Received: 17 October 2020 | Revised: 8 November 2021 | Accepted: 16 November 2021

DOI: 10.33012/navi.506

REGULAR PAPERS

Designing Low-Correlation GPS Spreading Codes with a Natural Evolution Strategy Machine-Learning Algorithm

Tara Yasmin Mina | **Grace Xingxin Gao**

Stanford University

Correspondence
Grace Xingxin Gao, Stanford University
Email: gracegao@stanford.edu
+1 650 725 3489

Abstract

With the birth of the next-generation GPS III constellation and the upcoming launch of the Navigation Technology Satellite-3 (NTS-3) testing platform to explore future technologies for GPS, we are indeed entering a new era of satellite navigation. Correspondingly, it is time to revisit the design methods of the GPS spreading code families. In this work, we develop a natural evolution strategy (NES) machine-learning algorithm with a Gaussian proposal distribution which constructs high-quality families of spreading code sequences. We minimize the maximum between the mean-squared auto-correlation and the mean-squared cross-correlation and demonstrate the ability of our algorithm to achieve better performance than well-chosen families of equal-length Gold codes and Weil codes, for sequences of up to length-1023 and length-1031 bits and family sizes of up to 31 codes. Furthermore, we compare our algorithm with an analogous genetic algorithm implementation assigned the same code evaluation metric. To the best of the authors' knowledge, this is the first work to explore using a machine-learning approach for designing navigation spreading code sequences.

Keywords

GPS spreading code design, machine learning, memory codes, natural evolution strategy

1 | INTRODUCTION

On January 13 of 2020, the US Air Force 2nd Space Operations Squadron (2 SOPS) issued a statement that the first GPS III satellite was marked healthy and available for use (Cozzens, 2020). This announcement officially marked the birth of the next-generation GPS constellation. In addition to broadcasting the new L1C signal, the modernized constellation was distinguished by its reprogrammable payload which allows it to evolve with new technologies and changing mission needs.

Furthermore, with the upcoming launch of the Navigation Technology Satellite-3 (NTS-3) testing platform in 2023 (Cozzens, 2021), the United States Air Force (USAF) seeks to explore technologies which will help shape future GPS constellations (Chapman et al., 2020). The NTS-3 will demonstrate the agility of the next-generation satellite-based navigation architecture and the ability to rapidly deploy new technological advancements and capabilities via the reprogrammable

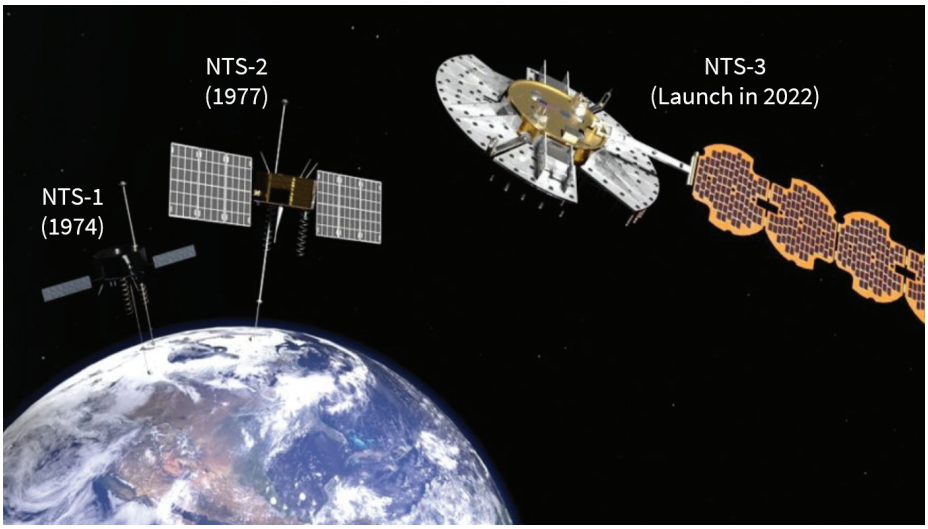


FIGURE 1: Illustrations of the satellite testing platforms used in each of three Navigation Technology Satellite (NTS) initiatives *Credit: Lt. Jacob Lutz, AFRL Space Vehicles Directorate*

nature of the upcoming GPS system. Indeed, this is the third Navigation Technology Satellite (NTS) mission, with the previous two (NTS-1 and NTS-2) developed in the 1970s in order to validate technologies, including the rubidium and cesium atomic clocks (Parkinson et al., 2010) that were integrated into the first generation of GPS satellites. Illustrations of the three NTS testing platforms are shown in Figure 1.

The NTS-3 program will further test several new technologies, including new signal designs for improved GPS security and interference mitigation (Chapman et al., 2020). According to a request-for-information announcement (Inside GNSS, 2016; United States Department of the Air Force, Air Force Materiel Command, 2016), the Air Force Research Lab (AFRL) has expressed interest in exploring modifications to all layers of the GPS signal in order to enhance positioning, navigation, and timing (PNT) resiliency and performance. As we enter a new era of satellite navigation, we believe it is time to revisit the design methods of the GPS spreading code families. Correspondingly, in this work, we leverage recent advances in machine-learning techniques to explore a new platform for learning high-quality spreading codes via a natural evolution strategy (NES) machine-learning framework.

1.1 | Advantages of Using Memory Codes for GPS

Currently, all broadcast GPS signals use algorithmically generable spreading code sequences. The legacy L1 C/A signal uses Gold codes (Gold, 1967), while the modernized L1C signal, to be broadcast on the GPS III satellites, uses Weil codes (Guohua & Quan, 2002; Rushanan, 2006), along with a seven-bit extension in order to satisfy the desired spreading code length of 10,230 bits. Having algorithmically generable sequences was an important design consideration during the initial development phase of GPS in the 1970s due to memory storage constraints, as well as the computation limitations which restricted the ability to search for better sequences. However, memory has become an increasingly inexpensive resource, allowing receivers to store complete families of spreading code sequences—though it should be acknowledged that currently, memory is still one of the main influencers of receiver chip size (van Diggelen, 2014, 2020). Furthermore, over

the past several decades, processing power has also significantly increased, which correspondingly has allowed the field of machine learning and related research advancements to flourish (Lu, 2017).

Lower bounds on the maximum periodic non-central auto- and cross-correlation value for spreading code families have been analytically derived in prior work (Stular & Tomazic, 2000), including the Welch bound (Welch, 1974), the Sarwate bound (Sarwate, 1979), and the Sidelnikov bound (Sidelnikov, 1971a, 1971b). The Welch bound is valid for any sequence with symbols of a fixed norm, including potentially complex spreading codes, while the Sidelnikov bound is a tighter lower bound for sequences with symbols that are roots-of-unity. As a result, for binary spreading codes, which fall into this roots-of-unity classification with symbols in the set $\{+1, -1\}$, the Sidelnikov bound provides a tighter lower bound than the Welch bound on the maximum periodic correlation performance.

Gold code families are known to be asymptotically optimal with respect to the Sidelnikov bound as the code length increases (Burroughs & Wilson, 1996). However, this optimality characteristic only applies for the specific sequence length and the complete set of code sequences provided by the Gold code family definition. For the length-1023 bit Gold codes, for example, this optimality characteristic only applies to the complete Gold family size, made up of over 1,000 spreading codes in total (Gold, 1967).

Indeed, the lower bounds on the maximum correlation sidepeak increase with the size of the spreading code family (Sarwate, 1979; Sidelnikov, 1971a, 1971b; Welch, 1974), suggesting that for the same sequence length, better performance could be found for a smaller code family size. Furthermore, there is debate as to whether the maximum correlation sidepeak is an ideal metric for spread spectrum system applications (Ganapathy et al., 2011a, 2011b; Pursley, 1977; Stular & Tomazic, 2001). Indeed, the multiple access interference (MAI) of a spread spectrum system (i.e., the signal noise caused by inter-signal interference within the shared channel) relates to the average correlation performance rather than the worst-case.

Spreading codes which are not algorithmically generable are commonly called *memory codes*, since these codes must be stored in memory. Memory codes are not limited to specific lengths of code sequences or specific code family sizes. Indeed, designing a nonconforming spreading code length, as was done for the GPS L2C, L5, and L1C signals, would require truncation or extension of the algorithmically generable sequences, which disturb the coveted correlation properties of these codes (Wallner et al., 2007). As a result, by expanding the design space to the set of all binary sequences, we have a greater range of possible code families and a better opportunity to find a superior set of codes; although the exponentially larger design space also greatly complicates the code design method.

1.2 | Related Prior Work

Several past works have designed memory codes using genetic algorithms (GAs), including for the E1 Open Service (OS) and the E6 Commercial Service (CS) signals of the Galileo satellite constellation (European Union, 2021; Wallner et al., 2007). *Genetic algorithms* (Holland, 1992) are optimization algorithms which mimic the process of biological evolution. In particular, GAs maintain a population of design points, while selecting and recombining high-performing candidate solutions at each iteration.

Genetic algorithms were utilized to design spreading codes for Galileo which exhibit the autocorrelation sidelobe zero property, in which the auto-correlation

is zero at a relative delay of ±1 chip (Wallner et al., 2007). GAs were also utilized to design spreading code families for improved indoor positioning applications (Ávila-Rodríguez et al., 2006). Work has also been done to define several cost parameters for selecting high quality GNSS spreading codes (Soualle et al., 2005; Winkel, 2011), including evaluating the mean-squared auto-correlation and cross-correlation components above the Welch bound (Welch, 1974).

Additionally, in our prior work, we designed a multi-objective GA platform for designing navigation spreading codes which improved two objectives simultaneously: the mean absolute auto-correlation and cross-correlation of the code family (Mina & Gao, 2019). Learning-based techniques have also been explored to design *error correcting codes* (ECCs; Huang et al., 2019), which are binary sequences that encode messages to improve the detection of communication transmission errors. In their work, Huang et al. explored policy gradient and advantage actor critic methods to design ECCs by optimizing the code set performance with regards to the block error rate (Huang et al., 2019).

1.3 | Objective and Key Contributions

In this work, we seek to explore using a machine-learning technique for the application of navigation spreading code design. This work is based on our recent ION GNSS+ 2020 conference paper (Mina & Gao, 2020), and to the best of our knowledge, this is the first work which explores using a machine-learning approach to design navigation spreading codes. In particular, the key contributions of our work are the following:

1. We develop a *natural evolution strategy* (NES) machine-learning algorithm which constructs high-quality families of spreading code sequences.
2. We utilize a Gaussian proposal distribution parameterized by a *generative neural network* (GNN) in order to model a search distribution over the design space of possible binary spreading code families.
3. We incorporate a baseline to reduce the variance of the NES gradient estimate and to improve the rate of learning.
4. We use a maximization evaluation metric to ensure the algorithm minimizes both the auto-correlation and cross-correlation characteristics of the spreading codes simultaneously.

With our algorithm, we demonstrate the ability to achieve low auto- and cross-correlation sidepeaks within the family of spreading codes. We further compare the correlation performance of the learned spreading codes with those of well-chosen families of equal-length Gold codes and Weil codes as well as with an analogous genetic algorithm implementation assigned the same code evaluation metric as our proposed algorithm.

1.4 | Paper Organization

The remainder of the paper is organized as follows: Section 2 provides relevant technical background for the paper; Section 3 describes our NES machine learning algorithm for the design of spreading code families; Section 4 presents our experimental validation setup and results; Section 5 concludes this paper; and Section A1 provides additional background on artificial neural networks.

2 | BACKGROUND

2.1 | Binary Sequence Representation

For mathematical convenience, we represent a binary sequence $\bar{x}^{(0,1)}$ with elements of the set {0, 1} as a sequence $\bar{x}$ with elements of the set {+1, −1} by using the following conversion:

$$\begin{aligned} \forall i : \bar{x}^{(0,1)}(i) = 0, \bar{x}(i) = +1 \\ \forall i : \bar{x}^{(0,1)}(i) = 1, \bar{x}(i) = -1 \end{aligned} \tag{1}$$

where $\bar{x}^{(0,1)}(i)$ and $\bar{x}(i)$ represent the i-th element of the two binary sequence representations. In this paper, unless otherwise specified, we represent sequences using the (+1, −1) binary representation. To denote sequences with the (0, 1) binary representation, we utilize an additional superscript of (0, 1), i.e. $\bar{x}^{(0,1)}$.

The (+1, −1) representation of the binary sequences simplifies the auto-correlation and cross-correlation computations performed on one or more binary sequences. In particular, an exclusive-OR operation (i.e., $b_1 \oplus b_2$) between two elements of the (0, 1) binary representation becomes a simple multiplication operation (i.e., $b_1 \cdot b_2$) between two elements of the (+1, −1) binary representation, as illustrated in Figure 2.

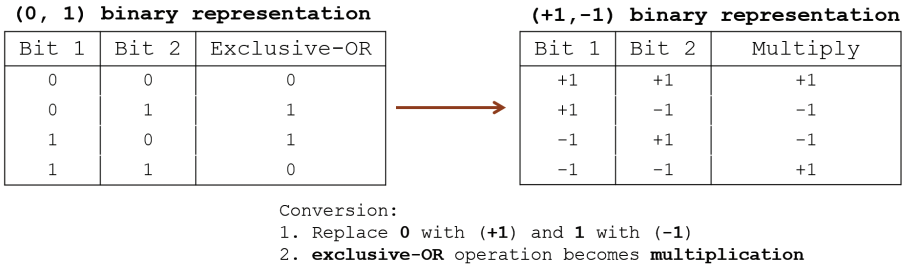


FIGURE 2: Illustration of the conversion from the (0, 1) binary sequence representation to the (+1, −1) representation used for mathematical convenience; consequently, the exclusive-OR operation between elements of the (0, 1) representation becomes a multiplication operation.

2.2 | Periodic Auto-Correlation and Cross-Correlation

Using the (+1, −1) binary sequence representation defined in Equation (1), let $\bar{x}_k$ represent the k-th binary sequence in a family of length-ℓ spreading code sequences. Let $\bar{x}_k(i) \in \{+1, -1\}$ represent the i-th element of the sequence, where $i \in \mathbb{Z} \cap [0, \ell - 1]$ and $\mathbb{Z}$ represent the set of all integer values. Because $\bar{x}_k$ is a periodic sequence, we have that $\bar{x}_k(i) = \bar{x}_k(i + \ell)$. The normalized, periodic, even auto-correlation of sequence k at a relative delay of δ bits is defined as:

$$R_k(\delta) := \frac{1}{\ell} \sum_{i=0}^{\ell-1} \bar{x}_k(i) \bar{x}_k(i - \delta) \tag{2}$$

Similarly, in Equation (3) we define the normalized, periodic, even cross-correlation between sequences k and m at a relative delay of δ bits as:

$$R_{k,m}(\delta) := \frac{1}{\ell} \sum_{i=0}^{\ell-1} \bar{x}_k(i) \bar{x}_m(i - \delta) \tag{3}$$

The even correlation is distinguished from the odd correlation, which defines the correlation in the case that a data flip occurs within the integration period of the correlators (Wallner et al., 2011; Winkel, 2011). Correspondingly, the odd correlation for sequence k would be computed by flipping the sign of the second multiplicative terms of Equation (2) and (3), i.e. $\bar{x}_k(i-\delta)$ and $\bar{x}_m(i-\delta)$ respectively, when $i < \delta$.

2.3 | Algorithmically Generable GPS Spreading Codes

Currently, all broadcast GPS signals use algorithmically generable spreading code sequences. The legacy L1 C/A GPS signal, for example, uses length-1023 Gold codes, which can be generated with two 10-bit linear feedback shift registers (LFSRs), as shown in Figure 3. Gold codes (Gold, 1967) are a class of binary spreading code families which can be generated from two *maximum-length sequences*, or *m*-sequences. Maximum-length sequences are pseudorandom sequences of length $\ell = 2^n - 1$, which are generated using an n-bit LFSR (Golomb, 2017).

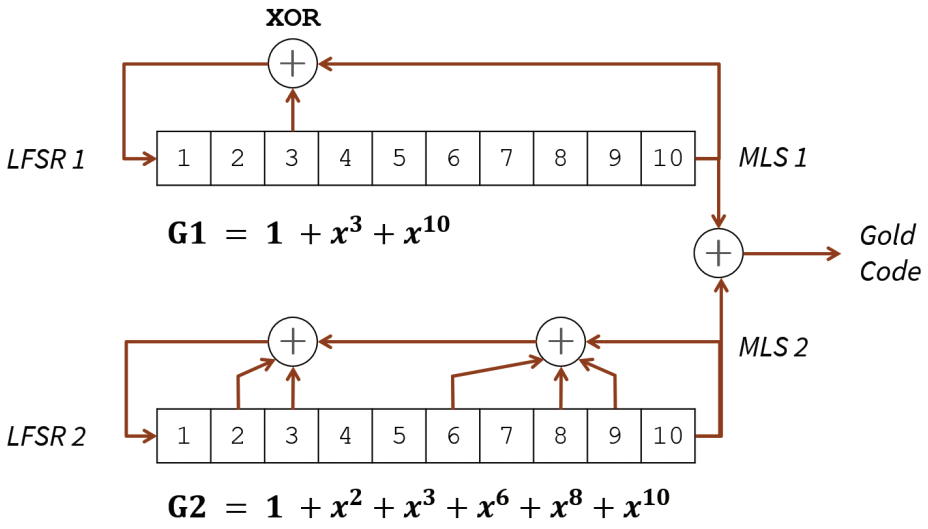


FIGURE 3 Diagram of the linear feedback shift register (LFSR) pair with their corresponding characteristic equations ($G1$ and $G2$) utilized to generate length-1023 bit Gold codes used in GPS L1 C/A (Navstar GPS Joint Program Office, 2021a); only two 10-bit LFSRs are required for the generation of the GPS L1 C/A spreading codes.

Recently developed for the modernized GPS L1C signal, Weil codes are families of spreading codes which were initially proposed by Guohua and Quan (2002) and further analyzed and proposed for the L1C signal by Rushanan (2006, 2007). A Weil code family exists for any prime number sequence length-p and is generated from the corresponding length-p Legendre sequence, constructed from the Legendre symbol (Legendre, 1808; Yan, 2002). In this work, we construct both Gold codes and Weil codes in order to compare the performance of these spreading code families utilized in GPS with the performance of the spreading codes generated from our proposed NES machine-learning algorithm.

2.4 | Natural Evolution Strategy Algorithms

Natural evolution strategies (NES; Kochenderfer & Wheeler, 2019; Rechenberg, 1973; Wierstra et al., 2014, 2008) are a subset of stochastic optimization methods

which model a probability distribution, or a *proposal distribution*, over the optimization design space. Each sample from this proposal distribution outputs a candidate design point. In the context of designing navigation spreading codes, for example, each sample of the proposal distribution would output a set of binary sequences. NES requires choosing a family of parameterized proposal distributions $\mathcal{P}_\Theta$, explicitly defined as:

$$\mathcal{P}_\Theta = \{p_\theta(x) : x \in \mathcal{X}, \theta \in \Theta\}$$

where Θ is the set of possible vector-valued parameters θ defining each distribution in the family and where $\mathcal{X}$ represents the design space over which we are optimizing.

Within the specified family of probability distributions, NES methods directly optimize the proposal distribution p_θ with respect to its parameterization θ in order to guide the search process towards higher performing regions of the design space. NES algorithms are especially useful for non-convex optimization problems with large design spaces, including combinatorial design spaces, which applies to our application of interest with the design of navigation spreading codes.

For high-dimensional design spaces, often diagonal covariance matrices are considered for the proposal distribution in order to improve computational efficiency with regards to the number of NES parameters to optimize as well as sample efficiency with regards to the number of required Monte Carlo samples to effectively estimate the search gradient during the optimization step (Schaul et al., 2011; Wierstra et al., 2014). Further variations of traditional NES methods have also been explored to handle highly multi-modal optimization landscapes by leveraging heavy-tailed distributions, including those with undefined variances, such as the Cauchy distribution (Schaul et al., 2011).

NES methods optimize the proposal distribution within the defined family $\mathcal{P}_\Theta$ by seeking to minimize the expected objective function J, defined as:

$$\begin{aligned} J(\theta) &:= \mathbb{E}_{x\sim p_\theta}\left[f(x)\right] \\ &= \int p_\theta(x) f(x) dx \end{aligned} \tag{4}$$

where x represents a particular point in the design space $\mathcal{X}$, $p_\theta(x)$ represents the probability of selecting x given the parameter θ, and $f(x)$ represents the objective function value of the design point x, which we seek to minimize. Thus, NES methods seek to find the best proposal distribution by solving the optimization problem:

$$\theta^* = \arg\min_\theta J(\theta) \tag{5}$$

To optimize the proposal distribution, the gradient of the expected objective function, i.e. $\nabla_\theta J(\theta)$, can be estimated using Monte Carlo sampling of the objective function using the current proposal distribution (Glynn, 1990; Kochenderfer & Wheeler, 2019; Mohamed et al., 2020):

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{x\sim p_\theta}\left[f(x)\right] \\ &= \int f(x) \nabla_\theta p_\theta(x) dx \\ &= \int f(x) \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} p_\theta(x) dx \qquad (6) \\ &= \int \left[f(x) \nabla_\theta \log p_\theta(x)\right] p_\theta(x) dx \qquad (7) \end{aligned}$$

$$= \mathbb{E}_{x\sim p_\theta}\left[f(x)\nabla_\theta \log p_\theta(x)\right] \tag{8}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} f(x^i)\nabla_\theta \log p_\theta(x^i) \tag{9}$$

where N represents the number of samples from the proposal distribution and x^i represents the i-th sample. Equations (6) and (7) demonstrate the so-called *log-derivative trick* which is observed in a variety of machine-learning contexts, including reinforcement learning and variational inference (Mohamed et al., 2020).

From this gradient estimate, using any first-order optimizer (e.g. stochastic gradient descent algorithm, Adam [Kingma & Ba, 2015]), NES improves its proposal distribution vector-valued parameter θ in order to minimize the expected objective function in Equation (4):

$$\begin{aligned} &\theta \leftarrow \theta - \delta\theta \\ &\delta\theta := g_{opt}\left(\nabla_\theta J(\theta)\right) \end{aligned} \tag{10}$$

where g_{opt} is the generic first-order optimizer utilized to iteratively improve the proposal distribution. As illustrated in Figure 4, this optimization process increases the probability of design points which lead to lower values of the objective function f to be minimized and decreases the probability of those which result in higher values of the objective function. Indeed, the stochasticity of the proposal distribution induces the NES algorithm to explore the high-dimensional design space and further improve its proposal distribution to output better performing design points.

3 | PROPOSED NATURAL EVOLUTION STRATEGY ALGORITHM FOR SPREADING CODE DESIGN

3.1 | High-Level Learning Framework

For the spreading code design application, the machine-learning framework utilized is represented at a high level in Figure 5. A *code generator* maintains a proposal distribution over the spreading code design space. Then, an *evaluator*

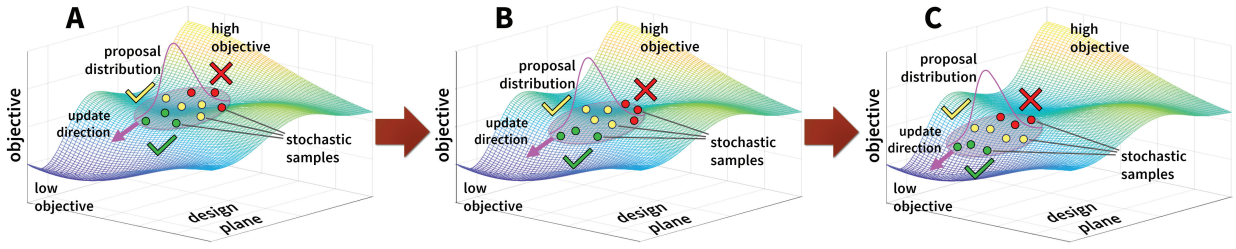


FIGURE 4 Illustration depicting how the NES algorithm optimizes the expected objective $J(\theta)$; from Instance **A** to Instance **B**, the NES optimizer updates the parameters of the proposal distribution in order to increase the likelihood of design points which lead to lower values of the objective function (indicated by the green sampled points), while decreasing the likelihood of design points which have higher objective function values (indicated by the red sampled points). This update then guides the proposal distribution down the objective function landscape in Instance **B** toward regions where the objective is lower. The optimizer again repeats this process from Instance **B** to Instance **C**, thereby further guiding the proposal distribution down the objective function landscape. Indeed, the stochasticity of the proposal distribution induces the NES algorithm to explore the design space; and the optimization step iteratively steers the search process toward better performing regions of the design space, where the objective function value is lower.

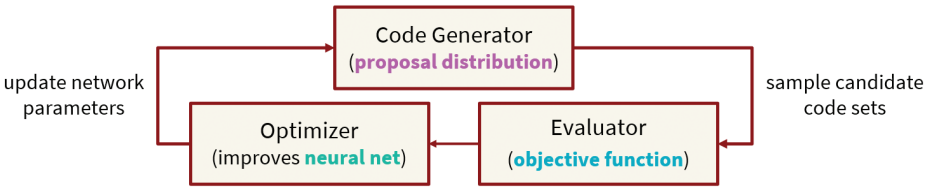


FIGURE 5 High-level learning framework utilized for spreading code generation; the *code generator* outputs a set of sampled candidate code sets which are assessed by the *evaluator*. From the sampled design points and corresponding objective function values, *the optimizer* improves the proposal distribution by updating its parameters as represented by a generative neural network.

assesses the performance of the code generator with respect to the objective function f by sampling from its proposal distribution.

The *objective function* represents an evaluation metric for the spreading code family based on the auto- and cross-correlation properties of the code set. Finally, as in Equation (9), the *optimizer* estimates the gradient of the expected objective function with respect to the parameters of the proposal distribution, as represented by a *generative neural network* (GNN). The optimizer then updates the network parameters in order to improve the proposal distribution for the code generator.

3.2 | Gaussian Proposal Distribution Representation

Using a neural network architecture, we represent a probability distribution over the design space of all possible binary spreading code families. Thus, the parameters of the proposal distribution θ correspond to the hidden layers of the network. As depicted in Figure 6, for the code generator to output a set of binary codes, it

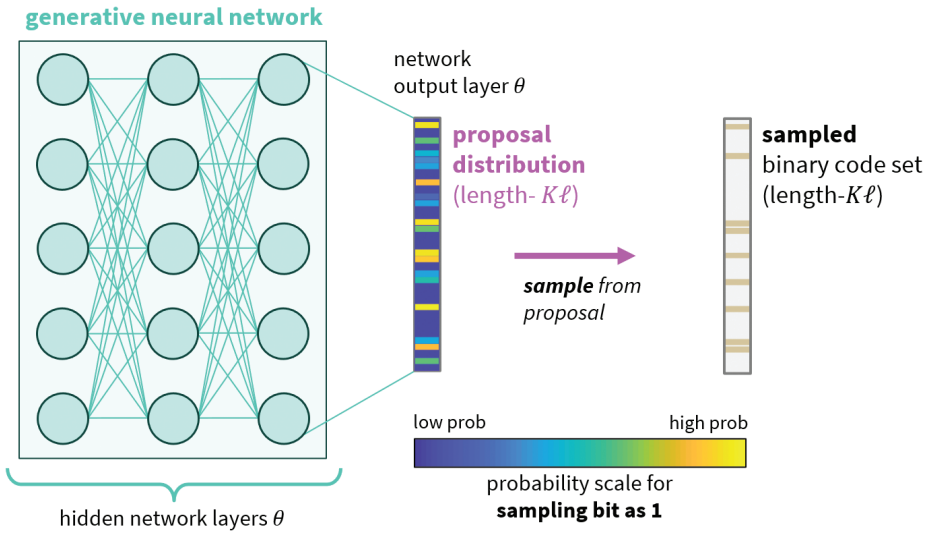


FIGURE 6 Illustration of *generative neural network* (GNN) and *proposal distribution representation*. The GNN outputs the mean vector of the multivariate Gaussian *proposal distribution* from which a sampled output code set for the complete spreading code family is generated. In particular, the sampled output code set is a binary vector of length-$K\ell$ bits, where K denotes the number of codes in the family and ℓ denotes the length of each spreading code sequence. With this representation of the proposal distribution, the network parameters correspond to the NES parameters θ, which are optimized during training.

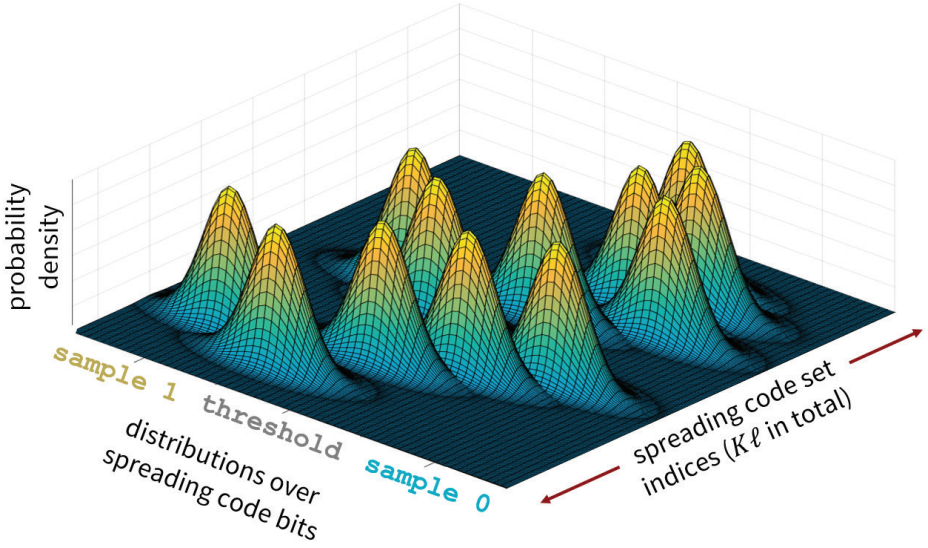


FIGURE 7 Illustration of the proposal distribution utilized for spreading code design; we model the code design proposal distribution as an uncorrelated multivariate Gaussian family, where each component corresponds to an index in the family of binary spreading codes. Correspondingly, the total length of the spreading code set is $K\ell$, where K denotes the number of codes in the family and ℓ denotes the length of each binary spreading code sequence.

must sample from this proposal distribution provided by the GNN with the currently learned network parameters θ. For a concise introduction to artificial neural networks, see Section A1.

We modeled the code design proposal distribution as an uncorrelated multivariate Gaussian family as depicted in Figure 7, in which each component corresponds to an index in the complete family of binary spreading codes. Our neural network model represents this proposal distribution by parameterizing the mean vector of the Gaussian distribution via a bounded output activation function, such as the logistic or hyperbolic tangent output activations. The variance σ^2 of each component is maintained as a constant value. We can represent this Gaussian proposal distribution in terms of the network parameters θ as:

$$p_\theta(x) = \mathcal{N}\left(\mu(\theta), \sigma^2 \mathcal{I}^{(K\ell)}\right) \tag{11}$$

where K denotes the number of codes in the family, ℓ denotes the length of each sequence in the code family, $\mathcal{I}^{(m)}$ represents the identity matrix of size $(m \times m)$ where m is a positive integer, and $\mu(\theta)$ represents the mean of the Gaussian proposal distribution, as output by the GNN with parameters θ. Note that the mean of p_θ is bounded, i.e. $\mu(\theta) \in [\mu_L, \mu_U]^{K\ell}$, due to the bounded nature of the output activation function of the network.

Due to the uncorrelated model of the multivariate Gaussian proposal distribution, each component represents a univariate Gaussian distribution over each binary value in the spreading code family. As depicted in Figure 7, to obtain a random output family of spreading codes, the code generator first samples directly from the multivariate Gaussian proposal distribution defined in Equation (11) (i.e., $x \sim p_\theta$) From this sampled vector x, the output code set is deterministically obtained by discretizing each component according to a threshold ζ defined in Equation (13), which we choose to be the midpoint of the bounded output range of

the GNN. Correspondingly, the output binary code family $\bar{x}$ can be defined in an element-wise manner as in Equation (14):

$$x_i \sim \mathcal{N}\left(\mu_i(\theta), \sigma^2\right) \tag{12}$$

$$\zeta := \frac{\mu_L + \mu_U}{2} \tag{13}$$

$$\bar{x}_i = \mathbb{1}\{x_i \geq \zeta\} \tag{14}$$

Thus, after sampling the design point x from the proposal distribution p_θ, if a component of x is above the threshold ζ, the corresponding sampled bit in the output binary code set $\bar{x}$ is a one; otherwise, the corresponding bit is a zero. Note that the design space $\mathcal{X}$, where $x \in \mathcal{X}$, is in fact a continuous design space. Indeed, we map each continuous design point x to a binary vector $\bar{x}$ according to the threshold ζ as in Equation (14). Correspondingly, we evaluate the performance of the design point x by utilizing its discretized binary vector $\bar{x}$ with the spreading code evaluation metric, as defined in Section 3.3.

3.3 | Spreading Code Evaluation Metric

In the NES framework, the evaluation metric provides feedback to the code generator in order to improve its proposal distribution. In this work, we consider the *mean-square non-central even auto-correlation* and the *mean-square even cross-correlation* as the two objectives to minimize, defined respectively as:

$$f_{AC} := \frac{1}{K\ell}\left(\sum_{k=1}^{K}\sum_{\delta=1}^{\ell-1}|R_k(\delta)|^2\right) \tag{15}$$

$$f_{CC} := \frac{1}{K_p\ell}\left(\sum_{k=1}^{K}\sum_{j=k+1}^{K}\sum_{\delta=0}^{\ell-1}|R_{k,j}(\delta)|^2\right) \tag{16}$$

where K represents the number of sequences in the output code family $\bar{x}$, ℓ represents the length of each sequence, $K_p := \binom{K}{2}$ represents the number of pairs of sequences in the code family of size K, R_k represents the auto-correlation of the k-th sequence in the output code family $\bar{x}$ as defined in Equation (2), and $R_{k,j}$ represents the cross-correlation of sequences k and j of the output code family $\bar{x}$ as defined in Equation (3).

In order to ensure the code generator reduces both objectives, we define the comprehensive objective function as the maximum of the two components from Equations (15) and (16):

$$f(\bar{x}) = \max\left(f_{AC}, f_{CC}\right) \tag{17}$$

Thus, to minimize the objective, the NES algorithm will seek to reduce both components of the objectives simultaneously, without compromising one component over the other. Indeed, modifications of the objective function in Equation (17) could be readily incorporated within this machine-learning framework. In particular, if seeking to design spreading codes with smaller auto-correlation sidelobes than cross-correlation sidelobes, one could scale f_{AC} by a corresponding factor

that is greater than one in Equation (17), which would encourage the algorithm to generate codes that reduce the auto-correlation metric in comparison to the cross-correlation metric by the corresponding user-defined factor.

Furthermore, additional components of objectives could be incorporated into the comprehensive objective function, including, for example, odd correlation performance metrics and relative Doppler frequency correlation metrics. These additional objectives could be incorporated as new arguments within the maximization function of Equation (17) with relative scaling factors if desired, or these objective could be combined via a weighted summation, as proposed in Soualle et al. (2005).

3.4 | NES Optimization

We optimize the network parameters θ, which define the proposal distribution p_θ, by following the NES optimization process described in Section 2.4. To reduce the variance in the gradient estimate, we incorporate a constant baseline b, which is subtracted from the objective function in the gradient estimate expression of Equation (9; Mohamed et al., 2020). Thus, with the baseline subtraction, the gradient estimate from Equation (9) becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\Big(f(x^i)-b\Big)\nabla_\theta \log p_\theta(x^i) \tag{18}$$

where N represents the cardinality of the batch of samples $\{x^1, x^2, \ldots, x^N\}$. For our constant baseline b, we utilize the mean return in the current batch of samples, defined as:

$$b := \frac{1}{N}\sum_{i=1}^{N} f(x^i) \tag{19}$$

Subtracting a constant value does not introduce any bias in the gradient estimate, but it can significantly reduce its variance (Mohamed et al., 2020; Wierstra et al., 2008). Indeed, by reducing the variance of the gradient estimate, we observe improved convergence and overall learning performance during training. Section 4.2 demonstrates the learning rate of the NES algorithm with and without the incorporation of this baseline from Equation (19).

4 | EXPERIMENTAL VALIDATION

4.1 | Details of Experimental Setup

We validated the ability of our NES machine-learning algorithm to devise low-correlation spreading codes and further compared its performance with that of well-chosen families of equal-length Gold codes and Weil codes. We compared the performance of our algorithm with Gold codes of length-63, 127, 511, and 1023 bits. Similarly, since Weil codes only exist for sequence lengths that correspond to a prime number length, we compared our algorithm with Weil codes of length-67, 127, 257, 521, and 1031 bits.

From our NES method, we generated sequences for family sizes of three codes up to 31 codes. We additionally compared our proposed algorithm with the best performing code family across 10, 000 sampled families of Gold codes and Weil codes. In a few of the sample runs of Gold and Weil codes, we observed a large deviation in the auto- and cross-correlation cost components which frequently led to worse

performance on the overall objective defined in Equation (17). In these instances, when possible to reduce the deviation in auto- and cross-correlation objectives, we would resample the conventional code families, leading to an improvement in the performance metric of the Gold and Weil codes. The Gold and Weil code performance results have been updated since those presented in our previous conference paper (Mina & Gao, 2020), resulting in similar or slightly worse performance of the Gold and Weil codes on the overall objective.

The details of our NES machine-learning algorithm are indicated in Table 1. We utilize a slightly smaller learning rate for learning families with sequences of over length-500 bits, since we observed more consistent performance with a reduced learning rate for these longer sequence test cases. We use the hyperbolic tangent function as our bounded output activation function and the Adam algorithm (Kingma & Ba, 2015) as the optimizer of the GNN.

Furthermore, we compared the performance of our algorithm with that of an analogous genetic algorithm which was set to optimize over the same maximization evaluation objective defined in Equation (17). The GA implementation used fitness proportionate selection (Holland, 1992), which stochastically selected a candidate solution from the population with a probability proportional to its normalized objective. Selected individuals were recombined using uniform crossover (Syswerda, 1989) which allowed for greater recombination of candidate solutions and was observed to have better performance during initial testing. We additionally incorporated elitism (Baluja & Caruana, 1995), which ensured that the genetic algorithm would continuously improve during its optimization process.

In Table 2, we describe the parameters of the analogous genetic algorithm. For the GA to be comparable with the NES method, we set its population size to be

TABLE 1
Design parameters of proposed NES machine learning method

Parameter Description	Value
Gaussian proposal variance (σ^2)	0.1
learning rate (for $\ell < 500$)	10^{-4}
learning rate (for $\ell > 500$)	$5 \cdot 10^{-5}$
hidden layer size	$2K\ell$
number hidden layers	2
output activation	tanh
network optimizer	Adam (Kingma & Ba, 2015)
batch size (N)	100
number iterations	10,000

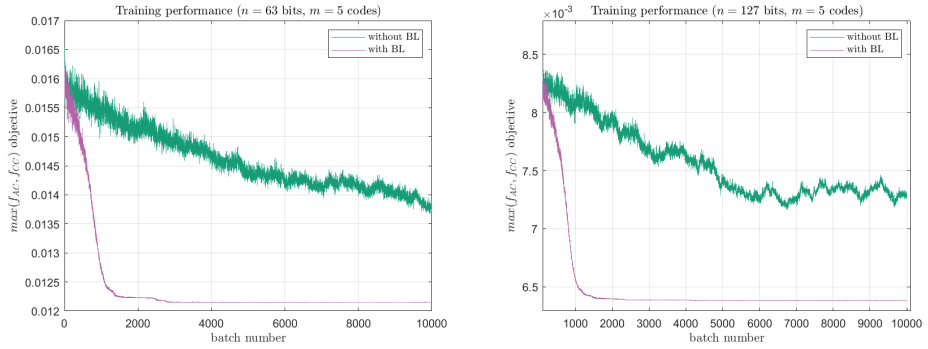
TABLE 2
Design parameters of genetic algorithm implementation

Parameter Description	Value
elite rate	0.01
mutation rate	0.005
selection method	fitness proportionate (Holland, 1992)
crossover method	uniform (Syswerda, 1989)
population size	100
number iterations	10,000

the batch size of the NES machine-learning algorithm, and we set the number of iterations for both algorithms to be the same.

4.2 | Effect of Baseline Incorporation on Learning Performance

Figure 8 demonstrates the effect of incorporating the constant mean baseline on the learning performance during training, as measured by the normalized maximum correlation objective, i.e. $\max\{f_{AC}, f_{CC}\}$, which the algorithm seeks to minimize. For both of the code length scenarios shown in Figure 8, we observed that the incorporation of the baseline, shown in violet, led to significant improvement in the learning performance during training due to the reduction of variance in the gradient estimate. In particular, the baseline incorporation allowed for more systematic learning and more consistent improvement on the correlation objective.



(a) 63 bits, 5 codes

(b) 127 bits, 5 codes

FIGURE 8 Training performance on the normalized maximum correlation objective, i.e. $\max(f_{AC}, f_{CC})$, of the NES machine-learning algorithm, with (violet) and without (green) baseline incorporation; we observe that baseline incorporation leads to faster and more consistent learning performance during training.

4.3 | Comparison with Best Performing Gold and Weil Codes

Figures 9 and 10 show the converged performance of our NES algorithm after training, comparing it with the best-performing sets of Gold and Weil code families of equal length. We plotted the final performance in terms of the maximization evaluation objective defined in Equation (17), i.e. $\max\{f_{AC}, f_{CC}\}$, as a function of the code family size. From these results, we empirically observed how designing spreading code sequences for larger family sizes generally leads to worse performance on the optimization objective for all types of code sequences. As a result, developing a flexible optimization framework to tailor the design of the code family to the desired, smaller number of spreading codes may lead to better performance of the complete navigation system.

From Figures 9 and 10, we observe that the proposed NES method in violet outperforms the best Gold code and Weil code families across all tested code lengths and code family sizes. To determine if this performance improvement against the best-performing sets of Gold and Weil codes would continue for larger families of

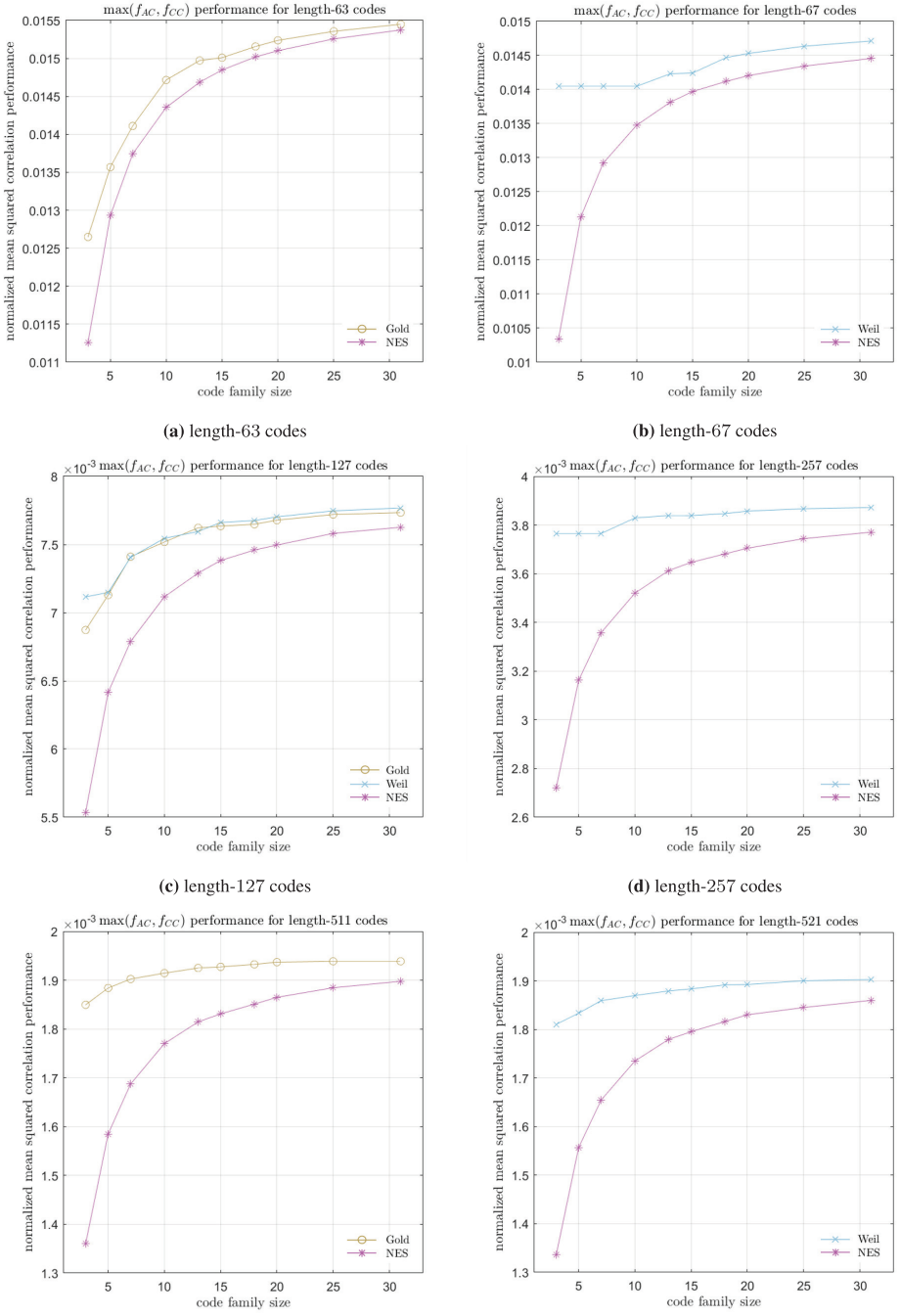
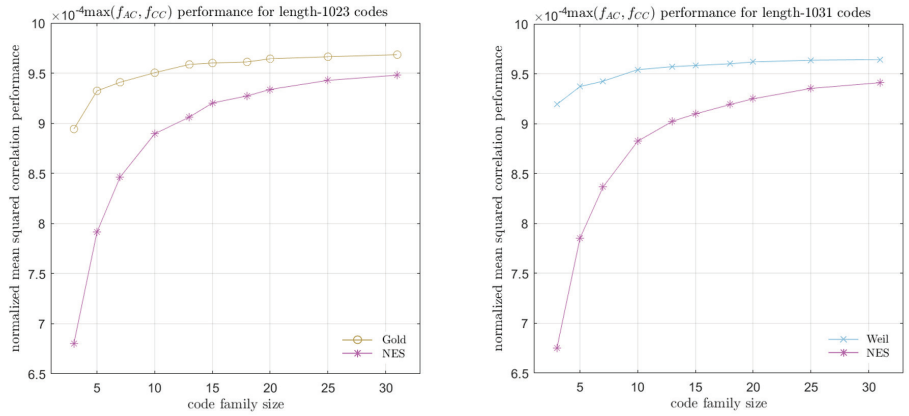


FIGURE 9 Comparison of the proposed NES machine-learning method with that of well-chosen Gold and Weil code families as a function of family size; performance is optimized in terms of the maximization evaluation objective, i.e. max(f_{AC}, f_{CC}), defined in Equation (17), plotted here in terms of the normalized correlations. The Gold code correlation performance is indicated in the gold line with circle markers denoting the particular, discrete set of family sizes tested in each case, while the Weil code performance is indicated in blue with × markers and the performance of the codes generated by our proposed NES algorithm is indicated in violet with asterisk markers. We observed that the NES method outperformed the best Gold and Weil codes across all tested sequence lengths and code family sizes.



(a) length-1023 codes

(b) length-1031 codes

FIGURE 10 Comparison of the proposed NES machine learning method with that of well-chosen Gold and Weil code families as a function of family size for length-1023 and length-1031 sequences; performance is optimized in terms of the maximization evaluation objective, i.e. $\max(f_{AC}, f_{CC})$, defined in Equation (17), plotted here in terms of the normalized correlations. The Gold code correlation performance is indicated in the gold line with circle markers denoting the particular, discrete set of family sizes tested in each case, while the Weil code performance is indicated in blue with × markers and the performance of the codes generated by our proposed NES algorithm is indicated in violet with asterisk markers. We observed that the NES method outperformed the best Gold and Weil codes across all tested code family sizes.

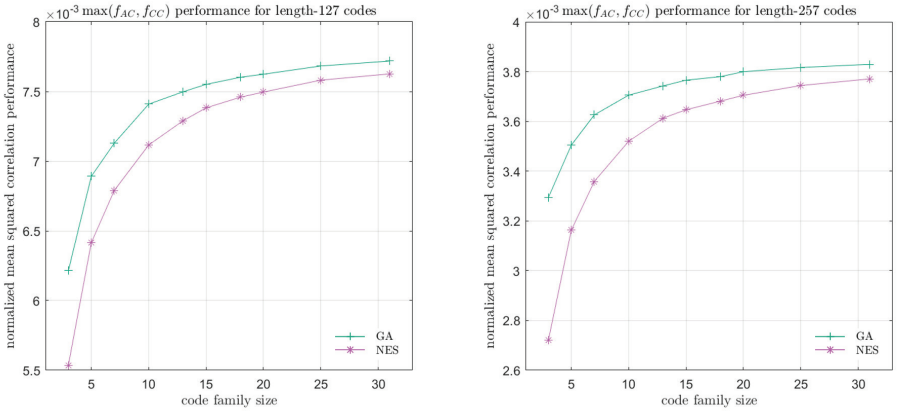
the NES-generated spreading codes, further testing would be required with more extensive memory and computational resources. In Section 4.5, we discuss how the proposed NES method scales with the spreading code length and the size of the code family, while discussing potential techniques to reduce the memory requirements and resources during the training process.

4.4 | Comparison with Analogous Genetic Algorithm and Elitism

Finally, we compared our NES method with that of the analogous genetic algorithm described in Section 4.1. Figure 11 shows separate plots of the normalized correlation performance from length-127 up to length-1031 bit spreading code families. Both optimization algorithms sought to minimize the same spreading code correlation objective defined in Equation (17). The GA additionally incorporates elitism to guarantee the algorithm continuously improves during its optimization process. We observe in Figure 11 that our NES method consistently outperforms the implemented GA with elitism across all tested sequence lengths and family sizes.

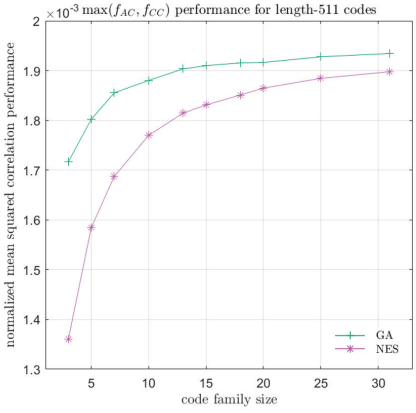
4.5 | Scaling of Network Size with the Size of the Learned Code Family

With this proposed method, the total number of parameters in the GNN, and correspondingly the memory required for the network to train, scales quadratically with the length of the total code set to be determined (i.e., $K\,\ell$ total binary values for length-ℓ binary sequences and K codes). Indeed, these network parameters

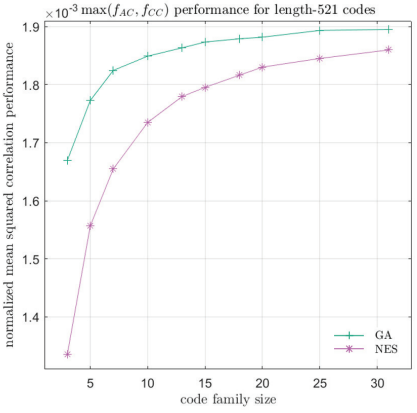


(a) length-127 codes

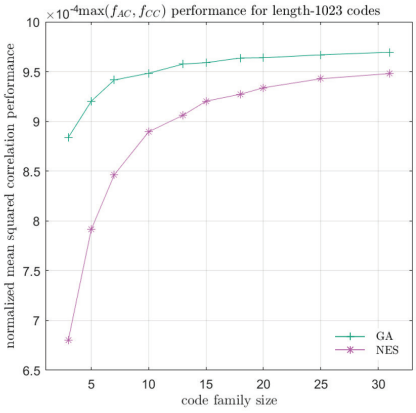
(b) length-257 codes



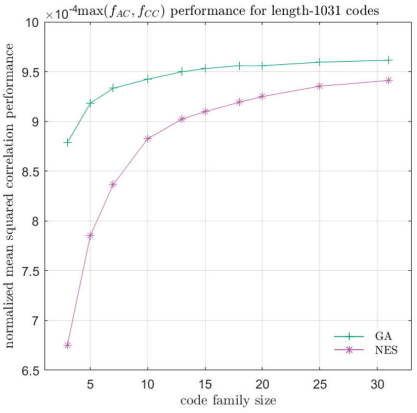
(c) length-511 codes



(d) length-521 codes



(e) length-1023 codes



(f) length-1031 codes

FIGURE 11 Comparison of the proposed NES machine-learning method with that of the analogous genetic algorithm with incorporated elitism; performance is optimized in terms of the maximization evaluation objective, i.e., $\max(f_{AC}, f_{CC})$, defined in Equation (17), plotted here in terms of the normalized correlations. The *genetic algorithm* (GA) performance is indicated in green with + markers denoting the particular, discrete set of family sizes tested in each case, while the performance of our proposed NES method is indicated in violet with asterisk markers. We observe that for all spreading code lengths, the NES method outperforms the GA implementation in designing spreading codes which minimize the spreading code evaluation metric in Equation (17).

are only necessary during the training phase of the algorithm while the codes are being learned. Once the algorithm converges to a final, optimized spreading code family, the network parameters are no longer needed and can safely be discarded. However, the memory required during training correspondingly scales quadratically with the length of the total code set, thereby requiring significantly greater investments in computing resources, especially for very large family sizes of longer spreading codes, such as a family of 420 length-10,230 sequences as used for GPS L1C (Navstar GPS Joint Program Office, 2021b).

Besides simplifying the neural network structure, one could reduce memory requirements for training large-scale neural networks by modeling network parameters with lower precision as well as by exploring the use of sparsity in the network model, where some of the network parameters are set to zero and are correspondingly no longer necessary to store in memory. In this work, we used a fully connected network model; however, leveraging sparsity has been shown to significantly reduce the number of network parameters required for deep networks while achieving similar or even better performance (Mostafa & Wang, 2019; Sohoni et al., 2019). Another advantage to utilizing sparse models is the corresponding reduction in the number of gradient computations as well as the size of the momentum buffer for the optimizer, thereby also leading to reduced memory requirements during training.

5 | CONCLUSION

In this work, we developed a flexible framework to devise a family of high-performing spreading code sequences which achieve low mean-square periodic auto- and cross-correlation objectives. We utilized a natural evolution strategy algorithm using a generative neural network to model a Gaussian proposal distribution over the space of binary spreading code families. Using the Monte-Carlo-based gradient estimate, we directly optimized the proposal distribution over the design search space. Furthermore, we believe this is the first work to develop a machine-learning method to design navigation spreading code families.

By minimizing the maximum between the mean-squared auto-correlation and the mean-squared cross-correlation, we demonstrated the ability of our algorithm to achieve better performance than well-chosen families of Gold and Weil codes, for sequences of up to length-1,023 and length-1,031 bits and family sizes of up to 31 codes. We additionally demonstrated that the spreading code families generated from our method consistently outperformed those obtained from a genetic algorithm with incorporated elitism on the overall objective function.

ACKNOWLEDGMENTS

This material is based upon work supported by the Air Force Research Lab (AFRL) at Kirtland Air Force Base in Albuquerque, New Mexico, under grant number FA9453-20-1-0002. This material is additionally based upon work supported by the National Science Foundation Graduate Research Fellowship under grant number DGE-1656518. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Air Force Research Lab or the National Science Foundation.

REFERENCES

Ávila-Rodríguez, J. A., Wallner, S., & Hein, G. W. (2006). How to optimize GNSS signals and codes for indoor positioning. *Proc. of the 19th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2006)*, Fort Worth, TX, 2418–2426.

Baluja, S., & Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. *Proc. of the 12th International Conference on Machine Learning*, Tahoe City, CA, 38–46. https://doi.org/10.1016/B978-1-55860-377-6.50014-1

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Burroughs, M. D., & Wilson, S. G. (1996). On the performance of 4-phase sequences in asynchronous CDMA systems. In T. S. Rappaport, B. D. Woerner, & J. H. Reed (Eds.), *Wireless personal communications: The evolution of personal communications systems* (pp. 171–182). Springer. https://dl.acm.org/doi/10.5555/251300.251346

Chapman, D., Hinks, J., & Anderson, J. (2020). Way, way out in front – Navigation Technology Satellite-3: The vanguard for space-based PNT. *Inside GNSS*, *15*(4), 32–35. https://insidegnss.com/way-way-out-in-front-navigation-technology-satellite-3-the-vanguard-for-space-based-pnt/

Cozzens, T. (2020). *First GPS III satellite now available*. GPS World. https://www.gpsworld.com/first-gps-iii-satellite-now-available

Cozzens, T. (2021). *NTS-3 experimental satellite launch delayed to 2023*. GPS World. https://www.gpsworld.com/nts-3-experimental-satellite-launch-delayed-to-2023

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, *12*(7), 2121–2159. https://dl.acm.org/doi/10.5555/1953048.2021068

European Union. (2021). *European GNSS (Galileo) open service signal-in-space interface control document* (OS SIS ICD Issue 2.0). https://www.gsc-europa.eu/sites/default/files/sites/all/files/Galileo_OS_SIS_ICD_v2.0.pdf

Ganapathy, H., Pados, D. A., & Karystinos, G. N. (2011a). New bounds and optimal binary signature sets–Part I: Periodic total squared correlation. *IEEE Transactions on Communications*, *59*(4), 1123–1132. https://doi.org/10.1109/TCOMM.2011.020411.090404

Ganapathy, H., Pados, D. A., & Karystinos, G. N. (2011b). New bounds and optimal binary signature sets–Part II: Aperiodic total squared correlation. *IEEE Transactions on Communications*, *59*(5), 1411–1420. https://doi.org/https://doi.org/10.1109/TCOMM.2011.020811.090405

Glynn, P. W. (1990). Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, *33*(10), 75–84. http://www.doi.org/10.1145/84537.84552

Gold, R. (1967). Optimal binary sequences for spread spectrum multiplexing (corresp.) *IEEE Transactions on Information Theory*, *13*(4), 619–621. http://www.doi.org/10.1109/TIT.1967.1054048

Golomb, S. W. (2017). *Shift register sequences: Secure and limited-access code generators, efficiency code generators, prescribed property generators, mathematical models* (3rd Ed.). Aegean Park Press. https://www.doi.org/10.1142/9361

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Guohua, Z., & Quan, Z. (2002). Pseudonoise codes constructed by Legendre sequence. *Electronics Letters*, *38*(8), 376–377. https://www.doi.org/10.1049/el:20020220

Hinton, G., Srivastava, N., & Swersky, K. (2012). *Lecture 6e, rmsprop: Divide the gradient by a running average of its recent magnitude* [Lecture notes]. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Holland, J. H. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press. https://www.doi.org/10.7551/mitpress/1090.001.0001

Huang, L., Zhang, H., Li, R., Ge, Y., & Wang, J. (2019). AI coding: Learning to construct error correction codes. *IEEE Transactions on Communications*, *68*(1), 26–39. https://www.doi.org/10.1109/TCOMM.2019.2951403

Inside GNSS. (2016). *Air force, Army reach for new GPS technologies*. Inside GNSS. https://insidegnss.com/air-force-army-reach-for-new-gps-technologies

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations,* San Diego, CA. https://arxiv.org/pdf/1412.6980.pdf

Kochenderfer, M. J., & Wheeler, T. A. (2019). *Algorithms for optimization*. MIT Press.

Legendre, A. M. (1808). *Essai sur la théorie des nombres* (2nd Ed.). Cambridge University Press. https://www.doi.org/10.1017/CBO9780511693199

Lu, C.-P. (2017). AI, native supercomputing and the revival of Moore's law. *APSIPA Transactions on Signal and Information Processing*, *6*(1). https://www.doi.org/10.1017/ATSIP.2017.9

Mina, T. Y., & Gao, G. X. (2019). Devising high-performing random spreading code sequences using a multi-objective genetic algorithm. *Proc. of the 32nd International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+ 2019)*, Miami, FL, 1076–1089. https://www.doi.org/10.33012/2019.17044

Mina, T. Y., & Gao, G. X. (2020). Designing low-correlation GPS spreading codes via a policy gradient reinforcement learning algorithm. *Proc. of the 33rd International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+ 2020)*, 1968–1983. https://www.doi.org/10.33012/2020.17650

Mohamed, S., Rosca, M., Figurnov, M., & Mnih, A. (2020). Monte Carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, *21*(132), 1–62. https://jmlr.org/papers/volume21/19-346/19-346.pdf

Mostafa, H., & Wang, X. (2019). Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *Proc. of the 36th International Conference on Machine Learning*, Long Beach, CA, 4646–4655. http://proceedings.mlr.press/v97/mostafa19a/mostafa19a.pdf

Navstar GPS Joint Program Office. (2021a). Interface specification: Navstar GPS space segment/navigation user interfaces (IS-GPS-200 Revision M).

Navstar GPS Joint Program Office. (2021b). Interface specification: Navstar GPS space segment/user segment L1C interface (IS-GPS-800 Revision H).

Parkinson, B. W., Powers, S. T., Green, G., Fruehauf, H., Strom, B., Gilbert, S., Melton, W., Huston, B., Martin, E., Spilker, J., Natali, F., Strada, J., Glazer, B., Schwartz, D., & Stansell, T. (2010). Part 1: The origins of GPS, and the pioneers who launched the system. *GPS World*, *25*(5), 30–41. https://www.gpsworld.com/origins-gps-part-1

Pursley, M. (1977). Performance evaluation for phase-coded spread-spectrum multiple-access communication - Part I: System analysis. *IEEE Transactions on Communications*, *25*(8), 795–799. https://www.doi.org/10.1109/TCOM.1977.1093915

Rechenberg, I. (1973). Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. Feddes Repertorium, 86(5), 337–337. https://www.doi.org/10.1002/fedr.19750860506

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, *22*(3), 400–407. https://www.doi.org/10.1214/aoms/1177729586

Rushanan, J. J. (2006). Weil sequences: A family of binary sequences with good correlation properties. *2006 IEEE International Symposium on Information Theory*, Seattle, WA, 1648–1652. https://www.doi.org/10.1109/ISIT.2006.261556

Rushanan, J. J. (2007). The spreading and overlay codes for the L1C signal. *NAVIGATION*, *54*(1), 43–51. https://www.doi.org/10.1002/j.2161-4296.2007.tb00394.x

Sarwate, D. (1979). Bounds on crosscorrelation and autocorrelation of sequences (corresp.) *IEEE Transactions on Information Theory*, *25*(6), 720–724. https://www.doi.org/10.1109/TIT.1979.1056116

Schaul, T., Glasmachers, T., & Schmidhuber, J. (2011). High dimensions and heavy tails for natural evolution strategies. *Proc. of the 13th Annual Conference on Genetic and Evolutionary Computation*, 845–852. https://www.doi.org/10.1145/2001576.2001692

Sidelnikov, V. M. (1971a). Cross correlation of sequences. *Probl. Kybern*, *24*, 15–42.

Sidelnikov, V. M. (1971b). On mutual correlation of sequences. *Doklady Akademii Nauk*, *196*(3), 531–534.

Sohoni, N. S., Aberger, C. R., Leszczynski, M., Zhang, J., & Ré, C. (2019). Low-memory neural network training: A technical report. https://arxiv.org/pdf/1904.10631.pdf

Soualle, F., Soellner, M., Wallner, S., Ávila-Rodríguez, J., Hein, G., Barnes, B., Pratt, T., Ries, L., Winkel, J., Lemenager, C., & Erhard, P. (2005). Spreading code selection criteria for the future GNSS Galileo. *Proc. of the European Navigation Conference GNSS*, 19–22.

Stular, M., & Tomazic, S. (2001). Mean periodic correlation of sequences in CDMA. *Proc. of IEEE Region 10 International Conference on Electrical and Electronic Technology (TENCON 2001)*, Singapore, 287–290. https://www.doi.org/10.1109/TENCON.2001.949598

Stular, M., & Tomazic, S. (2000). Maximum periodic correlation of pseudo-random sequences in CDMA. *Proc. of the 10th Mediterranean Electrotechnical Conference,* 420–423. https://www.doi.org/10.1109/MELCON.2000.880455

Sun, S., Cao, Z., Zhu, H., & Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. *IEEE Transactions on Cybernetics*, *50*(8), 3668–3681. https://www.doi.org/10.1109/TCYB.2019.2950779

Syswerda, G. (1989). Uniform crossover in genetic algorithms. *Proc. of the 3rd International Conference on Genetic Algorithms*, 2–9.

United States Department of the Air Force, Air Force Materiel Command. (2016). *Navigation Technology Satellite-3* (RFI-RVKVE-NTS-3). Kirtland AFB, NM, United States.

van Diggelen, F. (2014). Who's your daddy? Why GPS will continue to dominate consumer GNSS. *Inside GNSS*, *9*(2), 30–41. https://www.insidegnss.com/auto/marapr14-vanDiggelen.pdf

van Diggelen, F. (2020). High-sensitivity GNSS. In Y. T. Jade Morton, F. van Diggelen, J. J. Spilker Jr., B. W. Parkinson, S. Lo, & G. Gao (Eds.), *Position, navigation, and timing technologies in the 21st century: Integrated satellite navigation, sensor systems, and civil applications* (pp. 445–479). Wiley Online Library. https://www.doi.org/10.1002/9781119458449.ch18

Wallner, S., Ávila-Rodríguez, J., & Hein, G. W. (2011). Codes: The PRN family grows again. *Inside GNSS*, *6*(5), 83–92. https://insidegnss.com/codes-the-prn-family-grows-again

Wallner, S., Ávila-Rodríguez, J.-A., Hein, G. W., & Rushanan, J. J. (2007). Galileo E1 OS and GPS L1C pseudo random noise codes - requirements, generation, optimization, and comparison. *Proc. of the 20th International Technical Meeting of the Satellite Division of the Institute of*

Navigation (ION GNSS 2007), Fort Worth, TX, 25–28. https://www.ion.org/publications/abstract.cfm?articleID=7359

Welch, L. (1974). Lower bounds on the maximum cross correlation of signals (corresp.) *IEEE Transactions on Information Theory*, *20*(3), 397–399. https://www.doi.org/10.1109/TIT.1974.1055219

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., & Schmidhuber, J. (2014). Natural evolution strategies. *The Journal of Machine Learning Research*, *15*(1), 949–980. https://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf

Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008). Natural evolution strategies. *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, Hong Kong, China, 3381–3387. https://www.doi.org/10.1109/CEC.2008.4631255

Winkel, J. O. (2011). *Spreading codes for a satellite navigation system* [Patent No. US 8,035, 555 B2].

Yan, S. Y. (2002). *Number theory for computing*. Springer.

How to cite this article: Mina, T. & Gao, G. (2022) Designing low-correlation GPS spreading codes with a natural evolution strategy machine learning algorithm. *NAVIGATION*, *69*(1). https://doi.org/10.33012/navi.506

A1 | APPENDIX: BACKGROUND ON NEURAL NETWORKS

At a high-level, an *artificial neural network* is a collection of parameterized linear and nonlinear functions, which altogether represent a complex and flexible mathematical model (Bishop, 2006). The parameters of the network are optimized by the machine-learning algorithm, or *learned*, in order to recognize potentially complicated patterns which are difficult to describe within a provided set of data.

The key structural component of the neural network architecture is an *artificial neuron*, which is pictorally represented in Figure A1. A neuron consists of three main components: a) weight parameters, b) a bias parameter, and a c) nonlinear

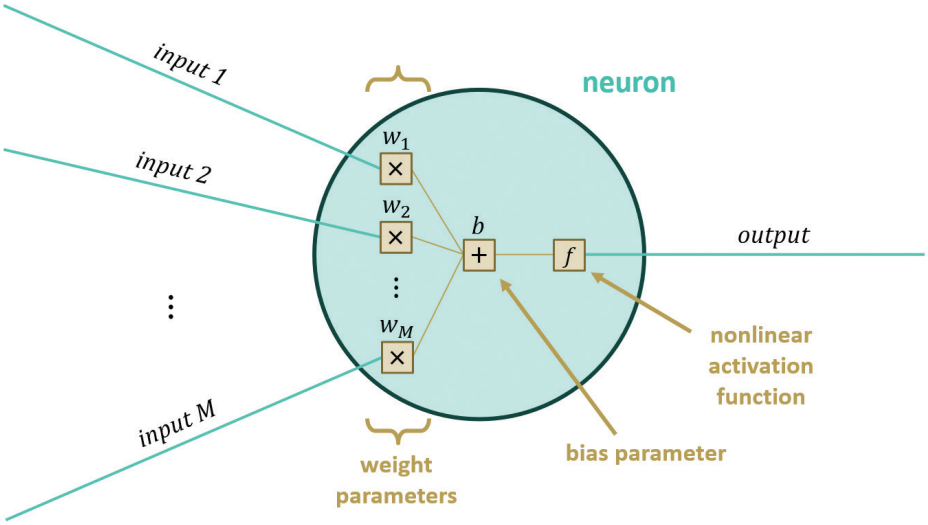


FIGURE A1 Depiction of an individual artificial neuron, the key structural component of an artificial neural network

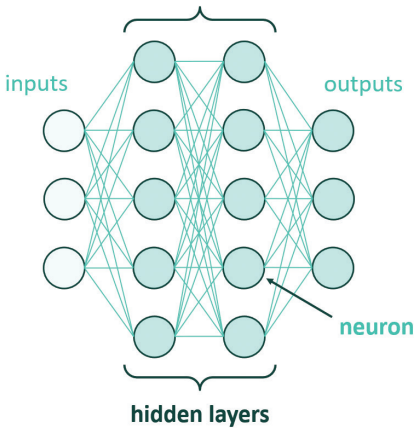


FIGURE A2 Depiction of an artificial neural network, consisting of layers of stacked artificial neurons

activation function as shown in Figure A1. A neuron can have any number of inputs; in Figure A1, we illustrate a neuron with M inputs. Each input $i \in \{1, \ldots, M\}$ is multiplied by its corresponding weight w_i which is a parameter of the neuron, then the weighted inputs are summed and added to the bias term b, which is also a parameter of the network.

Finally, the summed term, which is a scalar value for a particular neuron, is input to a nonlinear activation function represented as f. Indeed, the notation $f(\cdot)$ is commonly used in machine learning to represent the nonlinear activation function of a neuron, and in this case should not be confused with the objective function used for optimization. Thus, the explicit expression for the mathematical operations performed by an individual neuron is:

$$y = f\left(b + \sum_{i=1}^{M} x_i w_i\right) \tag{A1}$$

where y is the scalar value output by the neuron and x_i is the i-th input value to the neuron.

Nonlinear activation functions have a domain that is continuous and unbounded (i.e., $\mathbb{R}$), and generally a range that is continuous, which may or may not be bounded. Common bounded-output activation functions include the logistic sigmoid function function, commonly called a *sigmoid function*, as defined in Equation (A2), as well as the *hyperbolic tangent function* defined in Equation (A3):

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{A2}$$

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{A3}$$

A commonly utilized unbounded activation function is the *rectified linear unit*, commonly called *ReLU*, defined in Equation (A4):

$$\mathrm{ReLU}(x) = \max(0, x) \tag{A4}$$

An artificial neural network consists of layers of stacked artificial neurons, where the output of each layer of neurons acts as the inputs to the next layer of neurons, as depicted in Figure A2. The input layer is simply represented as a set of constant values and is correspondingly represented as a lighter shade in Figure A2, while the neurons in the output layer still maintain the complete neural structure of weights, biases, and an activation function as in Figure A1. Notice that the number of neurons in the output layer is equivalent to the dimension of the desired output from the neural network.

The layers of neurons which do not correspond to the input layer or the output layer are called the *hidden layers* of the network. The number of neurons in each hidden layer can be selected by the designer of the network architecture. Additionally, the neural network is typically fully connected, in which each neuron from the previous layer is input to all neurons in the following layer.

The parameters of the neural network (i.e., the weights and biases of each artificial neuron in the network) are optimized, or learned, during training via any desired optimization algorithm (Kochenderfer & Wheeler, 2019; Sun et al., 2019). Several optimization algorithms that are commonly utilized in the field of machine learning include stochastic gradient descent (Robbins & Monro, 1951), Adagrad (Duchi et al., 2011), RMSProp (Hinton et al., 2012), and Adam (Kingma & Ba, 2015).

For more background on neural network architectures or general machine-learning concepts, the authors recommend referencing Bishop (2006) and Goodfellow et al. (2016) for a valuable and accessible introduction.