that under these circumstances we achieve convergence with a limited number of connections.

Uniformly Continuous Neural Network Models

In this section, we address the question of guaranteeing uniform continuity on all of the input images. As we shall see, uniform continuity prevents abrupt changes in the outputs of neural network functions based on small changes in the inputs. Thus, we can think of uniform continuity as a form of stability guarantees on our neural networks.

We define **uniform continuity** over $S \subset \mathbf{R}^m$ by requiring that for any $\epsilon > 0$ we can find a δ that *is not a function of* v such that (see page 109 in [28]):

$$|f(v_1) - f(v_2)| < \epsilon \quad \text{whenever} \quad ||v_1 - v_2|| < \delta.$$

For one-dimensional functions, the basic idea is demonstrated in Fig. 5. Given an $\epsilon > 0$, we define a rectangle of size 2ϵ. We then get to choose the width of the rectangle 2δ so that the function will remain inside the rectangle at all points $x \in S$. In the example of Fig. 5, without providing a formal proof, we can see that a fixed rectangle is always possible for $a \leq x \leq b$. On the other hand, for $x < a$, as x approaches 0, the function grows to ∞. In this case, it is not possible to have uniform continuity because we cannot shrink δ sufficiently for all x. As $x \to 0$, $f(x)$ grows rapidly to ∞ and escapes any fixed rectangle (see Fig. 5).

To establish uniform continuity for our neural networks we require that the input images (or videos) remain bounded. This requirement is additional to our requirements for

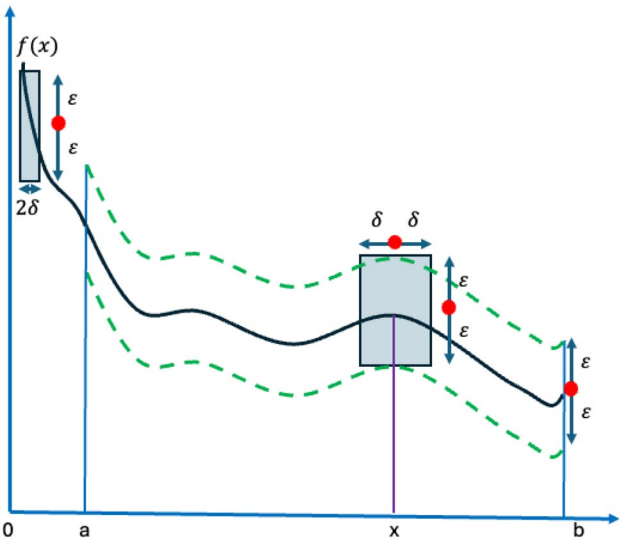

Fig. 5 Uniformly continuous function. A continuous function over a compact set is uniformly continuous

bounded weights and continuous models. We restrict our input signals between B_1 and B_2. For 8-bit images, we can set $B_1 = 0$ and $B_2 = 255$. More generally, prior to training, we bound an input signal using:

$$x_i' = \begin{cases} x_i & \text{if } B_1 \leq x_i \leq B_2, \\ B_1 & \text{if } B_1 > x, \\ B_2 & \text{otherwise.} \end{cases} \tag{6}$$

It is important to note that the combination of bounding the input weights (e.g., through (2)) and (6) restricts the domain of F_n to be over a **compact set**. Here, note that a compact set is both closed (containing all of its accumulation points) and bounded. Most importantly, a continuous function defined over a compact set is **uniformly continuous** (see theorem 3, page 16 in [27]). Thus, based on our restrictions, $F(W_*, X)$ given in (5) is uniformly continuous over the inputs and weights.

Results from Student Interviews

In this section, we summarize the student reactions to our efforts. We will begin with the student reactions to our middle-school program. For our efforts to introduce Real Analysis in Machine Learning optimization, we summarize the impact we had on the students who took the Numerical Optimization course.

We present an example video produced by middle-school students in Fig. 6. The students thoroughly enjoyed working on their projects and were very excited to get them to work.

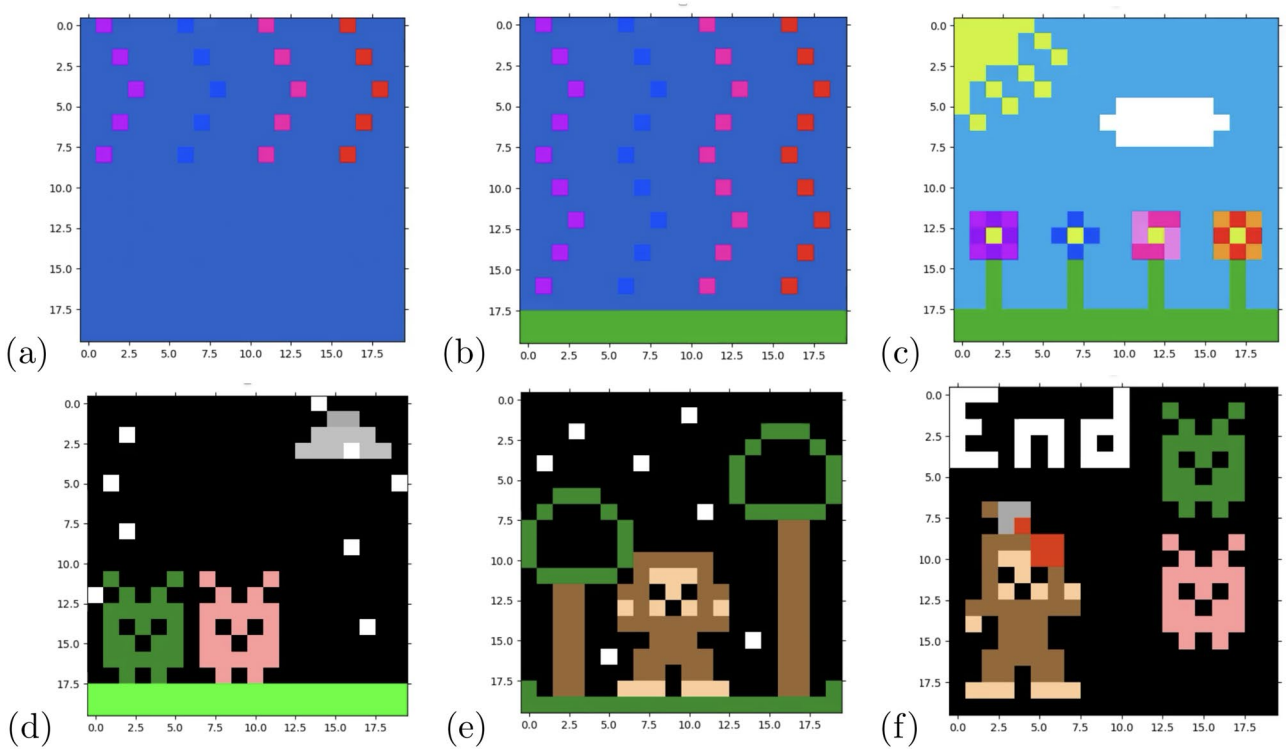

Fig. 6 Some video frames produced by middle-school students using hexadecimal values and NumPy arrays

We summarize the students' reactions to our efforts to build coding based on the underlying mathematics.

After conducting interviews with middle school students and using thematic analysis to determine common themes, it became clear that students' discoveries on the connections between mathematics and computer programming were one of the most prominent themes as was also presented in [4, 5]. As a result of learning the connections between mathematics and computer programming, students reported a rise in their enthusiasm for both subjects [4, 5]. For example, as Jesús, a middle school student who had taken on the co-facilitator role stated:

> They (Computer Programming and Mathematics) are connected. I just like computer programming better.... It's.. loops is like multiplication you could say. Instead of adding a number by itself again and again, you can just multiply it by how many times you wanted to add it. And then you could do that ...we use the loops to do stuff multiple times to get it done faster and to use less blocks, as we are using blocks of codes.

What is evident from this quote is how the student was able to make sense of how loops worked by using mathematics to make the algorithms more efficient. Students also explained their enjoyment of Mathematics and Computer Programming are related. In administering a pre, during, and post implementation questionnaire of the integrated curriculum with the undergraduate student facilitators and conducting an interview with them, a theme that emerged on the integration of computer programming and mathematics was related to the content. Several facilitators mentioned that the middle school student with whom they worked progressively exhibited a deeper understanding of the curriculum's core concepts and the crucial connections between mathematics and computer programming. The approach detailed in this research also contributed to the long-term development of the students' interest in STEM practices and their identity formation [6]. We include two selected quotes that reflect this finding:

> I feel like whenever we did the Guessing Game, that's when they like understood that if you do like to undo a number, like if you multiplied it, and then subtracted it, to divide then add, in whichever order, then they understand the order of operations with the math and how to reverse them. (Issac - Interview May 15, 2018)

> I think with like binary and the hexadecimal and binary conversions, and then getting to apply that to colors, the RGB, hex, that setup. I think that helped them make the connection that math gives you colors. (Shelby - Interview May 15, 2018)

The material on Real Analysis was introduced in the graduate course on optimization and student researchers. The

students were asked to provide short answer questions on the material to make sure they understood the applications of the theorems. The larger impact of the approach occurred when the students were working on training using large datasets for their final projects. Ultimately, motivated by the strong convergence results, students waited longer for convergence and they studied the gaps between the training and validation loss sequences. The deeper results on uniformly continuous neural network models were not tested in a classroom setting.

Conclusion

The paper summarizes the advantages of teaching coding in an integrated curriculum that builds understanding based on the relevant mathematics. The paper presented two examples. First, at the middle-school level, the students generate digital videos by manipulating NumPy arrays while building their understanding based on variables, linear equations, number representations, and coordinate systems. Second, at the University level, we provide an example of the use of Real Analysis to minimize the validation loss, establish the convergence of neural network model functions, and establish conditions that guarantee that the neural network functions are uniformly continuous. Our two examples follow our guiding principles that led us to design activities that build understanding based on the underlying mathematics, while being interesting, fun, and supportive of exploratory analysis and further experimentation.

Acknowledgements Some of the material is based upon work supported by the National Science Foundation under Grant No. 1949230 and Grant No. 1613637. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Data availability The data is not publicly available. However, interested researchers can contact the authors to gain access to the data provided they can provide a legitimate reason and undergo the necessary training to access sensitive information.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no Conflict of interest.

References

1. Sentance S, Barendsen E, Howard NR, Schulte C. Computer Science Education. New York: Bloomsbury Publishing; 2023.
2. Hurt T, Greenwald E, Allan S, Cannady MA, Krakowski A, Brodsky L, Collins MA, Montgomery R, Dorph R. The computational thinking for science (ct-s) framework: Operationalizing ct-s for k-12 science education researchers and educators. International Journal of STEM Education. 2023;10(1).
3. Lehman E, Leighton FT, Meyer AR. Mathematics for Computer Science. Cambridge, Massachusetts: Massachusetts Institute of Technology; 2010.
4. Lee HH, Celedón-Pattichis S, Pattichis MS, Johnson AR, Cantú E, Tovar I, López leiva CA. Knowing and enjoying: Expanding Latinx students' experiences with an integrated computer programming and mathematics curriculum. Paper presented at the Annual Meeting of the American Educational Research Association, Chicago, Illinois; 2023.
5. Lee HH, Celedón-Pattichis S, Pattichis MS, LópezLeiva CA, Cantú E. Integrating computer science and mathematics for enhanced learning using object-oriented programming and robotics: Mixed methods. In: Proceedings of the Annual Meeting of the American Educational Research Association (AERA 2024), Philadelphia, PA, USA; 2024.
6. Lee HH, Celedón-Pattichis S, LópezLeiva CA, Pattichis MS, Song Y. Emergent interests to well-developed interests in stem. In: Proceedings of the 18th International Conference of the Learning Sciences - ICLS 2024, New York, USA; 2024. International Society of the Learning Sciences
7. LópezLeiva CA, Noriega G, Celedón-Pattichis S, Pattichis MS. From students to cofacilitators: Latinx students' experiences in mathematics and computer programming. Teachers College Record. 2022;124(5):146–65.
8. Pattichis R, Pattichis MS. Understanding neural network systems for image analysis using vector spaces and inverse maps. In: Proceedings of the 2024 IEEE Southwest Symposium on Image Analysis and Interpretation; 2024.
9. Feurzeig W, Papert SA, Lawler B. Programming-languages as a conceptual framework for teaching mathematics. Interactive Learning Environments. 2011;19(5):487–501.
10. Forsström SE, Kaufmann OT. A literature review exploring the use of programming in mathematics education. Int J Learn Teaching Educ Res. 2018;17(12):18–32.
11. Kaufmann OT, Stenseth B. Programming in mathematics education. Int J Math Educ Sci Technol. 2021;52(7):1029–48.
12. Martin-Löf P. Constructive mathematics and computer programming. Philosophical Trans R Soc Lond Ser A Math Phys Sci. 1984;312(1522):501–18.
13. McCoy LP, Dodl NR. Computer programming experience and mathematical problem solving. J Res Comput Educ. 1989;22(1):14–25.
14. Milner SD. The effects of teaching computer programming on performance in mathematics. PhD thesis, ProQuest Information & Learning; 1973.
15. Psycharis S, Kallia M. The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. Inst Sci. 2017;45(5):583–602.
16. Rich PJ, Bly N, Leatham KR. Beyond cognitive increase: Investigating the influence of computer programming on perception and application of mathematical skills. J Comput Math Sci Teaching. 2014;33(1):103–28.
17. Goldenberg EP, Carter CJ. Programming as a language for young children to express and explore mathematics in school. Br J Educ Technol. 2021;52(3):969–85. https://doi.org/10.1111/bjet.13080.
18. Benton L, Hoyles C, Kalas I, Noss R. Bridging primary programming and mathematics: some findings of design research in England. Digital Exp Math Educ. 2017;3:115–38.
19. Solin P, Roanes-Lozano E. Using computer programming as an effective complement to mathematics education: Experimenting with the standards for mathematics practice in a multidisciplinary environment for teaching and learning with technology in

the 21st century. International Journal for Technology in Mathematics Education. 2020;27(3).
20. Wilensky UJ. Connected mathematics: building concrete relationships with mathematical knowledge. PhD thesis, Massachusetts Institute of Technology; 1993.
21. Sangwin CJ, O'Toole C. Computer programming in the uk undergraduate mathematics curriculum. Int J Math Educ Sci Technol. 2017;48(8):1133–52.
22. Olteanu C. Programming, mathematical reasoning and sensemaking. Int J Math Educ Sci Technol. 2022;53(8):2046–64.
23. Chollet F. Deep Learning with Python. Shelter Island, New York: Manning Publications Co.; 2021.
24. Lay SR. Analysis with an Introduction to Proof. 2nd ed. Englewood Cliffs, New Jersey: Prentice-Hall; 1990.
25. Cummings J. Real analysis: a Long-form Mathematics Textbook. LongFormMath.com: CreateSpace Independent Publishing Platform; 2019.
26. Knopp K. Infinite Series and Sequences. Mineola, New York: Dover; 1956.
27. Cheney EW. Analysis for Applied Mathematics. Berlin: Springer; 2001.
28. Hoffman K. Analysis in Euclidean Space. Mineola, New York: Dover; 2019.
29. Apostol TM. Mathematical Analysis. 2nd ed. Reading, Massachusetts: Addison-Wesley; 1974.