PITZER COLLEGE

Untying Knots with Neural Nets

Matthew vonAllmen

David Bachman, Advisor
, Reader

Submitted to Pitzer College in Partial Fulfillment
of the Degree of Bachelor of Arts

January 5, 2021

Department of Mathematics

Abstract

Neural networks can transform 3-dimensional data in a manner reminiscent of an ambient isotopy. With some modifications, a neural network can be trained to manipulate the vertices of a knot while respecting its topological structure. We use the discrete Möbius energy as a loss function to incentivize a neural network to smooth out curves in a knot, without performing illegal operations. By introducing unconventional neural network layers, we are able to untwist highly tangled polygonal knots until a human can visually recognize whether they are topologically equivalent to the unknot.

Contents

Acknowledgments

Thank you to Professor David Bachman, my mentor and advisor throughout the writing of this thesis. Your guidance and feedback was invaluable; you have my heartfelt appreciation for sticking with me even as the scope of this project ballooned far beyond its original parameters.

Chapter 1

Introduction and Definitions

In this paper, we introduce a new heuristic method for converting high-energy knots into lower energy variants in the same equivalence class. We accomplish this by running the vertices of a polygonal knot through a specialized neural network, designed to reduce knot energy via operations akin to Reidemeister moves.

We begin by defining some basic terms in knot theory, and will then connect these concepts to the world of neural networks.

1.1 Basic Knot Theory

1.1.1 Knots

A *knot* is defined as a closed curve in 3-dimensional space. More formally, we say that a knot is a continuous function $K : [0, 1] \to \mathbb{R}^3$ such that $K(0) = K(1)$ and for all $x, y \in [0, 1)$,

$$K(x) = K(y) \Rightarrow x = y.$$

In other words, K is "nearly" injective. The only two inputs which it does not map to different elements in $\mathbb{R}^3$ are 0 and 1.

A knot is *polygonal* if it is a piecewise linear function, whose image is the union of a finite set of line segments in $\mathbb{R}^3$ such that the intersection of any two line segments consists of at most a single point. The endpoints of these line segments are called the *vertices* of the polygonal knot. It is possible to represent a polygonal knot by an ordered list of its vertices. This representation is most suitable for neural networks, which operate on arrays of data.

1.1.2 Ambient Isotopies

In 3-dimensional space, an *ambient isotopy* $h : \mathbb{R}^3 \times [0,1] \to \mathbb{R}^3$ is a continuous map such that the curried function

$$h(\cdot, 0)$$

is the identity function, and for all $t \in [0,1]$,

$$h(\cdot, t)$$

is a homeomorphism from $\mathbb{R}^3$ to itself, where a *homeomorphism* is a continuous function with a continuous inverse. In order to simplify notation, we often write $h_t(x)$ instead of $h(x,t)$.

One can intuitively think of ambient isotopies as continuous deformations in space.

We can construct equivalence classes out of knots that are continuous deformations of one another. Formally, two knots K_1, K_2 are equivalent if there exists an ambient isotopy h such that

$$h_1 \circ K_1 = K_2.$$

We write $K_1 \sim K_2$ to denote their equivalence. We may also say that K_1 and K_2 are *ambient isotopic*.

1.1.3 Continuity and Metrics

On the half-open interval $[0,1)$, we define the *interval metric* d_I such that for all $t, t' \in [0,1)$,

$$d_I(t,t') = \min(|t-t'|, \min(t, 1-t) + \min(t', 1-t')).$$

The interval metric treats elements of $[0,1)$ as positions along a circle with unit-length circumference, and returns the shortest path between two positions along the circle.

Chapter 2

Neural Networks as Ambient Isotopies

Neural networks are constructed out of layers. Each layer accepts input from the previous layer, performs a set of operations upon the input, and then passes the result to the next layer. We can abstractly define a layer L as a function from $\mathbb{R}^{d_1}$ to $\mathbb{R}^{d_2}$, for $d_1, d_2 \in \mathbb{Z}^+$. A neural network as a whole is a composition of its layers. In this paper, all layers will map from $\mathbb{R}^3$ to $\mathbb{R}^3$.

We say that the composition of one or more neural network layers $N : \mathbb{R}^3 \to \mathbb{R}^3$ *implements an ambient isotopy* if and only if there exists an ambient isotopy h such that

$$h_1 = N.$$

Theorem 1. *For any two neural networks N, M that implement an ambient isotopy, their composition $N \circ M$ also implements an ambient isotopy.*

Proof. Let h^N and h^M be ambient isotopies implemented by N and M, respectively. Then, consider the function $f : \mathbb{R}^3 \times [0,1] \to \mathbb{R}^3$ defined as follows:

$$f(x,t) = \begin{cases} h^N_{2t-1}(h^M_1(x)) & t > \frac{1}{2} \\ h^M_{2t}(x) & t \leq \frac{1}{2} \end{cases}$$

For any $t \in [0, \frac{1}{2}]$, $f(\cdot, t) = h^M_{2t}$ is a homeomorphism. Since the composition of two homeomorphisms is also a homeomorphism, for any $t \in (\frac{1}{2}, 1]$, $f(\cdot, t) = h^N_{2t-1} \circ h^M_1$ is a homeomorphism as well. $f(\cdot, t)$ is thus a homeomorphism for all $t \in [0, 1]$.

Next, we note that $f(\cdot, 0) = h_0^M$, and as h^M is an ambient isotopy, $h_0^M = f(\cdot, 0)$ must be the identity function.

Additionally, since both h^N and h^M are continuous functions and $h_{2t-1}^N \circ h_1^M = h_{2t}^M$ when $t = \frac{1}{2}$, f is also a continuous function. This demonstrates that f satisfies all the properties of an ambient isotopy.

Lastly, since

$$f(\cdot, 1) = h_1^N \circ h_1^M = N \circ M,$$

$N \circ M$ implements an ambient isotopy. □

From theorem 1, we can conclude that if a neural network performs a sequence of operations that all implement ambient isotopies, then the neural network as a whole must implement an ambient isotopy.

2.1 Operating on Polygonal Knots

Often, neural network layers will act on an array of inputs, rather than a single input. As shorthand, we write

$$L([\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}}]) := [L(\overrightarrow{x_0}), ..., L(\overrightarrow{x_{n-1}})]$$

whenever a layer $L : \mathbb{R}^3 \to \mathbb{R}^3$ acts on an array of $n \in \mathbb{Z}^+$ inputs $\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}} \in \mathbb{R}^3$. By convention, elements of arrays will be zero-indexed.

Unfortunately, neural networks can only accept a finite number of inputs at the same time, and so cannot operate on the entire image of a polygonal knot. Instead, a neural network can operate on a polygonal knot's vertices, and we can establish a correspondence between those vertices and the knot itself.

2.1.1 The φ function

Consider an ordered list of n 3D coordinates $[\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}}]$ such that for any two integers $i, j \in \{0, ..., n-1\}$ where $i \neq j$,

1. $\overrightarrow{x_i} \neq \overrightarrow{x_j}$, and
2. For all $\lambda_1, \lambda_2 \in [0, 1)$,

$$\lambda_1 \overrightarrow{x_i} + (1 - \lambda_1)\overrightarrow{x_{i^+}} \neq \lambda_2 \overrightarrow{x_j} + (1 - \lambda_2)\overrightarrow{x_{j^+}}$$

where $i^+ \triangleq (i + 1) \bmod n$ and $j^+ \triangleq (j + 1) \bmod n$.

Let $\mathcal{K}$ denote the set of all finite-length arrays containing elements of $\mathbb{R}^3$ which satisfy these properties, and let $\overline{\mathcal{K}}$ denote the set of finite-length arrays which do not satisfy them. An immediate corollary is that all arrays in $\mathcal{K}$ must have at least 3 elements.

There is a convenient correspondence between $\mathcal{K}$ and the set of all polygonal knots. From any ordered list in $\mathcal{K} \cup \overline{\mathcal{K}}$ with n elements, we can construct a function $K : [0,1] \to \mathbb{R}^3$ where

$$K(t) = (1 - (nt \bmod 1))\overrightarrow{x_{i(t)}} + (nt \bmod 1)\overrightarrow{x_{i^+(t)}}$$

for $i(t) = \lfloor nt \rfloor \bmod n$ and $i^+(t) = (\lfloor nt \rfloor + 1) \bmod n$.

The function K linearly interpolates between the adjacent elements of $[\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}}]$, and then linearly interpolates between $\overrightarrow{x_{n-1}}$ and $\overrightarrow{x_0}$. It allocates an equally-sized chunk of the domain $[0,1]$ to each interpolating line segment. If $[\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}}] \in \mathcal{K}$, then K is a polygonal knot.

Let φ denote the function which takes elements of $\mathcal{K} \cup \overline{\mathcal{K}}$ and maps them to this linear interpolation between their elements. φ has several useful properties.

Theorem 2. *For $V = [\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}}], W = [\overrightarrow{y_0}, ..., \overrightarrow{y_{n-1}}] \in \mathcal{K} \cup \overline{\mathcal{K}}$ and scalars $a, b \in \mathbb{R}$,*

$$\varphi(aV + bW) = a\varphi(V) + b\varphi(W).$$

Proof. Let $i(t) \triangleq \lfloor nt \rfloor \bmod n$ and $i^+(t) \triangleq (\lfloor nt \rfloor + 1) \bmod n$. For $t \in [0,1]$, we compute:

$$\begin{aligned}
&\varphi(aV + bW)(t) \\
&= (1 - (nt \bmod 1))(a\overrightarrow{x_{i(t)}} + b\overrightarrow{y_{i(t)}}) + (nt \bmod 1)(a\overrightarrow{x_{i^+(t)}} + b\overrightarrow{y_{i^+(t)}}) \\
&= (1 - (nt \bmod 1))a\overrightarrow{x_{i(t)}} + (nt \bmod 1)a\overrightarrow{x_{i^+(t)}} + (1 - (nt \bmod 1))b\overrightarrow{y_{i(t)}} + (nt \bmod 1)b\overrightarrow{y_{i^+(t)}} \\
&= a((1 - (nt \bmod 1))\overrightarrow{x_{i(t)}} + (nt \bmod 1)\overrightarrow{x_{i^+(t)}}) + b((1 - (nt \bmod 1))\overrightarrow{y_{i(t)}} + (nt \bmod 1)\overrightarrow{y_{i^+(t)}}) \\
&= a\varphi(V)(t) + b\varphi(W)(t).
\end{aligned}$$

□

Theorem 3. *For all $V \in \mathcal{K} \cup \overline{\mathcal{K}}$ and $W \in \mathcal{K}$ such that both V and W contain at least 4 elements,*

$$\varphi(V) \circ \varphi(W)^{-1}$$

is Lipschitz, where $\varphi(V) \circ \varphi(W)^{-1}$ is the composition of $\varphi(V)$ with the inverse of $\varphi(W)$.

Proof. Let $V \in \mathcal{K} \cup \overline{\mathcal{K}}$ and $W \in \mathcal{K}$ be given. We will demonstrate that the value of

$$\frac{\|(\varphi(V) \circ \varphi(W)^{-1})(\vec{x}) - (\varphi(V) \circ \varphi(W)^{-1})(\vec{y})\|}{\|\vec{x} - \vec{y}\|}$$

is bounded above for all $\vec{x}, \vec{y}$ in the image of $\varphi(W)$, i.e. the domain of $\varphi(V) \circ \varphi(W)^{-1}$.

First, consider a restriction of $\varphi(W)$ and $\varphi(V)$ to the domain $[0, 1)$, and equip this domain with the interval metric. We will show that

$$\frac{\|\varphi(V)(t) - \varphi(V)(t')\|}{d_I(t, t')}$$

and

$$\frac{d_I(t, t')}{\|\varphi(W)(t) - \varphi(W)(t')\|}$$

are bounded above for all $t, t' \in [0, 1)$ such that $t \neq t'$. To do so, we will examine in turn values of t and t' that produce points on the same line segment, on adjacent line segments, and on non-adjacent line segments. This will demonstrate that the restriction of $\varphi(V)$ is Lipschitz and that the inverse of the restriction of $\varphi(W)$ is Lipschitz.

Let n_V and n_W be the number of vertices in V and W, respectively. Let $\vec{x}$ and $\vec{y}$ be adjacent vertices in V. If t and t' are in the closed interval that is the preimage of the line segment between $\vec{x}$ and $\vec{y}$, we have

$$\frac{\|\varphi(V)(t) - \varphi(V)(t')\|}{d_I(t, t')} = \frac{\|\vec{x} - \vec{y}\|}{1/n_V},$$

and since there are a finite number of pairs of adjacent vertices in V, these values can be bounded above. If instead $\vec{x}$ and $\vec{y}$ are adjacent vertices in W, we have

$$\frac{\|\varphi(W)(t) - \varphi(W)(t')\|}{d_I(t, t')} = \frac{\|\vec{x} - \vec{y}\|}{1/n_W}.$$

Since $W \in \mathcal{K}$, we have $\vec{x} \neq \vec{y}$, so

$$\frac{1/n_W}{\|\vec{x} - \vec{y}\|}$$

is finite and well-defined. This establishes an upper bound on

$$\frac{d_I(t,t')}{\|\varphi(W)(t)-\varphi(W)(t')\|}$$

for t, t' along the same line segment.

Now consider $\vec{x}$, $\vec{y}$, and $\vec{z}$, where $\vec{x}$ and $\vec{y}$ are adjacent vertices, and so are $\vec{y}$ and $\vec{z}$. In other words, the line segment from $\vec{x}$ to $\vec{y}$ is adjacent to the line segment from $\vec{y}$ to $\vec{z}$, and they share $\vec{y}$ as an endpoint.

Suppose t is in the closed interval between the preimage of $\vec{x}$ and $\vec{y}$ and t' is in the closed interval between the preimage of $\vec{y}$ and $\vec{z}$. Let $a \in [0,1]$ be the fraction of the distance traversed from $\vec{y}$ to $\vec{x}$ to reach the image of t, and let $b \in [0,1]$ be the fraction of the distance traversed from $\vec{y}$ to $\vec{z}$ to reach the image of t'. Then, we have

$$\begin{aligned}\frac{\|\varphi(V)(t)-\varphi(V)(t')\|}{d_I(t,t')} &= \frac{\|a(\vec{x}-\vec{y})+\vec{y}-b(\vec{z}-\vec{y})-\vec{y}\|}{\frac{a}{n_V}+\frac{b}{n_V}} \\ &= \frac{\|a(\vec{x}-\vec{y})-b(\vec{z}-\vec{y})\|}{(a+b)\frac{1}{n_V}} \\ &= \frac{\|\frac{a}{a+b}(\vec{x}-\vec{y})+\frac{b}{a+b}(\vec{y}-\vec{z})\|}{1/n_V}.\end{aligned}$$

The coefficients $\frac{a}{a+b}$ and $\frac{b}{a+b}$ range over values in $[0,1]$ that together sum to 1. Therefore, we can rewrite the above as

$$\frac{\|\frac{a}{a+b}(\vec{x}-\vec{y})+\frac{b}{a+b}(\vec{y}-\vec{z})\|}{1/n_V} = \frac{\|q(\vec{x}-\vec{y})+(1-q)(\vec{y}-\vec{z})\|}{1/n_V}$$

for $q \in [0,1]$. This is a continuous function of q over a compact domain, and so the function must have a maximum and minimum value. Therefore,

$$\frac{\|\varphi(V)(t)-\varphi(V)(t')\|}{d_I(t,t')}$$

is bounded above for t, t' in the preimage of adjacent line segments.

If instead $\vec{x}$, $\vec{y}$, and $\vec{z}$ are vertices in W, we have

$$\frac{\|\varphi(W)(t)-\varphi(W)(t')\|}{d_I(t,t')} = \frac{\|q(\vec{x}-\vec{y})+(1-q)(\vec{y}-\vec{z})\|}{1/n_W},$$

where

$$\|q(\vec{x}-\vec{y})+(1-q)(\vec{y}-\vec{z})\|$$

has a minimum equal to zero if and only if there exists a $q \in [0, 1]$ such that

$$q(\vec{x} - \vec{y}) = (1 - q)(\vec{z} - \vec{y}).$$

This would violate the properties of $\mathcal{K}$, and so the minimum value must be greater than zero. This establishes an upper bound on

$$\frac{d_I(t, t')}{\|\varphi(W)(t) - \varphi(W)(t')\|}$$

for t, t' along the preimages of adjacent line segments.

Lastly, we must consider the case where t and t' lie on the preimages of non-adjacent line segments. For non-adjacent line segments in V, the distance between the preimages of two vertices is bounded below by $1/n_V$. Then, since the image of $\varphi(V)$ is bounded, this establishes an upper bound on

$$\frac{\|\varphi(V)(t) - \varphi(V)(t')\|}{d_I(t, t')}$$

for t, t' in the preimages of non-adjacent line segments.

For non-adjacent line segments in W, we can make use of the fact that non-overlapping line segments have a minimum distance between them. Since there are a finite number of pairs of non-adjacent line segments in W and no two overlap, there is an upper bound for

$$\frac{d_I(t, t')}{\|\varphi(W)(t) - \varphi(W)(t')\|}$$

when t, t' are in the preimages of non-adjacent line segments.

This completes our proof that our restricted version of $\varphi(V)$ is Lipschitz and that the inverse of our restricted version of $\varphi(W)$ is Lipschitz. Since the composition of two Lipschitz functions is itself Lipschitz and $\varphi(V) \circ \varphi(W)^{-1}$ is the same function regardless of whether $\varphi(V)$ or $\varphi(W)$ have their domains restricted,

$$\varphi(V) \circ \varphi(W)^{-1}$$

is Lipschitz. □

Theorem 3 is true even without the extra constraint that V and W contain at least four elements, though the proof is more complicated. Both theorems 2 and 3 will be useful in demonstrating a correspondence between polygonal knots and arrays of their vertices.

2.1.2 Respecting polygonality

For a neural network $N : \mathbb{R}^3 \to \mathbb{R}^3$ and array of vertices $V \in \mathcal{K}$, we say N *respects the polygonality of* V if and only if

$$N(V) \in \mathcal{K} \quad \text{and} \quad N \circ \varphi(V) \sim \varphi(N(V)).$$

Less formally, suppose a neural network with this property operated on the vertices of a polygonal knot, and then line segments were drawn between the resulting points in $\mathbb{R}^3$. This knot would be equivalent to the image of the original polygonal knot when run through the neural network.

Theorem 4. *Let N, M be any two neural networks that are both homeomorphisms. If M respects the polygonality of a set of vertices $V \in \mathcal{K}$ and N respects the polygonality of $M(V)$, then their composition $N \circ M$ respects the polygonality of V.*

Proof. Since N respects the polygonality of $M(V)$, we have $N(M(V)) = (N \circ M)(V) \in \mathcal{K}$. Therefore, $N \circ M$ satisfies the first property needed to respect the polygonality of V.

Next, to show the second property, we begin by noting that since M respects the polygonality of V,

$$M \circ \varphi(V) \sim \varphi(M(V)).$$

Furthermore, because N is a homeomorphism, it preserves the above equivalence relation, and we have

$$N \circ M \circ \varphi(V) \sim N \circ \varphi(M(V)).$$

Then, since N respects the polygonality of $M(V)$, we have

$$N \circ \varphi(M(V)) \sim \varphi(N(M(V))),$$

and applying the transitive property of the $\sim$ relation to the above,

$$N \circ M \circ \varphi(V) \sim \varphi(N(M(V))).$$

This can be rewritten as $(N \circ M) \circ \varphi(V) \sim \varphi((N \circ M)(V))$. $N \circ M$ thus respects the polygonality of V. □

Theorem 4 demonstrates that if a neural network is composed of layers that are each homeomorphisms and respect the polygonality of the input

received from the layer that precedes them, and where the first layer respects the polygonality of the original input, then the neural network as a whole respects the polygonality of its input.

A neural network that both implements an ambient isotopy and respects the polygonality of its input is able to perform "safe" operations on a knot's vertices. When an observer reconstructs a polygonal knot from the neural network's output, they can be sure that it will be equivalent to the polygonal knot which corresponded to the original vertices. Or, more formally:

Theorem 5. *Let N be a neural network that both implements an ambient isotopy and respects the polygonality of an array of vertices $V \in \mathcal{K}$. Then,*

$$\varphi(N(V)) \sim \varphi(V).$$

Proof. As N implements an ambient isotopy,

$$N \circ \varphi(V) \sim \varphi(V),$$

and since N respects the polygonality of V,

$$N \circ \varphi(V) \sim \varphi(N(V)).$$

Therefore, since $\sim$ is an equivalence relation, we have

$$\varphi(N(V)) \sim \varphi(V).$$

□

There are numerous applications for neural networks that both implement an ambient isotopy and respect the polygonality of their input. For example, suppose such a neural network N acted on the vertices of a polygonal knot $V \in \mathcal{K}$, and the resulting polygonal knot $\varphi(N(V))$ was visually identifiable as the unknot. This would be a sufficient condition for the original polygonal knot $\varphi(V)$, no matter how visually complicated or tangled it may appear, to be equivalent to the unknot as well.

Unfortunately, guaranteeing that each layer of a neural network respects the polygonality of the input received from the previous layer is a difficult task. There is a limited class of functions that respect the polygonality of all possible sets of vertices, but the kinds of spatial deformations they can perform is highly constrained.

Theorem 6. *If $f : \mathbb{R}^3 \to \mathbb{R}^3$ is a positive affine transformation, then f both implements an ambient isotopy and respects the polygonality of all arrays of vertices $V \in \mathcal{K}$.*

Proof. If f is a positive affine transformation, there must exist a 3 by 3 matrix $\mathbf{A}$ and vector $\vec{b} \in \mathbb{R}^3$ such that for all $\vec{x} \in \mathbb{R}^3$,

$$f(\vec{x}) = \mathbf{A}\vec{x} + \vec{b},$$

where $\mathbf{A}$ has real entries and $\det(\mathbf{A}) > 0$.

First, we will demonstrate that f implements an ambient isotopy. Since $\mathbf{A}$ has a positive determinant, f is both invertible and orientation-preserving. Its inverse is the function f^{-1} where

$$f^{-1}(\vec{x}) = \mathbf{A}^{-1}\vec{x} - \mathbf{A}^{-1}\vec{b}.$$

Using a property of the determinant,

$$\det(A^{-1}) = \frac{1}{\det(A)} > 0,$$

meaning that f^{-1} is also orientation-preserving. Furthermore, since all bounded linear operators (such as $\mathbf{A}$ and $\mathbf{A}^{-1}$) are continuous, the translations defined by $\vec{x} \mapsto \vec{x} + \vec{b}$ and $\vec{x} \mapsto \vec{x} - \mathbf{A}^{-1}\vec{b}$ are continuous, and the composition of two continuous functions is itself continuous, f and f^{-1} are both continuous functions. Therefore, f is an orientation-preserving homeomorphism. Lastly, as every orientation-preserving homeomorphism is the function h_1 for some ambient isotopy h, f must implement an ambient isotopy.

Now, let $V = [\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}}] \in \mathcal{K}$ be given. We will show that f respects the polygonality of V. Recall that by the definition of φ, for $t \in [0, 1]$,

$$\varphi(V)(t) = (1 - (nt \bmod 1))\overrightarrow{x_{i(t)}} + (nt \bmod 1)\overrightarrow{x_{i^+(t)}},$$

where $i(t) = \lfloor nt \rfloor \bmod n$ and $i^+(t) = (\lfloor nt \rfloor + 1) \bmod n$. Composing the function $\varphi(V)$ with f thus yields

$$(f \circ \varphi(V))(t) = f((1 - (nt \bmod 1))\overrightarrow{x_{i(t)}} + (nt \bmod 1)\overrightarrow{x_{i^+(t)}}).$$

Since f is an affine function, and

$$(1 - (nt \bmod 1))\overrightarrow{x_{i(t)}} + (nt \bmod 1)\overrightarrow{x_{i^+(t)}}$$

is a convex combination of $\overrightarrow{x_{i(t)}}$ and $\overrightarrow{x_{i^+(t)}}$, we have

$$\begin{aligned} &f((1-(nt \bmod 1))\overrightarrow{x_{i(t)}} + (nt \bmod 1)\overrightarrow{x_{i^+(t)}}) \\ &= (1-(nt \bmod 1))f(\overrightarrow{x_{i(t)}}) + (nt \bmod 1)f(\overrightarrow{x_{i^+(t)}}). \end{aligned}$$

Therefore,

$$\begin{aligned} (f \circ \varphi(V))(t) &= (1-(nt \bmod 1))f(\overrightarrow{x_{i(t)}}) + (nt \bmod 1)f(\overrightarrow{x_{i^+(t)}}) \\ &= \varphi(f(V))(t) \end{aligned}$$

for all $t \in [0,1]$, so $f \circ \varphi(V) = \varphi(f(V))$. Since knot equality implies knot equivalence, we can also make the weaker statement that $f \circ \varphi(V) \sim \varphi(f(V))$, and thus f respects the polygonality of V. □

If a positive affine function were used as a neural network layer, then regardless of the input passed from the preceding layer, its polygonality would be respected. Sadly, positive affine transformations are closed under composition. As a result, a neural network created by stacking many such layers would be no more powerful than a neural network with just one.

A superior approach would require finding neural network layers that respect the polygonality of *particular* arrays of vertices, rather than *all* arrays of vertices, and ensuring that they only ever act upon inputs whose polygonality they respect.

2.2 Methods for Respecting Polygonality

Neural networks risk failing to respect polygonality when they try to pass one line segment through another. To demonstrate this problem, we'll use an example that can be easily displayed in two dimensions.

Let $(c_1, c_2) \in \mathbb{R}^2$ be some point in the unit circle. Consider a neural network $C : \mathbb{R}^3 \to \mathbb{R}^3$ that maps from cylindrical coordinates to rectangular coordinates as follows:

$$C(r,\theta,z) = \begin{cases} (r(\cos(\theta)-c_1)+c_1, r(\sin(\theta)-c_2)+c_2, z) & r < 1 \\ (r\cos(\theta), r\sin(\theta), z) & r \geq 1 \end{cases}$$

C only acts on points inside a cylinder with radius 1 that stretches along the z-axis. At each cross-section of the cylinder in the x, y-plane, it selects (c_1, c_2) to be the new "center" for the points in that unit disk, and drags them toward it.

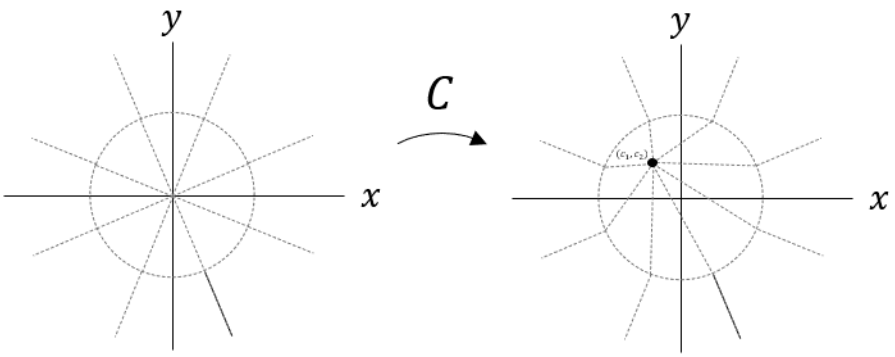
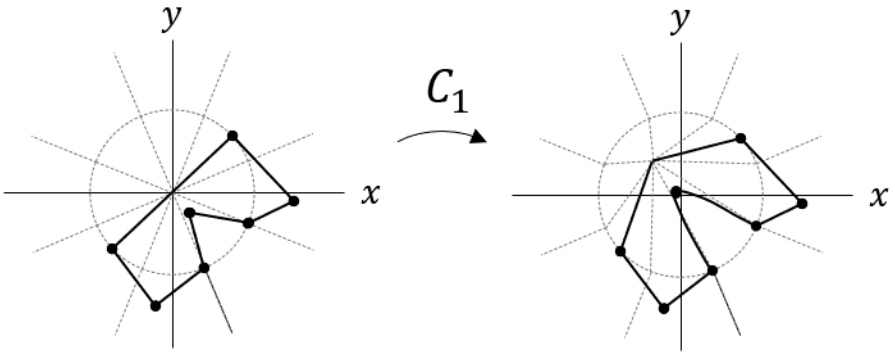


Figure 2.1: The dashed ring represents the unit circle. Lines from the unit circle to the old center become lines between the unit circle and the new center.

C implements an ambient isotopy. The family of functions

$$C_t(r, \theta, z) = tC(r, \theta, z) + (1 - t)((r\cos(\theta), r\sin(\theta), z))$$

is an ambient isotopy such that $C_1 = C$. As a result, C will never cause two parts of a knot to self-intersect as it continuously deforms $\mathbb{R}^3$. We can observe this phenomenon for the following polygonal knot with 7 vertices:



However, if it only acts on the vertices of the knot and we redraw the edges at each time step t, two edges will pass through each other on the way to their destination:

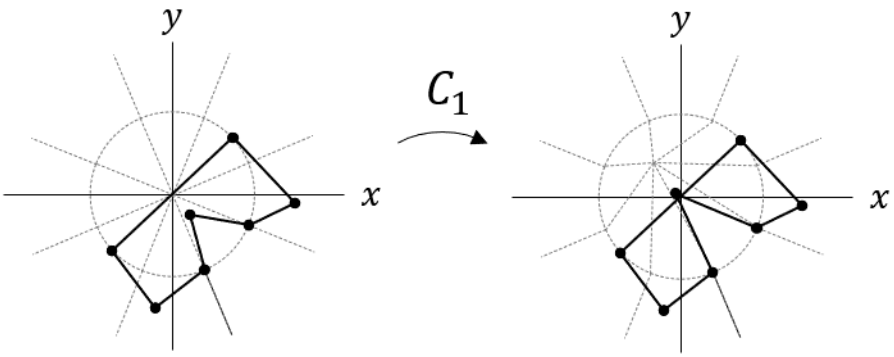


Figure 2.2: The continuous deformation fails when the knot is approximated by a finite number of vertices.

In other words, there will exist a $t \in [0, 1]$ such that $C_t(V) \notin \mathcal{K}$.

This is not a sufficient condition to show $\varphi(C(V)) \not\sim C \circ \varphi(V)$. For example, there may exist another ambient isotopy implemented by C where a self-intersection does not occur for any $t \in [0, 1]$. On the other hand, $\varphi(C(V)) \not\sim C \circ \varphi(V)$ implies that no continuous deformation will transform $\varphi(V)$ into $\varphi(C(V))$, since C implements an ambient isotopy and thus $C \circ \varphi(V) \sim \varphi(V)$. Avoiding intermediate self-intersections over the course of a *particular* continuous deformation ensures that a neural network respects the polygonality of its input.

Ideally, a neural network acting on a knot could identify when it risks an intermediate self-intersection, and change its behavior to avoid that self-intersection. There are two ways this can be accomplished. The first is to modify the neural network. The second is to modify the input.

2.2.1 Modifying the neural network

Consider a neural network N and the identity function $I : \mathbb{R}^3 \to \mathbb{R}^3$. We say that *N pushes to an ambient isotopy* if and only if

$$tN + (1 - t)I$$

is a homeomorphism for all $t \in [0, 1]$.

Theorem 7. *All neural networks that push to an ambient isotopy implement an ambient isotopy. Not all neural networks that implement an ambient isotopy push to an ambient isotopy.*

Proof. Let N be a neural network that pushes to an ambient isotopy, and let I denote the identity function. Then,

$$h_t := tN + (1-t)I$$

creates a continuous family of homeomorphisms for $t \in [0,1]$ such that h_0 is the identity function, which satisfies the definition of an ambient isotopy. Since $h_1 = N$, the neural network N implements an ambient isotopy.

Now, for any $t \in [0,1]$ consider the neural network $M_t : \mathbb{R}^3 \to \mathbb{R}^3$ which rotates its input πt radians around the z-axis, so that

$$M_t(x,y,z) = (\cos(\pi t)x - \sin(\pi t)y, \sin(\pi t)x + \cos(\pi t)y, z).$$

Rotations are homeomorphisms. Since varying t produces a continuous family of homeomorphisms where M_0 is the identity, M_1 implements an ambient isotopy.

In order for M_1 to push to an ambient isotopy, $tM_1 + (1-t)I$ must be a homeomorphism for all $t \in [0,1]$. However, the function $\frac{1}{2}M_1 + \frac{1}{2}I$ is not. For all $(x,y,z) \in \mathbb{R}^3$,

$$\begin{aligned}(\frac{1}{2}M_1 + \frac{1}{2}I)(x,y,z) &= \frac{1}{2}(\cos(\pi)x - \sin(\pi)y, \sin(\pi)x + \cos(\pi)y, z) + \frac{1}{2}(x,y,z) \\ &= \frac{1}{2}(-x,-y,z) + \frac{1}{2}(x,y,z) \\ &= (0,0,z),\end{aligned}$$

and so $\frac{1}{2}M_1 + \frac{1}{2}I$ is not invertible. Therefore, M_1 implements an ambient isotopy but does not push to an ambient isotopy. □

Neural networks that push to an ambient isotopy can deform space "part of the way," stopping before they violate the polygonality of their input. This can be demonstrated using the neural network C from earlier.

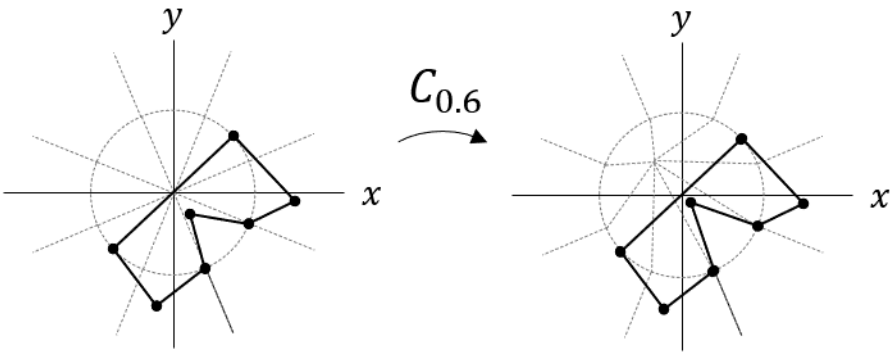


Figure 2.3: Cutting short the deformation prevents intermediate self-intersections.

For every neural network that pushes to an ambient isotopy, and for any desired fraction of the deformation it performs, there exists a neural network that corresponds to that partial deformation. At least one such neural network is capable of performing safe operations on the vertices of a knot.

Lemma 1. *Let N be a neural network that pushes to an ambient isotopy, and let I denote the identity function. Then, for all $m \in [0, 1]$,*

$$mN + (1 - m)I$$

pushes to an ambient isotopy.

Proof. Let $t \in [0, 1]$ be given. We compute:

$$\begin{aligned} t(mN + (1 - m)I) + (1 - t)I &= tmN + t(1 - m)I + (1 - t)I \\ &= tmN + (t(1 - m) + (1 - t))I \\ &= tmN + (1 - tm)I. \end{aligned}$$

Since $tm \in [0, 1]$ and N pushes to an ambient isotopy, $tmN + (1 - tm)I$ is a homeomorphism. Therefore, $mN + (1 - m)I$ pushes to an ambient isotopy. □

Lemma 2. *For all $\vec{x}, \vec{x}', \vec{y}, \vec{y}' \in \mathbb{R}^3$, there exists a unique $t \in [0, 1]$ such that*

$$t\vec{x}' + (1 - t)\vec{x} = t\vec{y}' + (1 - t)\vec{y}$$

if and only if

$$\vec{x}' - \vec{y}' \neq \vec{0} \quad \textit{or} \quad \vec{x} - \vec{y} \neq \vec{0}$$

and

$$0 = (\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2.$$

Proof. Before we begin our main proof, we will show that for all $t \in \mathbb{R}$, we have

$$t\vec{x}' + (1-t)\vec{x} = t\vec{y}' + (1-t)\vec{y}$$

if and only if the polynomial

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 t^2 + 2((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2)t + \|\vec{x} - \vec{y}\|_2^2$$

equals zero at that t.

Suppose the former condition holds. We will manipulate the equation solely using invertible operations, so that we show both the backward and forward direction of the proof at once. Subtracting $t\vec{y}' + (1-t)\vec{y}$ from both sides, we have

$$\begin{aligned} t\vec{x}' + (1-t)\vec{x} - t\vec{y}' - (1-t)\vec{y} &= \vec{0} \\ &= t(\vec{x}' - \vec{y}') + (1-t)(\vec{x} - \vec{y}). \end{aligned}$$

A vector in $\mathbb{R}^3$ is equal to $\vec{0}$ if and only if its dot product with itself is zero. Therefore,

$$\begin{aligned} 0 &= (t(\vec{x}' - \vec{y}') + (1-t)(\vec{x} - \vec{y})) \cdot (t(\vec{x}' - \vec{y}') + (1-t)(\vec{x} - \vec{y})) \\ &= t^2(\vec{x}' - \vec{y}') \cdot (\vec{x}' - \vec{y}') + 2t(1-t)(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + (1-t)^2(\vec{x} - \vec{y}) \cdot (\vec{x} - \vec{y}) \\ &= t^2\|\vec{x}' - \vec{y}'\|_2^2 + 2t(1-t)(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + (1-t)^2\|\vec{x} - \vec{y}\|_2^2. \end{aligned}$$

Breaking apart each coefficient, we have the following:

$$\begin{aligned} t^2 &= t^2 \\ 2t(1-t) &= 2t - 2t^2 \\ (1-t)^2 &= 1 - 2t + t^2 \end{aligned}$$

Therefore, we can rearrange the right hand side of our previous equation as

$$\begin{aligned} &t^2\|\vec{x}' - \vec{y}'\|_2^2 \\ &+ 2t(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - 2t^2(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) \\ &+ \|\vec{x} - \vec{y}\|_2^2 - 2t\|\vec{x} - \vec{y}\|_2^2 + t^2\|\vec{x} - \vec{y}\|_2^2. \end{aligned}$$

Grouping our terms, we thus have

$$\begin{aligned}\vec{0} =& t^2(\|\vec{x}' - \vec{y}'\|_2^2 - 2(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x} - \vec{y}\|_2^2) \\ &+ 2t((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2) \\ &+ \|\vec{x} - \vec{y}\|_2^2,\end{aligned}$$

which simplifies to

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 t^2 + 2((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2)t + \|\vec{x} - \vec{y}\|_2^2.$$

As a result, we know that there exists a unique $t \in [0, 1]$ such that

$$t\vec{x}' + (1 - t)\vec{x} = t\vec{y}' + (1 - t)\vec{y}$$

if and only if the polynomial

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 t^2 + 2((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2)t + \|\vec{x} - \vec{y}\|_2^2$$

has precisely one root in the interval $[0, 1]$.

We can now proceed to the main body of our proof. For the forward direction, assume for a contradiction that $\vec{x}' - \vec{y}' = \vec{x} - \vec{y}$. Then, the polynomial simplifies to

$$\begin{aligned}&\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 t^2 + 2((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2)t + \|\vec{x} - \vec{y}\|_2^2 \\ &= \|(\vec{x} - \vec{y}) - (\vec{x} - \vec{y})\|_2^2 t^2 + 2((\vec{x} - \vec{y}) \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2)t + \|\vec{x} - \vec{y}\|_2^2 \\ &= \|\vec{0}\|_2^2 t^2 + 2(\|\vec{x} - \vec{y}\|_2^2 - \|\vec{x} - \vec{y}\|_2^2)t + \|\vec{x} - \vec{y}\|_2^2 \\ &= \|\vec{x} - \vec{y}\|_2^2.\end{aligned}$$

We either have $\|\vec{x} - \vec{y}\|_2^2 = 0$ or $\|\vec{x} - \vec{y}\|_2^2 \neq 0$. If the former, then there are infinite values of $t \in [0, 1]$ such that the polynomial is equal to zero. If the latter, then there are no values of $t \in [0, 1]$ such that the polynomial is equal to zero. This contradicts our premise, so

$$\vec{x}' - \vec{y}' \neq \vec{x} - \vec{y}.$$

This furthermore implies that $\vec{x}' - \vec{y}'$ and $\vec{x} - \vec{y}$ are not both equal to $\vec{0}$, and so we have completed the first part of the forward direction. For the second part, we note that it also implies that the term

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2$$

is not equal to zero, and so we can apply the quadratic formula to find the roots t_1, t_2 of the polynomial:

$$t_1, t_2 = \frac{-2((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2) \pm \sqrt{D}}{2\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2}$$

for discriminant D, where

$$\begin{aligned} D &= 4((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2)^2 - 4\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 \|\vec{x} - \vec{y}\|_2^2 \\ &= 4(((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}))^2 - \|\vec{x}' - \vec{y}'\|_2^2 \|\vec{x} - \vec{y}\|_2^2). \end{aligned}$$

Dividing both the numerator and denominator of our roots by 2 gives us

$$t_1, t_2 = \frac{\|\vec{x} - \vec{y}\|_2^2 - (\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) \pm \sqrt{((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}))^2 - \|\vec{x}' - \vec{y}'\|_2^2 \|\vec{x} - \vec{y}\|_2^2}}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2}.$$

In order for there to exist a unique root in $[0, 1]$, there must at least exist a real root. In order for either of the roots to be real, however, we must have

$$((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}))^2 - \|\vec{x}' - \vec{y}'\|_2^2 \|\vec{x} - \vec{y}\|_2^2 \geq 0,$$

but by the Cauchy-Schwarz inequality,

$$((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}))^2 \leq \|\vec{x}' - \vec{y}'\|_2^2 \|\vec{x} - \vec{y}\|_2^2.$$

Therefore, if there indeed exists a unique root in $[0, 1]$, we must have

$$((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}))^2 = \|\vec{x}' - \vec{y}'\|_2^2 \|\vec{x} - \vec{y}\|_2^2.$$

We will use this fact to show that $(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 = 0$.

For a contradiction, suppose that

$$(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 \neq 0,$$

and thus

$$-(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) \neq \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2.$$

It could not then be the case that either $\vec{x}' - \vec{y}'$ or $\vec{x} - \vec{y}$ is the zero vector. However, since

$$((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}))^2 = \|\vec{x}' - \vec{y}'\|_2^2 \|\vec{x} - \vec{y}\|_2^2,$$

we must instead have

$$(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) = \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2.$$

Plugging this result into our equation for the roots, we have

$$\begin{aligned} t_1 = t_2 &= \frac{\|\vec{x} - \vec{y}\|_2^2 - (\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y})}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \\ &= \frac{\|\vec{x} - \vec{y}\|_2^2 - \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2}. \end{aligned}$$

We will obtain a second expression for the roots as follows:

$$\begin{aligned} t_1 = t_2 &= \frac{\|\vec{x} - \vec{y}\|_2^2 - \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \\ &= \frac{\|\vec{x} - \vec{y}\|_2^2 - 2\|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 + \|\vec{x}' - \vec{y}'\|_2^2 + (\|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 - \|\vec{x}' - \vec{y}'\|_2^2)}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \\ &= \frac{\|\vec{x} - \vec{y}\|_2^2 - 2(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x}' - \vec{y}'\|_2^2 + (\|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 - \|\vec{x}' - \vec{y}'\|_2^2)}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \\ &= \frac{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 + (\|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 - \|\vec{x}' - \vec{y}'\|_2^2)}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \\ &= 1 + \frac{\|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 - \|\vec{x}' - \vec{y}'\|_2^2}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \\ &= 1 - \frac{\|\vec{x}' - \vec{y}'\|_2^2 - \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2}. \end{aligned}$$

From these two expressions, it is clear that if the root is to be contained in $[0, 1]$, we must have both

$$\frac{\|\vec{x} - \vec{y}\|_2^2 - \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \geq 0$$

and

$$\frac{\|\vec{x}' - \vec{y}'\|_2^2 - \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \geq 0.$$

Now, recall that since neither $\vec{x}' - \vec{y}'$ nor $\vec{x} - \vec{y}$ are the zero vector, their norms are both strictly greater than 0. Additionally, since we have already shown $\vec{x}' - \vec{y}' \neq \vec{x} - \vec{y}$,

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2$$

must be strictly greater than 0 as well. Therefore, we can divide both the numerator and denominator of our first inequality by $\|\vec{x} - \vec{y}\|$ to get

$$\frac{\|\vec{x} - \vec{y}\|_2 - \|\vec{x}' - \vec{y}'\|_2}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 / \|\vec{x} - \vec{y}\|_2} \geq 0,$$

where

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 / \|\vec{x} - \vec{y}\|_2$$

is a positive real number. Therefore, we must have

$$\|\vec{x} - \vec{y}\|_2 - \|\vec{x}' - \vec{y}'\|_2 \geq 0.$$

Repeating the above steps for the second inequality, we obtain

$$\frac{\|\vec{x}' - \vec{y}'\|_2 - \|\vec{x} - \vec{y}\|_2}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 / \|\vec{x}' - \vec{y}'\|_2} \geq 0,$$

where

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 / \|\vec{x} - \vec{y}\|_2$$

is a positive real number, and so

$$\|\vec{x}' - \vec{y}'\|_2 - \|\vec{x} - \vec{y}\|_2 \geq 0.$$

Since we have both $\|\vec{x}' - \vec{y}'\|_2 \leq \|\vec{x} - \vec{y}\|_2$ and $\|\vec{x}' - \vec{y}'\|_2 \geq \|\vec{x} - \vec{y}\|_2$, we must have

$$\|\vec{x}' - \vec{y}'\|_2 = \|\vec{x} - \vec{y}\|_2.$$

However, this violates one of our earlier assumptions: that

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2$$

is strictly greater than 0. To show this, we compute the following:

$$\begin{aligned}
\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 &= \|\vec{x} - \vec{y}\|_2^2 - 2(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x}' - \vec{y}'\|_2^2 \\
&= \|\vec{x} - \vec{y}\|_2^2 - 2\|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 + \|\vec{x}' - \vec{y}'\|_2^2 \\
&= (\|\vec{x}' - \vec{y}'\|_2 - \|\vec{x} - \vec{y}\|_2)^2 \\
&= 0.
\end{aligned}$$

This is a contradiction, and so

$$(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 = 0.$$

This completes the forward direction of the proof.

For the backwards direction, assume both that either

$$\vec{x}' - \vec{y}' \neq \vec{0} \quad \text{or} \quad \vec{x} - \vec{y} \neq \vec{0}$$

and

$$0 = (\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2.$$

We will show that the polynomial

$$\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 t^2 + 2((\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) - \|\vec{x} - \vec{y}\|_2^2)t + \|\vec{x} - \vec{y}\|_2^2$$

possesses a unique root in the interval $[0, 1]$.

First, we can rewrite the first coefficient of our polynomial as follows:

$$\begin{aligned}
\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2 &= \|\vec{x}' - \vec{y}'\|_2^2 - 2(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x} - \vec{y}\|_2^2 \\
&= \|\vec{x}' - \vec{y}'\|_2^2 + 2\|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 + \|\vec{x} - \vec{y}\|_2^2 \\
&= (\|\vec{x}' - \vec{y}'\|_2 + \|\vec{x} - \vec{y}\|_2)^2.
\end{aligned}$$

Since either

$$\vec{x}' - \vec{y}' \neq \vec{0} \quad \text{or} \quad \vec{x} - \vec{y} \neq \vec{0},$$

we know that

$$(\|\vec{x}' - \vec{y}'\|_2 + \|\vec{x} - \vec{y}\|_2)^2$$

must be strictly greater than 0, and so we can use the quadratic formula to find any roots of our polynomial. Using similar computations to before, we find that

$$\begin{aligned}
t_1 = t_2 &= \frac{\|\vec{x} - \vec{y}\|_2^2 - (\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) \pm \sqrt{0}}{\|(\vec{x}' - \vec{y}') - (\vec{x} - \vec{y})\|_2^2} \\
&= \frac{\|\vec{x} - \vec{y}\|_2^2 + \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2}{\|\vec{x}' - \vec{y}'\|_2^2 - 2(\vec{x}' - \vec{y}') \cdot (\vec{x} - \vec{y}) + \|\vec{x} - \vec{y}\|_2^2} \\
&= \frac{\|\vec{x} - \vec{y}\|_2^2 + \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2}{\|\vec{x}' - \vec{y}'\|_2^2 + 2\|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2 + \|\vec{x} - \vec{y}\|_2^2} \\
&= \frac{\|\vec{x} - \vec{y}\|_2^2 + \|\vec{x}' - \vec{y}'\|_2 \|\vec{x} - \vec{y}\|_2}{(\|\vec{x}' - \vec{y}'\|_2 + \|\vec{x} - \vec{y}\|_2)^2}.
\end{aligned}$$

Since the denominator is a positive real number and the numerator is non-negative, the root must be greater than or equal to 0. Furthermore, by rearranging the expression, we find

$$\begin{aligned} t_1 = t_2 &= \frac{\|\vec{x}-\vec{y}\|_2^2 + \|\vec{x}'-\vec{y}'\|_2\|\vec{x}-\vec{y}\|_2}{(\|\vec{x}'-\vec{y}'\|_2 + \|\vec{x}-\vec{y}\|_2)^2} \\ &= \frac{\|\vec{x}-\vec{y}\|_2^2 + 2\|\vec{x}'-\vec{y}'\|_2\|\vec{x}-\vec{y}\|_2 + \|\vec{x}'-\vec{y}'\|_2^2 - \|\vec{x}'-\vec{y}'\|_2^2 - \|\vec{x}'-\vec{y}'\|_2\|\vec{x}-\vec{y}\|_2}{(\|\vec{x}'-\vec{y}'\|_2 + \|\vec{x}-\vec{y}\|_2)^2} \\ &= \frac{(\|\vec{x}'-\vec{y}'\|_2 + \|\vec{x}-\vec{y}\|_2)^2 - \|\vec{x}'-\vec{y}'\|_2^2 - \|\vec{x}'-\vec{y}'\|_2\|\vec{x}-\vec{y}\|_2}{(\|\vec{x}'-\vec{y}'\|_2 + \|\vec{x}-\vec{y}\|_2)^2} \\ &= 1 - \frac{\|\vec{x}'-\vec{y}'\|_2^2 + \|\vec{x}'-\vec{y}'\|_2\|\vec{x}-\vec{y}\|_2}{(\|\vec{x}'-\vec{y}'\|_2 + \|\vec{x}-\vec{y}\|_2)^2}. \end{aligned}$$

Therefore, the root must be less than or equal to 1. Combining these results, we know that there exists a unique root for the polynomial and it is contained within the interval $[0, 1]$.

This completes the backwards direction of the proof. □

Theorem 8. *Let N be a neural network that pushes to an ambient isotopy, let V be an array of vertices in $\mathcal{K}$, and let I denote the identity function. Then, there exists a non-zero $t_{max} \in (0, 1]$ such that for all $a \in [0, t_{max})$,*

$$aN(V) + (1-a)V$$

is an element of $\mathcal{K}$ and

$$aN + (1-a)I$$

implements an ambient isotopy.

Proof. First, by lemma 1, that N pushes to an ambient isotopy implies that for all $m \in [0, 1]$, $mN + (1-m)I$ pushes to an ambient isotopy. Then, by theorem 7, $mN + (1-m)I$ implements an ambient isotopy.

Next, theorem 2 tells us that

$$\begin{aligned} \varphi((mN + (1-m)I)(V)) &= \varphi(mN(V) + (1-m)V) \\ &= m\varphi(N(V)) + (1-m)\varphi(V). \end{aligned}$$

Varying m should produce a continuous family of polygonal knots so long as $mN(V) + (1-m)V \in \mathcal{K}$. Therefore, we should expect $\varphi((mN + (1-$

$m)I)(V))$ to have at least one self-intersection (and thus $mN(V) + (1 - m)V \notin \mathcal{K}$) if there exists $t, t' \in [0, 1)$ where $t \neq t'$ such that

$$m\varphi(N(V))(t) + (1 - m)\varphi(V)(t) = m\varphi(N(V))(t') + (1 - m)\varphi(V)(t').$$

By lemma 2, there exists an m such that this equation holds for a particular t and t' if and only if

$$\varphi(N(V))(t) - \varphi(N(V))(t') \neq \vec{0} \quad \text{or} \quad \varphi(V)(t) - \varphi(V)(t') \neq \vec{0}$$

and

$$\begin{aligned}\vec{0} =&(\varphi(N(V))(t) - \varphi(N(V))(t')) \cdot (\varphi(V)(t) - \varphi(V)(t')) \\ &+ \|\varphi(N(V))(t) - \varphi(N(V))(t')\|_2 \|\varphi(V)(t) - \varphi(V)(t')\|_2.\end{aligned}$$

The first of these two conditions is automatically satisfied, since the fact that $t \neq t'$ implies $\varphi(V)(t) \neq \varphi(V)(t')$. Though the second condition will not be satisfied by all t, t', we possess a formula that returns the value of m in the case where the second condition is satisfied. Therefore, any lower bound on that formula will also lower bound the set of all m where both the first and second condition are satisfied.

We derived in the proof of lemma 2 that if the second condition is satisfied, the self-intersection occurs for a value of m equal to

$$\frac{\|\varphi(V)(t) - \varphi(V)(t')\|_2^2 + \|\varphi(N(V))(t) - \varphi(N(V))(t')\|_2 \|\varphi(V)(t) - \varphi(V)(t')\|_2}{(\|\varphi(N(V))(t) - \varphi(N(V))(t')\|_2 + \|\varphi(V)(t) - \varphi(V)(t')\|_2)^2}.$$

We can simplify this expression further. Since $\varphi(V)(t) \neq \varphi(V)(t')$, the value of $\|\varphi(V)(t) - \varphi(V)(t')\|_2^2$ must be greater than 0. Dividing both the numerator and denominator of our formula for m, we have

$$\begin{aligned}m &= \frac{1 + \frac{\|\varphi(N(V))(t) - \varphi(N(V))(t')\|_2}{\|\varphi(V)(t) - \varphi(V)(t')\|_2}}{(\frac{\|\varphi(N(V))(t) - \varphi(N(V))(t')\|_2}{\|\varphi(V)(t) - \varphi(V)(t')\|_2} + 1)^2} \\ &= \frac{1}{1 + \frac{\|\varphi(N(V))(t) - \varphi(N(V))(t')\|_2}{\|\varphi(V)(t) - \varphi(V)(t')\|_2}}.\end{aligned}$$

By theorem 3, we know that

$$\varphi(N(V)) \circ \varphi(V)^{-1}$$

is Lipschitz. There must therefore exist some constant M such that

$$\frac{\|\varphi(N(V))(t) - \varphi(N(V))(t')\|_2}{\|\varphi(V)(t) - \varphi(V)(t')\|_2} \leq M$$

for all $\varphi(V)(t) \neq \varphi(V)(t')$. This means that

$$t_{\text{max}} = \frac{1}{1+M}$$

is a lower bound on the set of all values $m \in [0,1]$ where the image of the function

$$\varphi(mN(V) + (1-m)V)$$

has self-intersections, and thus would not be a polygonal knot. Note that t_{max} is strictly greater than 0. Furthermore, since it is less than all values where a self-intersection might occur, all values of $a \in [0, t_{\text{max}})$ will also avoid a self-intersection. □

Recall from theorem 5 that a neural network N which both respects the polygonality of its input $V \in \mathcal{K}$ and implements an ambient isotopy is able to perform "safe" operations, i.e. $\varphi(N(V)) \sim \varphi(V)$. Theorem 8 tells us that neural networks which push to an ambient isotopy and are modified according to the procedure described above possess this property. This is a powerful sufficient condition. The downside to this approach, however, is that it requires a differentiable formula for t_{max}. Otherwise, a neural network that uses this method will not be able to train via gradient descent.

The advantage to this method is that it has an analogue in how a human might attempt to untangle a knotted piece of string. A person will try to pull different parts of the string to new locations, but stop once they realize that two overlapping parts of the string are being pulled in opposite directions. The above method allows a neural network to identify when it's making a similar error.

Finding a differentiable formula for t_{max} is beyond the scope of this paper. We describe this method here as a potential avenue for future research.

2.2.2 Modifying the input

Suppose that we select a finite number of points in the image of a knot $K : [0,1] \to \mathbb{R}^3$, and construct an inscribed polygonal approximation to K by drawing line segments between each point and its successor.

Figure 2.4: Constructing an inscribed polygonal approximation to a knot.

Intuitively, as the number of vertices in the polygonal approximation grows larger, it will better adhere to K's shape. By placing some restrictions on the properties of K, we can demonstrate that a sufficiently precise polygonal approximation will be ambient isotopic to the original knot.

The literature has discovered several well-behaved knot varieties which are ambient isotopic to an inscribed polygonal approximation. Li and Peters[2] prove this for knots in differentiability class C^2. They show that for any such knot $\mathcal{C}$, for any sequence of inscribed piecewise linear curves $\{L_i\}_1^\infty$ that converges pointwise to $\mathcal{C}$ and uniformly converges to $\mathcal{C}$ in total curvature, a positive integer N can be found such that for all $i > N$, L_i is ambient isotopic to $\mathcal{C}$. Additionally, they show that such a sequence of inscribed piecewise linear curves exists.

A classic theorem from Milnor[3] establishes a similar result for knots with finite total curvature. All such knots are ambient isotopic to an inscribed polygonal curve, and thus tame. Additionally, Sullivan[6] finds that a rectifiable curve has finite total curvature if and only if its unit tangent vector is of bounded variation. Therefore, a neural network could be made to respect polygonality if it mapped polygonal knots to rectifiable curves with this property.

Unfortunately, these conditions are more suited to characterizing individual knots than every possible knot that a neural network layer might return. It would be ideal if there were conditions that held for a broad class of neural networks, which guarantee that they will produce knots which are ambient isotopic to sufficiently fine-grained inscribed polygonal approximations. There is a correspondence between such neural networks and whether they respect polygonality.

Let N be a neural network that implements an ambient isotopy and let V be a set of vertices in $\mathcal{K}$. Then, $\varphi(N(V))$ can be viewed as an inscribed

polygonal approximation to $N \circ \varphi(V)$.

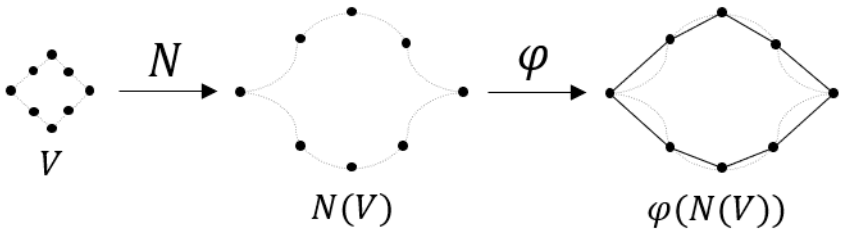


Figure 2.5: N warps space, then φ linearly interpolates between elements of $N(V)$. $\varphi(N(V))$ is inscribed in $N \circ \varphi(V)$.

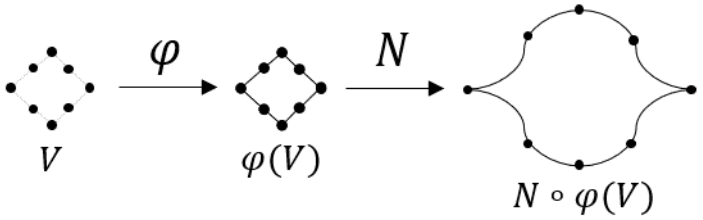


Figure 2.6: Reversing the order in which φ and N are applied will produce $N \circ \varphi(V)$, a visually different knot.

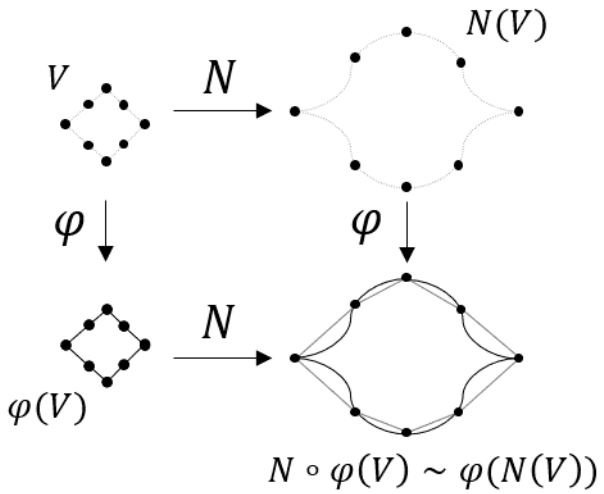


Figure 2.7: For a sufficiently precise approximation, this diagram commutes and $N \circ \varphi(V) \sim \varphi(N(V))$.

However, if this polygonal approximation is inexact, then $N \circ \varphi(V) \not\sim \varphi(N(V))$. In other words, N will not respect the polygonality of V. We therefore need conditions for when a sufficiently fine-grained approximation to $N \circ \varphi(V)$ can be obtained by introducing additional vertices to $\varphi(N(V))$.

First, we will introduce an operation on arrays that increases the number of vertices they contain, without distorting the polygonal knot they represent. For $k \in \mathbb{Z}^+$, let the function $\mathcal{A}_k : \mathcal{K} \cup \overline{\mathcal{K}} \to \mathcal{K} \cup \overline{\mathcal{K}}$ be defined such that

$$\begin{aligned}\mathcal{A}_k([\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}}]) = [&(1 - \frac{0}{k})\overrightarrow{x_0} + \frac{0}{k}\overrightarrow{x_1}, ..., (1 - \frac{k-1}{k})\overrightarrow{x_0} + \frac{k-1}{k}\overrightarrow{x_1},\\ &(1 - \frac{0}{k})\overrightarrow{x_1} + \frac{0}{k}\overrightarrow{x_2}, ..., (1 - \frac{k-1}{k})\overrightarrow{x_1} + \frac{k-1}{k}\overrightarrow{x_2},\\ &...,(1 - \frac{0}{k})\overrightarrow{x_{n-1}} + \frac{0}{k}\overrightarrow{x_0}, ..., (1 - \frac{k-1}{k})\overrightarrow{x_{n-1}} + \frac{k-1}{k}\overrightarrow{x_0}].\end{aligned}$$

Intuitively, $\mathcal{A}_k$ places $k-1$ evenly spaced new vertices between any two adjacent vertices in the original array. The function possesses the important property that it does not modify linear interpolations constructed using φ.

Theorem 9. *For $V \in \mathcal{K} \cup \overline{\mathcal{K}}$,*

$$\varphi(\mathcal{A}_k(V)) = \varphi(V)$$

for all $k \in \mathbb{Z}^+$.

Proof. For convenience, we will write out the elements of V as $[\overrightarrow{x_0}, ..., \overrightarrow{x_{n-1}}]$ and the elements of $\mathcal{A}_k(V)$ as $[\overrightarrow{y_0}, ..., \overrightarrow{y_{nk-1}}]$. Note that as V is of length n, $\mathcal{A}_k(V)$ will be of length nk.

Given $t \in [0, 1]$,

$$\varphi(\mathcal{A}_k(V))(t) = (1 - (nkt \bmod 1))\overrightarrow{y_{i(t)}} + (nkt \bmod 1)\overrightarrow{y_{i^+(t)}}$$

for $i(t) = \lfloor nkt \rfloor \bmod nk$ and $i^+(t) = (\lfloor nkt \rfloor + 1) \bmod nk$. We will show that this is equal to $\varphi(V)(t)$ for all $t \in [0, 1]$.

Let t be given. Then, there exists a unique $q \in \mathbb{N}$ and $a \in [0, 1)$ such that

$$t = \frac{q + a}{n}.$$

We thus can rewrite $i(t)$ and $i^+(t)$ as

$$\begin{aligned} i(t) &= \lfloor nk\frac{q+a}{n} \rfloor \bmod nk \\ &= \lfloor kq + ka \rfloor \bmod nk \\ &= (kq + \lfloor ka \rfloor) \bmod nk \end{aligned}$$

and

$$\begin{aligned} i^+(t) &= (\lfloor nk\frac{q+a}{n} \rfloor + 1) \bmod nk \\ &= (\lfloor kq + ka \rfloor + 1) \bmod nk \\ &= (kq + \lfloor ka \rfloor + 1) \bmod nk. \end{aligned}$$

This decomposes $i(t)$ and $i^+(t)$ into two separate components: the index of the start of an interpolated interval (i.e. kq) and the index within the interpolated interval (i.e. $\lfloor ka \rfloor$ and $\lfloor ka \rfloor + 1$). Let i' denote $q \bmod n$, and let $i^{+\prime}$ denote $(q + 1) \bmod n$. By our construction of $\mathcal{A}_k$, we thus have

$$\overrightarrow{y_{i(t)}} = (1 - \frac{\lfloor ka \rfloor}{k})\overrightarrow{x_{i'}} + \frac{\lfloor ka \rfloor}{k}\overrightarrow{x_{i^{+\prime}}}$$

and

$$\overrightarrow{y_{i^+(t)}} = (1 - \frac{\lfloor ka \rfloor + 1}{k})\overrightarrow{x_{i'}} + \frac{\lfloor ka \rfloor + 1}{k}\overrightarrow{x_{i^{+\prime}}}.$$

Furthermore, we can rewrite $nkt \bmod 1$ as

$$\begin{aligned} nkt \bmod 1 &= nk\frac{q+a}{n} \bmod 1 \\ &= (kq + ka) \bmod 1 \\ &= ka \bmod 1. \end{aligned}$$

Therefore,

$$(1 - (nkt \bmod 1))\overrightarrow{y_{i(t)}} = (1 - (ka \bmod 1))((1 - \frac{\lfloor ka \rfloor}{k})\overrightarrow{x_{i'}} + \frac{\lfloor ka \rfloor}{k}\overrightarrow{x_{i^+{}'}})$$

and

$$\begin{aligned} (nkt \bmod 1)\overrightarrow{y_{i^+(t)}} &= (ka \bmod 1)((1 - \frac{\lfloor ka \rfloor + 1}{k})\overrightarrow{x_{i'}} + \frac{\lfloor ka \rfloor + 1}{k}\overrightarrow{x_{i^+{}'}}) \\ &= (ka \bmod 1)((1 - \frac{\lfloor ka \rfloor}{k} - \frac{1}{k})\overrightarrow{x_{i'}} + (\frac{\lfloor ka \rfloor}{k} + \frac{1}{k})\overrightarrow{x_{i^+{}'}}) \\ &= (ka \bmod 1)((1 - \frac{\lfloor ka \rfloor}{k})\overrightarrow{x_{i'}} + \frac{\lfloor ka \rfloor}{k}\overrightarrow{x_{i^+{}'}} + \frac{\overrightarrow{x_{i^+{}'}} - \overrightarrow{x_{i'}}}{k}). \end{aligned}$$

Summing the above expressions to find $\varphi(\mathcal{A}_k(V))(t)$, we have

$$\begin{aligned} \varphi(\mathcal{A}_k(V))(t) &= (1 - (nkt \bmod 1))\overrightarrow{y_{i(t)}} + (nkt \bmod 1)\overrightarrow{y_{i^+(t)}} \\ &= (1 - \frac{\lfloor ka \rfloor}{k})\overrightarrow{x_{i'}} + \frac{\lfloor ka \rfloor}{k}\overrightarrow{x_{i^+{}'}} + (ka \bmod 1)\frac{\overrightarrow{x_{i^+{}'}} - \overrightarrow{x_{i'}}}{k} \\ &= (1 - \frac{\lfloor ka \rfloor + (ka \bmod 1)}{k})\overrightarrow{x_{i'}} + \frac{\lfloor ka \rfloor + (ka \bmod 1)}{k}\overrightarrow{x_{i^+{}'}} \\ &= (1 - (a \bmod 1))\overrightarrow{x_{i'}} + (a \bmod 1)\overrightarrow{x_{i^+{}'}} \\ &= (1 - ((nt - q) \bmod 1))\overrightarrow{x_{i'}} + ((nt - q) \bmod 1)\overrightarrow{x_{i^+{}'}} \\ &= (1 - (nt \bmod 1))\overrightarrow{x_{i'}} + (nt \bmod 1)\overrightarrow{x_{i^+{}'}} \\ &= \varphi(V)(t). \end{aligned}$$

□

Corollary 9.1. *For any $V \in \mathcal{K}$ and $W \in \overline{\mathcal{K}}$,*

$$\mathcal{A}_k(V) \in \mathcal{K} \quad \textit{and} \quad \mathcal{A}_k(W) \in \overline{\mathcal{K}}.$$

Proof. This follows directly from theorem 9. If $\varphi(\mathcal{A}_k(V)) = \varphi(V)$ and $\varphi(V)$ is a polygonal knot, then $\varphi(\mathcal{A}_k(V))$ is a polygonal knot and $\mathcal{A}_k(V) \in \mathcal{K}$. Similarly, if $\varphi(\mathcal{A}_k(W)) = \varphi(W)$ and $\varphi(W)$ is not a polygonal knot, then $\varphi(\mathcal{A}_k(W))$ is not a polygonal knot and $\mathcal{A}_k(V) \in \overline{\mathcal{K}}$. □

Corollary 9.2. *For $V \in \mathcal{K}$, if a neural network N respects the polygonality of $\mathcal{A}_k(V)$, then*

$$\varphi(N(\mathcal{A}_k(V))) \sim N \circ \varphi(V).$$

If N also implements an ambient isotopy, then

$$\varphi(N(\mathcal{A}_k(V))) \sim \varphi(V).$$

Proof. From theorem 9, $\varphi(\mathcal{A}_k(V)) = \varphi(V)$, which implies $\varphi(\mathcal{A}_k(V)) \sim \varphi(V)$. Since N respects the polygonality of $\mathcal{A}_k(V)$, we also have $\varphi(N(\mathcal{A}_k(V))) \sim N \circ \varphi(\mathcal{A}_k(V))$. Therefore, as $\sim$ is an equivalence relation,

$$\varphi(N(\mathcal{A}_k(V))) \sim N \circ \varphi(V).$$

Next, assuming that N implements an ambient isotopy, $\varphi(V) \sim N \circ \varphi(V)$. Thus

$$\varphi(N(\mathcal{A}_k(V))) \sim \varphi(V).$$

□

As with the method described in section 2.2.1, proving conditions for when applying $\mathcal{A}_k$ to sets of vertices forces a neural network to respect their polygonality for sufficiently large values of k is beyond the scope of this paper.

Presuming, however, that a neural network meets the preconditions of corollary 9.2, increasing the number of vertices has definite advantages over trying to detect intermediate self-intersections. It is simpler to implement, and doesn't restrict the flexibility of the neural network.

Brute force solutions also have their downsides, however. The disadvantages of this approach are two-fold. First, increasing the number of inputs to a neural network has computational costs. It takes longer to train a neural network to untangle a knot with more vertices. Second, it is not clear how many vertices are required. Even if a sufficiently large set of vertices will prevent illegal manipulations of the underlying knot, that number will change as the neural network fine-tunes its parameters. This can prove dangerous for any fixed number of vertices, no matter how large.

Neither of the two methods described here are without drawbacks. The best way to avoid polygonality violations is to avoid situations where they might occur to begin with. This can be achieved by altering a neural network's loss function.

2.3 Knot Energy as Loss Function

In order for a neural network to determine which of its parameter configurations most successfully untangles an input knot, we require a loss function that represents how "tangled" the output is. Geometric knot theorists have developed functionals on the space of knots, called knot energies, which capture this property.

The *Möbius energy* is a particularly common knot energy. For a knot $K : [0, 1] \to \mathbb{R}^3$ the Möbius energy functional is defined by the map

$$K \mapsto \int_{u=0}^{1} \int_{v=0}^{1} \left(\frac{1}{\|K(u) - K(v)\|^2} - \frac{1}{D_{\text{arc}}(K(u), K(v))^2} \right) \|\dot{K}(u)\| \|\dot{K}(v)\| dv \, du,$$

where $D_{\text{arc}}(K(u), K(v))$ is the minimum distance between $K(u)$ and $K(v)$ along the curve defined by the image of K.

Since neural networks act on a discrete number of inputs, we will make use of a discretized version of the Möbius energy. For a polygonal knot K with vertex set $V \in \mathcal{K}$, the map defined by the discrete Möbius energy functional is

$$K \mapsto \sum_{x_i \neq x_j \in V} \left(\frac{1}{\|x_i - x_j\|^2} - \frac{1}{D_{\text{arc}}(x_i, x_j)^2} \right) \|x_i - x_{i+1 \bmod n}\| \|x_j - x_{j+1 \bmod n}\|,$$

where n is the number of vertices in V. For polygonal knots with few vertices, a better-behaved version of the discrete Möbius energy can be calculated by replacing $\|x_i - x_{i+1 \bmod n}\|$ and $\|x_j - x_{j+1 \bmod n}\|$ in the above formula with

$$\frac{\|x_i - x_{i+1 \bmod n}\| + \|x_i - x_{i-1 \bmod n}\|}{2}$$

and

$$\frac{\|x_j - x_{j+1 \bmod n}\| + \|x_j - x_{j-1 \bmod n}\|}{2},$$

respectively.

The advantage of the discrete Möbius energy is that the continuous Möbius energy of a smooth knot will correspond closely to the discrete Möbius energy of a sufficiently similar inscribed polygonal knot.[4] Furthermore, the discrete Möbius energy is easily differentiable with respect to the vertices of a polygonal knot, which makes it an ideal candidate for a

loss function. Other discrete knot energies which do not possess this property, such as the minimum distance energy[5], are less suited for training a neural network via gradient descent.

Additionally, we can modify our loss function so that the neural network actively avoids situations where it might violate the polygonality of its input. Since these situations most often arise when the edges of a polygonal knot are unequally sized, we construct an index of how poorly a polygonal knot's edge lengths conform to the edge lengths of an ideal knot:

$$\exp\left(\sqrt{\sum_{i=0}^{n-1}\frac{1}{n}\ln\left(\frac{\|x_i - x_{i+1 \bmod n}\|}{L/n}\right)^2}\right)$$

L denotes the desired arclength of our knot.

Multiplying this index by the discrete Möbius energy produces a loss function that is both sensitive to polygonality violations and the primary objective of untangling the knot.

2.4 Pitfalls of Conventional Layers

Traditionally, the layers of a neural network are a composition of three functions: a linear mapping, a translation, and an activation function. First, the layer runs its input through a matrix with real entries. Next, it adds a vector with real entries–called the *bias*–to the output of the linear transformation. Finally, it applies a continuous function pointwise to each entry of the resulting vector, which is called an *activation function*.

As a result, we can represent a conventional neural network layer $L : \mathbb{R}^3 \to \mathbb{R}^3$ acting on an input $\vec{x} \in \mathbb{R}^3$ as

$$L(\vec{x}) = f(\mathbf{A}\vec{x} + \vec{b})$$

for 3 by 3 matrix $\mathbf{A}$, bias term $\vec{b} \in \mathbb{R}^3$, and activation function $f : \mathbb{R} \to \mathbb{R}$.

So long as the matrix $\mathbf{A}$ has a positive determinant, the map without the activation function defined by $\vec{x} \mapsto \mathbf{A}\vec{x} + \vec{b}$ both implements an ambient isotopy and respects the polygonality of all arrays of vertices. Including the activation function in this map, however, poses some difficulties. First, the activation function may not respect the polygonality of its input. Second, many common activation functions are not bijective, and therefore cannot

be homeomorphisms from $\mathbb{R}$ to $\mathbb{R}$. Only bijective neural network operations can implement ambient isotopies.

Fortunately, this second difficulty can be circumvented. If f is strictly monotonically increasing–as is the case for many common activation functions–then there is a way to transform the activation function so that it implements an ambient isotopy.

Lemma 3. *Let $I : \mathbb{R} \to \mathbb{R}$ denote the identity function, and let m be any real number in $[0, 1)$. If a continuous function $f : \mathbb{R} \to \mathbb{R}$ is strictly monotonically increasing, then the function $mf + (1 - m)I$ is a homeomorphism.*

Proof. For $mf + (1 - m)I$ to be a homeomorphism, it must be a bijective continuous function with a continuous inverse.

First, since $mf + (1 - m)I$ is a convex combination of two continuous functions, it is itself continuous.

Next, we will show that $mf + (1 - m)I$ is injective. Let $x, y \in \mathbb{R}$ be given such that $x < y$. f preserves the $<$ relation because it is strictly monotonically increasing, so we have

$$f(x) < f(y).$$

Multiplying by the non-negative value of m on both sides, this becomes

$$mf(x) \leq mf(y).$$

Since m can equal 0, this replaces the strict inequality with a non-strict inequality. However, as $(1-m) > 0$, we can repeat the process for the relation $x < y$ while retaining the strict inequality:

$$(1 - m)x < (1 - m)y$$

Adding the two inequalities together, we have

$$mf(x) + (1 - m)x < mf(y) + (1 - m)y,$$

and so $mf+(1-m)I$ is strictly monotonically increasing. All strictly monotonically increasing functions are injective.

To show that $mf+(1-m)I$ is surjective, first consider that by a property of monotonically increasing functions,

$$\lim_{x\to\infty}(mf(x)) \quad \text{and} \quad \lim_{x\to-\infty}(mf(x))$$

always exist. $\lim_{x\to\infty}(mf(x))$ is either a finite real number or $+\infty$, while $\lim_{x\to-\infty}(mf(x))$ is either a finite real number or $-\infty$. Meanwhile, since $(1-m)>0$,

$$\lim_{x\to\infty}((1-m)x)=+\infty \quad \text{and} \quad \lim_{x\to-\infty}((1-m)x)=-\infty,$$

and so regardless of the asymptotic values of $mf(x)$,

$$\lim_{x\to\infty}((mf+(1-m)I)(x))=+\infty \quad \text{and} \quad \lim_{x\to-\infty}((mf+(1-m)I)(x))=-\infty.$$

Recall that $mf+(1-m)I$ is a continuous function. The intermediate value theorem thus applies and for any $y\in\mathbb{R}$ there will exist some $x\in\mathbb{R}$ such that $(mf+(1-m)I)(x)=y$. $mf+(1-m)I$ is therefore a bijection.

Lastly, the inverse of a strictly monotonically increasing function will always be continuous.[1] This is the final property needed to show that $mf+(1-m)I$ is a homeomorphism. □

Note also that $mf+(1-m)I$ is continuous in m. Using lemma 3, we can implement an ambient isotopy using a modified activation function:

Theorem 10. *Let $I:\mathbb{R}\to\mathbb{R}$ denote the identity function, and let m be any real number in $[0,1)$. If a continuous function $f:\mathbb{R}\to\mathbb{R}$ is strictly monotonically increasing, then applying the function $mf+(1-m)I$ pointwise to elements of $\mathbb{R}^3$ implements an ambient isotopy.*

Proof. From lemma 3, $af+(1-a)I$ is a homeomorphism for all $a\in[0,1)$. Since $m\in[0,1)$, we can make the weaker claim that $af+(1-a)I$ is a homeomorphism for all $a\in[0,m]$.

Applying a homeomorphism pointwise is itself a homeomorphism. We set $a=mt$ for $t\in[0,1]$ and, using the fact that $af+(1-a)I$ is continuous in a, this produces an ambient isotopy $h:\mathbb{R}^3\times[0,1]\to\mathbb{R}^3$ where

$$h_t(\vec{x})=mtf(\vec{x})+(1-mt)\vec{x}.$$

Since $h_1=mf+(1-m)I$, this demonstrates that $mf+(1-m)I$ implements an ambient isotopy. □

Corollary 10.1. *Applying $mf+(1-m)I$ pointwise pushes to an ambient isotopy.*

Proof. Rearranging the ambient isotopy constructed in theorem 10, we have

$$\begin{aligned}h_t(\vec{x})&=mtf(\vec{x})+(1-mt)\vec{x}\\&=mtf(\vec{x})+(1-m)t\vec{x}+(1-t)\vec{x}\\&=t(mf(\vec{x})+(1-m)\vec{x})+(1-t)\vec{x}.\end{aligned}$$

This is a linear interpolation between $mf + (1 - m)I$ and I, which implies $mf + (1 - m)I$ pushes to an ambient isotopy. □

Since m can be made as close to 1 as desired, many common activation functions (such as the sigmoid, hyperbolic tangent, or ReLU functions) can be easily modified to implement ambient isotopies without heavily distorting their behavior. In fact, some activation functions (e.g. leaky ReLU) already implement ambient isotopies.

In addition, corollary 10.1 allows us to modify strictly increasing activation functions so that they respect the polygonality of their input, using the method described in section 2.2.1.

Unfortunately, the other traits of conventional neural networks prevent them from untying knots effectively. Training dense layers while keeping the determinant of their matrix transformations positive requires significant changes to the training process. Additionally, most activation functions were not built to exploit the particular structure of knots. In practice, they are unable to learn operations that could untangle a complicated set of vertices.

We were able to outperform conventional layers by creating customized alternatives that take advantage of a typical knot's structure. Stacked together, they avoid many of the difficulties with traditional neural networks and yield superior results.

Chapter 3

Custom Layers

We experimented with several different custom neural network layers. All implemented ambient isotopies. Some were significantly more effective than others at untangling knots. In addition, composing several layers of one type could often mimic the functionality of a single layer of a different type.

An overriding design philosophy behind the custom layers was to make sure that at least one configuration of their trainable parameters would cause them to become the identity function. As a result, if it would be locally optimal for a custom layer to stop acting upon the knot entirely (e.g. because it is causing the knot to become more twisted than it began), the custom layer can "drop out" of the neural network. Custom layers are thus able to recognize when they do more harm than good, and act accordingly.

Additionally, this design philosophy allows us to more easily demonstrate that the custom layers implement ambient isotopies. If

1. The map implemented by a custom layer is a continuous function of its parameters,

2. There exists a continuous curve in the custom layer's parameter space which starts at its current parameters and ends at a parameter configuration that would yield the identity function, and

3. All parameter configurations along the curve yield homeomorphisms,

then the layer itself, at its current parameters, implements an ambient isotopy.

For a layer with n parameters that can take on values in some interval of the real numbers, proving that a layer implements an ambient isotopy

becomes even simpler. Suppose a layer has a parameter space of the form $S_1 \times ... \times S_n$, where each S_i is an interval of the real numbers. Then, for any two parameter configurations, there will exist a continuous curve between them in the parameter space. Therefore, if at least one parameter configuration yields the identity function, property (2) will always hold, *regardless of the layer's current parameters*.

We can thus produce a simplified sufficient set of conditions for a custom layer to implement an ambient isotopy. If

1. The map implemented by a custom layer is a continuous function of its parameters,
2. The custom layer has n parameters, and a parameter space of the form $S_1 \times ... \times S_n$, where each S_i is an interval of real numbers,
3. All possible parameter configurations of the custom layer yield homeomorphisms, and
4. There exists a parameter configuration that yields the identity function,

then the custom layer, at all possible parameter configurations, implements an ambient isotopy. Each custom layer investigated in this section satisfies all of these 4 properties.

3.1 TwistLayer

This layer mimics the first Reidemeister move. It twists or untwists a segment of the knot in a direction of its choice (i.e. either clockwise or counterclockwise).

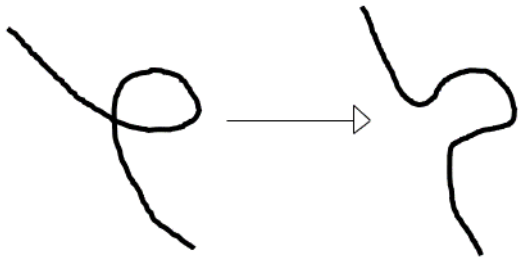
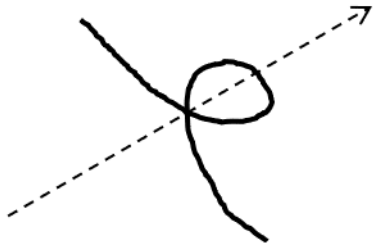
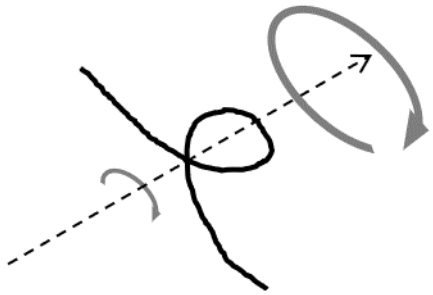


Figure 3.1: Reidemeister 1

The `TwistLayer` begins by identifying a line in 3D space, ideally radial to a loop in the knot:



It then applies a rotation along the axis defined by this line. The angle of rotation, however, isn't uniform across the entire space. Locations further in the direction of the line are rotated more, while locations in the opposite direction are rotated less.



The result is that the `TwistLayer` applies a "vortex-like" transformation to the data, performing scarcely any rotation near the base of a twist in the knot, and much more intensive rotation inside the looped region. If the line of rotation is placed correctly, this successfully duplicates the functionality of the first Reidemeister move.

One trainable parameter r controls the rate and direction of the twist, while another trainable parameter $m \in [0, 1)$ controls the maximal degree of rotation applied in the limit to an input as it asymptotically moves in the direction of the axis. If $m = 0$, then the `TwistLayer` becomes the identity function.

In order to ensure that the `TwistLayer` pushes to an ambient isotopy, the maximal degree of rotation as m approaches 1 is capped at π radians. As demonstrated in theorem 7, a rotation of π radians does not push to an ambient isotopy.

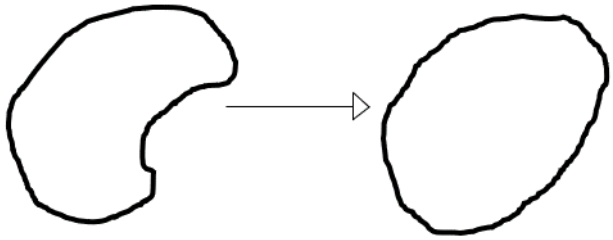
Efficacy

The `TwistLayer` is able to remove twists, though newly untwisted areas are often jagged and in need of smoothing afterwards.

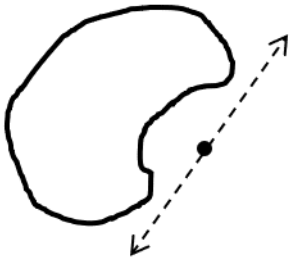
Additionally, the `TwistLayer` sometimes has trouble identifying locations at which to perform twists. Neural networks constructed with `TwistLayer`s can become stuck in local minima, in which the parameters for the axis of rotation are unable to move into the correct position to undo a twist in the knot without first moving through a region that causes the knot to become more twisted than it started. This is a problem that exists (to varying degrees) with all custom layers that rely upon specifying some line in space before performing their operations. It will be discussed more comprehensively in the `GravityLayer` section.

3.2 GravityLayer

The `GravityLayer` pushes space either away or toward a line. It's intended to help deformed segments of a knot become more curved.



The layer begins by identifying a point in space, called a *planet*. It then finds a line that intersects the planet, which we'll refer to as the *planetary line*.



The layer then computes the distance between each vertex in the knot and the planet. Vertices closer to the planet will be shifted more dramatically, while far away locations will remain relatively undisturbed. At the same time, the layer computes the line segment of shortest distance between each vertex and the planetary line. Vertices will be pulled or pushed in the direction of this line of shortest distance.

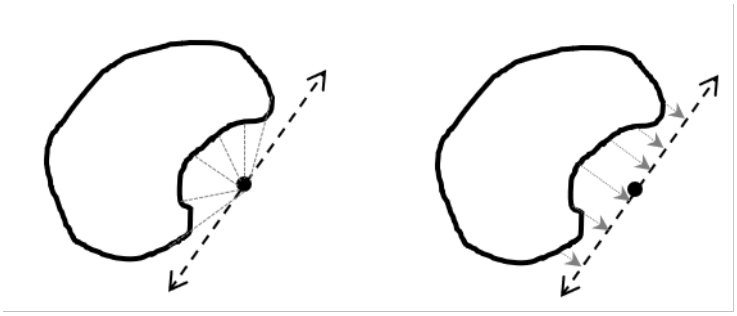


Figure 3.2: The length of the dashed grey lines are computed, and the dotted grey lines are stored as vectors.

This makes the name "`GravityLayer`" slightly misleading–while the gravitational (or inversely gravitational) force each vertex experiences is dependent on their distance from the planet, they are pulled toward (or pushed away from) the planetary line rather than the planet itself.

The formula for the distance moved by each vertex in the direction of the planetary line is

$$\|\vec{l_i}\|(1 - \frac{1}{1 + \frac{1-s}{s}\exp(-r * d_i)}),$$

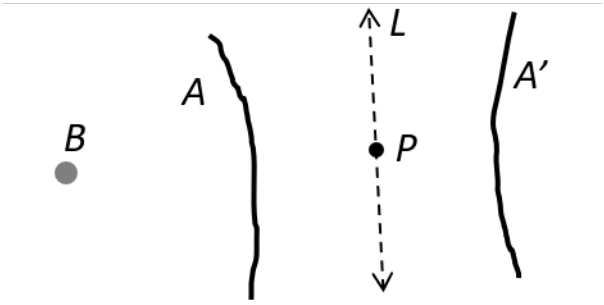
where $\vec{l_i}$ is the shortest vector from vertex i to the planetary line, d_i is the distance from vertex i to the planet, and $s, r \in (0, \infty)$ are trainable parameters. Intuitively, r corresponds to how fast the gravitational effect disappears as one moves farther from the planet, and s corresponds to how strong the gravitational effect is overall. A value of $s \in (0, 1)$ causes vertices to move *away* from the planetary line, while a value of $s \in (1, \infty)$ causes vertices to move *toward* the planetary line. More extreme values of s (i.e. $s >> 1$ or $s << 1$) cause a stronger effect. When $s = 0$, no force is applied.

Efficacy

The `GravityLayer` is the most powerful of the custom layers we tested. However, local modifications to its weights often fail to approach a minimum in the loss function unless the weights are initialized to positions already close to a minimum.

Consider the following example. Suppose the loss function of a knot would be decreased if strands A and A' were to be pushed away from lo-

cation *B* by a `GravityLayer`, but the planet *P* and the planetary line *L* are located in between the two strands:



Suppose the `GravityLayer` has a value of s contained in $(0, 1)$. The planetary line is thus pushing strand *A* in a direction that increases the value of the loss function, but pushes *A'* in a direction that decreases the value of the loss function. Modifying the values of s or r would therefore cause the loss function to increase or decrease further in equal measure. Gradient descent will leave those two parameters unchanged.

Ideally, in order to minimize the loss function, we would place the planet and planetary line on the opposite side of strand *A*, at positions *P** and *L**:

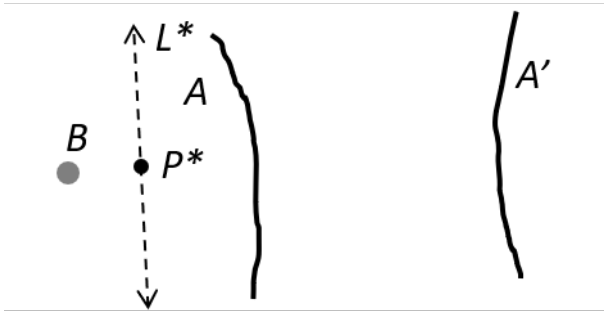


Figure 3.3: The optimal location for *P* and *L*.

However, since gradient descent only allows for local parameter changes, the planetary line must first pass through (or go around) strand *A* to get to this optimal location. Unfortunately, the closer the planetary line gets to strand *A*, the more it pushes *A* toward point *B*, and the less it pushes *A'* away from point *B*. This increases the loss.

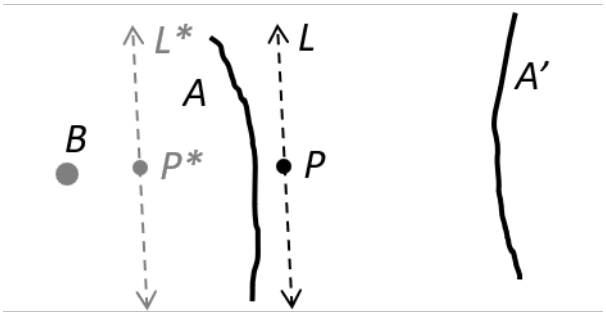


Figure 3.4: The position of L, while closer to L^*, results in a higher loss.

The `GravityLayer` would achieve a lower loss if it instead moved the planetary line toward A', which is farther away from its optimal location.

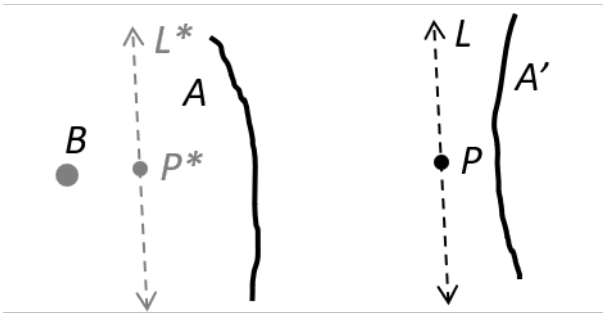


Figure 3.5: The position of L obtains a lower loss, but is farther away from L^*.

As a result, gradient descent moves the planet and planetary line away from their optimal locations.

In practice, this problem can be cirumvented with using several stacked `GravityLayer`s, where each layer's parameters are initialized randomly. Some of the layers will end up with parameters close to their optimal values, at which point gradient descent can successfully guide the planetary line to the right spot. Meanwhile, though the layers whose parameters are initialized to values far away from an optimum might fail to help decrease the loss function, they will not increase it. In the worst case scenario, they can update the value of their s parameters to be equal to 1. A `GravityLayer` with $s = 1$ is the identity function, and so adding additional `GravityLayer`s does not cause a knot to become more tangled than

it started.

3.3 SmoothLayer

This layer performs a "soft" projection of the knot onto the unit sphere around a specific point $\vec{c}$. The coordinates of this point are trainable parameters.

A projection onto the unit sphere around $\vec{c}$ would transform each vertex $\vec{v_i}$ as follows:

$$\vec{v_i} \mapsto \frac{\vec{v_i} - \vec{c}}{\|\vec{v_i} - \vec{c}\|} + \vec{c}$$

This map has a discontinuity at $\vec{v_i} = \vec{c}$, which prevents it from qualifying as a homeomorphism.

Unlike a normal projection, however, the `SmoothLayer` avoids this discontinuity by tapering off the projection as $\vec{v_i}$ approaches $\vec{c}$. The transformation implemented by the `SmoothLayer` is as follows:

$$\vec{v_i} \mapsto (m\frac{\tanh(s\|\vec{v_i} - \vec{c}\|)}{\|\vec{v_i} - \vec{c}\| + \epsilon}(\vec{v_i} - \vec{c}) + (1 - m)(\vec{v_i} - \vec{c})) + \vec{c}$$

ϵ is an arbitrarily small positive constant.[1] The parameter $m \in [0, 1)$ determines the "strength" of the transformation, and linearly interpolates between the layer's usual output and its input. As $m \to 0$, the `SmoothLayer` becomes the identity function.[2]

The parameter $s \in (0, \infty)$ indicates how close to $\vec{c}$ an input point must be before the projection begins to taper off. At $s \approx 1.4$, most points within the unit sphere around $\vec{c}$ will be unmoved, while points outside the unit sphere (and those inside and near the boundary of the unit sphere) will be projected onto the unit sphere's surface.

[1] This is used purely to avoid errors when running the `SmoothLayer`. Regardless of ϵ's size, the layer will implement an ambient isotopy and the image of line segments under a `SmoothLayer` will have a continuous tangent vector.

[2] For convenience, the version of the `SmoothLayer` used to obtain this paper's results restricted m to the open interval $(0, 1)$. This does not prevent the `SmoothLayer` from implementing an ambient isotopy.

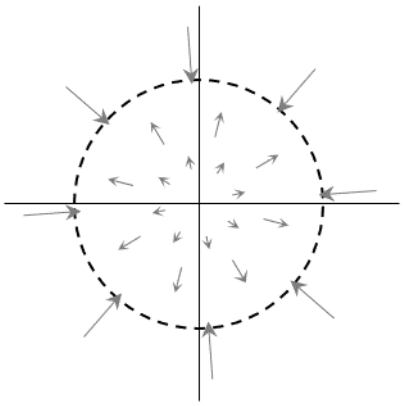


Figure 3.6: Spatial distortion for $s \approx 1.4$. Dotted line represents the unit circle.

As $s \to \infty$, points ever closer to $\vec{c}$ will be projected onto the surface of the unit sphere, and the projection becomes less "soft."

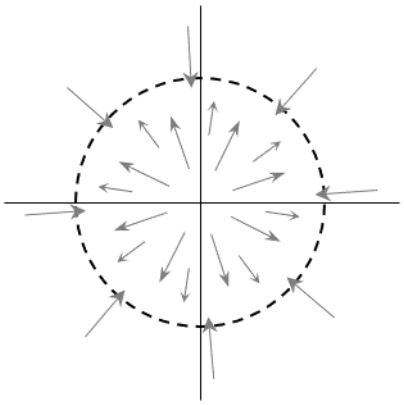


Figure 3.7: Spatial distortion as $s \to \infty$.

As $s \to 0$, points far outside the unit sphere will still be projected onto its surface, but closer points will instead be strongly drawn toward $\vec{c}$.

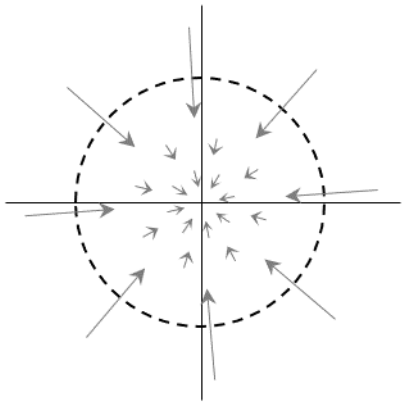


Figure 3.8: Spatial distortion as $s \to 0$.

Expected behavior for the `SmoothLayer` is for s to end up greater than 1.4. Values of s less than 1.4 are rarely helpful in untangling knots.

Efficacy

The `SmoothLayer` is most effective when placed among the final layers in a neural network. If it accepts as input an unknot that has been untangled to the point where it resembles a jagged loop, the `SmoothLayer` immediately converts it into an almost perfect circle.

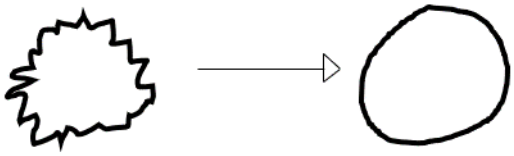


Figure 3.9: An ideal input and output for the `SmoothLayer`.

This same effect could be accomplished with several layers that perform local smoothing transformations on the knot, each operating at a different location. The advantage of the `SmoothLayer` is that those operations can be reduced to a single layer.

3.4 FoldLayer

This layer folds a plane of its choice into a parabolic surface. First, it reorients space via a trainable rotation matrix $\mathbf{R}$ and a translation vector $\vec{c}$ such that

$$\vec{v_i} \mapsto \mathbf{R}(\vec{v_i} - \vec{c}).$$

This maps a particular plane in $\mathbb{R}^3$ to the x, y plane. Next, the `FoldLayer` curls this new x, y plane upwards or downwards along the z-axis, creating a parabolic surface via the map

$$(x, y, z) \mapsto (x, y, z + ay^2),$$

where $a \in \mathbb{R}$ is a trainable strength parameter. Positive values of a correspond to an upwards curl, while negative values result in a downwards curl. The curl becomes more pronounced when $|a|$ is large.

Once this transformation is complete, the `FoldLayer` returns space back to its original orientation, first applying the inverted rotation matrix $\mathbf{R}^{-1}$ and then adding the translation vector $\vec{c}$.

Efficacy

Of the layers listed in this section, the `FoldLayer` proved the least powerful. It could only effectively untangle knots when placed into a neural network that contained other kinds of custom layers. This was likely due to two factors.

1. Nearby points are stretched farther from each other at locations more distant from the base of the parabolic surface. These spatial distortions can cause the vertices of a polygonal knot to become unevenly distributed, which makes the discrete Möbius energy a worse approximation to the continuous Möbius energy.

2. The `FoldLayer` is unable to perform small, local perturbations upon a knot. These are often a necessity when correcting entanglements. As a result, while the `FoldLayer` might be able to fix macro-scale tangles, it requires the assistance of other layers to do so on the micro scale.

Chapter 4

Untangling Knots

The custom layers described in the previous section were able to successfully untangle several sample knots.

4.1 Experimental Setup

We used the same neural network architecture when untangling each of our sample knots. It consisted of 24 stacked trios of a `TwistLayer`, followed by a `FoldLayer`, followed by a `GravityLayer`. Then, at the end, a single `SmoothLayer` was added. This produced a neural network with 73 layers total.

The networks were implemented in Keras, and trained using the Nadam optimizer.[1] Each network was trained for 3000 epochs on a different polygonal knot with 400 vertices, all of which were approximations to some commonplace smooth knot.

Additionally, some of our training knots were "pre-tangled" in order to make the task of untangling them harder for the neural network to accomplish. To tangle the knots, we ran their vertices through a function that applied a different 3D rotation matrix to the data depending on how far they were from the origin. A vertex $\vec{v}$ with norm $\|\vec{v}\|_2$ would be run through a rotation with parameters for its angles as follows:

$$\begin{aligned} \text{Yaw} &= \cos(4\|\vec{v}\|_2) * \sin(2\|\vec{v}\|_2) \\ \text{Pitch} &= \cos(2\|\vec{v}\|_2) * \sin(4\|\vec{v}\|_2) \\ \text{Roll} &= -\cos(2\|\vec{v}\|_2) * \sin(2\|\vec{v}\|_2) \end{aligned}$$

[1] In practice, Nesterov momentum helped prevent polygonality violations.

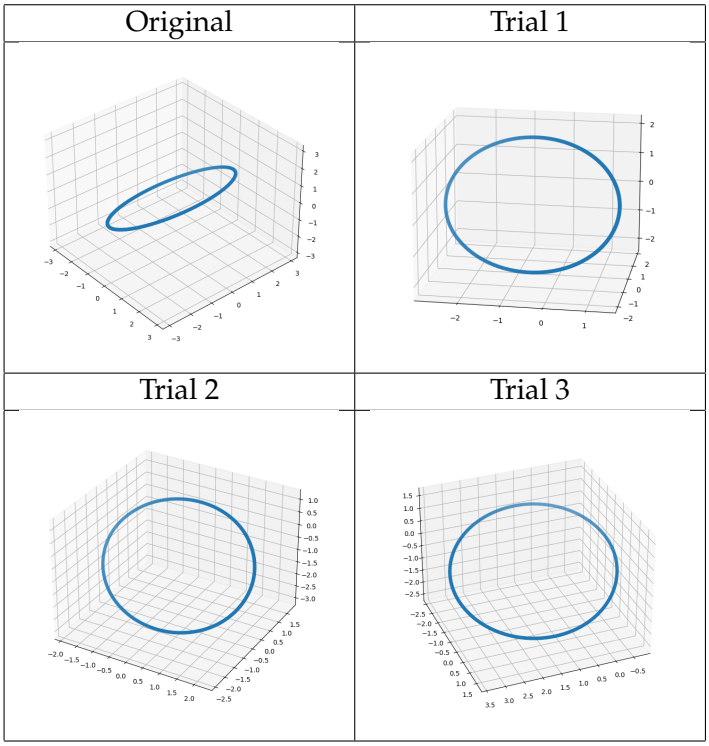
This transformation implements an ambient isotopy, and so should make a knot more complicated without changing the equivalence class to which it belongs.

Since there is some randomness in the selection of our neural networks' initial parameters, we performed the training process 3 times on each knot to showcase that the neural networks' performance was not heavily dependent on their initial conditions. The results of these trials are detailed below.

4.2 Results

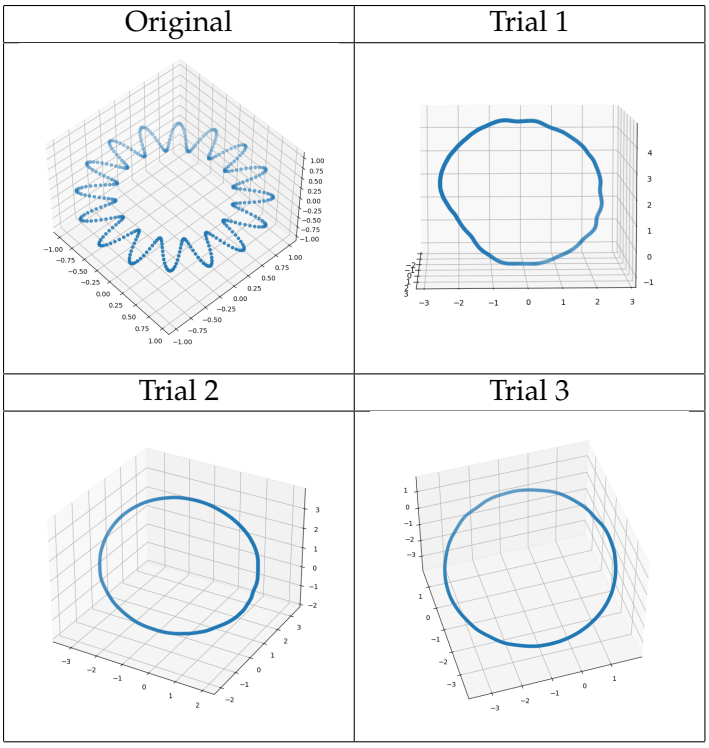
We will begin with less visually complex knots, then proceed to more complex ones at the end of the section.

Oval



We begin with an extremely obvious unknot: an oval. The unknot with minimum energy is a circle, so our neural network merely compresses its input into a ring-shape.

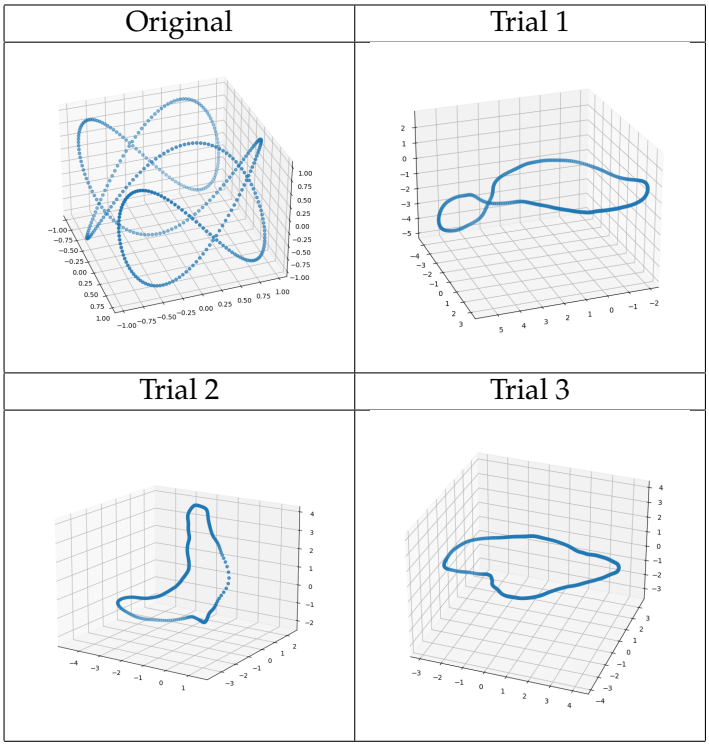
Sixteen-point star



This variant of the unknot is a flat, sixteen-point star. It tests the neural network's ability to smooth several jagged locations at once.

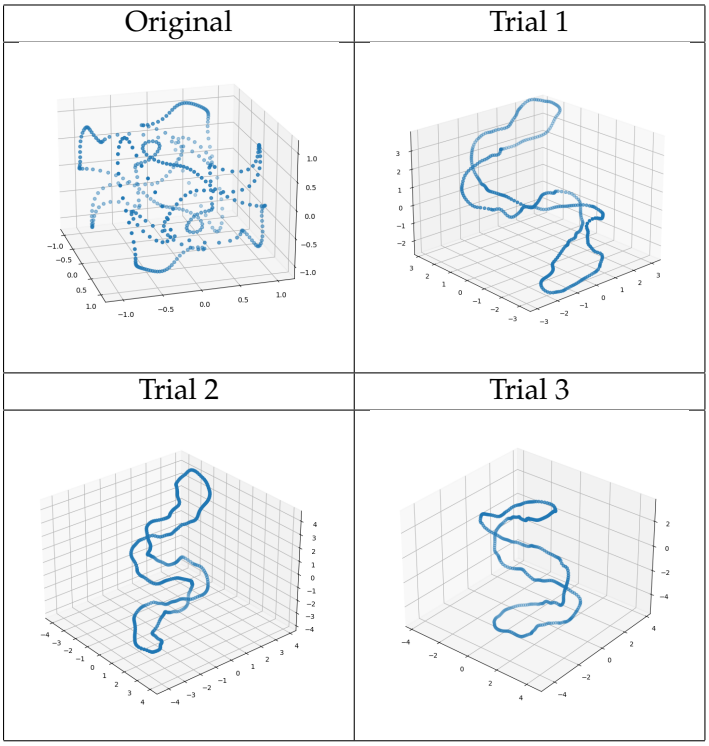
The neural network accomplishes its task well. Each of the trials resulted in a near-perfect circle.

Lissajous unknot



This is a very simple Lissajous unknot. The neural network is able to untangle it quite easily, though it occasionally has difficulties removing one or two twists. In one of our trials, it left the Lissajous knot in the form of a figure-eight.

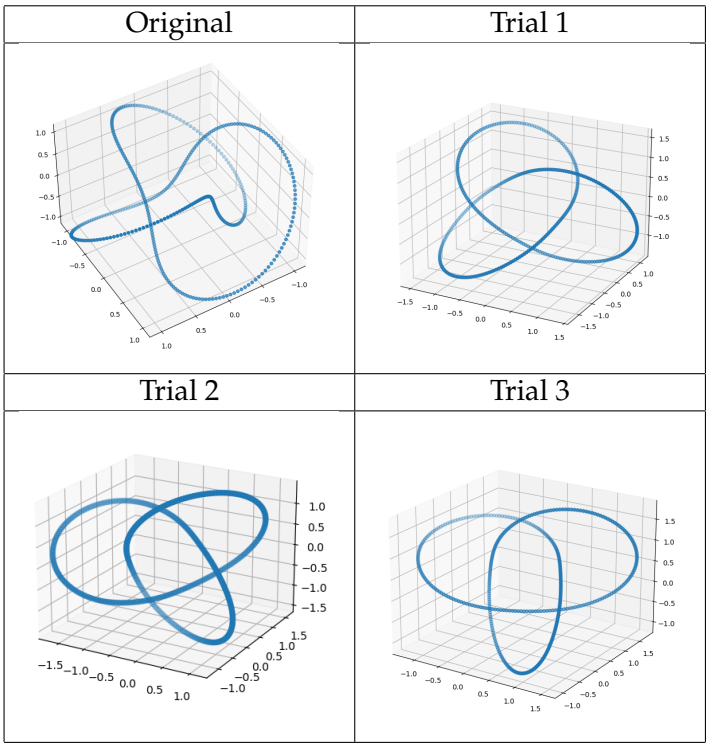
Lissajous unknot (tangled)



After harshly tangling the Lissajous unknot, our neural network begins to leave more twisted segments in place, though it is still able to reduce the knot energy of its input considerably. All three trials left two crossings in place, and trial 1 comes very close to leaving three.

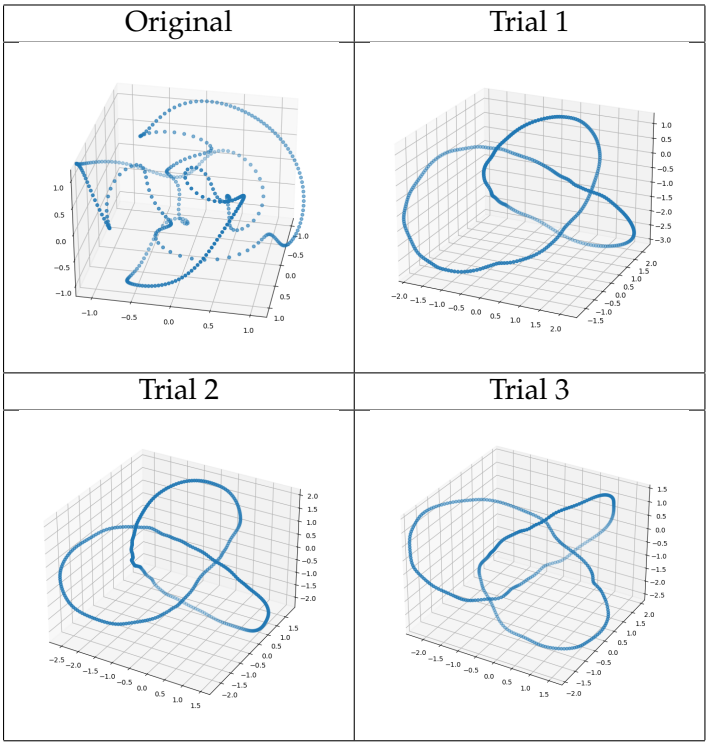
This displays a limitation of the neural network architecture: the concluding `SmoothLayer` cannot be trained to remove the final crossings on its own, and so local parameter changes will be less effective at removing these seemingly trivial twists. An architecture that concluded with a `TwistLayer` could potentially surmount this problem. Alternatively, a version of the loss function that took into account the Möbius energy of the knot produced at intermediate layers, rather than solely the Möbius energy of the output, might be able to avoid the problem as well.

(2, 3)-torus knot



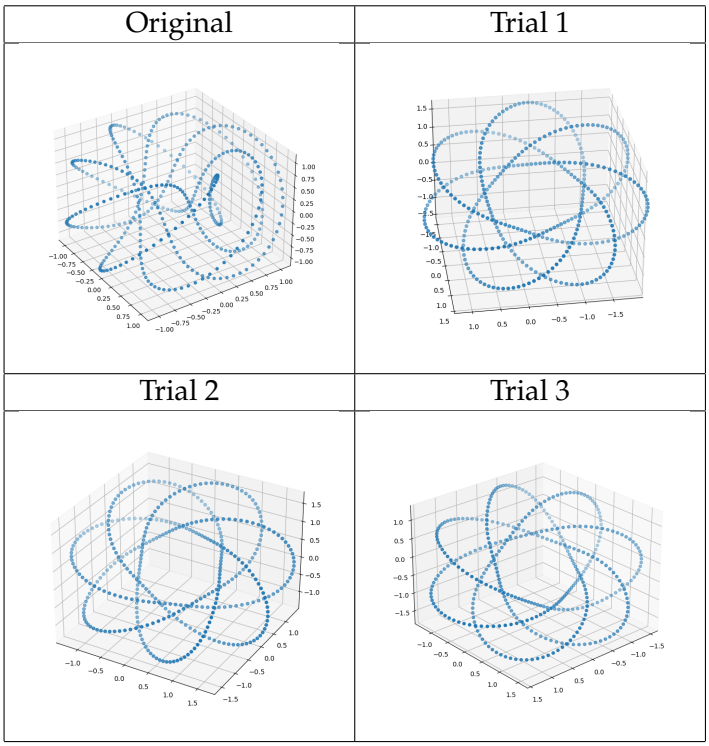
The $(2, 3)$-torus knot, or the trefoil knot, is non-trivial. Since this version of the trefoil knot already possesses a low knot energy, the neural network behaves correctly and barely modifies it.

(2, 3)-torus knot (tangled)



This version of the trefoil knot has been tangled, and has a high initial knot energy. The neural network successfully smooths the knot but is (correctly) unable to reduce it to the unknot.

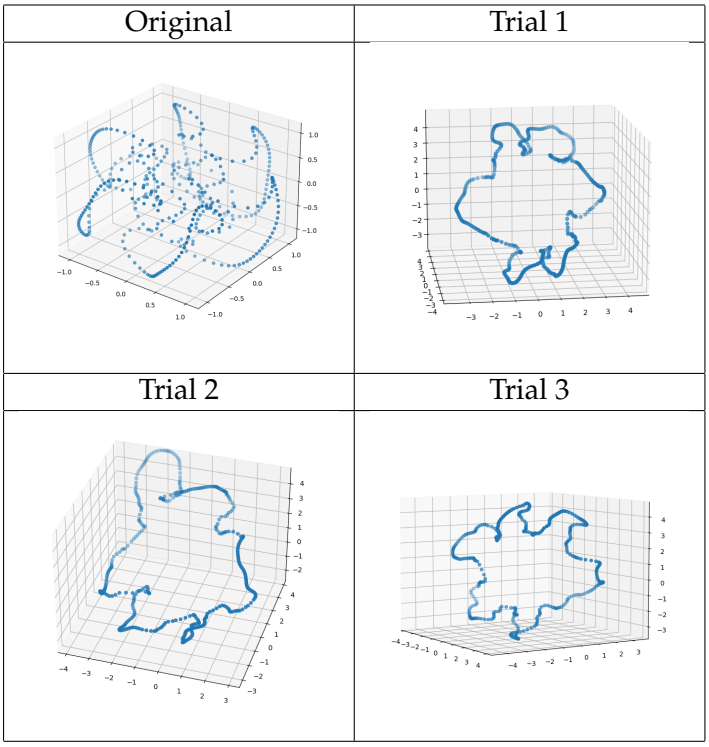
(4, 7)-torus knot



The $(4, 7)$-torus knot is significantly more visually complicated than the trefoil knot, but this variant is still very close to reaching a minimal knot energy. The neural network intelligently avoids disturbing the existing energy minimum in all three of our trials, making only minor improvements to the knot energy each time.

Since the $(4, 7)$-torus knot is non-trivial, we are unable to convert it into a visually recognizable unknot. As with the $(2, 3)$-torus knot, the neural network behaves correctly on this front.

(1, 6)-torus knot (tangled)



The $(1, 6)$-torus knot is equivalent to the unknot. Tangling it, however, makes it difficult to recognize as such.

The neural network succeeds in transforming the torus knot into a ring-like shape, albeit one with several twists along its perimeter. The knot energy of the output is significantly reduced.

4.3 Conclusions and Future Work

Our neural network shows a lot of promise. It is able to untangle complicated knots that the human eye would be unable to easily identify as the unknot. There is, however, clear room for improvement. The crossed segments from the tangled Lissajous and the $(1, 6)$-torus knot trials remain, despite their existence contributing to a higher knot energy. Additional layers, or a modified neural network architecture, might be able to resolve the

problem.

Furthermore, the two methods for preventing polygonality violations discussed in section 2.2 could be further developed. At present, whether a neural network can transform a set of vertices into a visually recognizable unknot is only a heuristic for whether the polygonal knot represented by the vertices is actually equivalent to the unknot. Polygonality violations, though unlikely in practice, can still occur with the present neural network architecture.

If polygonality violations could be ruled out altogether, the polygonal knots whose vertices are represented by our neural network's input and output would always be ambient isotopic to one another, and whether the output is visually recognizable as the unknot would be a sufficient condition for whether the input is the unknot as well. Even without these guarantees, the neural network performs quite well–polygonality violations are rare, and the neural network constrains itself to legal operations when trying to reduce a knot's energy. Nevertheless, future work can and should attempt to reduce the frequency of polygonality violations to zero.

Bibliography

[1] Austin, A. K. (1985). Two curiosities. *The Mathematical Gazette*, 69:42–44.

[2] Li, J. and Peters, T. J. (2013). Isotopic convergence theorem. *Journal of Knot Theory and Its Ramifications*, 22.

[3] Milnor, J. W. (1950). On the total curvature of knots. *The Annals of Mathematics*, 52:248–257.

[4] Scholtes, S. (2014). Discrete möbius energy. *Journal of Knot Theory and Its Ramifications*, 23.09.

[5] Simon, J. K. (1994). Energy functions for polygonal knots. *Journal of Knot Theory and Its Ramifications*, 3.03:299–320.

[6] Sullivan, J. M. (2008). Curves of finite total curvature. *Discrete Differential Geometry*, 38:137–161.