Can a neural network learn planar topology?

Jean-Michel Morel
Ecole Normale Supérieure Paris-Saclay, France
Joint work with Adrien Courtois and Pablo Arias

Conference at CUHK MATH-IMS Applied Maths series, April 16, 2021

In this talk we address the simplest instance of the figure-ground problem, raised in 1921 by the cognitive psychologist Edgar Rubin. The problem is to understand how human perception segments an image into foreground figures and their background.

The simplest instance of the figure-ground problem is the closure problem. For psychologists, the interior of a Jordan curve is perceived as a figure, and its exterior as its background.

For mathematicians, this is just the Jordan curve theorem. It is easily solved numerically: There is an elementary algorithm tagging the interior points of any rasterized Jordan curve in a digital image.

This problem is therefore a good playground to evaluate how far we stand in artificial intelligence.
We shall evaluate the performance of neural networks on this problem and attempt to explain it based on recent mathematical observations on the learning power of neural networks.

Plan:

Illustration first : learning the Jordan curve theorem by CNN

Analysis of a theorem by Pedro Domingos (*)

Various reformulations, consequences, discussion

(*) DOMINGOS, Pedro. Every Model Learned by Gradient Descent Is Approximately a Kernel Machine. *arXiv preprint arXiv:2012.00152*, 2020.

References

On perception theory

Rubin, Edgar. Figure and ground. *Readings in perception*, 1958.

On shapes and implementation of perception theory

Courtois, Adrien, JMM, Arias, Pablo (2021)
Non-local layers reducing network depth for global analysis, *submitted.*

Alvarez, L., Monzón, N., & Morel, J. M. (2018). Interactive design of random aesthetic abstract textures by composition principles. *Leonardo*, 1-11, MIT Press

On structure of learning, and the neural tangent kernel

Minsky, M., & Papert, S. A. (1969). Perceptrons: An introduction to computational geometry. *MIT press*, *2017 reedition.*

Domingos, Pedro (2020). Every Model Learned by Gradient Descent Is Approximately a Kernel Machine. *arXiv preprint arXiv:2012.00152*.

Jacot, A., Gabriel, F., & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Adv. Neural Inf. Proc. Sys., 31:8571{8580, 2018.*

Charpiat, G., Girard, N., Felardos, L., & Tarabalka, Y. (2019, December). Input Similarity from the Neural Network Perspective. In *NeurIPS 2019-33th Annual Conference on Neural Information Processing Systems*.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., & Wang, R. (2019). On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*.

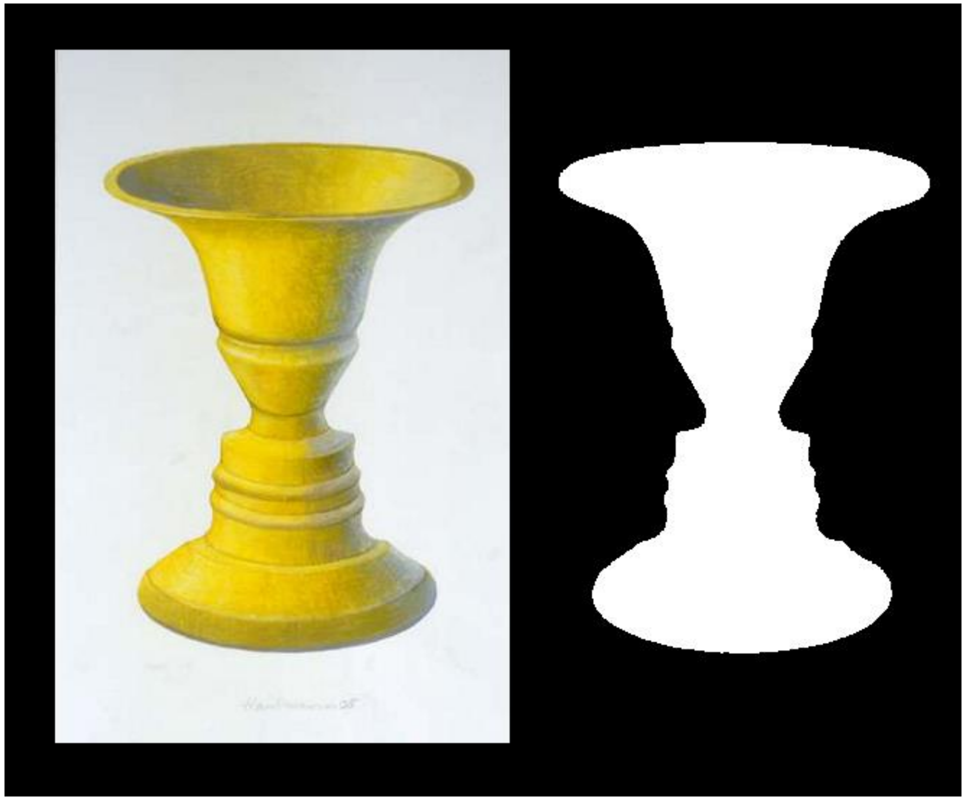
Plan:

Illustration first : learning the Jordan curve theorem by CNN

Analysis of a theorem by Pedro Domingos (*)

Various reformulations, consequences, discussion

(*) DOMINGOS, Pedro. Every Model Learned by Gradient Descent Is Approximately a Kernel Machine. *arXiv preprint arXiv:2012.00152*, 2020.



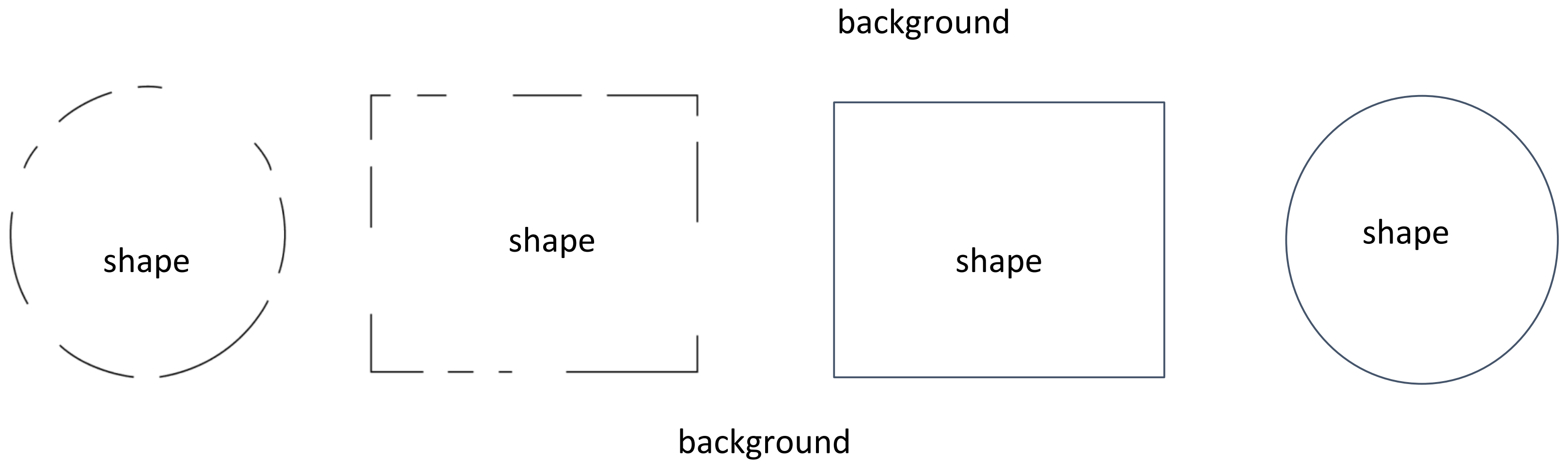
A vase or two human profiles? **The figure ground dilemma by Edgar Rubin (1921)**

Distinguishing frontal objects from their background is a fundamental problem of perception

RUBIN, Edgar. Figure and ground. *Readings in perception*, 1958.

MLA

Fig: https://en.wikipedia.org/wiki/Edgar_Rubin#/media/File:Rubin2.jpg



The closure law
There is a perceptual tendency to close a curve and to perceive its interior as an object, its exterior as background. This one geometric formulation of the figure/background dilemma.

Wertheimer, Max. "Untersuchungen zur Lehre von der Gestalt. II." *Psychologische forschung* 4.1 (1923): 301-350.

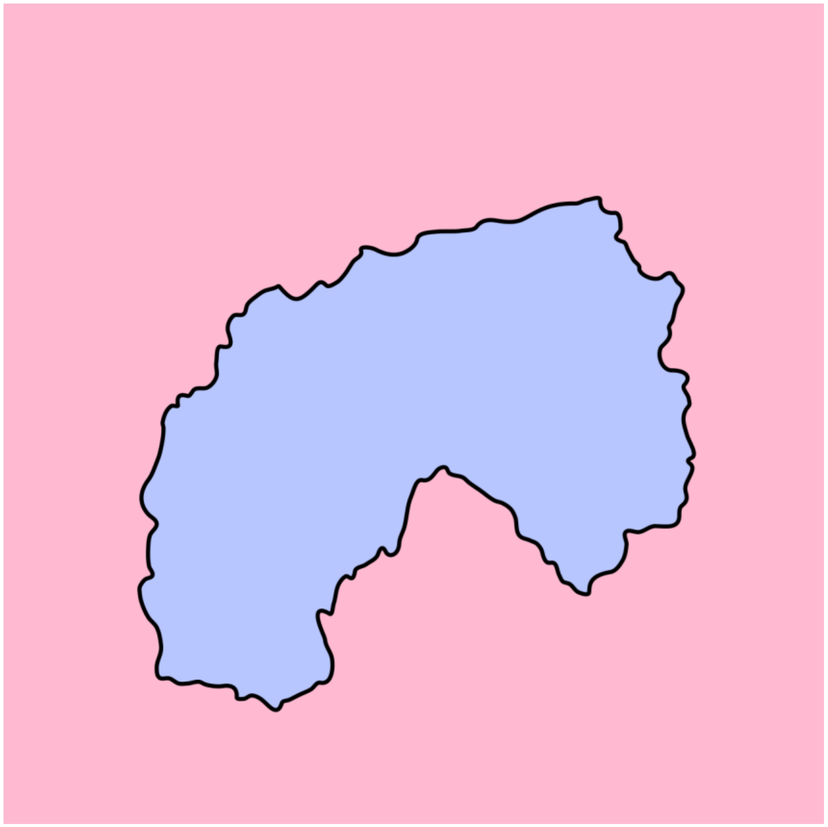
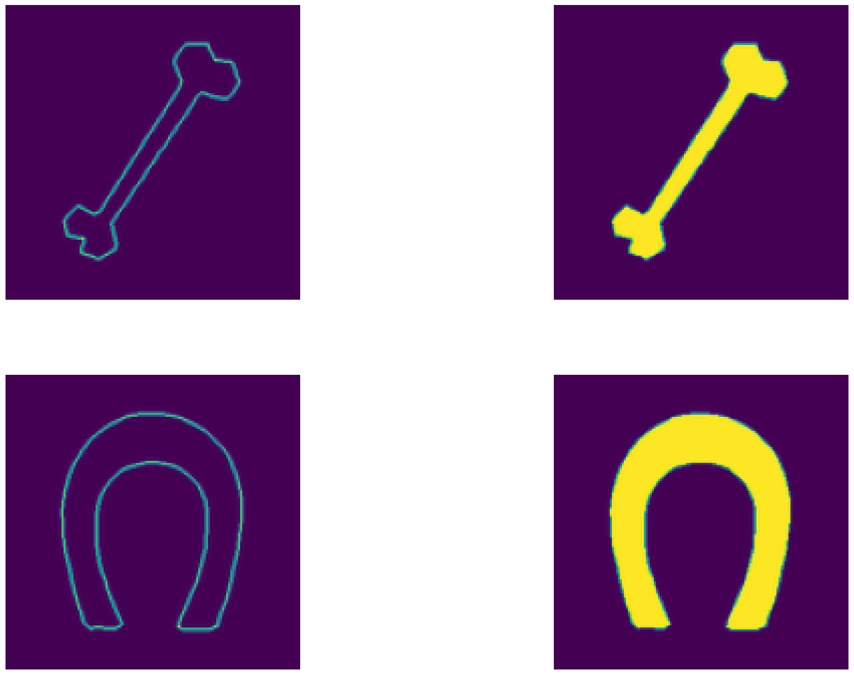


Illustration of the Jordan curve theorem. A Jordan curve (drawn in black) divides the plane into an "inside" region (light blue) and an "outside" region (pink). This is the simplest version of the perception law of closure. The inside region is the “figure”. The outside region the “background”

Jordan, Camille (1887), *Cours d'analyse*

Fig: https://en.wikipedia.org/wiki/Jordan_curve_theorem

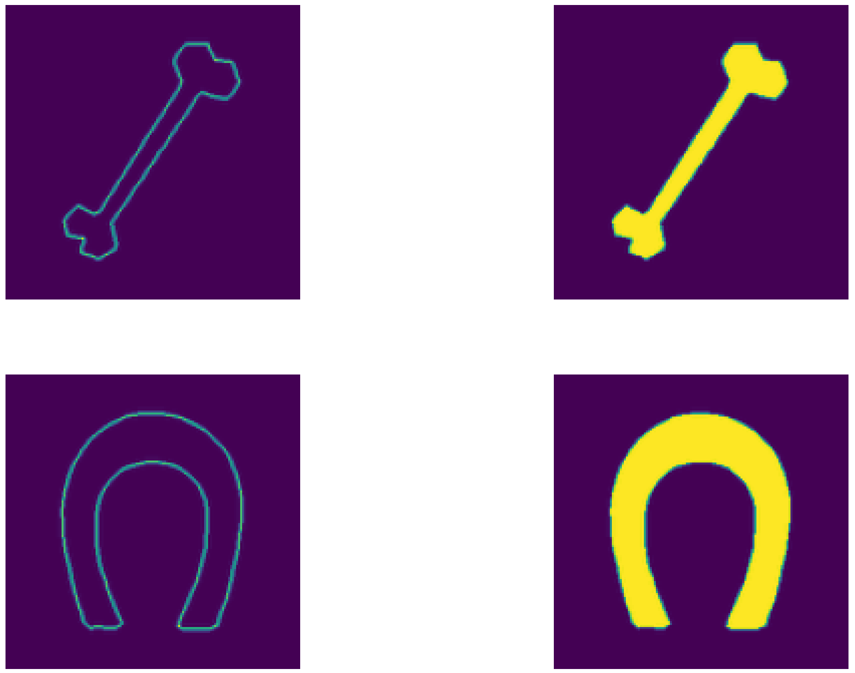
Illustration: a Shape Dataset



Given the boundary of a shape, the network has to reconstruct the shape
i.e classify each pixel of the image as belonging to the shape or not

1,115 training images, data augmentation by random affine
distortions, 140 test images, images of size 128x128

Illustration: the Shape Dataset



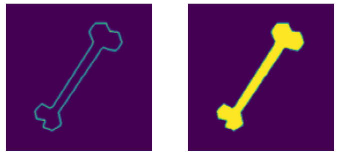
Particularity: it can be solved with 2 lines of code!

(Compute the parity of the number of crossings on a horizontal half-line starting from each pixel)

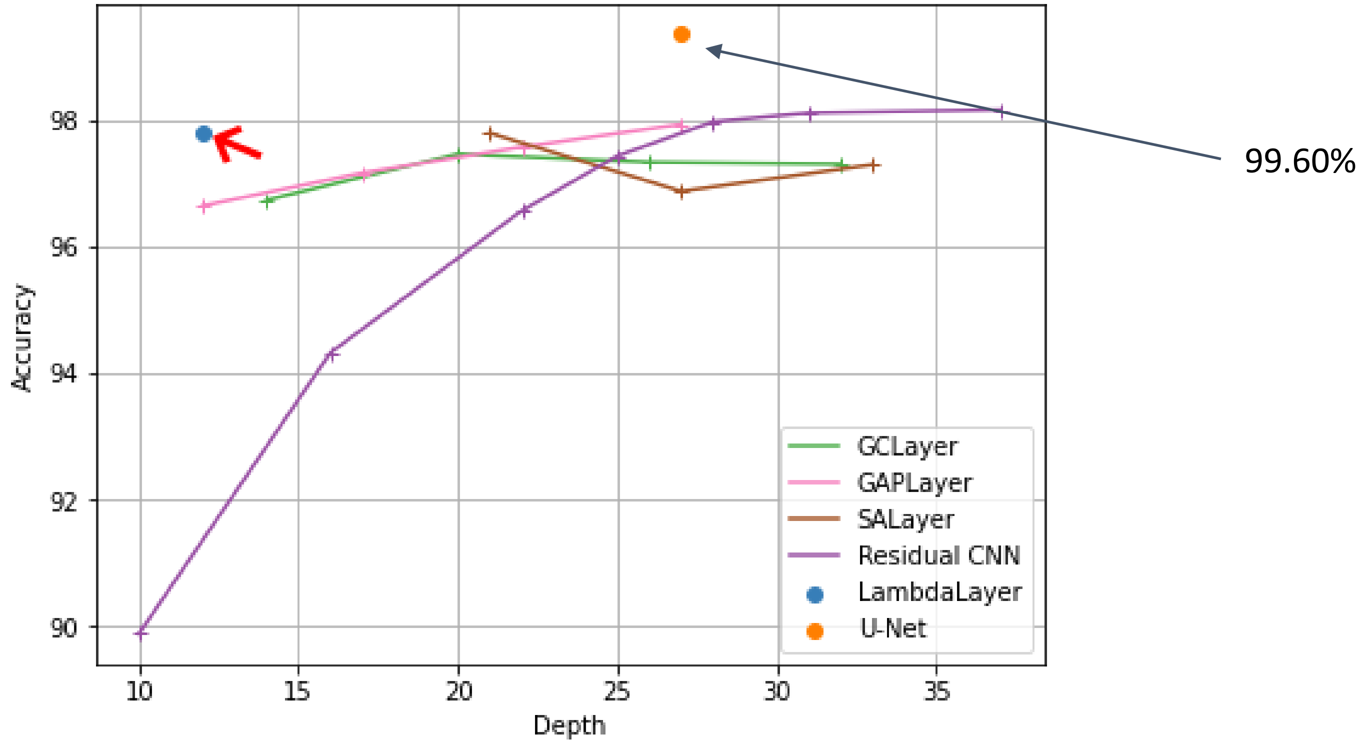
A lightweight CNN programmed *ad hoc* <u>can do</u> this task provided the receptive field is large enough, but then the question is: can this simple algorithm be learnt from examples? 🤔

Minsky, M., & Papert, S. A. (1969). Perceptrons: An introduction to computational geometry. *MIT press*, *2017 reedition.*

Illustration: the Shape Dataset



Can a network learn this very simple operation?

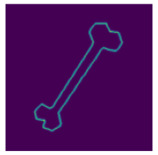


GAP: global average pooliing. SA: self-attention, GC: Global Context

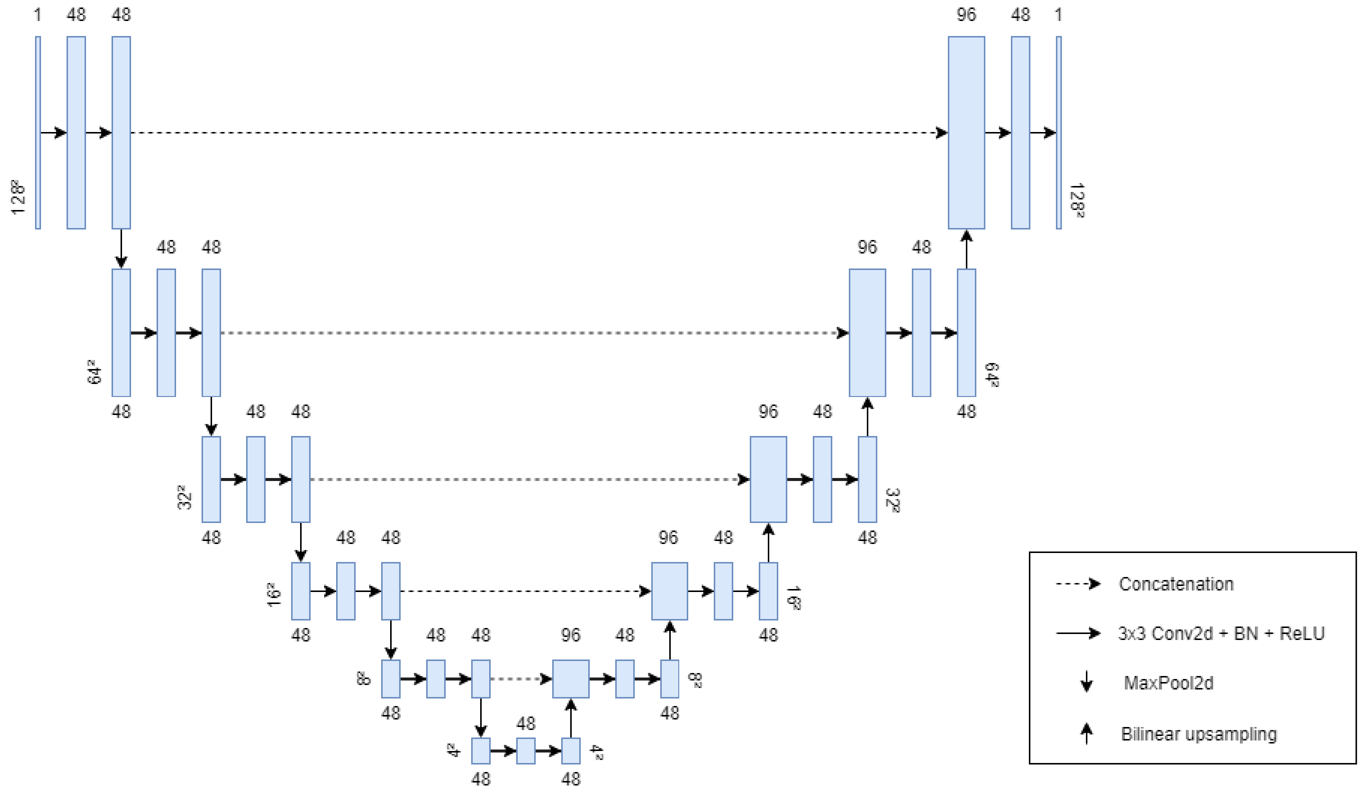
It’s an impossible task for CNNs working locally (*):
It requires a network with a significant depth (or non-local layers), or a U-net

(*) Minsky, M., & Papert, S. A. (1969). Perceptrons: An introduction to computational geometry. *MIT press, 2017 reedition.*

Illustration: the Shape Dataset



Can a network learn this very simple operation?



A simple candidate network: A U-Net with 6 scales

Trained with Ranger for 150 epochs, binary cross-entropy loss, batch size 32

Illustration: the Shape Dataset

Can a network learn this very simple operation?

Examples of results (left: input, right: output)

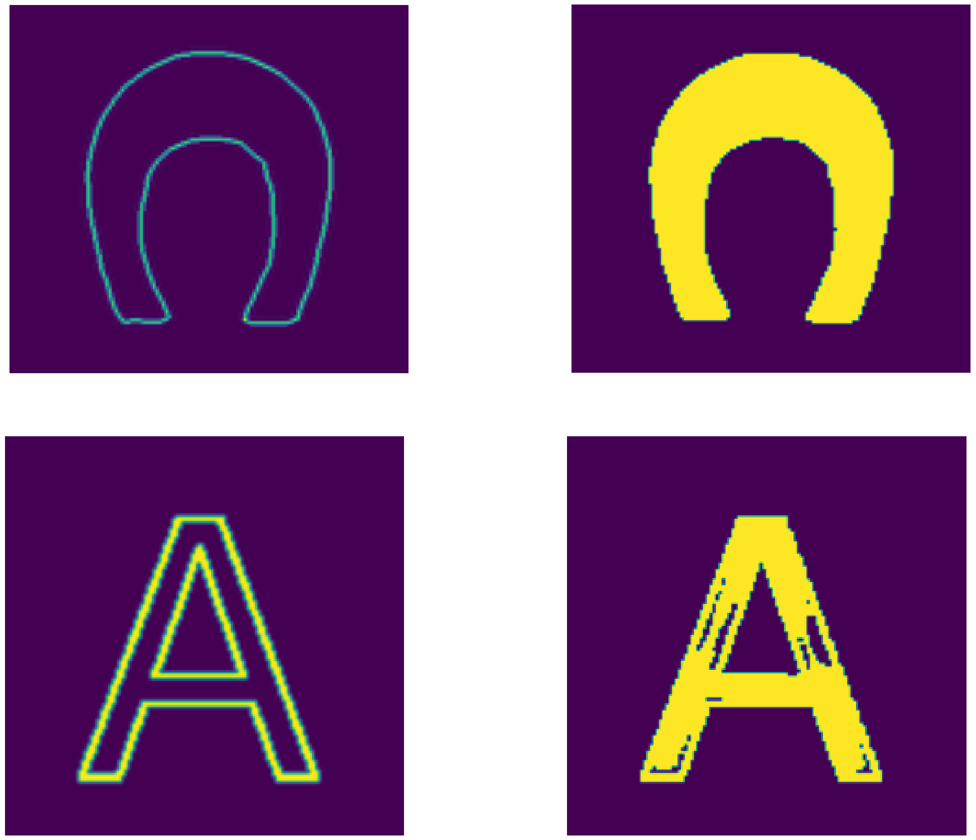
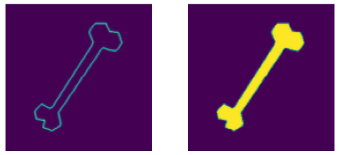
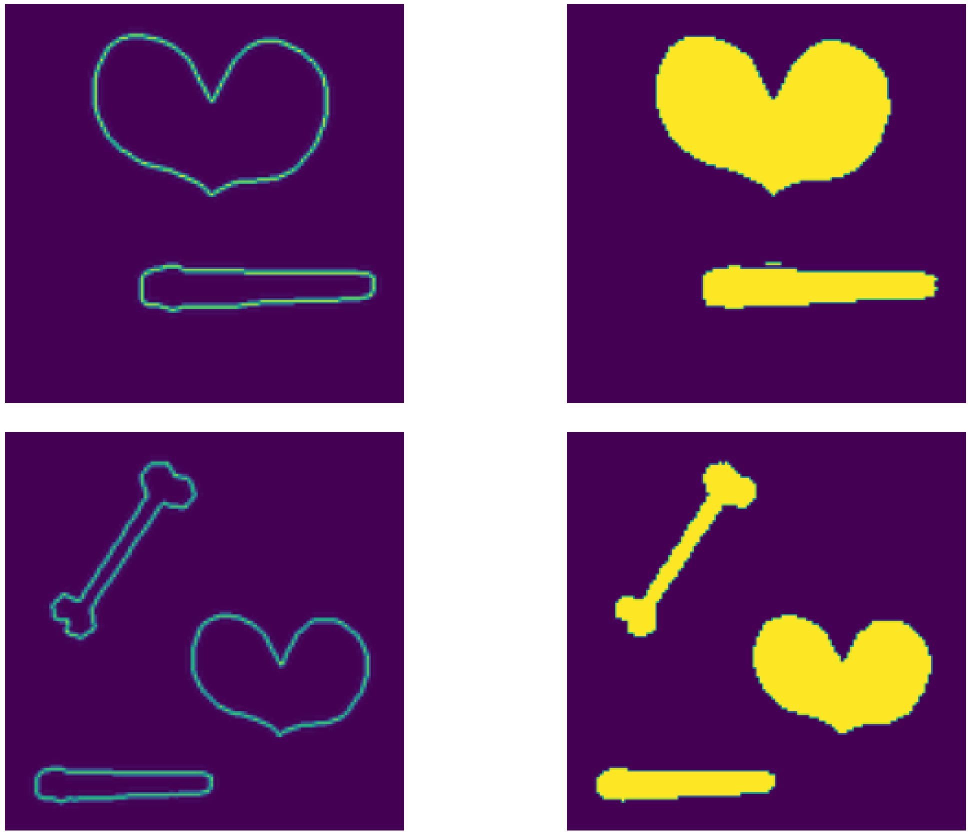


Illustration: the Shape Dataset



Can a network learn this very simple operation?

The results are fine, even when tested on two or three shapes (the train set contains only one):

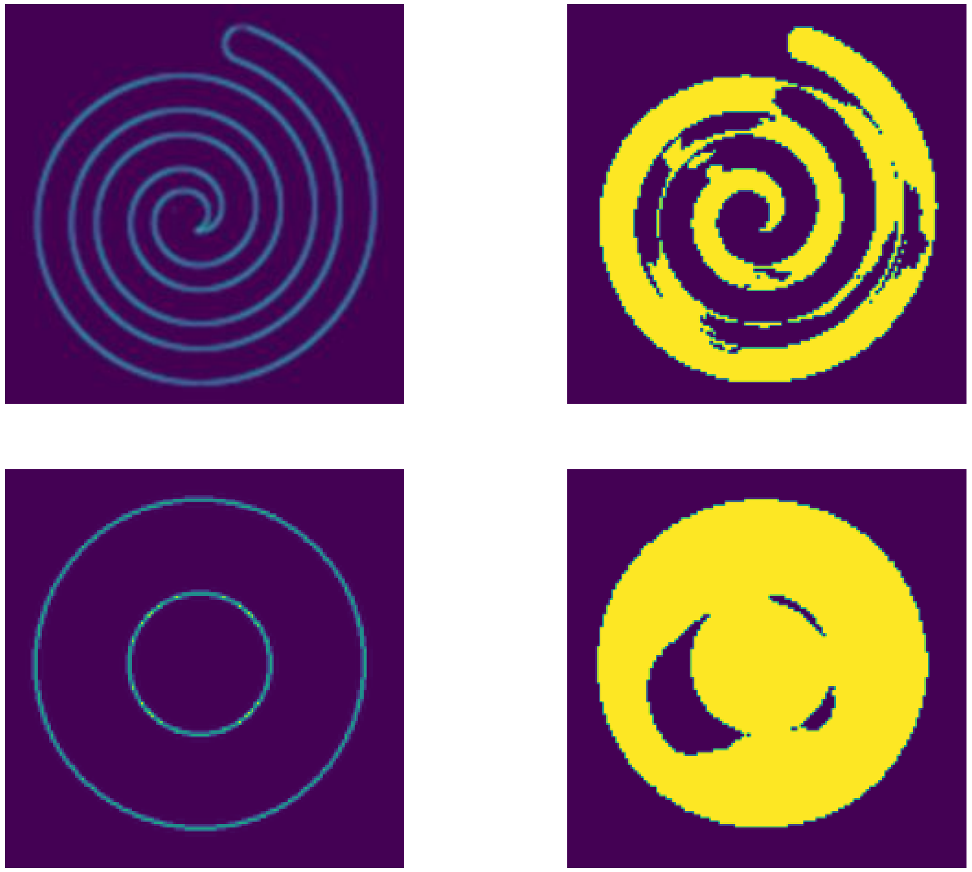


But does it mean it learned the best algorithm?

Illustration: the Shape Dataset

Can a network learn this very simple operation?

But does it mean it learned the best algorithm? **NO!**



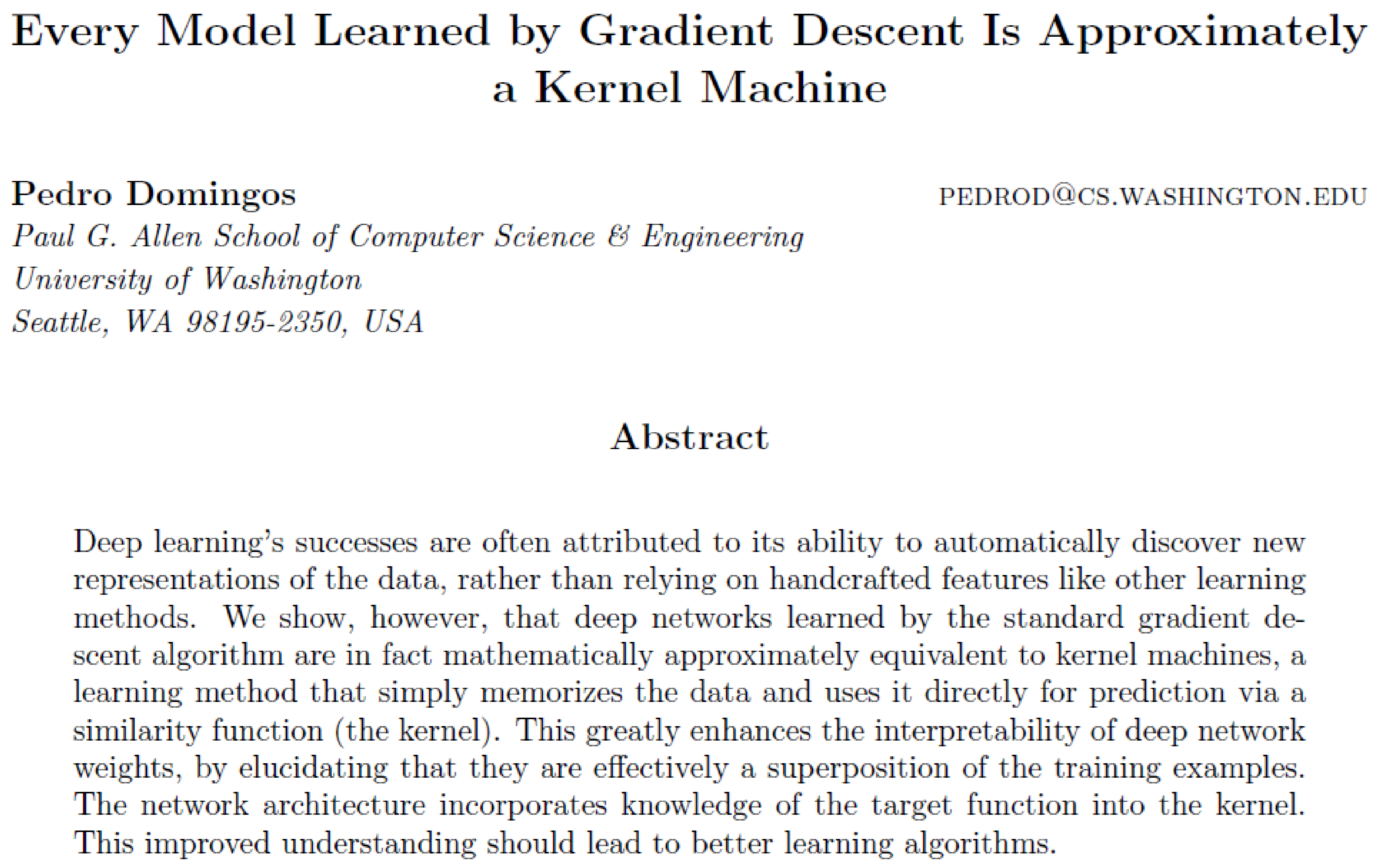
Plan:

Illustration first : learning the Jordan curve theorem by CNN

Analysis of a theorem by Pedro Domingos (*)

Various reformulations, consequences, discussion

(*) DOMINGOS, Pedro. Every Model Learned by Gradient Descent Is Approximately a Kernel Machine. *arXiv preprint arXiv:2012.00152*, 2020.



Every Model Learned by Gradient Descent Is Approximately a Kernel Machine

Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350, USA

Abstract

Deep learning's successes are often attributed to its ability to automatically discover new representations of the data, rather than relying on handcrafted features like other learning methods. We show, however, that deep networks learned by the standard gradient descent algorithm are in fact mathematically approximately equivalent to kernel machines, a learning method that simply memorizes the data and uses it directly for prediction via a similarity function (the kernel). This greatly enhances the interpretability of deep network weights, by elucidating that they are effectively a superposition of the training examples. The network architecture incorporates knowledge of the target function into the kernel. This improved understanding should lead to better learning algorithms.

Classic example of kernel:

$$K(x, x_i) = e^{-||x-x_i||^2}$$

A kernel machine is a model of the form

$$y = g\left(\sum_i a_i K(x, x_i) + b\right),$$

where x is the query data point, the sum is over training data points x_i, g is an optional nonlinearity, the a_i's and b are learned parameters, and the kernel K measures the similarity of its arguments (Schölkopf and Smola, 2002). In supervised learning, a_i is typically a linear function of y_i^*, the known output for x_i. Kernels may be predefined or learned (Cortes et al., 2009). Kernel machines, also known as support vector machines, are one of the most developed and widely used machine learning methods. In the last decade, however, they have been eclipsed by deep networks, also known as neural networks and multilayer perceptrons, which are composed of multiple layers of nonlinear functions. Kernel machines can be viewed as neural networks with one hidden layer, with the kernel as the nonlinearity.

Schölkopf, B., Smola, A. J., & Bach, F. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Whether a representable function is actually learned, however, depends on the learning algorithm. Most deep networks, and indeed most machine learning models, are trained using variants of gradient descent (Rumelhart et al., 1986). Given an initial parameter vector w_0 and a loss function $L = \sum_i L(y_i^*, y_i)$, gradient descent repeatedly modifies the model's parameters w by subtracting the loss's gradient from them, scaled by the learning rate ϵ:

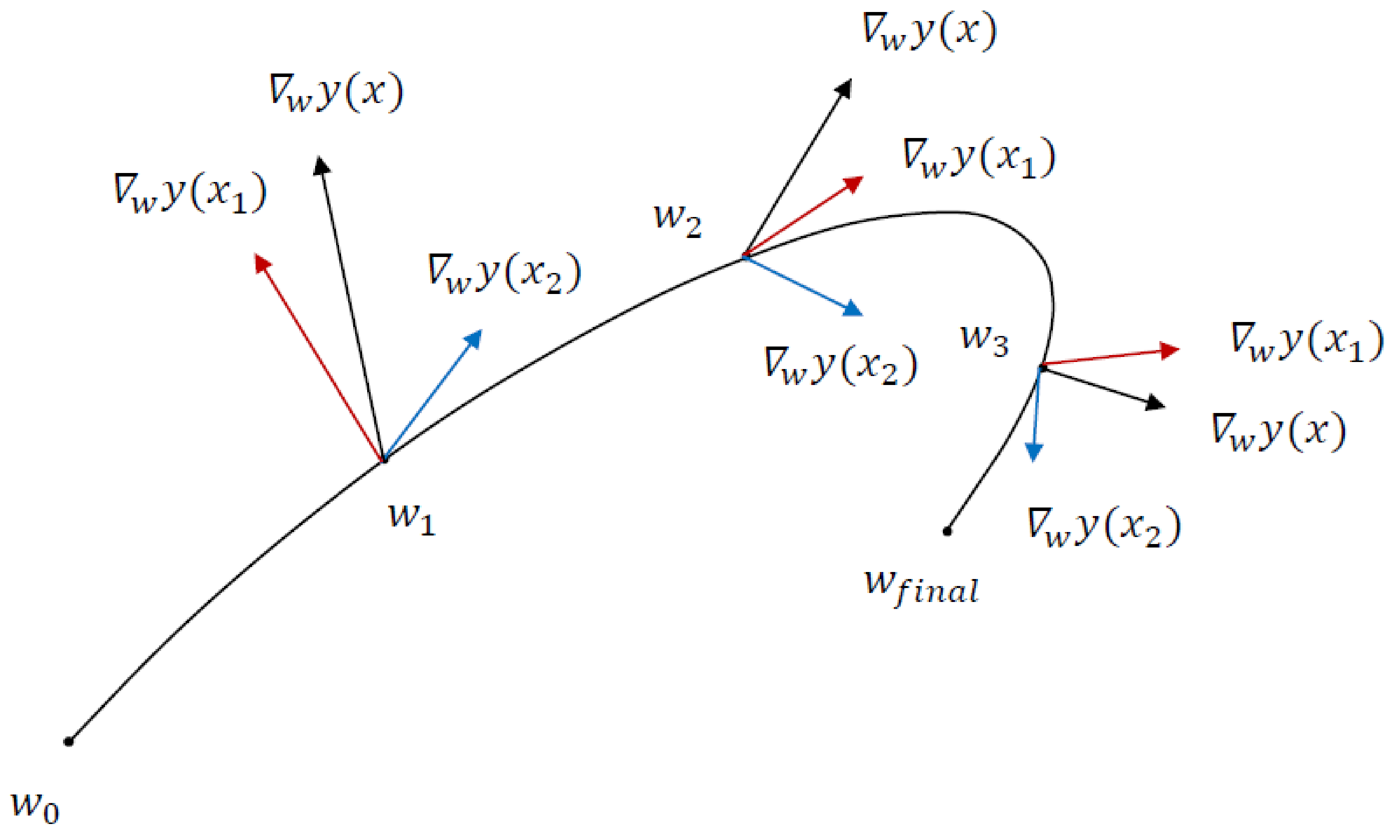
$$w_{s+1} = w_s - \epsilon \nabla_w L(w_s).$$

The process terminates when the gradient is zero and the loss is therefore at an optimum (or saddle point). Remarkably, we have found that learning by gradient descent is a strong enough constraint that the end result is guaranteed to be approximately a kernel machine, regardless of the number of layers or other architectural features of the model.

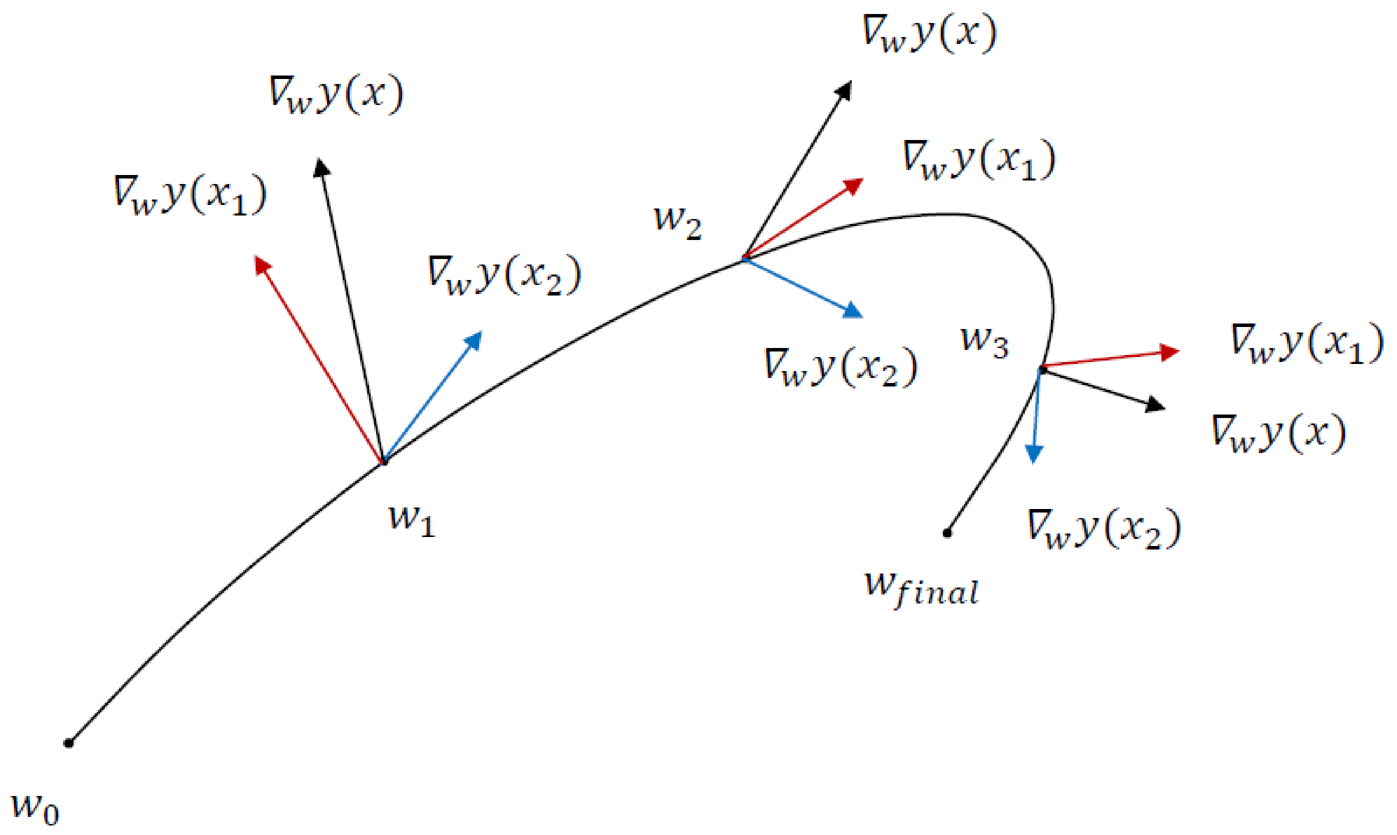
Specifically, the kernel machines that result from gradient descent use what we term a *path kernel*. If we take the learning rate to be infinitesimally small, the path kernel between two data points is simply the integral of the dot product of the model's gradients at the two points over the path taken by the parameters during gradient descent:

$$K_f(x, x') = \int_0^t \nabla_w f_{w(t)}(x) \cdot \nabla_w f_{w(t)}(x')$$

where $c(t)$ is the path. Intuitively, the path kernel measures how similarly the model at the two data points varies during learning. The more similar the variation for x and x', the higher the weight of x' in predicting y. Fig. 1 illustrates this graphically.



How the path kernel measures similarity between examples. In this two-dimensional illustration, as the weights follow a path on the plane during training, the model's gradients (vectors on the weight plane) for x, x_1 and x_2 vary along it. The kernel $K(x, x_1)$ is the integral of the dot product of the gradients $\nabla_w y(x)$ and $\nabla_w y(x_1)$ over the path, and similarly for $K(x, x_2)$. Because on average over the weight path $\nabla_w y(x) \cdot \nabla_w y(x_1)$ is greater than $\nabla_w y(x) \cdot \nabla_w y(x_2)$, y_1 has more influence than y_2 in predicting y, all else being equal.



Definition 1 *The* tangent kernel *of a learning function* $f_w(x)$ *is* $K_{f,w}(x,x') = \nabla_w f_w(x) \cdot \nabla_w f_w(x')$.

Definition 2 *The* path kernel *associated with a learning function* $f_w(x)$ *and a learning path* $w(s)_{s\in[0,t]}$ *is*

$$K_{f,w}(x,x') = \int_0^t \nabla_w f_{w(s)}(x) \cdot \nabla_w f_{w(s)}(x') ds.$$

Notation

- Learning data: the samples $(x_i, y_i^\star)$
- Parameters of the learning device: $w = (w_j)_{j=1\dots d}$
- Parametric machine learning device $y = f(w, x), \quad y_i = f(w, x_i)$
- Loss function : $\mathbf{L}(w) = \sum_i L(y_i, y_i^\star) = \sum_i L(f(w, x_i), y_i^\star)$
- For example: $L(y_i, y_i^\star) = \big(f(w, x_i) - y_i^\star\big)^2$
- Gradient descent: $w(s + \epsilon) = w(s) - \epsilon \nabla_w \mathbf{L}(w(s)), \quad \frac{dw(t)}{dt} = -\nabla_w \mathbf{L}(w(t))$
- Learning path: $\mathbf{w} = (w(s))_{s \in [0,t]}, \quad y_i(s) = f(w(s), x_i)$
- "Slave learning path" of input x: $y(s) = f(w(s), x)$, $s \in [0, t]$
- Tangent kernel $K_{f,w}(x, x') = \nabla_w f(w, x) \cdot \nabla_w f(w, x')$
- Path kernel $K_{f,\mathbf{w}}(x, x') = \int_0^t \nabla_w f(w(s), x) \cdot \nabla_w f(w(s), x') ds$

Theorem 1 *Consider a differentiable learning machine model* $y = f(w,x)$. *Let* w *be the parameter learned from a training set* $(x_i, y_i^\star)_{i=1,\dots m}$ *by gradient descent of a loss function* $\mathbf{L}(w) = \sum_i L(y_i^\star, f(w,x_i))$. *Then*

$$y = f(w,x) = \sum_{i=1}^{m} a_i K_{f,\mathbf{w}}(x, x_i) + b,$$

where $K_{f,\mathbf{w}}$ *is the path kernel associated with* $f(w,x)$ *and the learning path* $\mathbf{w}$ *taken during gradient descent,* a_i *is the average* $\frac{\partial L}{\partial y_i}$ *along the path weighted by the corresponding tangent kernel, and* $b = y(0)$ *is the initial model.*

Definition 1 *The* tangent kernel *of a learning function* $f_w(x)$ *is*

$$K_{f,w}(x,x') = \nabla_w f(w,x) \cdot \nabla_w f(w,x').$$

Definition 2 *The* path kernel *associated with a learning function* $f_w(x)$ *and a learning path* $\mathbf{w} = w(s)_{s\in[0,t]}$ *is*

$$K_{f,\mathbf{w}}(x,x') = \int_0^t \nabla_w f(w(s),x) \cdot \nabla_w f(w(s),x') ds.$$

Proof a bit rewritten

Proof When its step tends to 0, the gradient descent method becomes

$$\frac{dw(t)}{dt} = -\nabla_w \mathbf{L}(w(t)). \tag{1}$$

Let $y(t) = f(w(t), x)$ be the "slave learning path". We have by the chain rule,

$$\frac{dy}{dt} = \sum_{j=1}^{d} \frac{\partial f}{\partial w_j}(w(t), x) \frac{\partial w_j(t)}{\partial t},$$

where $w(t) = (w_j(t))_{j=1,\cdots,d}$. Using (1), this yields

$$\frac{dy}{dt} = \sum_{j=1}^{d} \frac{\partial f}{\partial w_j}(w(t), x) \left(-\frac{\partial \mathbf{L}}{\partial w_j} \right),$$

But $\mathbf{L}(w) = \sum_i L(y_i, y_i^\star) = \sum_i L(f(w, x_i), y_i^\star)$, and using again the chain rule,

$$\frac{dy}{dt} = \sum_{j=1}^{d} \frac{\partial f}{\partial w_j}(w(t), x) \left(-\sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \frac{\partial f(w, x_i)}{\partial w_j} \right).$$

$$\frac{dy}{dt} = \sum_{j=1}^{d} \frac{\partial f}{\partial w_j}(w(t), x) \left(-\frac{\partial \mathbf{L}}{\partial w_j} \right),$$

But $\mathbf{L}(w) = \sum_i L(y_i, y_i^\star) = \sum_i L(f(w, x_i), y_i^\star)$, and using again the chain rule,

$$\frac{dy}{dt} = \sum_{j=1}^{d} \frac{\partial f}{\partial w_j}(w(t), x) \left(-\sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \frac{\partial f(w, x_i)}{\partial w_j} \right).$$

Rearranging terms:

$$\frac{dy}{dt} = -\sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \sum_{j=1}^{d} \frac{\partial f(w, x)}{\partial w_j} \frac{\partial f(w, x_i)}{\partial w_j}.$$

Let $L'(y_i^\star, y_i) = \frac{\partial L}{\partial y_i}$. Then using the definition of the tangent kernel

$$K_{f,w}(x, x') = \nabla_w f(w, x) \cdot \nabla_w f(w, x') = \sum_{j=1}^{d} \frac{\partial f(w, x)}{\partial w_j} \frac{\partial f(w, x')}{\partial w_j},$$

$$\frac{dy}{dt} = -\sum_{i=1}^{m} L'(y_i^\star, y_i(t)) K_{f,w(t)}(x, x_i).$$

$$K_{f,w}(x,x') = \nabla_w f(w,x) \cdot \nabla_w f(w,x') = \sum_{j=1}^{d} \frac{\partial f(w,x)}{\partial w_j} \frac{\partial f(w,x')}{\partial w_j},$$

$$\frac{dy}{dt} = -\sum_{i=1}^{m} L'(y_i^\star, y_i(t)) K_{f,w(t)}(x, x_i).$$

Integrating between 0 and t and recording that $y_i(t) = f(w(t), x_i)$

$$y(t) = y(0) - \int_0^t \sum_{i=1}^{m} L'(y_i^\star, y_i) K_{f,w(s)}(x, x_i) ds.$$

Multiplying and dividing by $\int_0^t K_{f,w(s)}(x, x_i) ds$, we get

$$y(t) = y(0) - \sum_{i=1}^{m} \left(\frac{\int_0^t L'(y_i^\star, y_i) K_{f,w(s)}(x, x_i) ds}{\int_0^t K_{f,w(s)}(x, x_i) ds} \right) \int_0^t K_{f,w(s)}(x, x_i) ds,$$

$$y = f(w,x) = \sum_{i=1}^{m} a_i K_{f,\mathbf{w}}(x, x_i) + b,$$

where $K_{f,\mathbf{w}}$ is the path kernel associated with $f(w,x)$ and the learning path $\mathbf{w}$ taken during gradient descent, a_i is the average $\frac{\partial L}{\partial y_i}$ along the path weighted by the corresponding tangent kernel, and $b = y(0)$ is the initial model.

Reformulation of the result

Let $L'(y_i^\star, y_i) = \frac{\partial L}{\partial y_i}$ be the loss derivative for the i-th output. Then

$$\frac{dy}{dt} = -\sum_{i=1}^{m} L'(y_i^\star, y_i) \nabla_w f(w(t), x) \cdot \nabla_w f(w(t), x_i)$$

$$y(t) = y(0) - \int_0^t \sum_{i=1}^{m} L'(y_i^\star, y_i(s)) \nabla_w f(w(s), x) \cdot \nabla_w f(w(s), x_i) ds$$

$$y(t) = y(0) - \sum_{i=1}^{m} \int_0^t L'(y_i^\star, y_i(s)) \nabla_w f(w(s), x) \cdot \nabla_w f(w(s), x_i) ds$$

If for example $L(y_i^\star, y_i) = \frac{1}{2}(y_i^\star - y_i)^2$, then

$$y(t) = y(0) - \sum_{i=1}^{m} \int_0^t (y_i(s) - y_i^\star) \nabla_w f(w(s), x) \cdot \nabla_w f(w(s), x_i) ds$$

Comparison of versions of the main formula

Version 1:

$$y(t) = y(0) - \sum_{i=1}^{m} \left(\frac{\int_0^t L'(y_i^\star, y_i) K_{f,w(s)}(x, x_i) ds}{\int_0^t K_{f,w(s)}(x, x_i) ds} \right) \int_0^t K_{f,w(s)}(x, x_i) d$$

$$y(t) = y(0) - \sum_{i=1}^{m} \bar{L}'(y_i^\star, y_i) K_{f,\mathbf{w}}(x, x_i)$$

$$y(t) = \sum_{i=1}^{m} a_i K_{f,\mathbf{w}}(x, x_i) + b$$

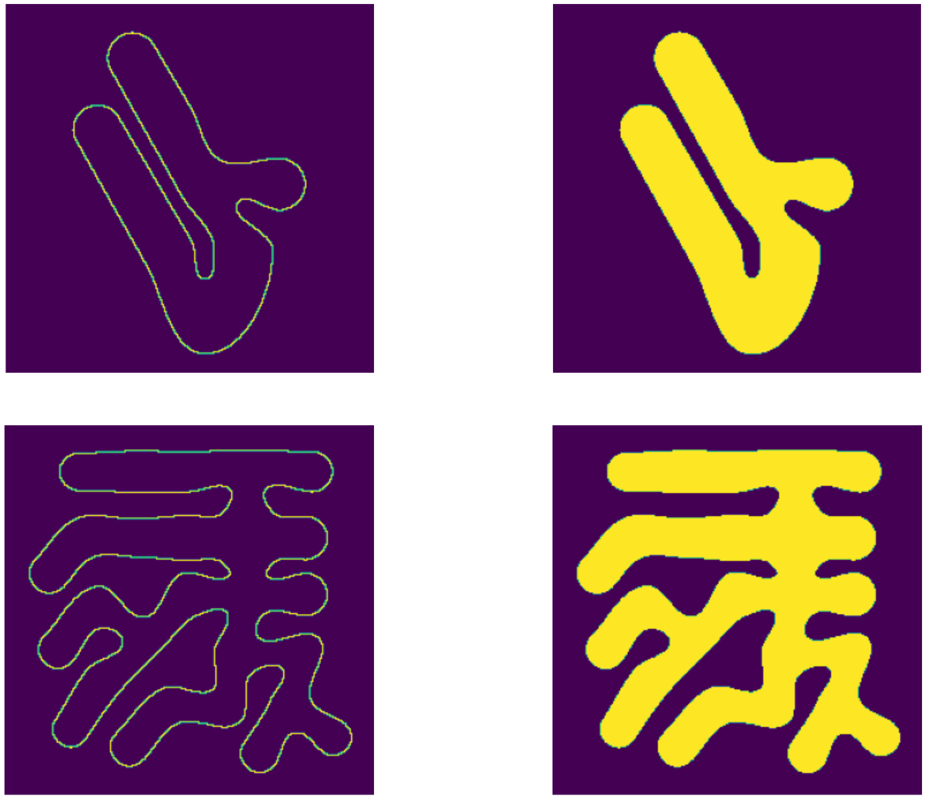
Version 2:

$$y(t) = y(0) - \sum_{i=1}^{m} \int_0^t L'(y_i^\star, y_i(s)) \nabla_w f(w(s), x) \cdot \nabla_w f(w(s), x_i) ds$$

$$y(t) = y(0) - \sum_{i=1}^{m} \int_0^t (y_i(s) - y_i^\star) \nabla_w f(w(s), x) \cdot \nabla_w f(w(s), x_i) ds$$

Another illustration: a bigger training dataset

Alvarez, L., Monzón, N., & M., J. M. (2018). Interactive design of random aesthetic abstract textures by composition principles. *Leonardo*, 1-11, MIT Press

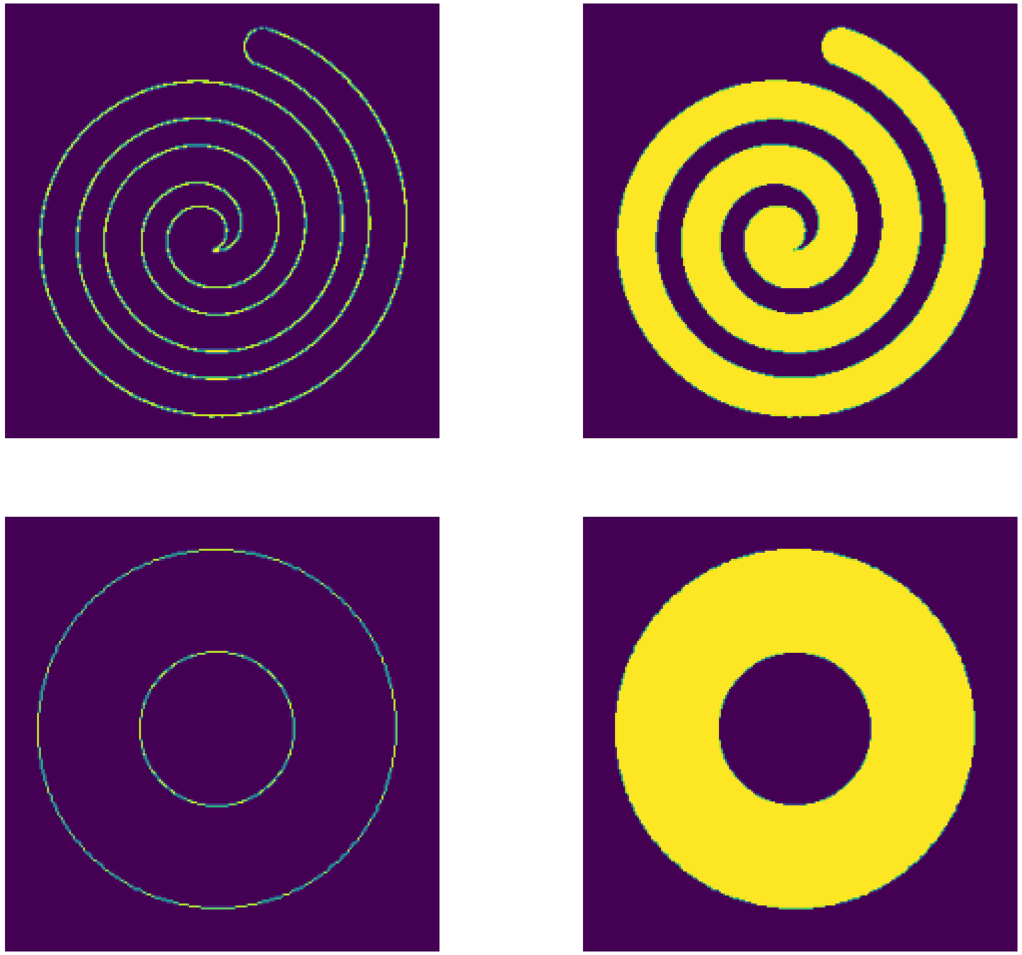


Given the boundary of a shape, the network has to reconstruct the shape i.e classify each pixel of the image as belonging to the shape or not

10,000 training images, data augmentation by random horizontal and vertical flips, 1,000 test images, images of size 256x256.

Another illustration: a bigger training dataset

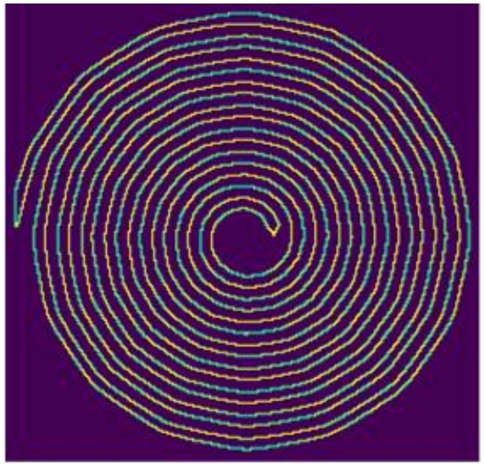
It performs better now

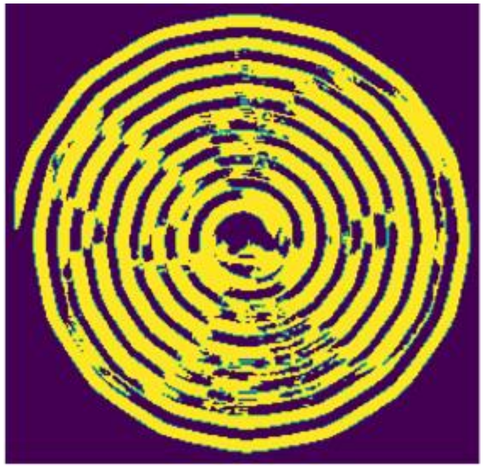


Another illustration: a bigger training dataset

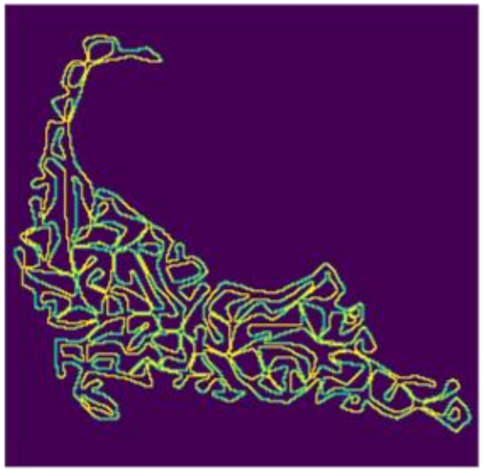
But it's still not perfect!

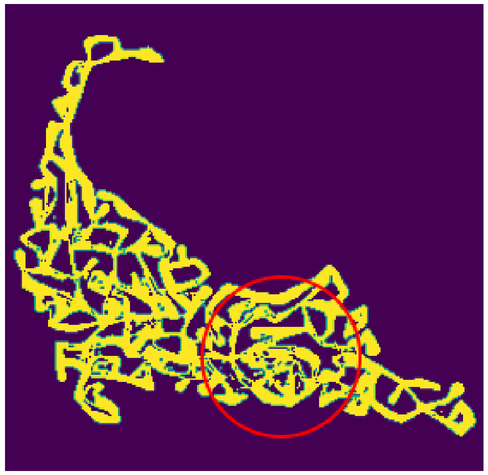
Alvarez, L., Monzón, N., & M., J. M. (2018). Interactive design of random aesthetic abstract textures by composition principles. *Leonardo*, 1-11, MIT Press





IOU: 51%





IOU: 85%

Plan:

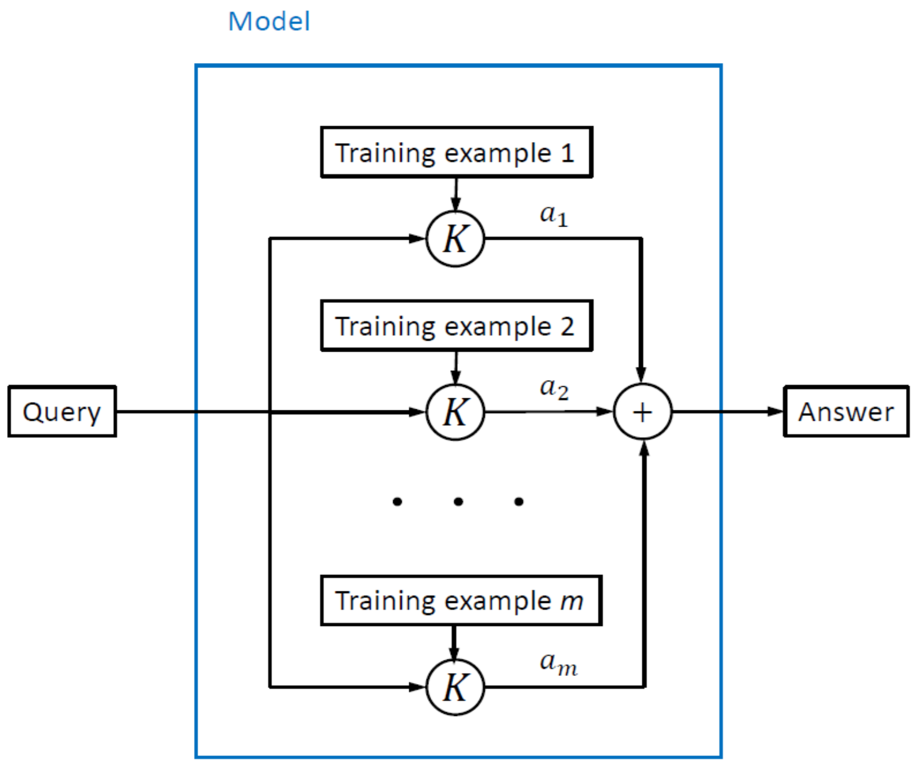
Illustration first : learning the Jordan curve theorem by CNN

Analysis of a theorem by Pedro Domingos (*)

Various reformulations, consequences, discussion

(*) DOMINGOS, Pedro. Every Model Learned by Gradient Descent Is Approximately a Kernel Machine. *arXiv preprint arXiv:2012.00152*, 2020.

An illustrative figure: a deep network as « a superposition of training examples »



Deep network weights as superpositions of training examples. Applying the learned model to a query example is equivalent to simultaneously matching the query with each stored example using the path kernel and outputting a weighted sum of the results.

Some « deep remark » by the author

Theorem 1 *Consider a differentiable learning machine model $y = f(w, x)$. Let w be the parameter learned from a training set $(x_i, y_i^\star)_{i=1,\dots m}$ by gradient descent of a loss function $\mathbf{L}(w) = \sum_i L(y_i^\star, f(w, x_i))$. Then*

$$y = f(w, x) = \sum_{i=1}^{m} a_i K_{f,\mathbf{w}}(x, x_i) + b,$$

where $K_{f,\mathbf{w}}$ is the path kernel associated with $f(w, x)$ and the learning path $\mathbf{w}$ taken during gradient descent, a_i is the average $\frac{\partial L}{\partial y_i}$ along the path weighted by the corresponding tangent kernel, and $b = y(0)$ is the initial model.

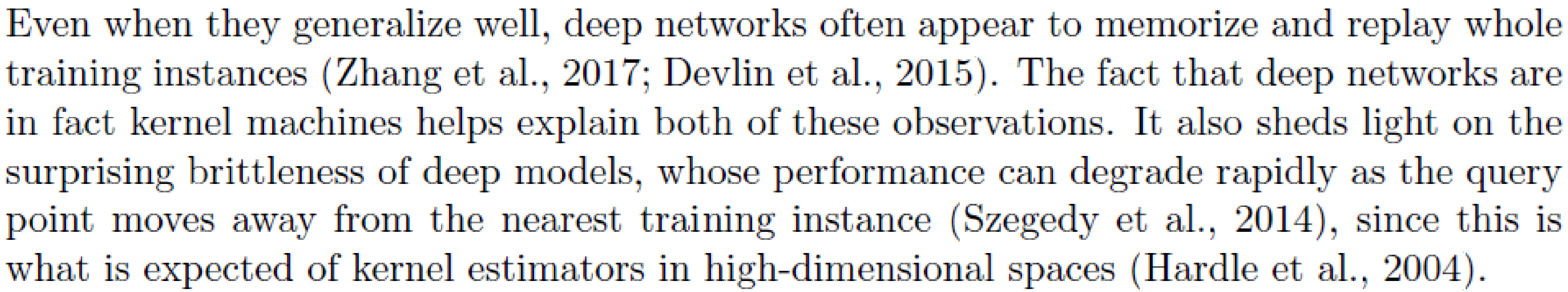
$$y(t) = y(0) - \sum_{i=1}^{m} \left(\frac{\int_0^t L'(y_i^\star, y_i) K_{f,w(s)}(x, x_i) ds}{\int_0^t K_{f,w(s)}(x, x_i) ds} \right) \int_0^t K_{f,w(s)}(x, x_i) ds,$$

Remark 1 *This differs from typical kernel machines in that the a_i's and b depend on x. Nevertheless, the a_i's play a role similar to the example weights in ordinary SVMs and the perceptron algorithm: examples that the loss is more sensitive to during learning have a higher weight. b is simply the prior model, and the final model is thus the sum of the prior model and the model learned by gradient descent, with the query point entering the latter only through kernels. Since Theorem 1 applies to every y_i as a query throughout gradient descent, the training data points also enter the model only through kernels (initial model aside).*

Some « deep remarks » by the author

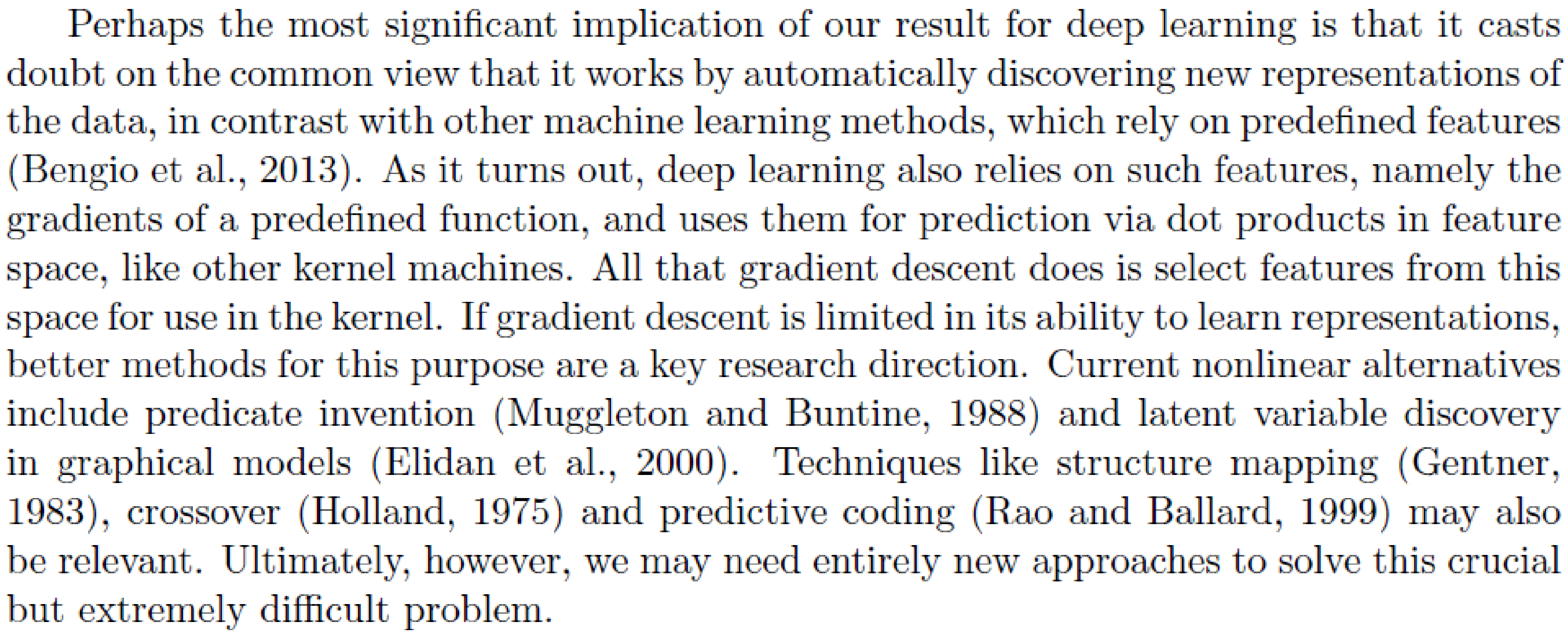
Remark 5 *The proof above is for batch gradient descent, which uses all training data points at each step. To extend it to stochastic gradient descent, which uses a subsample, it suffices to multiply each term in the summation over data points by an indicator function $I_i(t)$ that is 1 if the ith data point is included in the subsample at time t and 0 otherwise. The only change this causes in the result is that the path kernel and average loss derivative for a data point are now stochastic integrals. Based on previous results (Scieur et al., 2017), Theorem 1 or a similar result seems likely to also apply to further variants of gradient descent, but proving this remains an open problem.*

Some « deep remark » by the author



Even when they generalize well, deep networks often appear to memorize and replay whole training instances (Zhang et al., 2017; Devlin et al., 2015). The fact that deep networks are in fact kernel machines helps explain both of these observations. It also sheds light on the surprising brittleness of deep models, whose performance can degrade rapidly as the query point moves away from the nearest training instance (Szegedy et al., 2014), since this is what is expected of kernel estimators in high-dimensional spaces (Hardle et al., 2004).

Some « deep remark » by the author



Perhaps the most significant implication of our result for deep learning is that it casts doubt on the common view that it works by automatically discovering new representations of the data, in contrast with other machine learning methods, which rely on predefined features (Bengio et al., 2013). As it turns out, deep learning also relies on such features, namely the gradients of a predefined function, and uses them for prediction via dot products in feature space, like other kernel machines. All that gradient descent does is select features from this space for use in the kernel. If gradient descent is limited in its ability to learn representations, better methods for this purpose are a key research direction. Current nonlinear alternatives include predicate invention (Muggleton and Buntine, 1988) and latent variable discovery in graphical models (Elidan et al., 2000). Techniques like structure mapping (Gentner, 1983), crossover (Holland, 1975) and predictive coding (Rao and Ballard, 1999) may also be relevant. Ultimately, however, we may need entirely new approaches to solve this crucial but extremely difficult problem.

Antecedents, Tangent neural kernel in the “infinitely wide limit”

Lemma 3.1. *Consider minimizing the squared loss* $\ell(\boldsymbol{\theta})$ *by gradient descent with infinitesimally small learning rate:* $\frac{\mathrm{d}\boldsymbol{\theta}(t)}{\mathrm{d}t} = -\nabla\ell(\boldsymbol{\theta}(t))$. *Let* $\mathbf{u}(t) = (f(\boldsymbol{\theta}(t), \boldsymbol{x}_i))_{i\in[n]} \in \mathbb{R}^n$ *be the network outputs on all* $\boldsymbol{x}_i$*'s at time* t*, and* $\boldsymbol{y} = (y_i)_{i\in[n]}$ *be the desired outputs. Then* $\mathbf{u}(t)$ *follows the following evolution, where* $\boldsymbol{H}(t)$ *is an* $n \times n$ *positive semidefinite matrix whose* (i,j)*-th entry is* $\left\langle \frac{\partial f(\boldsymbol{\theta}(t),\boldsymbol{x}_i)}{\partial\boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}(t),\boldsymbol{x}_j)}{\partial\boldsymbol{\theta}} \right\rangle$*:*

$$\frac{\mathrm{d}\boldsymbol{u}(t)}{\mathrm{d}t} = -\boldsymbol{H}(t)\cdot(\boldsymbol{u}(t)-\boldsymbol{y}). \tag{3}$$

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., & Wang, R. (2019). On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*.

Lemma 3.1. *Consider minimizing the squared loss $\ell(\boldsymbol{\theta})$ by gradient descent with infinitesimally small learning rate: $\frac{\mathrm{d}\boldsymbol{\theta}(t)}{\mathrm{d}t} = -\nabla \ell(\boldsymbol{\theta}(t))$. Let $\boldsymbol{u}(t) = (f(\boldsymbol{\theta}(t), \boldsymbol{x}_i))_{i\in[n]} \in \mathbb{R}^n$ be the network outputs on all $\boldsymbol{x}_i$'s at time t, and $\boldsymbol{y} = (y_i)_{i\in[n]}$ be the desired outputs. Then $\boldsymbol{u}(t)$ follows the following evolution, where $\boldsymbol{H}(t)$ is an $n \times n$ positive semidefinite matrix whose (i,j)-th entry is $\left\langle \frac{\partial f(\boldsymbol{\theta}(t),\boldsymbol{x}_i)}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}(t),\boldsymbol{x}_j)}{\partial \boldsymbol{\theta}} \right\rangle$:*

$$\frac{\mathrm{d}\boldsymbol{u}(t)}{\mathrm{d}t} = -\boldsymbol{H}(t) \cdot (\boldsymbol{u}(t) - \boldsymbol{y}). \tag{3}$$

The statement of Lemma 3.1 involves a matrix $\boldsymbol{H}(t)$. Below we define a deep net architecture whose width is allowed to go to infinity, while fixing the training data as above. In the limit, it can be shown that the matrix $\boldsymbol{H}(t)$ remains *constant* during training i.e., equal to $\boldsymbol{H}(0)$. Moreover, under a random initialization of parameters, the random matrix $\boldsymbol{H}(0)$ converges in probability to a certain deterministic kernel matrix $\boldsymbol{H}^*$ as the width goes to infinity, which is the *Neural Tangent Kernel* $\ker(\cdot,\cdot)$ (Equation (2)) evaluated on the training data. If $\boldsymbol{H}(t) = \boldsymbol{H}^*$ for all t, then Equation (3) becomes

$$\frac{\mathrm{d}\boldsymbol{u}(t)}{\mathrm{d}t} = -\boldsymbol{H}^* \cdot (\boldsymbol{u}(t) - \boldsymbol{y}). \tag{4}$$

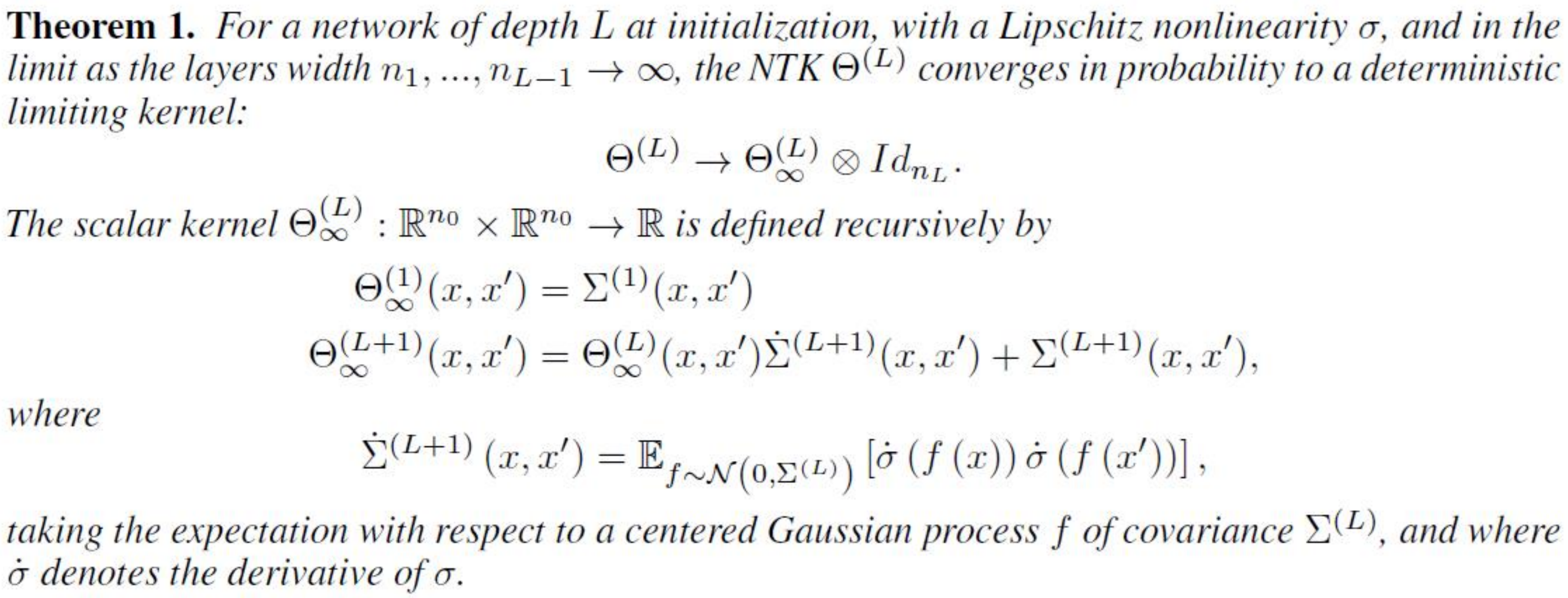
Note that the above dynamics is identical to the dynamics of *kernel regression* under gradient flow, for which at time $t \to \infty$ the final prediction function is (assuming $\boldsymbol{u}(0) = \boldsymbol{0}$)

$$f^*(\boldsymbol{x}) = (\ker(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \ker(\boldsymbol{x}, \boldsymbol{x}_n)) \cdot (\boldsymbol{H}^*)^{-1}\boldsymbol{y}. \tag{5}$$

In Theorem 3.2, we rigorously prove that a fully-trained sufficiently wide ReLU neural network is equivalent to the kernel regression predictor (5) on any given data point.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., & Wang, R. (2019). On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*.

Antecedents, Tangent neural kernel in the "infinitely wide limit"



Theorem 1. *For a network of depth L at initialization, with a Lipschitz nonlinearity σ, and in the limit as the layers width $n_1, ..., n_{L-1} \to \infty$, the NTK $\Theta^{(L)}$ converges in probability to a deterministic limiting kernel:*

$$\Theta^{(L)} \to \Theta_\infty^{(L)} \otimes Id_{n_L}.$$

The scalar kernel $\Theta_\infty^{(L)} : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}$ is defined recursively by

$$\Theta_\infty^{(1)}(x, x') = \Sigma^{(1)}(x, x')$$
$$\Theta_\infty^{(L+1)}(x, x') = \Theta_\infty^{(L)}(x, x')\dot{\Sigma}^{(L+1)}(x, x') + \Sigma^{(L+1)}(x, x'),$$

where

$$\dot{\Sigma}^{(L+1)}(x, x') = \mathbb{E}_{f \sim \mathcal{N}(0, \Sigma^{(L)})}\left[\dot{\sigma}(f(x))\,\dot{\sigma}(f(x'))\right],$$

taking the expectation with respect to a centered Gaussian process f of covariance $\Sigma^{(L)}$, and where $\dot{\sigma}$ denotes the derivative of σ.

Jacot, A., Gabriel, F., & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Adv. Neural Inf. Proc. Sys., 31:8571-8580, 2018.*

Thank you, questions ?