A Novel Approach to Regularising 1NN classifier for Improved Generalization

Aditya Challa, Sravan Danda, Laurent Najman

A Novel Approach to Regularising 1NN classifier for Improved Generalization

Aditya Challa *adityac@goa.bits-pilani.ac.in*
Department of CS&IS
BITS Pilani KK Birla Goa Campus

Sravan Danda *dandas@goa.bits-pilani.ac.in*
Department of CS&IS
BITS Pilani KK Birla Goa Campus

Laurent Najman *laurent.najman@esiee.fr*
Department of Computer Science
Université Gustave Eiffel

Abstract

In this paper, we propose a class of non-parametric classifiers, that learn arbitrary boundaries and generalize well.

Our approach is based on a novel way to regularize 1NN classifiers using a *greedy* approach. We refer to this class of classifiers as *Watershed Classifiers*. 1NN classifiers are known to trivially over-fit but have very large VC dimension, hence do not generalize well. We show that watershed classifiers can find arbitrary boundaries on any dense enough dataset, and, at the same time, have very small VC dimension; hence a watershed classifier leads to good generalization.

Traditional approaches to regularize 1NN classifiers are to consider K nearest neighbours. Neighbourhood component analysis (NCA) proposes a way to learn representations consistent with $(n-1)$ nearest neighbour classifier, where n denotes the size of the dataset. In this article, we propose a loss function which can learn representations consistent with watershed classifiers, and show that it outperforms the NCA baseline.

1 Introduction

Deep learning classifiers have obtained state-of-the-art results on the classification problem, but they predominantly use a parametric linear classifier as the classification layer. At the opposite end of the spectrum of classifiers to the linear classifiers lies non-parametric classifiers, such as K-Nearest-Neighbours Hastie et al. (2009). To our knowledge, non-parametric classifiers and their counterpart loss functions have not been used recently for classification.

However, non-parametric class of classifiers form an aspect of ML systems with a lot of flexibility to learn arbitrary boundaries. It is known that 1NN classifiers trivially overfit the data. Traditionally, one uses K Nearest Neighbors to regularize this. Goldberger et al. (2004) proposes a loss function – Neighbourhood Component Analysis (NCA), to learn KNN classifiers. *Is there a better way to regularize 1NN classifiers?*

Contributions: (i) In this article, we propose a novel way to regularize 1NN classifiers using a greedy approach. This class of classifiers are referred to as *Watershed Classifiers*. (ii) Interestingly, the only hyperparameter for this class is `N_SEEDS` which directly controls the VC dimension (see section 3). Moreover, even with a small VC dimension the watershed classifier can learn arbitrarily complex boundaries, assuming that the data is dense enough. This ensures a very small generalization gap theoretically. (iii) To suitably learn the embedding, we also propose a novel loss function. Although the loss function is highly non-convex,

SGD approaches worked surprisingly well, as we show in section 5. (iv) As delineated in section 5, the proposed classifier surpasses NCA in performance. To the best of our knowledge, this represents the first occurrence in which a non-parametric classifier matches or surpasses the accuracy of a linear classifier.

2 Related Works

NCA and Metric Learning: As stated in Goldberger et al. (2004), NCA learns the representations consistent with KNN classifiers. However, the approach does not scale well to large datasets. In Wu et al. (2018) the authors make suitable modifications to scale this approach to large datasets. Nevertheless, NCA based approaches usually do not lead to state-of-the-art results when compared to linear classifiers. Hence, to our knowledge, NCA based approaches have not been widely used in the context of classification in favour of linear classifiers. However, these have been widely used for *metric learning*. Teh et al. (2020) proposes `ProxyNCA++` which uses proxy points in the embedding space, based on Movshovitz-Attias et al. (2017). In this article, our focus is mainly on the classification aspect and do not consider the metric learning as an objective.

Relation with Watersheds: Watersheds have been widely used for image classification and related tasks before the deep learning approaches. To our knowledge, the authors in Falcao et al. (1999); Cousty et al. (2010; 2009) are the first to propose watersheds on edge-weighted graphs. The authors in Challa et al. (2019) propose to use watersheds for classification, but do not learn the representations. In Challa et al. (2022), the authors try to learn representations consistent with watershed using Triplet loss as a proxy loss function. As the authors state, their method depends heavily on the graph. In Turaga et al. (2009); Wolf et al. (2017) the authors propose novel loss functions to learn watershed segmentation for images. In Sanmartin et al. (2019; 2021) the authors propose (directed) probabilistic watersheds which assumes an underlying KNN graph. All these approaches work for segmenting images, but do not fare well for other tasks. In contrast, we provide a complexity analysis (section 3), propose a more consistent loss function (section 4) to learn the representations for multi-class classification. Our approach implicitly learns the graph, and works for other tasks than image segmentation.

Ultrametric Learning: Closely related to learning watersheds is the problem of learning ultrametrics. Learning ultrametrics is related to unsupervised clustering of data, and there exists a large body of work Ackerman & Ben-David (2016); Dasgupta (2016); Yarkony & Fowlkes (2015) exploring the links. In Chierchia & Perret (2019) the authors assume an underlying weighted graph and propose a cost function to learn the closest ultrametric. Most of the related approaches assume that the underlying graph is known. In contrast, the loss function proposed here implicitly learns a graph, specifically a minimum spanning tree, for the purposes of classification.

3 Watershed Classifier (a.k.a. greedy 1-NN classifier)

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y(\boldsymbol{x}_i)\}$ denote the dataset where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y(\boldsymbol{x}_i)$ denote the corresponding labels. We assume that $\mathcal{D}$ consists of both labelled and unlabelled samples. So, we assume that $y(\boldsymbol{x}) \in \{0, 1, \cdots, K\}$ (classes) for labelled samples and $y(\boldsymbol{x}) = -1$ for unlabelled samples. **Remark:** For developing the theory, we assume $y(\boldsymbol{x}) \in \{0, 1\}$ (binary classification) for simplicity. In practice, we extend this to multi-class using one-vs-rest approach.

We implicitly assume a *complete* graph on $\mathcal{D}$ – $\mathcal{G}_{\mathcal{D}} = (V, E, W)$, where (i) each vertex corresponds to the data point, $V = \{\boldsymbol{x}_i\}$ (ii) $E = V \times V$ and (iii) $W : E \to \mathbb{R}^+$ denotes the edge-weights which are restricted to be positive and denote the dissimilarity between the points. Unless otherwise specified, we use $W(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ as the measure of dissimilarity.

The aim of the classifier is to assign labels to all the unlabelled points – $\{\boldsymbol{x} \mid y(\boldsymbol{x}) = -1\}$. Given any arbitrary labelling on the entire dataset – $\tilde{y}(\boldsymbol{x}) \in \{0, 1, \cdots K\}$, define MARGIN as follows:

$$\text{MARGIN} = \min_{\tilde{y}(\boldsymbol{x}_i) \neq \tilde{y}(\boldsymbol{x}_j)} W(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{1}$$

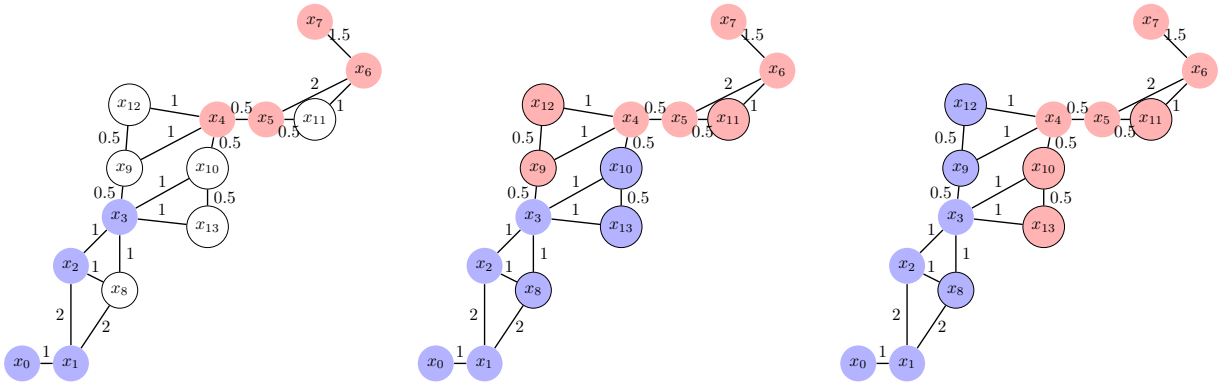


Figure 1: Illustrating the labelling preference using watershed classifier. (a) illustrates an arbitrary set of data points. Blue dots indicate class 0, Red dots indicate class 1. The unlabelled points have no colour. Few selected edges with corresponding edge-weights are included. (b) and (c) indicates two different labelling. Observe that MARGIN of (b) is 0.5, while margin of (c) is 1. Hence (*c*) is considered a better labelling than (b) as per the MAXIMUM MARGIN PRINCIPLE.

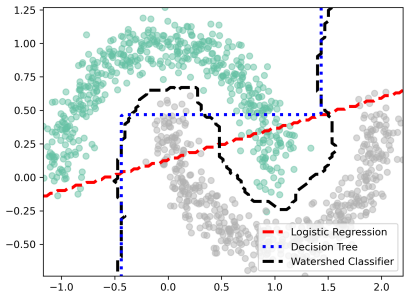


Figure 2: Comparison of Watershed Classifier with Linear Classifier and Decision Tree. Considering a simple toy example, and a fixed VC-dimension of 3, observe that watershed classifier can find the right boundary.

that is, the edge-weight between the "closest" pair of points with different labels.

The MAXIMUM MARGIN PRINCIPLE then states –

> The right labelling for the unlabelled points is obtained from the labelling with the largest MARGIN.

fig. 1 illustrates the preference of the labelling with respect to the MAXIMUM MARGIN PRINCIPLE. The above definitions are straight-forward extensions to general edge-weighted graphs of the margin and maximum margin principle defined for support vector machines Schölkopf & Smola (2002).

The above classification rule is the same as greedy-1NN-classification: There exists a simple rule to obtain the maximum margin labelling – *From the set of unlabelled points, identify the point which has the smallest edge-weight to the labelled set and label it accordingly. And repeat this step until all points are labelled.* This is the same as performing *greedy-1NN-classification*. **Remark**: Ties are broken arbitrarily.

In the example of fig. 1a – (i) We first label $\boldsymbol{x}_9$ as blue, (ii) label $\boldsymbol{x}_{10}$ as red (iii) We first label $\boldsymbol{x}_{12}$ as blue, (iv) label $\boldsymbol{x}_{13}$ as red. This gives the maximum margin labelling.

Table 1: Comparison of Watershed Classifiers with Linear Classifier and Decision Trees

Property	Watershed Classifier	Linear Classifier	Decision Trees
Complexity	`N_SEEDS`	Number of features	Number of Splits
Non-Linear Boundaries	Yes	No	Axis-Aligned-Steps
Computation	$\mathcal{O}(n \log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Training	Building a space partitioning data structure.	Fitting the parameters of the hyperplane.	Identify the best splits for each feature.
Inference for 1 sample	1-Nearest-Neighbor	Classification rule	Parsing the decision tree
Inference for n samples	Minimum Spanning Tree	Classification rule	Parsing the decision tree
Inference Depends on other samples	Yes	No	No

Notation: Observe that the data is assumed to contain both labelled and unlabelled samples. To differentiate between these, we refer to the labelled samples as `SEEDS`. Also, we implicitly assume that there exists *at least one* labelled sample per class in the set of `SEEDS`. Let `N_SEEDS` denote the number of labelled samples in each class. **Remark:** Although it is not necessary that the number of seeds be the same for all the classes, we assume so for the sake of simplicity.

3.1 Complexity Analysis of Watershed Classifier:

Note that watershed classifier has *no trainable parameters* and has only one hyperparameter `N_SEEDS`. In this section, we analyse the complexity of the watershed classifier.

Recall that VC-Dimension is defined as the size of the largest set of which the algorithm can shatter Shalev-Shwartz & Ben-David (2014). We then have the following result –

Theorem 1 *Let $\{\boldsymbol{x}_i\}$ denotes the set of data points and $\mathcal{G}_{\mathcal{D}} = (V, E, W)$ denotes the complete graph. Assume that $W(\boldsymbol{x}_i, \boldsymbol{x}_j) \neq W(\boldsymbol{x}_k, \boldsymbol{x}_l)$ for all $i \neq k$ or $j \neq l$. That is, all edge weights are assumed to be distinct. Let the number of classes be 2 (binary classification). Then, VC-dimension of the watershed classifier with `N_SEEDS` is $2 \times$ `N_SEEDS`.*

Proof 1 *The proof follows from the fact that – (a) if all edge-weights are distinct, then the 1-Nearest Neighbour is unique for all points and (b) Once the seeds are fixed, then the labelling is unique.*

Let k denote the number of points such that all possible configurations in $\{0, 1\}^k$ can be obtained using the watershed classifier. Observe that $k \leq 2 \times$ `N_SEEDS` since, when the $2 \times$ `N_SEEDS` are fixed, the labels of the other data points are fixed and unique. And it is easy to see that all possible configurations can be obtained when $k = 2 \times$ `N_SEEDS`. Hence, the VC dimension is $2 \times$ `N_SEEDS`.

Although the above result is relatively simple, there are a few significant implications - (i) The only hyperparameter `N_SEEDS` directly control the VC dimension, and hence generalization gap. (ii) If the features are dense enough, then by simply taking `N_SEEDS` $= 1$ can give very good generalization bounds. Apart from this the watershed classifier also exhibits several properties which are different from any of the existing classifiers.

Comparison with existing classifiers: table 1 shows the summary of the comparison of watershed classifiers with linear classifiers and decision trees. Several properties of the watershed classifier are inherited from the nearest-neighbour classifier. Training involves building a space partitioning data structure for efficient similarity search Dong et al. (2011); Johnson et al. (2021). Inference for a single sample is based on a simple 1-nearest-neighbor search.

Watershed classifier differs from nearest-neighbours for inference on several samples. Note that Linear Classifiers, Decision Trees and KNN classifiers do not distinguish between a single sample vs multiple sample inference. In case of watershed classifier, one should classify the *closest points* before classifying other points. This is similar to the active learning framework Xie et al. (2021) except that 1NN rule provides an efficient way and removes the high computational training of the models.

Regarding complexity, it is known that nearest neighbour approaches are prone to overfitting, with 1NN classifier which overfits in a single shot. Traditional approaches use KNN to reduce the complexity. Watershed classifier, unconventionally, uses a greedy approach to reduce the overfitting. This approach to reducing complexity is probably the key reason why watershed out-performs linear classifiers when working with neural networks (See section 5 for details).

Linear classifiers have a complexity which scales as $\mathcal{O}(d)$, where d is the number of features. However, the complexity of Watershed classifier does not depend on the embedding dimension. Decision stumps (Decision Trees with a single split) only search for axis aligned splits [1]. Interestingly, the VC-dimension of a decision stump is 2, which is equal to the VC-dimension of the Watershed classifier with `N_SEEDS` $= 1$. However, Watershed classifier is capable of having non-linear boundaries.

Regarding computational requirement, watershed classifier requires[2] $\mathcal{O}(n \log(n))$ to classify n points. However, both linear classifier and Decision Trees have a lower complexity ($\mathcal{O}(n)$) comparatively.

A property which makes watershed classifier unique compared to the rest, is the fact that the labelling actually depends on *density of the samples*. In presence of very high noise in the features, Watershed classifier does not work well. In comparison, both linear models and decision trees are capable of handling noisy inputs comparatively well. However, this issue is easily rectified when using watershed classifier along with neural nets (see section 4).

fig. 2 illustrates the difference between Watershed classifier with Linear and Decision Tree Classifiers. Even with the small VC-dimension of 3, we have that watershed classifier can identify the non-linear boundary. (See appendix B for more details about the visualization of the boundaries).

4 Learning Neural Networks to get Watershed Representations

Recall that the Watershed Classifier in section 3 does not have any trainable parameters and hence is not powerful enough to handle highly noisy datasets. To rectify this issue, in this section, we provide a way to combine the Watershed classifier with neural nets. Formally, we wish to solve

$$\arg\max_{\theta} \text{Acc}(\text{Watershed Classifier}(f_\theta(X)) \tag{4}$$

Here f_θ denotes the neural network architecture which learns the representations, $X = \{\boldsymbol{x}_i\}$ denotes the input data-points. In words – we wish to learn the representations of the data such that when watershed is applied we get as large accuracy as possible.

Watershed Loss Function: We propose a corresponding loss function described in algorithm 1 to train the network f_θ. Computing the loss function consists of three stages – (i) Propagating the labels (ii) Selecting the "closest" correctly labelled sample from each class (iii) Computing the cross-entropy loss with respect to the closest labels.

(i) Propagating watershed labels: Let $X = \{\boldsymbol{x}_i\}$ denote the input data-points, and $f_\theta(X)$ denote the representations of the data-points. First select a *random* subset of points of size `N_SEEDS` from each class. Using this subset of points as seeds, propagate the labels using greedy 1-Nearest-Neighbor approach.

[1] While it is possible to search for non-axis aligned splits Heath et al. (1993), most of the current state-of-art approaches rely on axis-aligned splits

[2] Greedy 1NN search is similar in complexity to constructing a Euclidean minimum spanning tree. Arya & Mount (2016) shows that $\epsilon-$approximate EMST for n points can be found in $\mathcal{O}(n \log(n) + (\epsilon^{-2} \log^2(1/\epsilon))n$ time.

Algorithm 1 Computing Watershed Loss

1: **Input:** Representations X__REP $= \{f_\theta(\boldsymbol{x}_i)\}$, Labels $\{y_i\}$, Hyper parameter `N_SEEDS`
2: For each class, select `N_SEEDS` points randomly from X__REP. We refer to this set as X__SEEDS.
3: Propagate the labels from the X__SEEDS to the rest of data points.
4: Let X__CORRECT denote the set of points which have the correct labels. (**Remark:** Note that the propagated labels may/may-not match with the input labels y_i. Clearly, X__SEEDS $\subseteq$ X__CORRECT.)
5: Further, let X__CORRECT__L denote the subset of correctly labelled point with label L. **Remark:** Thus we have,

$$\text{X\_\_CORRECT} = \bigcup_{\text{L}} \text{X\_\_CORRECT\_\_L} \tag{2}$$

6: Let 1NN($\boldsymbol{x}_i$, X__CORRECT__L) denote the 1 nearest neighbor of the point x_i in X__CORRECT__L. For each data point $\boldsymbol{x}_i$ compute the probabilities and the corresponding loss as -

$$\begin{aligned} \boldsymbol{x}_{i,1nn,L} &= \text{1NN}(f_\theta(\boldsymbol{x}_i), \text{X\_\_CORRECT\_\_L} \setminus \{f_\theta(\boldsymbol{x}_i)\}) \\ p_{i,L} &\propto \exp\left(-||f_\theta(\boldsymbol{x}_i) - f_\theta(\boldsymbol{x}_{i,1nn,L})||\right) \\ \text{Loss}(\boldsymbol{x}_i) &= \sum_{k=1}^{K} I[y_i = K] \log(p_{i,K}) \end{aligned} \tag{3}$$

where $I[.]$ denotes an indicator function.
7: **return** sum of losses for all data points.

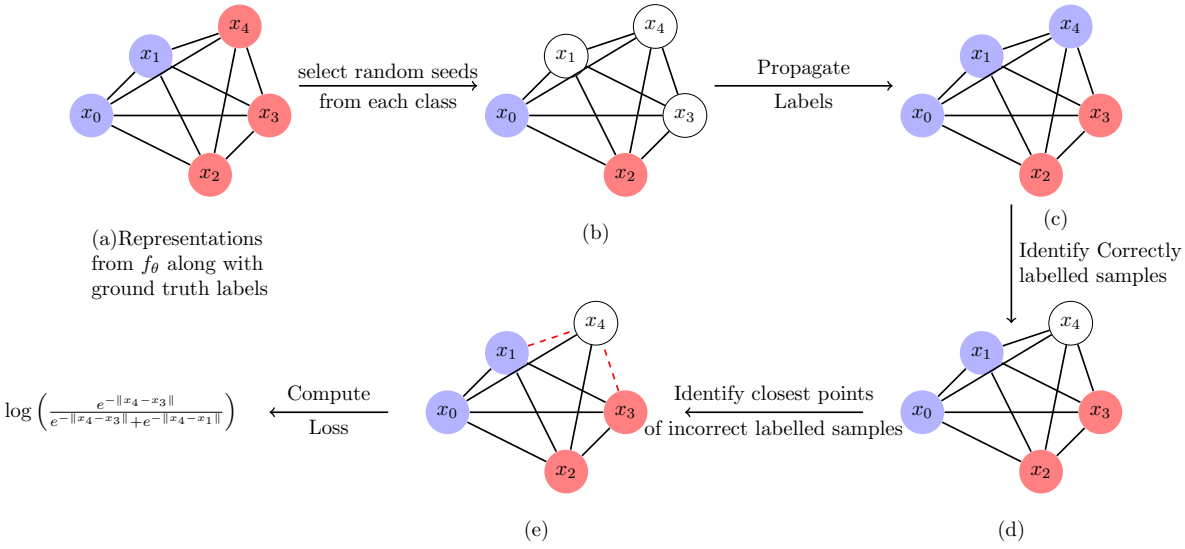


Figure 3: Illustrating the computation of watershed loss. (a) shows the toy representations obtained from f_θ along with the ground-truth labels. (b) shows the seeds selected from each class. We assume `N_SEEDS` $= 1$ in this case. (c) illustrates the propagation of the labels. Note that x_4 has a ground-truth label of red, but is labelled blue with label propagation. All the other labels match the ground-truth. (d) identifies the correctly labelled samples. (e) identifies closest samples which are correctly labelled. In this case, for x_4, the closest blue sample is x_1 and the closest red sample is x_3. Note that, one should compute this for all samples. We only choose x_4 here for illustration. Finally, we compute the loss for each sample using eq. (6) and eq. (7).

(ii) Selecting the closest correctly labelled sample from each class: Note that the propagated labels may or may not match the true labels. Consider only the subset of points which are correctly labelled – X__CORRECT. Also consider the subset of points which are correctly labelled and has a class L –

X_CORRECT_L. For each data point, we select the closest point as follows:

$$\boldsymbol{x}_{i,1nn,L} = \text{1NN}(f_\theta(\boldsymbol{x}_i), \text{X\_CORRECT\_L} \setminus \{f_\theta(\boldsymbol{x}_i)\}) \tag{5}$$

Here 1NN denote the computation of the 1-nearest-neighbor in the set X_CORRECT_L excluding itself ($f_\theta(\boldsymbol{x}_i)$).

(iii) Computing the loss: Finally, the loss is computed akin to the cross entropy loss. We construct the probabilities from the Euclidean distances as

$$p_{i,L} = \frac{\exp\left(-||f_\theta(\boldsymbol{x}_i) - f_\theta(\boldsymbol{x}_{i,1nn,L})||\right)}{\sum_{k=1}^{K} \exp\left(-||f_\theta(\boldsymbol{x}_i) - f_\theta(\boldsymbol{x}_{i,1nn,k})||\right)} \tag{6}$$

and then construct the cross-entropy loss based on these probabilities:

$$\text{Loss}(\boldsymbol{x}_i) = \sum_{k=1}^{K} I[y_i = k] \log(p_{i,k}) \tag{7}$$

where K denotes the number of classes.

A simple illustration of the computation of the loss function is given in fig. 3.

Intuition behind the loss function: Why does the above loss function learn representations consistent with watershed classifier? – Intuitively, at each stage of optimizing the loss function we – (i) make sure that the correct propagation is preserved and (ii) wrong propagation of labels is corrected. This ensures that the labels are correctly propagated. So, the correct propagation from the seeds can be considered an invariant which only improves with learning. The experiment in appendix A verifies this aspect of the loss function.

4.1 Training f_θ using watershed loss

Watershed loss is Non-Convex: A generic rule of thumb when designing a loss function for neural networks is – replacing the neural-net with a simple matrix should result in a convex loss function. This holds true for almost all known widely used loss functions. However, the watershed loss function which is proposed here does not have this property. Interestingly, we empirically find that stochastic gradient descent (SGD) performs surprisingly well on this loss function.

Does not distribute over samples: Almost all the loss functions – $\ell(.,.)$ distribute over the samples, i.e., they can be written as – $\sum_i \ell(f_\theta(\boldsymbol{x}_i), y_i)$. This does not hold true for watershed loss. Hence, the batch size B plays a crucial role in training the network f_θ. In practice – we first draw N random samples from the dataset, each with size B – $\{X_{b_i} = \{\boldsymbol{x}_{b_i,j}\}_{j=1}^{B}, y_{b_i} = \{y_{b_i,j})\}_{j=1}^{B}\}$ and train f_θ using the loss $L(\theta) = (1/N)\sum_{i=1}^{N} \text{Watershed Loss}(f_\theta(X_{b_i}), y_{b_i})$ where the Watershed Loss is obtained using algorithm 1 and $f_\theta(X_{b_i}) = \{f_\theta(\boldsymbol{x}) \mid \boldsymbol{x} \in X_{b_i}\}$.

Evaluating f_θ trained using watershed loss To be consistent with the training procedure, we evaluate the network trained by – (i) First select N random samples of size B each – $\{X_{b_i} = \{\boldsymbol{x}_{b_i,j}\}_{j=1}^{B}, y_{b_i} = \{y_{b_i,j})\}_{j=1}^{B}\}$ (ii) For each test sample $\boldsymbol{x}_i^+$ we perform 1NN classifier on each of the sample B and predict the label of $\boldsymbol{x}_i^+$. The final label is taken to be the *most frequent label* in the predictions for each of the samples.

5 Experiments and Analysis

The watershed classifier is unique in comparison to other classifiers in two major ways – (i) Its representational capacity is close to 1NN, i.e., it can learn arbitrarily complex boundaries and (ii) The generalization is however much better than 1NN, and is in-fact comparable to the linear classifiers. In this section, we perform simple experiments to verify these properties. **Remark:** The code to generate these results is provided in the supplementary material.

Table 2: Watershed Classifier vs NCA vs Linear Classifier on FashionMNIST. f_θ is taken to be the linear embedding with `EMBED_DIM` = 4 or 16.

		Watershed Classifier						NCA	Linear Classifier
	N SEEDS	1	5	10	20	40	100		
EMBED-DIM	BATCH-SIZE								
4	1020	0.7802	0.8211	0.8272	0.8296	0.8320	–		
	2040	0.7960	0.8265	0.8288	0.8300	0.8294	0.8307	0.4375	0.8120
	4090	0.8078	0.8252	0.8256	0.8304	0.8296	0.8305		
16	1020	0.8509	0.8716	0.8739	0.8769	0.8792	–		
	2040	0.8533	0.8743	0.8768	0.8792	0.8838	0.8809	0.6969	0.8600
	4090	0.8474	0.8736	0.8740	0.8807	0.8812	0.8815		

Table 3: Watershed Classifier vs NCA vs Linear Classifier on CIFAR10,CIFAR100, FashionMNIST. f_θ is taken to be three different types of architectures – M3LC, MR18, M3FF. Observe that while on FashionMNIST all approaches give similar results, Watershed classifier outperforms NCA and works comparably to Linear classifier on CIFAR10/100 datasets.

DATASET	MODEL	Width	Watershed Classifier	NCA	Linear Classifier
FashionMNIST	M3LC	256	0.9229	0.9164	0.9218
	M3LC	64	0.9120	0.9030	0.9089
	M3FF	256	0.8918	0.8725	0.8799
	M3FF	64	0.8801	0.8737	0.8760
	MR18		0.9356	0.9217	0.9246
CIFAR10	M3LC	256	0.8616	0.8040	0.824
	M3LC	64	0.8287	0.7359	0.789
	M3FF	256	0.6268	0.5585	0.6128
	M3FF	64	0.5491	0.5012	0.5439
	MR18		0.9191	0.8512	0.870
CIFAR100	M3LC	256	0.5788	0.3199	0.5478
	M3LC	64	0.4899	0.2602	0.4702
	M3FF	256	0.3208	0.1841	0.3290
	M3FF	64	0.2516	0.1603	0.2517
	MR18		0.6989	0.3494	0.6066

5.1 Verifying Representational Capacity of Watershed Classifier

Experiment Design: Consider the popular Spiral Dataset where we have a parameter `N_REV` which dictates how many times the spiral revolves around the origin. Hence, larger the `N_REV`, larger the complexity of the boundary. Assume that the embedding network f_θ is a matrix of size $2 \times$ `EMBED_DIM`. We then investigate – What is the minimum width `EMBED_DIM` required to get 100% training accuracy? We expect that if the classifier has large representational capacity, then `EMBED_DIM` would be small. Otherwise, it would be large. As baselines, we consider the NCA loss Goldberger et al. (2004) and the standard linear classifier with the same f_θ.

We measure the best loss one can achieve with different values of `EMBED_DIM`. The loss for watershed is the one computed in algorithm 1, loss for NCA is computed using the loss proposed in Goldberger et al. (2004),

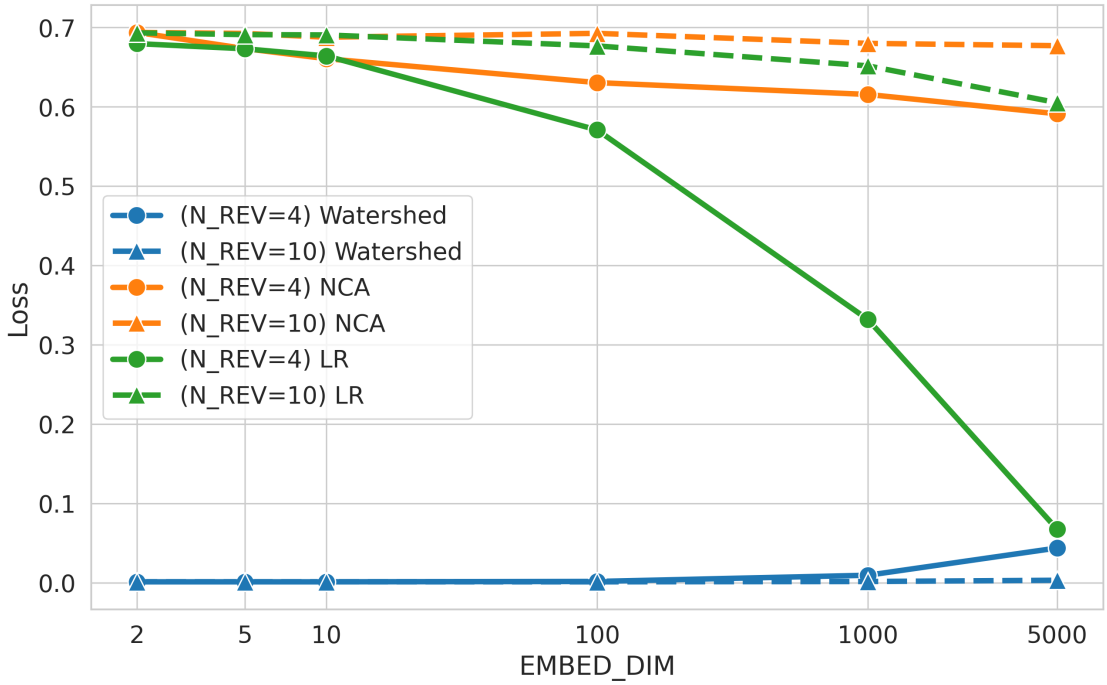


Figure 4: Representation Capacity of Watershed Classifier.

and the loss for linear classifiers is computed using the *binary cross-entropy* loss. We consider two values of `N_REV` – 4 and 10.

Observations: The results are plotted in fig. 4. Illustrations can be found in appendix C. Firstly, as it is intuitive, with large enough `EMBED_DIM`, one can get the loss to arbitrary small. However, for `N_REV` $= 4$ NCA and Linear classifiers require a high width. But watershed classifier can obtain negligible training loss even with `EMBED_DIM` $= 2$. For `N_REV` $= 10$, we could not reach 0 loss, even with `EMBED_DIM` $= 5000$ for both NCA and Linear classifiers. But, watershed is able to fit the data with `EMBED_DIM` $= 2$.

This shows that there is a very high representational capacity for watershed classifier. Indeed, it is known that 1NN classifiers can trivially overfit the dataset, and watershed classifier inherits this property from 1NN classifier. However, the only caveat is that – We assume that the density of the points is high, as is the case with the Spiral dataset considered here.

5.2 Generalization for Linear Embedding

Experiment Design: To investigate the generalization performance of the watershed classifier, we consider the case where f_θ is a linear embedding, and use FashionMNIST dataset. We consider two values of `EMBED_DIM` – 4 and 16. As baselines, we compare the performance with NCA and linear classifiers. table 2 shows the results obtained.

Observation 1 – Dependence on `N_SEEDS`: Recall that `N_SEEDS` controls the complexity of the watershed classifier. Hence, for very small `EMBED_DIM`, having a high complex classifier is expected to increase the accuracy. This is observed in table 2 – where, with `EMBED_DIM` $= 4$, we obtain a score of ≈ 0.79 with `N_SEEDS` $= 1$, and increases to ≈ 0.83 with `N_SEEDS` $= 100$. For comparison, the linear classifier obtains a score of 0.81 with the same `EMBED_DIM`.

Observation 2 – Dependence on `BATCH_SIZE`: Watershed classifier expects that the representations are dense to work effectively. When training large values of `BATCH_SIZE`, this can be achieved efficiently, and hence we see a slight increase in the score when considering `EMBED_DIM` $= 4$ and increasing the `BATCH_SIZE` from 1020 to 4090. However, for a higher `EMBED_DIM` $= 16$, there is not much effect, since dense manifolds can be learned much more effectively with larger `EMBED_DIM`.

Observation 3 – Comparison with NCA: Both NCA and Watershed classifier learns representations consistent with KNN classifiers. However, the strategy for regularization is different – NCA considers K nearest neighbours while Watershed considers a greedy approach. table 2 shows that greedy regularization is clearly superior to the alternate of considering K nearest neighbours.

Tuning `N_SEEDS`, we achieve the best score of 0.83 with `EMBED_DIM` $= 4$, and 0.883 with `EMBED_DIM` $= 16$. In comparison, NCA achieves 0.44 and 0.69, respectively. This illustrates two facts about watershed classifier – (i) Since the VC dimension is low, the generalization is much better and (ii) While both NCA and Watershed Classifier requires the representations to be dense, Watershed classifier can also work effectively with lesser dense representations. To see this, consider the difference between the scores – we get a difference of 0.4 for `EMBED_DIM` $= 4$ and 0.18 for `EMBED_DIM` $= 16$.

Observation 4 – Comparison with Linear Classifiers: Linear Classifiers and Nearest Neighbour classifiers can be considered as two opposing strategies for classification Hastie et al. (2009). Since Watershed classifier is inherited from Nearest Neighbour classifier, these classes share a lot of properties. On the other hand, while Linear classifiers try to separate the classes with linear boundaries, Watershed classifiers try to obtain dense representations so that the margin between the classes is maximized. Interestingly, although Watershed classifier outperforms the Linear classifier by a few points, we observe that both Linear classifiers and Watershed classifier perform similarly well.

5.3 Performance with large embedding networks f_θ

Finally, we validate the watershed classifier with a large embedding networks f_θ in this section. More details about the experiments in this section can be found in appendix D.

Experiment Design: To maintain diversity we consider three different architectures – (i) M3LC – A 3 layer convolution network, (ii) M3LFF – A 3 layer fully connected feed-forward network, (iii) M3LR18 – Resnet18 architecture modified for CIFAR datasets. Further, we also use widths 64 and 256 for M3LC and M3LFF networks. We use 3 datasets – FashionMNIST, CIFAR10 and CIFAR100 for comparison. Results can be found in table 3.

Observations: Most of the observations are consistent with the linear embedding networks – Watershed works on par with the linear classifiers, but out-performs NCA by a large extent. Thus, providing further evidence that greedy regularization of 1NN is more effective than KNN approaches.

Conjecture on complexity of Linear Classifier: It is surprising that linear classifiers work on par with greedy regularization of 1NN classifiers, although coming from two opposite ends of the spectrum Hastie et al. (2009). While the linear classifier has VC dimension which scales with the `EMBED_DIM`, the watershed classifier has a VC dimension of `N_SEEDS`. Specifically, we observe that considering `N_SEEDS` to be between 1 and 5 watersheds perform similarly well to linear classifiers. Thus, we conjecture that – *The complexity of Linear classifiers actually scale with the number of classes and not with `EMBED_DIM`.* The reason we expect is that – When learning a linear classifier, we are actually searching for a subspace with dimension – `N_CLASSES`. However, the VC dimension does not consider the optimization aspect and hence over-estimates the complexity of the classifier.

6 Conclusion and Future work

Parametric linear classifiers have largely been used with deep networks to obtain state-of-the art results. In comparison, to our knowledge, Non-parametric methods such as KNN do not obtain results comparable to linear classifiers. Neighbourhood Component Analysis (NCA) aims to learn the embedding consistent with KNN classifier. However, as we argue in this article – *It is better to regularize 1NN classifier using greedy approach rather than K-Nearest Neighbour approach.*

Using the greedy regularization, we propose *Watershed Classifiers*. Watershed classifiers have a single hyper-parameter – `N_SEEDS`, which control the VC dimension. We illustrate that, even with small VC-dimension, we can fit arbitrarily complex boundaries, assuming the high enough density. We propose a loss function in algorithm 1 to train the embedding. Interestingly, we observe that SGD approaches work very well, even if the loss function itself is highly non-convex. In section section 5, we show that watershed classifiers outperform the NCA for classification, and either outperform or match the performance of linear classifiers. To our knowledge, this is the first time a non-parametric classifier could match or outperform the parametric linear classifier.

7 Broader impact

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Margareta Ackerman and Shai Ben-David. A characterization of linkage-based hierarchical clustering. *J. Mach. Learning Research*, 2016.

Sunil Arya and David M. Mount. A fast and simple algorithm for computing approximate euclidean minimum spanning trees. In *Symp. on Discrete Algorithms*, 2016.

Aditya Challa, Sravan Danda, B. S. Daya Sagar, and Laurent Najman. Watersheds for semi-supervised classification. *IEEE Signal Process. Lett.*, 26(5):720–724, 2019.

Aditya Challa, Sravan Danda, B. S. Daya Sagar, and Laurent Najman. Triplet-watershed for hyperspectral image classification. *IEEE Trans. Geosci. Remote. Sens.*, 2022.

Giovanni Chierchia and Benjamin Perret. Ultrametric fitting by gradient descent. In *Neural Inform. Process. Syst.*, 2019.

Jean Cousty, Gilles Bertrand, Laurent Najman, and Michel Couprie. Watershed cuts: Minimum spanning forests and the drop of water principle. *Trans. Pattern Anal. Mach. Intell.*, 2009.

Jean Cousty, Gilles Bertrand, Laurent Najman, and Michel Couprie. Watershed cuts: Thinnings, shortest path forests, and topological watersheds. *Trans. Pattern Anal. Mach. Intell.*, 2010.

Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Symp. on Theory of Computing*, 2016.

Wei Dong, Moses Charikar, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Int. Conf. on World Wide Web*, 2011.

Alexandre Xavier Falcao, J Stolfi, RA Lotufo, BS da Cunha, and G Araujo. Image foresting transform. *Trans. Pattern Anal. Mach. Intell.*, 1999.

Jacob Goldberger, Sam T. Roweis, Geoffrey E. Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Neural Inform. Process. Syst.*, 2004.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, 2009.

David G. Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *AAAI Conf. on Artificial Intelligence*, 1993.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 2021.

Yair Movshovitz-Attias, Alexander Toshev, Thomas K. Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *Proc. Int. Conf. Comput. Vision*, 2017.

Enrique Fita Sanmartin, Sebastian Damrich, and Fred A. Hamprecht. Probabilistic watershed: Sampling all spanning forests for seeded segmentation and semi-supervised learning. In *Neural Inform. Process. Syst.*, 2019.

Enrique Fita Sanmartin, Sebastian Damrich, and Fred A. Hamprecht. Directed probabilistic watershed. In *Neural Inform. Process. Syst.*, 2021.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, 2002.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

Eu Wern Teh, Terrance DeVries, and Graham W. Taylor. Proxynca++: Revisiting and revitalizing proxy neighborhood component analysis. In *European Conf. Comput. Vision*, 2020.

Srinivas C. Turaga, Kevin L. Briggman, Moritz Helmstaedter, Winfried Denk, and H. Sebastian Seung. Maximin affinity learning of image segmentation. In *Neural Inform. Process. Syst.*, 2009.

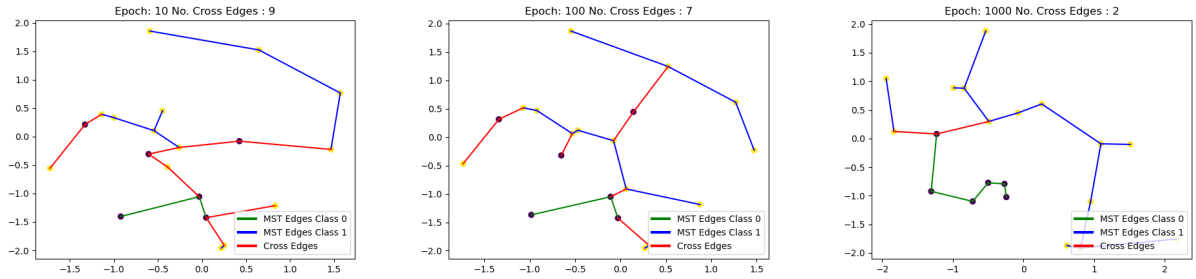
Steffen Wolf, Lukas Schott, Ullrich Köthe, and Fred A. Hamprecht. Learned watershed: End-to-end learning of seeded segmentation. In *Proc. Int. Conf. Comput. Vision*, 2017.

Zhirong Wu, Alexei A. Efros, and Stella X. Yu. Improving generalization via scalable neighborhood component analysis. In *European Conf. Comput. Vision*, 2018.

Yichen Xie, Masayoshi Tomizuka, and Wei Zhan. Towards general and efficient active learning. *arXiv:abs/2112.07963*, 2021.

Julian Yarkony and Charless C. Fowlkes. Planar ultrametrics for image segmentation. In *Neural Inform. Process. Syst.*, 2015.

A Verifying the Watershed Loss Function



(a) (b) (c)

Figure 5: Illustrating that the loss function in section 4 indeed is consistent with greedy 1NN propagation. The purple dots indicate class 1 and the yellow dots indicate class 0. Observe that as training progresses we have that number of cross-edges in the minimum spanning tree reduces. Note that greedy 1NN propagation is similar to the Prim's algorithm for constructing a minimum spanning tree. And hence the number of cross-edges provide a good measure on the efficacy of propagation.

Here, we empirically verify that the loss function in section 4 improves the correct label propagation.

Experimental Setup: Consider 20 arbitrary points on a 2d plane consisting of both class 0 and class 1. To experimentally verify the loss function, we directly optimize the co-ordinates of the points. We consider the simple SGD optimizer with learning rate 0.1 and no momentum.

Measuring the efficacy of propagation: To measure if the propagation is happening correctly, we look at number of cross edges – edges for which the end-points do not have the same labels, on the minimum spanning tree. Note that Greedy-1NN propagation is similar to the Prim's algorithm for constructing the minimum spanning tree. And hence, the number of cross-edges capture the efficacy of propagation quite well.

fig. 5 shows the result of optimizing the loss function. Observe that at epoch 10 we have 9 cross edges, at epoch 100 we have 7 and at epoch 1000 we only have 2 cross edges. This verifies that the loss function indeed learns the representations which are consistent with greedy 1NN propagation.

B Remark on fig. 2

We consider the two moons dataset[3] from scikit-learn with 1000 samples, `noise` $= 0.1$ and `random_state` $= 0$. Both the linear classifier and the decision tree classifier are also implemented using scikit-learn package.

Recall that inference for multiple samples differs in case of watershed classifiers and other classifiers. For visualization of the boundary, one should estimate the class for each point in the entire 2d grid. It is imperative that all the grid points are not considered together in the case of watershed classifier. This is because – A random sample technically should be from the same distribution as the training data. Considering all the grid-points would violate this assumption.

C Spiral Dataset Illustrations

figs. 6b to 6e shows the boundary for different widths of the hidden layer. Note that one needs at least 100 hidden nodes to learn the function. When considering `N_REV` $= 5$, even a width of 5000 did not suffice (as illustrated in fig. 6h). However, even width 2 suffices for the watershed classifier, irrespective of the `N_REV`. This is illustrated in figs. 6f and 6g.

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html`

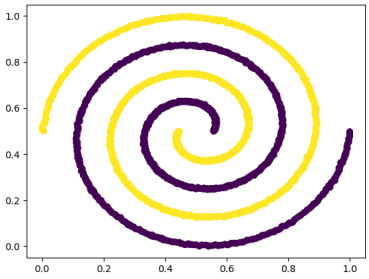
D Experimental Details for section 5.3

Network Architectures :

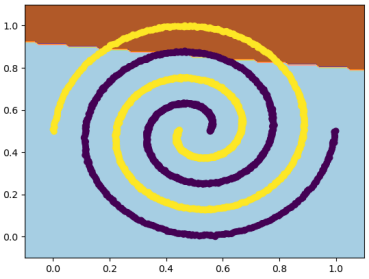
1. MR18: This is the ResNet18 architecture. The implementation is taken from `https://github.com/kuangliu/pytorch-cifar/blob/master/models/resnet.py`.
2. M3LC: We consider 3 convolution layers with width 256 and 64, with kernel size 5 and alternating maxpool layers with kernel size 2 and stride 2. The linear classifier has an additional classification layer on top of this.
3. M3LFF: We consider 3 fully connected feed-forward networks with width 256 and 64. The linear classifier has an additional classification layer on top of this.

Training Details: All experiments are run using A6000 Nvidia GPU with 16 GB memory. We use Adam optimizer with the learning rate $3e-4$. Experiments with learning rate schedules and other optimizers gave similar results. For watershed results, we run the experiment with $N\_SEEDS = 1, 5, 10, 20, 40, 100$ and report the best performance. We use $N = 256$ and $B = 2040$ for all the experiments. We use `early_stopping` with `patience` $= 20$ epochs on the validation data. We use $80:20$ train/valid split.

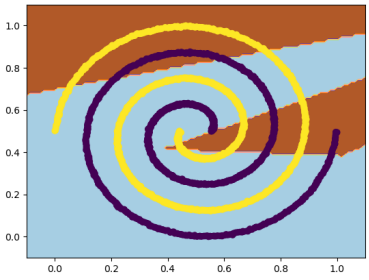
Timing Details: Using the batch size $B = 2040$ with $N = 256$ takes around 14 seconds on the system above with M3LC and M3LFF networks, and takes around 70*seconds* on MR18 backbone.



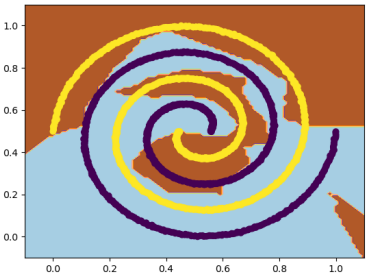
(a) Spiral dataset



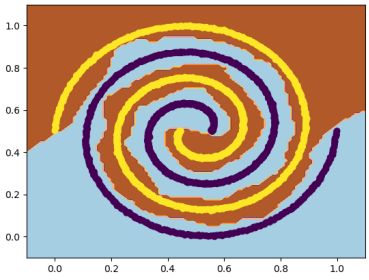
(b) LC – Width 2



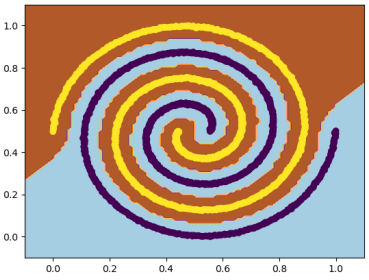
(c) LC – Width 10



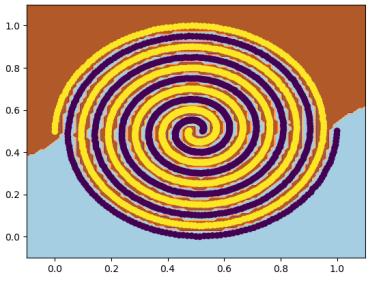
(d) LC – Width 100



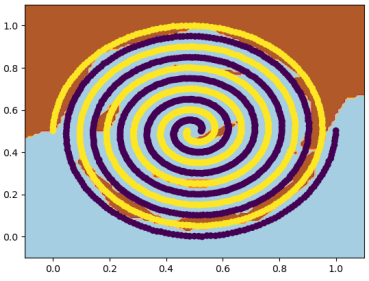
(e) LC – Width 500



(f) WS – Width 2



(g) WS – Width 2



(h) LC – Width 5000

Figure 6: Watershed vs. Linear Classifier as last layer on spiral dataset, illustrated in (a). For a simple spiral dataset, standard linear classifiers need a large number of hidden neurons to estimate the boundary. This is shown in (b)-(e). However, the with watershed classifier, even width 2 suffices as shown in (f). When the number of revolutions are increased, even 5000 hidden neurons did not suffice for linear classifier as shown in (h). On the other hand, width 2 suffices for watershed classifier, illustrated in (g).