CS 6170: Computational Topology, Spring 2019
Lecture 18

Topological Data Analysis for Data Scientists

Dr. Bei Wang

School of Computing
Scientific Computing and Imaging Institute (SCI)
University of Utah
www.sci.utah.edu/~beiwang
beiwang@sci.utah.edu

March 7, 2019

Persistence Image

Computing persistence image

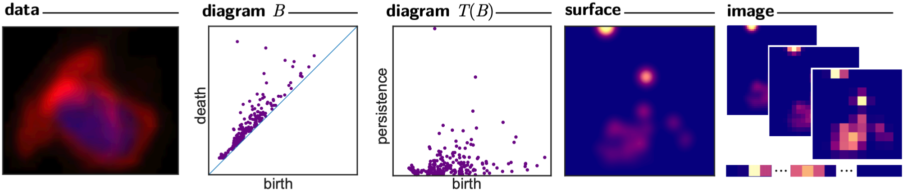


Figure 1: Algorithm pipeline to transform data into a persistence image.

Adams et al. (2017)

Computing persistence image

- Given a normalized symmetric Gaussian with mean $u = (u_x, u_y) \in \mathbb{R}^2$ and variance σ^2:

$$g_u(x, y) = \frac{1}{2\pi\sigma^2} e^{-[(x-u_x)^2+(y-u_y)^2]/2\sigma^2}$$

- Fix a nonnegative weighting function $f : \mathbb{R}^2 \to \mathbb{R}$ that is zero along the horizontal axis, continuous, and piecewise differentiable.
- For a persistence diagram B, the corresponding persistence surface $\rho_B : \mathbb{R}^2 \to \mathbb{R}$ is the function

$$\rho_B(z) = \sum_{u \in T(B)} f(u)\phi_u(z).$$

- Fix a grid in the plane with n boxes (pixels) and assign to each the integral of ρ_B over that region.

Adams et al. (2017)

Classification using persistence image

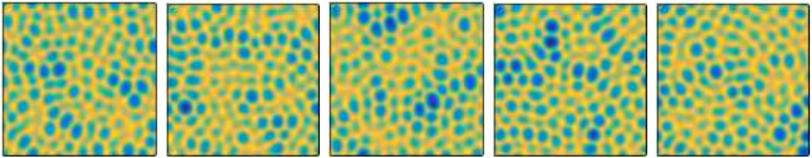


Figure 7: To illustrate the difficulty of our classification task, consider five instances of surfaces $u(x, y, 3)$ for $r = 1.75$ or $r = 2$, plotted on the same color axis. These surfaces are found by numerical integration of Equation (4), starting from random initial conditions. Can you group the images by eye?

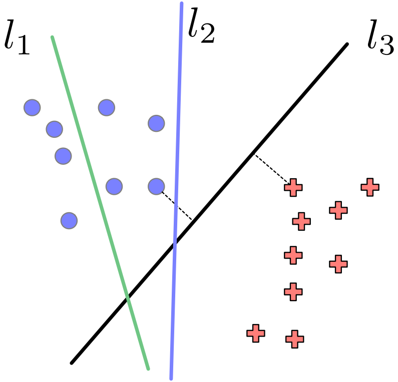
Answer: (from left) $r = 1.75, 2, 1.75, 2, 2$.

Adams et al. (2017)

SVM and Kernel SVM

SVM

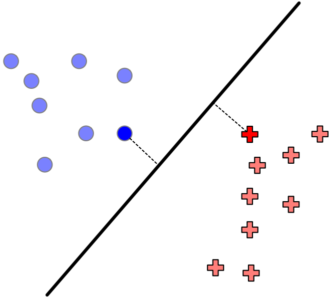
- SVM: Separating the training points with the maximal *margin*
- Margin: distance to the nearest training point of any class
- If margin increases, then generalization error decreases
- Perceptron does not *optimize the separation distance*.



l_1: not a good linear classifier; l_2: small margin; l_3: maximal margin.

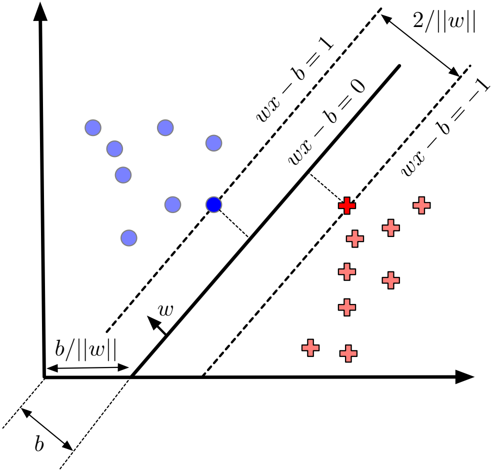
SVM

- Training data: $(x_1, y_1), \cdots, (x_n, y_n)$, where $x_i \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$.
- Goal: Find *maximum margin* hyperplane that separates the training data points with $+1$ and -1 labels.
- *Margin*: distance between the hyperplane and the nearest point.
- *Support vectors*: points on the margin.
- Move a support vector moves the decision boundary.
- Move thee other points/vectors has no effect on the decision boundary.



SVM: margins

- w: a normal vector defining the hyperplane (not necessarily normalized)
- $b/||w||$: offset of hyperplane from the origin along normal vector w.



SVM: hard margin

- Assume the training data is linearly separable
- Constraint: for each x_i
  - Either $wx_i - b \geq 1$ if $y_i = 1$
  - Or $wx_i - b \leq -1$ if $y_i = -1$
- Each training data point must lie on the correct side of the margin
- Rewrite the constraint as

$$y_i(wx_i - b) \geq 1, \forall 1 \leq i \leq n$$

- Problem statement as an optimization: *minimize $||w||$ subject to the above constraint*.
- Equivalently, *maximize the margin $1/||w||$ subject to the above constraint.*
- w^*, b^* that solve the optimization problem determines our classifier: assign each test data point x a label of $\text{sgn}(w^* x - b^*)$.

SVM: soft margin

- Assume the training data is not linearly separable
- Define *Hinge loss* for a training point x_i:

$$c_i = \max(0, 1 - y_i(wx_i - b))$$

- Problem statement: *minimize* the following loss function

$$\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(wx_i - b)) + \lambda||w||^2$$

- λ: parameter that determines the tradeoff between increasing the margin size and ensuring x_i lies on the correct side.
- If λ is sufficiently small, $\lambda||w||^2$ is negligible, similar to the hard margin.

SVM: Primal

- $c_i = \max(0, 1 - y_i(wx_i - b))$
- c_i is the smallest nonnegative number satisfying $y_i(w \cdot x_i - b) \geq 1 - c_i$.
- Optimization problem:

$$\text{minimize} \quad \frac{1}{n}\sum_{i=1}^{n} c_i + \lambda ||w||^2$$

$$\text{subject to} \quad y_i(w \cdot x_i - b) \geq 1 - c_i \ \text{ and } \ c_i \geq 0, \text{ for all } i.$$

SVM: Dual

- Rewrite the optimization problem as a dual maximization problem:

$$\text{maximize} \quad f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j,$$

$$\text{subject to} \quad \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

- This is a quadratic function of the c_i subject to linear constraints, it is efficiently solvable by quadratic programming.
  - $w = \sum_{i=1}^{n} c_i y_i x_i$.
  - Let s_i be a support vector.
  - $y_i(w \cdot s_i - b) = 1 \iff b = w \cdot s_i - y_i$.

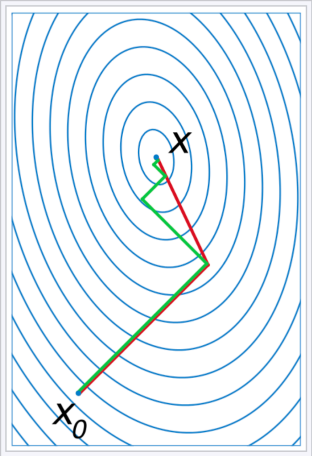
Quadratic Programming

- The quadratic programming problem with n variables and m constraints can be formulated as follows:
  - $\mathbf{c}$: a real-valued, n-dimensional vector
  - Q: an $n \times n$-dimensional real symmetric matrix
  - A: an $m \times n$-dimensional real matrix
  - $\mathbf{b}$: an m-dimensional real vector.
- Find an n-dimensional vector $\mathbf{x}$, that will

$$
\begin{aligned}
\text{minimize} \quad & \frac{1}{2}\mathbf{x}^{\mathrm{T}} Q \mathbf{x} + \mathbf{c}^{\mathrm{T}} \mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}
\end{aligned}
$$

Quadratic Programming

- Commonly used methods: Conjugate gradient, etc.



https://en.wikipedia.org/wiki/Conjugate_gradient_method

From SVM to Kernel SVM

- A subset of the training data points $x_1, \cdots, x_n$ are support vectors, denoted as $s_1, \cdots, s_k$.
- SVM: w can be written as a linear combination of the support vectors:

$$w = \sum_{i=1}^{n} c_i y_i s_i.$$

- Kernel SVM: w is rewritten in the transformed space,

$$w = \sum_{i=1}^{n} c_i y_i \Phi(s_i).$$

- Kernel $K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle = \Phi(x_i) \cdot \Phi(x_j)$

Kernel SVM: Dual

$$\text{maximize} \quad f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i K(x_i, x_j) y_j c_j,$$

$$\text{subject to} \quad \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \le c_i \le \frac{1}{2n\lambda} \text{ for all } i.$$

This is a quadratic function of the c_i subject to linear constraints, it is efficiently solvable by quadratic programming.

- $w = \sum_{i=1}^{n} c_i y_i \Phi(x_i)$.
- Let s_i be a support vector.
- $b = w \cdot \Phi(s_i) - y_i$.

Deep Learning with Topological Features

Deep learning and TDA

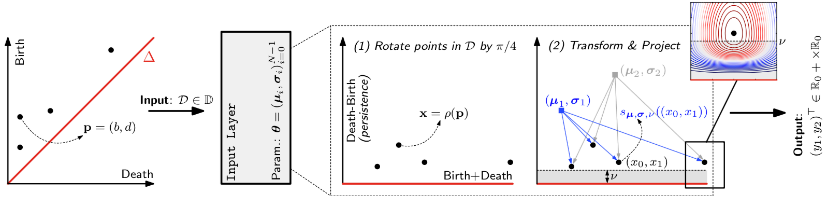


Figure 1: Illustration of the proposed network *input layer* for topological signatures. Each signature, in the form of a persistence diagram $\mathcal{D} \in \mathbb{D}$ (*left*), is projected w.r.t. a collection of *structure elements*. The layer's learnable parameters $\boldsymbol{\theta}$ are the locations $\boldsymbol{\mu}_i$ and the scales $\boldsymbol{\sigma}_i$ of these elements; $\nu \in \mathbb{R}^+$ is set a-priori and meant to discount the impact of points with low persistence (and, in many cases, of low discriminative power). The layer output $\mathbf{y}$ is a concatenation of the projections. In this illustration, $N = 2$ and hence $\mathbf{y} = (y_1, y_2)^\top$.

Hofer et al. (2017)

Main idea: transform persistent diagram via an input layer to be used by a neuron network

Project topological signature

Definition 3. *Let* $\boldsymbol{\mu} = (\mu_0, \mu_1)^\top \in \mathbb{R} \times \mathbb{R}^+, \boldsymbol{\sigma} = (\sigma_0, \sigma_1) \in \mathbb{R}^+ \times \mathbb{R}^+$ *and* $\nu \in \mathbb{R}^+$. *We define*

$$s_{\boldsymbol{\mu},\boldsymbol{\sigma},\nu} : \mathbb{R} \times \mathbb{R}_0^+ \to \mathbb{R}$$

as follows:

$$s_{\boldsymbol{\mu},\boldsymbol{\sigma},\nu}\big((x_0, x_1)\big) = \begin{cases} e^{-\sigma_0^2(x_0-\mu_0)^2 - \sigma_1^2(x_1-\mu_1)^2}, & x_1 \in [\nu, \infty) \\ e^{-\sigma_0^2(x_0-\mu_0)^2 - \sigma_1^2(\ln(\frac{x_1}{\nu})+\nu-\mu_1)^2}, & x_1 \in (0, \nu) \\ 0, & x_1 = 0 \end{cases} \tag{3}$$

A persistence diagram $\mathcal{D}$ *is then projected w.r.t.* $s_{\boldsymbol{\mu},\boldsymbol{\sigma},\nu}$ *via*

$$S_{\boldsymbol{\mu},\boldsymbol{\sigma},\nu} : \mathbb{D} \to \mathbb{R}, \qquad \mathcal{D} \mapsto \sum_{\mathbf{x} \in \mathcal{D}} s_{\boldsymbol{\mu},\boldsymbol{\sigma},\nu}(\rho(\mathbf{x})) \ . \tag{4}$$

Hofer et al. (2017)

w_1-stable input layer

$$\mathrm{w}_p^q(\mathcal{D}, \mathcal{E}) = \inf_{\eta} \left(\sum_{\mathbf{x} \in \mathcal{D}} ||\mathbf{x} - \eta(\mathbf{x})||_q^p \right)^{\frac{1}{p}}$$

Lemma 1. *Let*

$$s : \mathbb{R}_\star^2 \cup \mathbb{R}_\Delta^2 \to \mathbb{R}_0^+$$

have the following properties:

(i) s *is Lipschitz continuous w.r.t.* $\|\cdot\|_q$ *and constant* K_s

(ii) $s(\mathbf{x}) = 0$, *for* $\mathbf{x} \in \mathbb{R}_\Delta^2$

Then, for two persistence diagrams $\mathcal{D}, \mathcal{E} \in \mathbb{D}$, *it holds that*

$$\left| \sum_{x \in \mathcal{D}} s(x) - \sum_{y \in \mathcal{E}} s(y) \right| \leq K_s \cdot \mathrm{w}_1^q(\mathcal{D}, \mathcal{E}) \ . \tag{5}$$

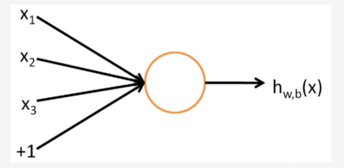
Hofer et al. (2017)

Neural Networks in a Nutshell

Reading Materials

- Neural Network: a type of non-linear classification/regression model.
- The goal of this lecture:
  - Not a complete overview of neural networks or deep learning
  - But rather a high level view of the technique and its connection to TDA
- `http://neuralnetworksanddeeplearning.com/`
- `http://deeplearning.stanford.edu/tutorial/`
- `http://www.deeplearningbook.org/`
- More on class schedule page...

A Single Neuron



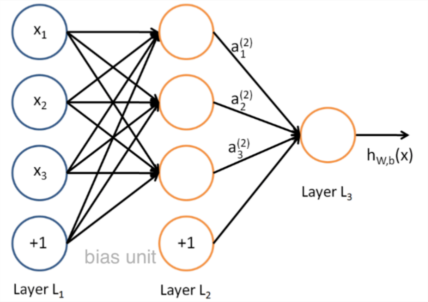
This "neuron" is a computational unit that takes as input x_1, x_2, x_3 (and a +1 intercept term), and outputs $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$, where $f : \Re \mapsto \Re$ is called the **activation function**. In these notes, we will choose $f(\cdot)$ to be the sigmoid function:

$$f(z) = \frac{1}{1 + \exp(-z)}.$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

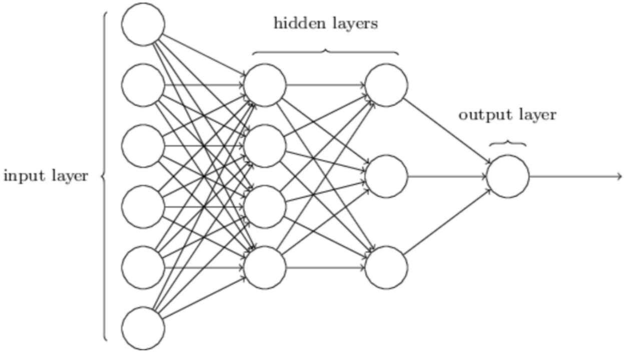
A Neural Network

A neural network is put together by hooking together many of our simple "neurons," so that the output of a neuron can be the input of another. For example, here is a small neural network:



http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

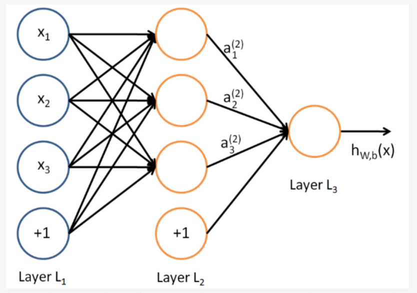
A Neural Network



http://neuralnetworksanddeeplearning.com/chap1.html

Forward propagation

Multiplying input with weights and add bias before applying activation function at each node.



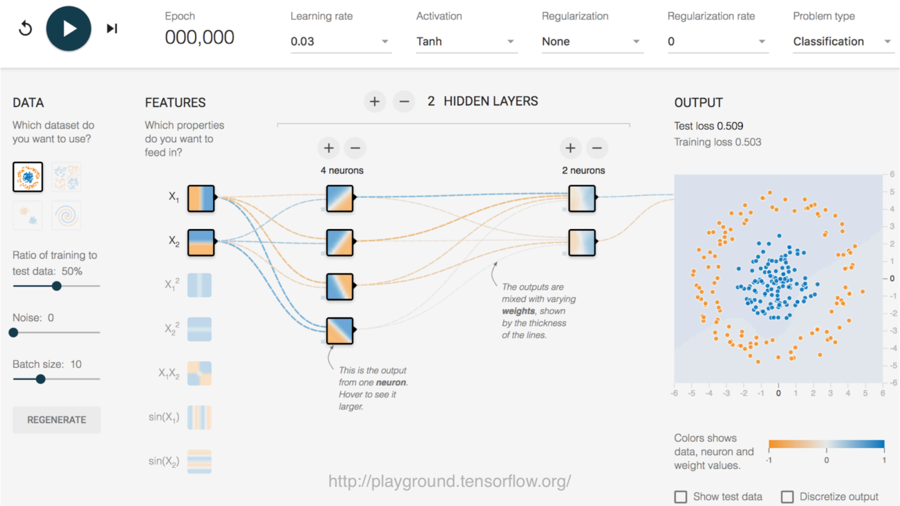
$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) = a_1^{(3)} &= f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

$$
\begin{aligned}
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
h_{W,b}(x) = a^{(3)} &= f(z^{(3)})
\end{aligned}
$$

$$
\begin{aligned}
z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\
a^{(l+1)} &= f(z^{(l+1)})
\end{aligned}
$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Visualizing the inner working of neural networks



http://playground.tensorflow.org/

Topological spaces that are not triangulable

Freedman's E8 Manifold

- Topological manifolds of dimensions 2 and 3 are always triangulable by an essentially unique triangulation (up to piecewise-linear equivalence).
- Some compact 4-manifolds have an infinite number of triangulations, all piecewise-linear inequivalent.
- For dimension greater than 4, there exist manifolds that do not have piecewise-linear triangulations.
- There exist compact manifolds of dimension 5 (and hence of every dimension greater than 5) that are not homeomorphic to a simplicial complex, i.e., that do not admit a triangulation.
- Freedman's E8 manifold (in 4-dimension): it is not triangulable as a simplicial complex.

https://people.math.osu.edu/davis.12/talks/Milwaukee-13short.pdf
https://en.wikipedia.org/wiki/Triangulation_(topology)

References I

Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., and Ziegelmeier, L. (2017). Persistence images: A stable vector representation of persistent homology. *The Journal of Machine Learning Research*, 18(1):218–252.

Hofer, C., Kwitt, R., Niethammer, M., and Uhl, A. (2017). Deep learning with topological signatures. *Neural Information Processing Systems Conference (NIPS)*.