Johann Radon Institute for
Computational and Applied Mathematics
Austrian Academy of Sciences (ÖAW)

# Approximation of convex polygons by polygons

C. Koutschan, A. Ponomarchuk, J. Schicho

RICAM-Report 2021-27

# Approximation of convex polygons by polygons

Christoph Koutschan
*RICAM*
*Austrian Academy of Sciences*
Linz, Austria
0000-0003-1135-3082

Anton Ponomarchuk
*RICAM*
*Austrian Academy of Sciences*
Linz, Austria
Anton.Ponomarchuk@oeaw.ac.at

Josef Schicho
*RISC*
*Johannes Kepler University*
Linz, Austria
Josef.Schicho@risc.jku.at

***Abstract*—Motivated by the study of shallow rectifier neural networks, we propose an algorithm that approximates a polygon in the plane by an outer polygon with fewer vertices. The algorithm minimizes the area locally in an iterative manner, and hence is guaranteed to find a local minimum and not necessarily the global optimum. Nevertheless, experiments indicate that in practice it yields a reasonably good approximation.**

***Index Terms*—ReLU network, convex polygon, shape approximation**

## I. Introduction

The motivation of our study originates in the context of machine learning, by the desire to acquire structural insights into the functionality of neural networks by geometric reasoning. Deep neural networks have achieved superior results in a number of tasks, such as pattern recognition, data generation, compression. Despite such results, there is a lack of theory concerning analysis and understanding from a structural mathematical point of view. Neural networks as they are used today are hard to analyse leading to well-known black-box problems, as for example explainability and instabilities related to $\varepsilon$-perturbations in an adversarial setting.

Deep models specify the parameterized input-output function $F$ of a network as multi-layered structure of computations. It results from iteratively applying the composition of an affine pre-activation function $f_k(z_{k-1}) := W_k z_{k-1} + b_k$ and a subsequent component-wise application of the non-linear activation function $g\colon \mathbb{R} \to \mathbb{R}$, where $z_0 = x \in \mathbb{R}^n$ (the input data) and $z_k = g \circ f_k(z_{k-1}) \in \mathbb{R}^{n_k}$ is the output of the $k$-th layer and $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$ denotes the weight matrix and $b_k \in \mathbb{R}^{n_k}$ the bias vector in the $k$-th layer.

The choice of the activation function is an important design question that influences the overall performance. The breakthrough of deep learning was also related to this question by utilizing the so-called rectified linear unit (ReLU), $g(a) := \max\{0, a\}$, which avoids and rectifies the vanishing gradient problem of previously preferred activation functions such as sigmoid or tanh.

From the above definitions it becomes apparent that such a ReLU network describes a function that is piecewise linear on linear regions of the input space, since each row of the Matrix $W_1$, together with the corresponding entry in the vector $b_1$, defines a hyperplane in the input space, and similarly for the subsequent layers, which finally induces a tesselation of the input space. The maximal number of linear regions of functions computed by a shallow (i.e. $L = 1$) ReLU network with $n$ inputs and $n_1$ hidden units (that is, the number of hyperplanes) is $\sum_{j=0}^{n} \binom{n_1}{j}$, but also for deeper networks this number can be estimated and bounded [8], [9]. It is then natural to separate these regions into those where the function value is 0 and those where it is positive, corresponding to a one-class classification problem, or in a more fine-grained way, according to the activations of the hidden units [10].

Here, we focus on shallow ReLU networks, i.e., neural networks with a single hidden layer ($L = 1$) and whose input space is $\mathbb{R}^2$. It is not difficult to see that the union of linear regions on which the function computed by the network is zero, i.e., the 0-preimage of that function, is a convex polygon, which is obtained by a finite intersection of half-spaces. That means, if the network has $n_1$ hidden units in the first layer, then its 0-preimage is described by a polygon $K$ with at most $n_1$ vertices. We study the question, how one can construct a simpler network, i.e., one with fewer units, that behaves as similar as possible compared to the original one. More precisely, we ask for a network with smallest possible 0-preimage that still contain the original 0-preimage $K$. Geometrically speaking, we propose an algorithm for polygon shape simplification. The algorithm provides an outer approximation of a given polygon $K$ with another polygon that has fewer vertices.

The problem of convex shape approximation with respect to pre-defined distance functions between convex polygons is studied in [7], [11]. The authors in [11] explored the problem of minimizing the volume of the symmetric difference of two convex polygons $K$ and $K'$ under all possible scalings and translations of $K'$. Moreover, in [7] an algorithm is provided that approximates a convex polygon $K$ by another one with smaller number of vertices by using the Hausdorff metric as the distance function. In contrast to the construction of outer convex polygon approximations, in [4], [7] approaches for convex internal approximation are described. In [7] an approach is presented that builds a convex inner polytope $K'$ that approximates the given polytope $K$ with a limited ratio of volumes $|K \setminus K'|/|K'|$. More detailed information about convex polytopes can be found in [3] and about polygon approximation in [1].

The research reported in this paper has been partly funded by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.

## II. METHOD

Given a convex bounded polygon $K \subset \mathbb{R}^2$, we denote by $V(K) \subset \mathbb{R}^2$ the set of its vertices. A vertex $v$ of $K$ is a point $v \in K$ such that there exists a line $\ell$ with the properties that $\ell \cap K = \{v\}$ and all other points of $K$ lie completely on one side of $\ell$. A line $\ell$ is called a *supporting line* of $K$ if and only if $\ell \cap K \neq \emptyset$ but the half-space on one side of $\ell$ does not intersect with $K$. Further, we denote by $E(K)$ the set of lines $\ell$ that contain the edges of $K$. More precisely, $E(K)$ contains all supporting lines $\ell$ of $K$ with the property $|\ell \cap K| \geq 2$, or equivalently $|\ell \cap V(K)| = 2$.

For any convex set $K \subset \mathbb{R}^2$ the area of $K$ will be denoted by $A(K)$. We denote by $\mathrm{int}(K)$ all interior points of $K$, i.e., all $p \in K$ such that $p \cap \ell = \emptyset$ for all $\ell \in E(K)$. Similarly, by $\mathrm{bd}(K)$ we denote the boundary of $K$, i.e., $\mathrm{bd}(K) := K \setminus \mathrm{int}(K)$. For two distinct points $v, w \in \mathbb{R}^2$, we denote by $\mathrm{line}(v, w)$ the unique line going through $v$ and $w$, i.e.,

$$\mathrm{line}(v, w) := \{\lambda v + (1 - \lambda)w \mid \lambda \in \mathbb{R}\}.$$

For a vertex $v \in V(K)$ of a convex polygon $K$, the family of all lines that contain the vertex $v$ but do not intersect $\mathrm{int}(K)$ is denoted by $\mathcal{L}(v)$:

$$\mathcal{L}(v) := \{\ell \subset \mathbb{R}^2 \mid v \in \ell,\, \ell \cap \mathrm{int}(K) = \emptyset\}.$$

For our algorithm, we further assume that the vertices $V(K) = \{v_1, \ldots, v_N\}$ are given in cyclic order. That is, assuming we pick an arbitrary point $p \in \mathrm{int}(K)$ and rotate $\mathrm{line}(p, v_1)$ in clockwise direction around $p$, then it will meet the points $v_2, \ldots, v_N$ in the given order. For sake of simplicity, we impose that the indices behave according to this cyclic order without making this explicit, i.e., we tacitly use the convention that $v_{N+1} = v_1$, $v_0 = v_N$, etc. The same convention is applied to the set of lines $E(K)$.

The main purpose of this paper is to introduce an algorithm that approximates a bounded convex polygon $K$ formed by $N$ edges by a convex polygon $K^*$ with a smaller number of edges $n < N$. The algorithm that is proposed here takes as input the cyclically ordered set of vertices $V(K)$ and a predefined number of edges $n$ for the desired approximating polygon. It starts by picking $n$ lines from $E(K)$ to form a set of lines: $E(K_0) \subset E(K)$. The set $E(K_0)$ defines a convex polygon $K_0$, whose vertices are obtained by intersecting neighboring lines from $E(K_0)$. $K_0$ is the first approximation of the polygon $K$. Then the algorithm checks whether by rotating some line $\ell_j \in E(K_0)$ around one of the vertices $v_r, v_{r+1} \in V(K)$ with $\{v_r, v_{r+1}\} \subset \ell_j$ by a certain angle $\varphi \in (0, \pi)$ (see Fig. 1), we could transform $K_0$ into a new convex polygon $K_1$ with the following properties:

$$A(K_1) < A(K_0), \tag{1}$$

$$K \subset K_1. \tag{2}$$

Equations (1) and (2) signify that the generated polygon $K_1$ has smaller area than $K_0$, while it still contains the input polygon $K$ as a subset. More generally, we test this property for all vertices $v_s, \ldots, v_t \subset V(K)$ in the polygon chain between the two neighboring lines $\ell_{j-1}$ and $\ell_{j+1}$, that is, $\ell_{j-1} \cap V(K) = \{v_s, v_{s-1}\}$ and $\ell_{j+1} \cap V(K) = \{v_t, v_{t+1}\}$. Fig. 1 shows an example, where this polygon chain is given by the vertices $v_{r-2}, \ldots, v_{r+3}$, the shaded area corresponding to the polygon $K$. Our algorithm identifies, among all lines $\ell \in \mathcal{L}(v_s) \cup \ldots \cup \mathcal{L}(v_t)$, one that minimizes $A(K_1)$, where $K_1$ denotes the polygon that is obtained from replacing $\ell_j$ in $K_0$ by $\ell$. If it succeeds to find $K_1$ with the properties (1) and (2), then the algorithm continues with $K_1$. If $A(K_0)$ cannot be reduced by moving the line $\ell_j$, then the algorithm proceeds by picking another line from $E(K_0)$ and repeats the process for this new line.

The algorithm continues to check the lines from $E(K_1)$ whether one of them could be rotated in order to reduce the area of $K_1$. The check and update processes continues until for some set of lines $E(K^*)$ there does not exist a line $\ell \in E(K^*)$ whose rotation would give rise to a polygon with smaller area. This means that the current set of lines $E(K^*)$ is (locally) optimal and we cannot get a convex polygon with smaller area than $K^*$ by rotating any one of its edges.

Summarizing, our algorithm (see Algorithm 1) builds a convex $n$-gon $K^*$ with the following properties:

- $n = |E(K^*)| < |E(K)|$,
- $K^* \supset K$,
- $A(K^*) < A(K)$, and the area of $K^*$ is locally minimal with respect to changing any one of its edges.

In Section II-A we provide information about the structure of the algorithm's input and output. In Section II-B we give a detailed and step-by-step explanation of the algorithm. Section II-C contains the main theoretical results that are necessary for proving the algorithm's correctness.

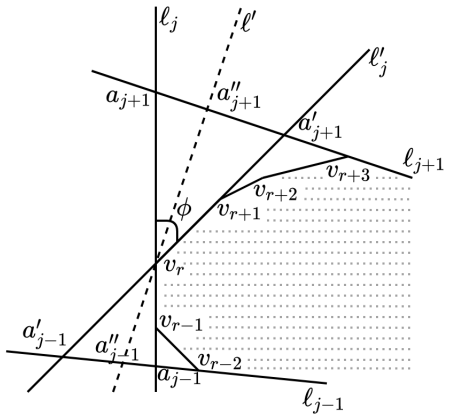



Fig. 1. This image captures a typical situation in the update step of Algorithm 1. A line $\ell_j$ is picked and it is checked whether there exists a line $\ell' \in \mathcal{L}(v_r)$ such that after replacing $\ell_j$ by $\ell'$ the resulting polygon $K_m$ satisfies: $A(K_m) < A(K_{m-1})$ and $K_m \supset K$. The maximal angle by which the algorithm can rotate $\ell_j$ is denoted by $\phi$. The grey shaded area indicates the interior of the input polygon $K$.

### A. Structure of input and output

Algorithm 1 takes as input a pair $(V(K), n)$, where $V(K)$ is the ordered set of vertices of the input polygon $K$, and $n \in \mathbb{N}$ denotes the number of edges of the desired approximating outer polygon. We assume, that $K$ is convex, that its interior is non-empty, $\operatorname{int} K \neq \emptyset$, and that $A(K) < \infty$ so that $K$ is a bounded set.

First, Algorithm 1 initializes a convex polygon $K_0$ by selecting $n$ lines from $E(K)$, denote them by $\ell_1, \ldots, \ell_n$, such that these lines define a bounded polygon. Note that the only situation where this is not possible is when we aim at approximating a parallelogram by a triangle. This situation could be treated as a special case, but is ignored in the following for sake of simplicity. The set $K_0$ then necessarily has the property $K \subseteq K_0$.

As output the algorithm returns an ordered family of lines $E(K^*)$ with $|E(K^*)| = n$. Each line in $E(K^*)$ contains an edge of output polygon $K^*$ that approximates the input polygon $K$. Furthermore, the output polygon $K^*$ is convex and bounded.

### B. The algorithm

Here we present the algorithm, that builds for the convex polygon defined in Section II-A, an outer convex polygon with a pre-defined number of edges $n$. A pseudo-code listing of the algorithm is given on page 4. Our algorithm is iterative: we start with some outer polygon and try to make the error smaller step by step.

We say that an outer polygon $K^*$ with $n$ edges is *locally optimal* if there is no convex polygon $K' \supset K$ whose area is smaller than the area of $K^*$ and such that $K^*$ and $K'$ have $n-1$ edges in common. In other words, a locally optimal polygon cannot be improved by changing a single supporting line. Clearly, an optimal polygon is locally optimal, but the converse is not true in general. Our algorithm computes only a locally optimal polygon. A globally optimal solution would be better, but quite complicated to compute, even for moderate sizes of $n$ (say, $n = 20$); at least, we could not think of an algorithm that computes a global optimum efficiently. Note that exhaustively trying all possibilities (when we impose the restriction that the supporting lines of the solution should also be supporting lines of the original polygon $K$) already leads to a combinatorial explosion, as there are $\binom{N}{n}$ such possibilities, where $N$ denotes the number of edges of $K$.

The idea of the algorithm is the following: we pick one of the $n$ support lines by random and rotate it around a point which it shares with $K$, minimizing the area. This is a one-dimensional problem which can be explicitly solved. Then we mark the picked edge as "already optimized" and proceed to the next edge. Additional care must be taken because the problems are not independent: a change of the line also changes the parameters of the two neighboring support lines. So, if one line is changed, we may mark this line as already optimized but we need to unmark the two neighboring lines.

The algorithm stops if all lines are marked. It is at this point not so clear if this ever happens, because in each step we mark one line but maybe we unmark two. This question is treated in the next section.

Let us now consider the one-dimensional problem more closely. Let us call the support line which we are allowed to change the *pivot line*. Apparently, the optimal support line depends only on the two neighboring support lines and on the polygon chain that connects these two support lines. Any vertex in this chain is a point around which we may rotate the pivot line; we therefore call these points *anchor points*. For any anchor point, there is an interval of possible choices of the pivot line. It is bounded by the two lines supporting the two edges attached to the anchor point.

The restriction to the interval determined by an anchor point is a one-dimensional optimization problem where the objective function is analytic, in particular differentiable. Indeed, we have to optimize the area of a triangle where two of the three lines and one point is given. We will later show that this function has only one stationary point (i.e., a point where the derivative of the objective function is zero), and in this point, the given point is the midpoint a triangle edge. Since the objective function is differentiable, we conclude that the optimum is either achieved at a boundary or at a stationary point. So, we have only three candidates for each anchor point. Also, the left boundary point of one anchor point is the right boundary point for the next anchor point.

In the examples we tested, the number of iterations is finite. It would be nice to prove that this is always the case, for instance as a consequence of the statement that all candidates that are ever tested come from a finite set. Indeed, the number of support lines that support an edge of $K$ is finite, which takes care of the candidates of the one-dimensional problem on the boundary of intervals. For each one-dimensional problem, there is only a single candidate in the middle; however, the one-dimensional problems depend on the neighbor support lines. For the cases where the neighbor support lines are support lines of edges of $K$, or support lines from the initialization, the number is still finite. But we also have other support lines arising in the computation, namely those that are itself optima in the interior of the interval. Let us call these support lines *swinging lines*. We may have potentially infinitely many candidates if – and only if – there are two neighboring swinging support lines.

It is possible to construct examples with two neighboring swinging support lines. Indeed, there are at most two swinging support lines unless $n = 3$. But if there are two, then they have to be neighbors. This can be seen as follows: a swinging support line is only possible if the two neighboring support lines form an *opening angle*, i.e., the sum of the two outer angles of the pivot lines with the two neighboring lines is bigger than $\pi$. But in a closed polygon, the sum of the outer angles is always equal to $2\pi$.

In order to achieve a finite set of possible candidates (thereby ensuring the termination of the algorithm), we slightly change the algorithm described above. If a situation with two neighboring swinging lines is possible – which is checked by computing the two sums of outer angles –, then we

avoid solving a one-dimensional optimization problem with a swinging neighboring support line. Instead, we are solving a two-dimensional optimization problem, optimizing the two possible swinging lines simultaneously.

As we will prove in the next section, the two-dimensional optimization problem also has just one unique stationary point. Assume that the two pivot lines are $\ell_j$ and $\ell_{j+1}$. The neighboring lines $\ell_{j-1}$ and $\ell_{j+2}$ are fixed, and fixed is also the anchor point $v$ for $\ell_j$ and the anchor point $w$ for $\ell_{j+1}$. According to Theorem 1 (see below), the area is minimized if $v$ divides the edge given by $\ell_j$ into two segments of equal length, and similarly $w$ for $\ell_j$. Theorem 2 in the next section states an explicit construction for two such lines. The other possibility that needs to be checked is that the minimum is attained at the boundary of definition.

### C. Correctness of Algorithm 1

This section proves the correctness of Algorithm 1. We first prove, that the optimal line through a vertex $v_r$ that minimizes the area of the approximating convex polygon, is an element of the set $\mathcal{L}(v_r)$ that lies either at the boundary, or in the interior at a very specific location. Since this very same construction is used in Algorithm 1 this will imply that after the $m$-th update step of the algorithm, the inner area of the approximating convex polygon $K_m$ decreases, $A(K_{m-1}) > A(K_m)$, and that our choice is (locally) optimal.

**Theorem 1.** *Let $\ell_{j-1}, \ell_j, \ell_{j+1} \in E(K_{m-1})$ be three consecutive lines of a polygon $K_{m-1} \supset K$, such that some vertex $v_r \in V(K)$ lies on $\ell_j$, while its neighbor $v_{r+1}$ does not. Denote by $a_{j+1} \in \ell_{j+1} \cap \ell_j$ and $a_{j-1} \in \ell_j \cap \ell_{j-1}$ the intersection points of the line $\ell_j$ with the other two lines. Further let $\ell'_j := \mathrm{line}(v_r, v_{r+1})$ be the line through $v_r$ and $v_{r+1}$, and denote by $a'_{j-1} \in \ell_{j-1} \cap \ell'_j$ and $a'_{j+1} \in \ell_{j+1} \cap \ell'_j$ the intersection points of this new line with the other two lines, as depicted in Fig. 1.*

*The line $\ell''_j \in \mathcal{L}(v_r)$ that minimizes the area of the polygon $K_m$, which is obtained by replacing $\ell_j$ in $K_{m-1}$ by $\ell''_j$, is one of the following three: $\ell''_j = \ell_j$ or $\ell''_j = \ell'_j$ or $\ell''_j$ is the unique line satisfying $\|v_r - a''_{j-1}\| = \|v_r - a''_{j+1}\|$, where $a''_{j-1} \in \ell_{j-1} \cap \ell'$ and $a''_{j+1} \in \ell_{j+1} \cap \ell'$.*

*Proof.* Without loss of generality, we can assume that the point $v_r$ is the origin (if not, we can translate our figure accordingly). Let $(x, y)$ and $(x', y')$ be the coordinates of the points $a_{j-1}$ and $a'_{j-1}$, respectively. The unknown coordinates of $a''_{j-1}$ are given by

$$(x'', y'') = \big(tx + (1-t)x', ty + (1-t)y'\big) \tag{3}$$

for some real number $t \in [0, 1]$. Then the coordinates of the points $a_{j+1}$, $a'_{j+1}$, and $a''_{j+1}$ are given by $c \cdot (x, y)$, $c' \cdot (x', y')$, and $c'' \cdot (x'', y'')$, respectively, for some negative real numbers $c, c', c''$. The colinearity of these three points is expressed by the equation

$$(cx - c''x'')(c'y' - c''y'') = (c'x' - c''x'')(cy - c''y''). \tag{4}$$

**Algorithm 1:** Convex polygon approximation

**Input:** $V(K)$ – set of vertices of a convex polygon $K$, $n$ – number of edges to approximate the set $K$.
**Output:** family of lines that defines a locally optimal $n$-vertex polygon.
**Initiation:** Select $n$ lines from $E(K)$ in cyclic order (denote them by $\ell_1, \ldots, \ell_n$) such that they constitute a convex polygon.

$E_{\mathrm{in}} := \{1, \ldots, n\}$
**while** $|E_{\mathrm{in}}| \neq 0$ **do**
  $j :=$ sample$(E_{\mathrm{in}})$
  $\ell_{\min} := \ell_j$
  $A_{\min} :=$ area of the polygon formed by $\ell_1, \ldots, \ell_n$
  **for** *each edge $e$ in the polygonal chain connecting the lines $\ell_{j-1}$ and $\ell_{j+1}$* **do**
    $\ell :=$ the support line of $e$
    $A :=$ area of polygon with $\ell$ replacing $\ell_j$
    **if** $A < A_{\min}$ **then**
      $(\ell_{\min}, A_{\min}) := (\ell, A)$
  **for** *each vertex $v_r$ in the polygonal chain connecting the lines $\ell_{j-1}$ and $\ell_{j+1}$* **do**
    **if** *$\ell_{j-1}$ and $\ell_{j+1}$ are non-parallel support lines of edges in $K$* **then**
      $\ell :=$ line in $\mathcal{L}(v_r)$ such that $v_r$ is equidistant to $\ell \cap \ell_{j-1}$ and $\ell \cap \ell_{j+1}$
      $A :=$ area of polygon with $\ell$ replacing $\ell_j$
      **if** $A < A_{\min}$ **then**
        $(\ell_{\min}, A_{\min}) := (\ell, A)$
  $E_{\mathrm{in}} := E_{\mathrm{in}} \setminus \{j\}$
  **if** $\ell_{\min} \neq \ell_j$ **then**
    $\ell_j := \ell_{\min}$
    $E_{\mathrm{in}} := E_{\mathrm{in}} \cup \{j-1, j+1\}$
  **if** *$\angle(\ell_{j-1}, \ell_{j+1}) > \pi$ and $\angle(\ell_j, \ell_{j+2}) > \pi$* **then**
    **for** *each ordered pair of vertices $v_r, v_s$ in the polygonal chain connecting the lines $\ell_{j-1}$ and $\ell_{j+2}$* **do**
      $p := \rho_{v_r}(\ell_j) \cap \rho_{v_s}(\ell_{j+1})$
      $\ell :=$ line connecting $v_r$ and $p$
      $\ell' :=$ line connecting $v_s$ and $p$
      $A :=$ area of the polygon $K'$ with $\ell, \ell'$ replacing $\ell_j, \ell_{j+1}$
      **if** *$K \subseteq K'$ and $A < A_{\min}$* **then**
        $\ell_j := \ell$
        $\ell_{j+1} := \ell'$
        $E_{\mathrm{in}} := E_{\mathrm{in}} \cup \{j-1, j+2\}$
      $\ell :=$ line through $v_{r-1}$ and $v_r$
      $\ell' :=$ line in $\mathcal{L}(v_s)$ such that $v_s$ is equidistant to $\ell' \cap \ell$ and $\ell' \cap \ell_{j+2}$
      Repeat test in 26–30 (and analogously for $\ell' :=$ line through $v_s, v_{s+1}$ and $\ell \in \mathcal{L}(v_r)$)
    $E_{\mathrm{in}} := E_{\mathrm{in}} \setminus \{j, j+1\}$
**Return:** $\{\ell_1, \ldots, \ell_n\}$

Our goal is to find the line $\ell_j'' \in \mathcal{L}(v_r)$ that minimizes the area of the polygon that is obtained from $K_{m-1}$ by replacing $\ell_j$ by $\ell_j''$. The only parts of the polygon that are affected by this change are the two triangles formed by $\{v_r, a_{j-1}, a_{j-1}''\}$ and $\{v_r, a_{j+1}', a_{j+1}''\}$. As we rotate the line $\ell_j''$, one of these triangles gets smaller while the area of the other increases, and hence, we have to find the location for $\ell_j''$ where the sum of their two areas is minimal. Using (3) the latter is calculated as follows:

$$\frac{1}{2}\det\begin{pmatrix} x'' & x \\ y'' & y \end{pmatrix} + \frac{1}{2}\det\begin{pmatrix} c'x' & c''x'' \\ c'y' & c''y'' \end{pmatrix} = \frac{1}{2}\big(x'y - x\,y'\big)\big(t\,(c'c'' - 1) + 1\big). \quad (5)$$

We first look at the special case $c = c'$, which corresponds to the lines $\ell_{j-1}$ and $\ell_{j+1}$ being parallel. In this case (4) implies that $c''$ does not depend on $t$ and we have $c'' = c$. Hence the area is a linear function in $t$ and therefore attains its minimum at one of the boundaries of the interval $[0, 1]$.

Next, we deal with the general case assuming $c \neq c'$. Using (3), Equation (4) allows us to express $t$ in terms of $c''$ (note that the line $\ell_j''$ is uniquely determined by $c''$):

$$t = \frac{c\,(c' - c'')}{c''\,(c' - c)}.$$

Substituting this into (5) (and omitting the irrelevant factor of $\frac{1}{2}$), we obtain the following objective function $s(c'')$, which we will have to minimize:

$$s(c'') = (x'y - x\,y') \cdot \frac{c'}{c - c'} \cdot \frac{(1 - c'c'')\,c - (1 - c\,c'')\,c''}{c''}.$$

By omitting the constant factor in front, and by taking the derivative, we obtain the expression

$$c \cdot \frac{(c'')^2 - 1}{(c'')^2},$$

which vanishes for $c'' = -1$ (recall that we assume $c''$ to be negative). But we have to take into account that not all $c'' \in (-\infty, 0)$ yield a line $\ell_j''$ that lies in $\mathcal{L}(v_r)$. More precisely, $c''$ must lie in the closed interval defined by $c$ and $c'$. Hence, the claim is proven by observing that the minimum is attained at $c'' = c$ or $c'' = c'$ or $c'' = -1$ (the latter can only happen if $c \leq -1 \leq c'$ or $c' \leq -1 \leq c$). □

We define the reflection of a point $p$ across another point $v$ as follows:

$$\rho_v(p) := 2v - p.$$

The reflection of a line $\ell \subset \mathbb{R}^2$ across a point $v$ is then obtained by reflecting all points of $\ell$ across $v$:

$$\rho_v(\ell) := \{\rho_v(p) \mid p \in \ell\}.$$

The following theorem tells us how we have to move two neighboring swinging lines such that the area is minimized. More precisely, we give a construction of the optimal intersection point of the two lines.

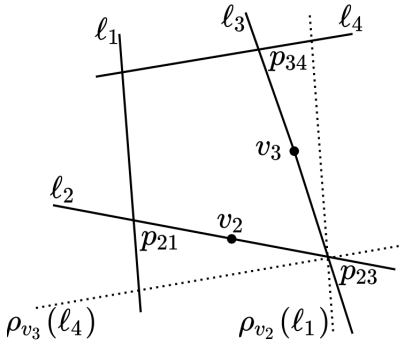



Fig. 2. Illustration for the proof of Theorem 2; the dotted lines are the reflections of $\ell_1$ and $\ell_4$.

**Theorem 2.** *Let $\ell_1, \ell_2, \ell_3, \ell_4$ be distinct lines in the plane such that $\ell_1$ and $\ell_4$ are not parallel. Let $v_2 \in \ell_2$ and $v_3 \in \ell_3$. For $i, j$ distinct in $\{1, 2, 3, 4\}$, let $p_{ij}$ denote the intersection point of $\ell_i$ and $\ell_j$ (undefined if the lines are parallel), see Fig. 2. Assume that*

$$\|v_2 - p_{12}\| = \|v_2 - p_{23}\|, \quad (6)$$
$$\|v_3 - p_{23}\| = \|v_2 - p_{34}\| \quad (7)$$

*hold. Then $p_{23}$ is the intersection of $\rho_{v_2}(\ell_1)$ and $\rho_{v_3}(\ell_4)$.*

*Proof.* Condition (6) can be reformulated as

$$\rho_{v_2}(p_{23}) \in \ell_1, \quad (8)$$

i.e., if we reflect the point $p_{23}$ across $v_2$ we must land on $\ell_1$. Applying the involution $\rho_{v_2}$ on both sides of (8), we obtain $p_{23} \in \rho_{v_2}(\ell_1)$. Doing the same reasoning on condition (7), we obtain $p_{23} \in \rho_{v_3}(\ell_4)$. The claim follows. □

## III. Experimental results and discussion

The natural question to ask now is: how well does our algorithm perform in practice? We have already noted that the Algorithm 1 finds a local minimum and is not guaranteed to deliver the global minimum. Indeed, one can easily construct examples where the algorithm gets stuck in a local minimum. An example where a hexagon is to be approximated by a triangle is displayed in Fig. 3: clearly none of the three lines can be moved to a neighboring edge without increasing the area of the dashed triangle. In order to move to the global minimum, all three lines would have to be moved at the same time. This example illustrates another phenomenon: the error that our algorithm produces cannot be bounded by the error of the optimal solution. In the example, if we make the short edges of the hexagon shorter and shorter, the error of the optimal solution goes to zero, while the error of the unfavorable local minimum basically stays the same.

Luckily the example from Fig. 3 is a very degenerate one, so that we can hope that our algorithm performs better on more generic examples. We carried out some experiments where we aim at approximating a polygon $K$ with 10 vertices

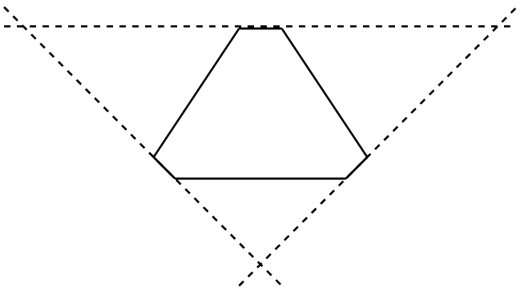


Fig. 3. Example of a locally optimal solution that is far away from the global optimum.

by a simpler one. For this purpose, we randomly generated convex 10-gons and compared the solution of our algorithm with the "optimal" solution. The latter was found by a brute-force approach, trying all ordered subsets of $E(K)$ that formed a convex bounded polygon. The results for $n = 3, 5, 7$ are displayed in Figs. 4 and 5, where we define the approximation error to be the area of the approximating polygon divided by the area of the original polygon $K$.

Some interesting phenomena occur here. The most striking observation is that in the case $n = 3$, the approximation error of our algorithm is often smaller than the one of the brute-force algorithm. This can be explained by the occurrence of swinging lines, which the brute-force algorithm does not take into account. For $n = 5$ we see that our algorithm overshoots the area of $K$ up to $13\%$, while the error of the optimal solution is between $10\%$ and $12\%$ in our sample set. We note that for more than half of the samples (51 out of 100), the output of our algorithm is as good as the solution of the brute-force algorithm. For $n = 7$ the approximation error becomes smaller in general, as expected, and our algorithm gets closer to the optimal solution.

In contrast to the above experiments, a polygon approximation via neural networks does not provide the same controllability and predictability. An output convex polygon depends on the architecture of the neural network and the values of the network's weights. The weights are calculated during their optimization by using a predefined loss function and optimization algorithm. The loss function and the optimization algorithm have a bunch of predefined parameters, for example: the learning rate during the weights optimization, the weight decay constant, the momentum factor, etc. All these factors imply how the optimization process is done and what network we will get at the end. Moreover, the size and quality of the training data set has a direct impact on the output polygon.

As a result, by using the same neural network architecture, optimization method and loss function, and by changing their parameters we can get convex polygons with different shapes, inner area and number of boundary edges. We carried out some experiments where we aimed to approximate a polygon $K$ with ten vertices by a neural network. For this purpose we randomly generated a convex 10-gon and trained several neural networks with a single hidden layer that consisted of five units. We used the ReLU function as non-linear function in the hidden layer and the sigmoid function for the output value. As a loss function, we used binary cross entropy and as an optimizer we used ADAM [6].

We trained networks for data sets with size $n = 10$, $n = 100$, $n = 1000$. The results are displayed in Figs. 6 and 7. Note that the output polygons usually do not completely contain the input polygon $K$. Moreover, different sizes of the data set can provide output polygons with a different number of boundary edges, which can even be open (unbounded). The most ambiguous output convex set is obtained by neural networks that were trained only on ten inner and outer points. The bigger the data set is, the better is the achieved approximation, see Fig 6.

The dependence of the approximation convex polygon shape from the type of loss function, optimization algorithm, data set, etc. is depicted in Fig 7. We used the following optimization algorithms: ADAM, RMSProp [5] and SGD [2], to train a neural network on the same data set with 1000 inner and outer points. We achieved approximation convex polygons with different shapes but with the same number of edges – 6.

After all these experiments, the natural question arises: how can one tune all these parameters to be able to build an approximation polygon that fully contains the input polygon $K$ and whose area is as small as possible? Moreover, is there an approach to build a neural network incorporating geometric properties of the input convex set $K$? The future research of these questions can lead to a better understanding how neural networks work and how to train them in a more controllable way.

## REFERENCES


[1] Helmut Alt, and Leonidas J. Guibas. "Discrete geometric shapes: Matching, interpolation, and approximation." Handbook of computational geometry. North-Holland, 2000, pp. 121–153.

[2] Lon Bottou. "Stochastic gradient descent tricks." Neural networks: Tricks of the trade. Springer, Berlin, Heidelberg, 2012. 421-436.

[3] Arne Brondsted. An introduction to convex polytopes. Vol. 90. Springer Science & Business Media, 2012.

[4] Yehoram Gordon, Mathieu Meyer, and Shlomo Reisner. "Constructing a polytope to approximate a convex body." Geometriae Dedicata 57.2 1995 pp. 217–222.

[5] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent." 2012.

[6] Diederik P. Kingma, and Jimmy Ba. "Adam: A Method for Stochastic Optimization". 3rd International Conference on Learning Representations, ICLR 2015.

[7] Mario A. Lopez, and Shlomo Reisner. "Hausdorff approximation of convex polygons." Computational Geometry 32.2 (2005) pp. 139–158.

[8] Guido Montufar et al. On the Number of Linear Regions of Deep Neural Networks. Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. NIPS14. Montreal, Canada, 2014, pp. 2924–2932.

[9] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. "On the number of response regions of deep feed forward networks with piece-wise linear activations." arXiv preprint arXiv:1312.6098 (2013).

[10] Natalia Shepeleva, et al. "ReLU Code Space: A Basis for Rating Network Quality Besides Accuracy." arXiv preprint arXiv:2005.09903 2020.

[11] Juyoung Yon, et al. "Approximating convex shapes with respect to symmetric difference under homotheties." 32nd International Symposium on Computational Geometry (SoCG 2016). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

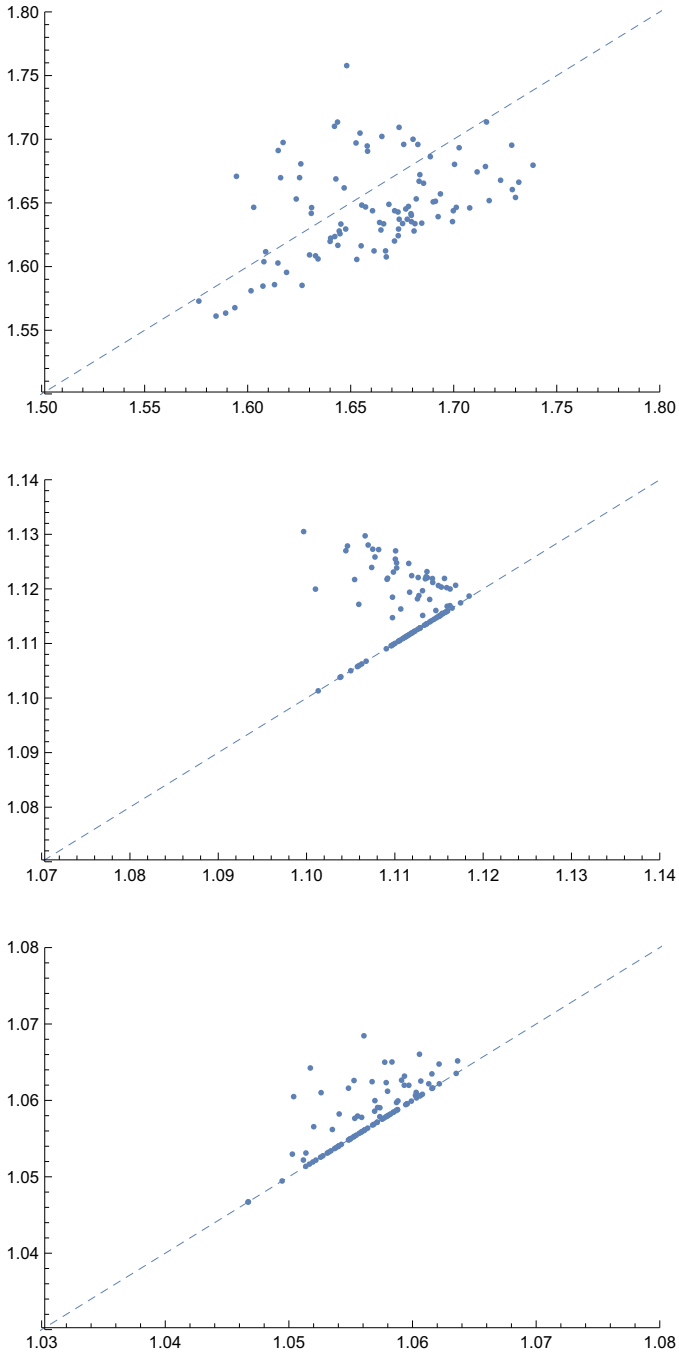



Fig. 4. Plot of the approximation error of the brute-force algorithm (horizontal axis) versus the approximation error of the proposed algorithm (vertical axis) for $n = 3, 5, 7$.

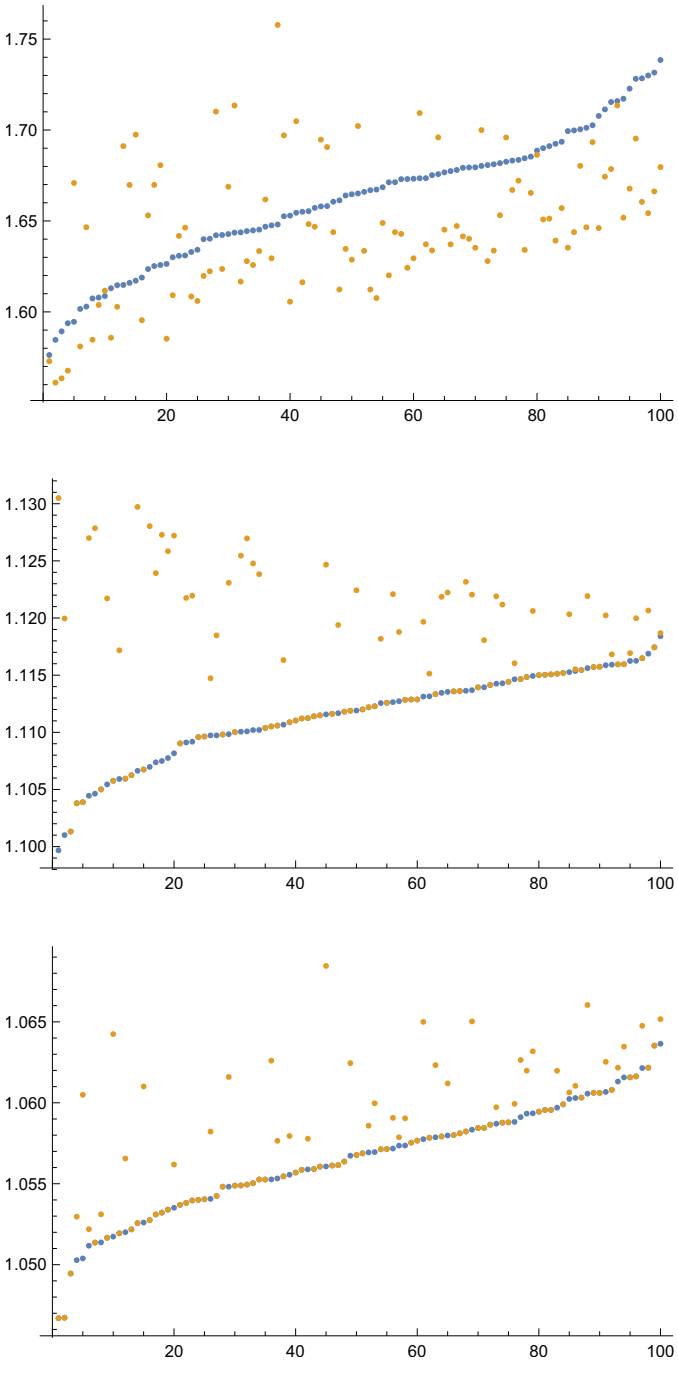
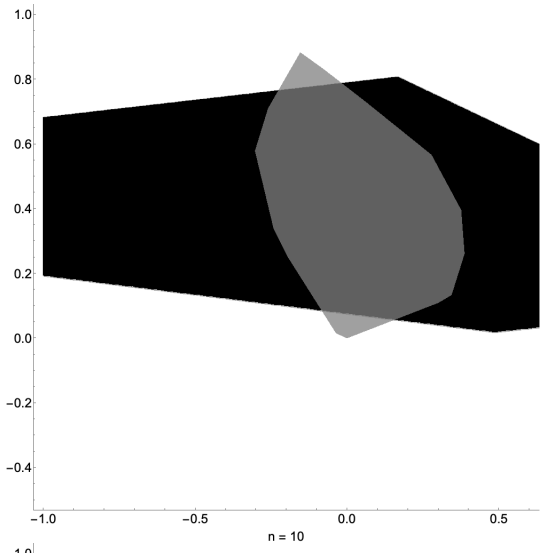
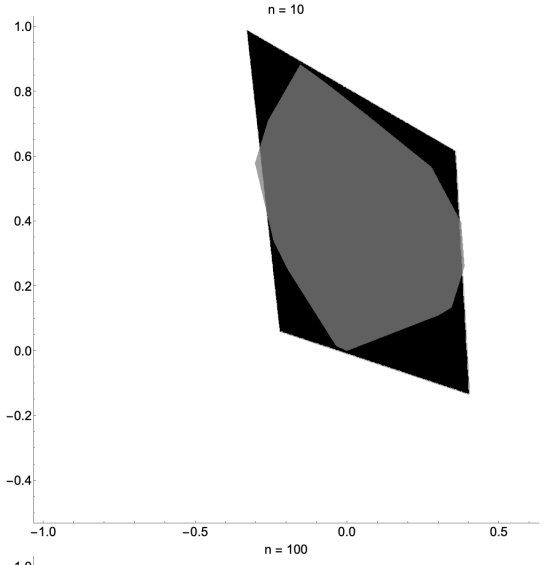



Fig. 5. Plot of the approximation errors of the two algorithms case by case (100 samples) for $n = 3, 5, 7$. The results are sorted by the error of the brute-force algorithm (blue dots). The error of our algorithm for the same sample appears as an orange dot with the same $x$-coordinate.



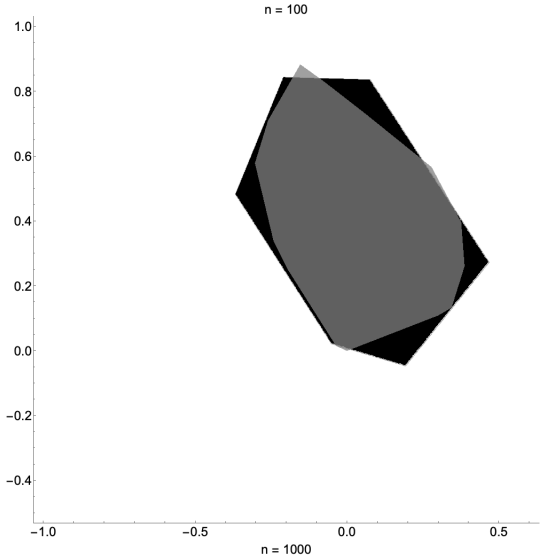
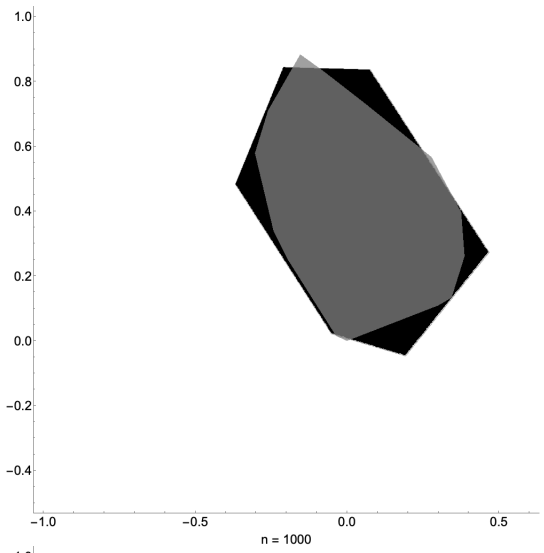
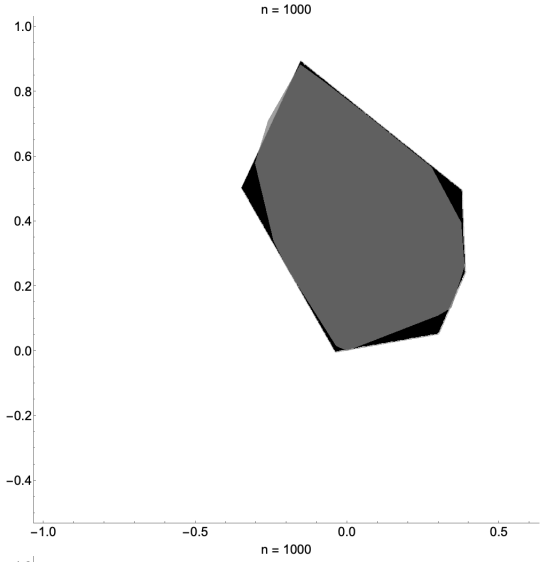




Fig. 6. Plot of the approximation sets (black) of the input convex set (gray) with different size of training data set $n = 10$, $n = 100$, $n = 1000$. For different data sets the approximation polygons have different sizes, shapes and number of edges. Also, from the plots it can be seen that the approximation polygons do not fully contain input polygon $K$.



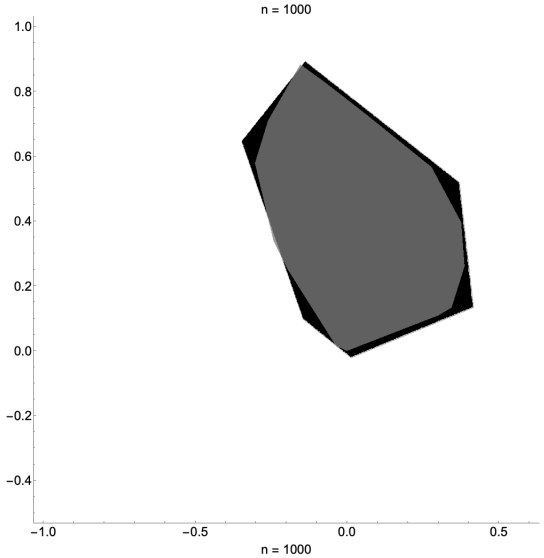




Fig. 7. Plot of the approximation polygons (black) of the input convex set (gray) for the data set with size $n = 1000$ by using different optimization algorithms (ADAM, RMSProp, SGD).