

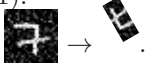
The objective of this lab is for you to program in Matlab radial basis function (RBF) networks and polynomials, both for nonlinear regression, apply them to some datasets and observe their behavior. The TA will first demonstrate the results of the algorithms on several datasets, and then you will program them, replicate those results, and further explore the datasets with the algorithms. You can use the textbook, lecture notes and your own notes.

**Important:** for RBF networks, develop your code so it works with inputs  $\mathbf{x} \in \mathbb{R}^D$  and outputs  $\mathbf{y} \in \mathbb{R}^{D'}$  for any dimensions  $D, D' \geq 1$ . This is particularly easy with RBF networks. For polynomials, focus on 1D regression problems only, i.e., with inputs  $\mathbf{x} \in \mathbb{R}$  and outputs  $\mathbf{y} \in \mathbb{R}$ .

## I Datasets

Firstly, construct your own toy datasets to visualize the result easily and be able to get the algorithm right. Take the input instances  $\{\mathbf{x}_n\}_{n=1}^N$  in  $\mathbb{R}$  or  $\mathbb{R}^2$  and the labels  $\{y_n\}_{n=1}^N$  in  $\mathbb{R}$ . Generate a noisy sample from a known function, e.g.  $y_n = f(x_n) + \epsilon_n$  where  $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$  and  $f(x) = ax + b$  or  $f(x) = \sin(x)$ .

Then, try the MNIST dataset of handwritten digits, with instances  $\mathbf{x} \in \mathbb{R}^D$  (where  $D = 784$ ). Create a ground-truth mapping  $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$  as follows (as in the previous lab about gradient descent and linear models):

- A random mapping with output dimension  $D'$ , e.g. take  $\mathbf{A}_{D' \times D}$  and  $\mathbf{b}_{D' \times 1}$  with elements in  $\mathcal{N}(0, 1)$ .
- A mapping that rotates, scales, shifts and possibly clips the input image  $\mathbf{x}$  and adds noise to it, e.g. . This makes it easy to visualize the result, since the desired output  $\mathbf{f}(\mathbf{x})$  for an image  $\mathbf{x}$  should look like  $\mathbf{x}$  but transformed accordingly.

## II RBF networks for nonlinear regression

We minimize the least-squares error

$$E(\{\mathbf{w}_h, \boldsymbol{\mu}_h\}_{h=1}^H, \sigma) = \frac{1}{2} \sum_{n=1}^N \|\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n)\|^2 + \lambda \sum_{h=1}^H \|\mathbf{w}_h\|^2 \quad (1)$$

where  $\lambda \geq 0$  is a regularization user parameter which controls the smoothness of  $\mathbf{f}$ , and  $\mathbf{f}$  is an RBF network:

$$\mathbf{f}(\mathbf{x}) = \sum_{h=1}^H \mathbf{w}_h \phi_h(\mathbf{x}) \quad \phi_h(\mathbf{x}) = \exp\left(-\frac{1}{2} \|\mathbf{x} - \boldsymbol{\mu}_h\|^2 / \sigma^2\right) \quad \mathbf{x} \in \mathbb{R}^D, \mathbf{f}(\mathbf{x}) \in \mathbb{R}^{D'} \quad (2)$$

where the radial basis functions  $\{\phi_h(\cdot)\}_{h=1}^H$  are (proportional to) Gaussians with centroids  $\{\boldsymbol{\mu}_h\}_{h=1}^H \subset \mathbb{R}^D$  and common width  $\sigma$ , and  $\{\mathbf{w}_h\}_{h=1}^H \subset \mathbb{R}^{D'}$  are weights. We take  $\phi_1(\mathbf{x}) \equiv 1$  if we want to use a *bias*.

We will train RBF networks in an approximate but simple and fast way as follows:

1. Set the centroids  $\{\boldsymbol{\mu}_h\}_{h=1}^H$  in an unsupervised way using only the input points  $\{\mathbf{x}_n\}_{n=1}^N$ , by simply selecting  $H$  points at random. If you want, you can further refine this by using those points as initial centroids for  $K$ -means.
2. Set the width  $\sigma$  by hand to some reasonable value (we will cross-validate it, see below).
3. Given the centroids and width, the values  $\phi_h(\mathbf{x}_n)$  are fixed, and the weights  $\{\mathbf{w}_h\}_{h=1}^H$  are determined by optimizing  $E$ , which reduces to a simple linear regression. We solve the linear system:

$$(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\mathbf{I})\mathbf{W} = \boldsymbol{\Phi}\mathbf{Y}^T \quad \boldsymbol{\Phi}_{H \times N} = (\phi_h(\mathbf{x}_n))_{hn}, \mathbf{W}_{H \times D'} = (\mathbf{w}_1, \dots, \mathbf{w}_H)^T, \mathbf{Y}_{D' \times N} = (\mathbf{y}_1, \dots, \mathbf{y}_N). \quad (3)$$

There are 3 hyperparameters for the user to set: the number of basis functions  $H$ , the width  $\sigma$ , and the regularization parameter  $\lambda$ . We set them by cross-validation using a grid search. For example, we can use  $H \in \{3, 5, 10, 50\}$ ,  $\sigma \in \{2^{-2}, 2^0, 2^2, 2^4\}$  and  $\lambda \in \{0, 10^{-5}, 10^{-3}, 10^{-1}\}$  (the actual values will depend on your problem, particularly for  $\sigma$ ). We train an RBF network (on the training set) for each combination of values of  $(H, \sigma, \lambda)$  and pick the one with lowest error on a validation set.

### III Polynomial regression

We minimize the same error as in eq. (1) but using the canonical basis for polynomials as basis functions (in 1D):

$$f(x) = \sum_{h=0}^H w_h \phi_h(x) \quad \phi_h(x) = x^h \quad x \in \mathbb{R}, f(x) \in \mathbb{R}. \quad (4)$$

Since the basis functions have no parameters (width, centroids),  $\phi(x) = (1, x, x^2, \dots, x^H)^T$  is fixed given  $x$ , so finding the optimal weights can be solved exactly as in eq. (3) by solving a linear regression. The hyperparameters  $H$  and  $\lambda$  are set by cross-validation as before.

### IV What you have to do

**Toy problem** This applies to both RBF networks and polynomials.

- Firstly, verify that optimizing the least-squares error (1) over the weights  $\{\mathbf{w}_h\}_{h=1}^H$  alone, given the fixed basis function values  $\phi_h(\mathbf{x}_n)$  for  $n = 1, \dots, N$  and  $h = 1, \dots, H$ , can indeed be done by solving the linear system (3).
- Implement the linear system solution in Matlab with `linsolve` or the “\” operator. You can also use `inv` but this is numerically more costly and less stable.
- Write code to compute the basis function values at each data point,  $\phi_h(\mathbf{x}_n)$  for  $n = 1, \dots, N$  and  $h = 1, \dots, H$ . For RBF networks, set the centroids and width as described above.
- Having done this, you can determine the weights given a training set, and hence train an RBF network (no need for iterative optimization or initial weight values). Then:
  - Observe how the resulting RBF network fits the data (see plots below).
  - Train several RBF networks for a set of hyperparameter values  $(H, \sigma, \lambda)$ , evaluate their validation error, and pick the best.

Questions to consider:

- How does the RBF network look like if we vary one of the hyperparameters keeping the rest fixed, that is:
  - if you increase  $H$ ?
  - if you increase  $\sigma$ ?
  - if you increase  $\lambda$ ?

And, how does this affect the training and the validation error? For example, what value of  $H$  (or  $\lambda$ , or  $\sigma$ ) gives the lowest training error? How about the lowest validation error?

- What is the value of the RBF net  $f(x)$  for a point  $x$  that is far from any training point?
- For a given point  $x$ , how many hidden units (i.e., BFs) have nonnegligible value  $\phi_h(x)$ ?
- What gives lower error, using a random subset of points as centroids for the RBFs, or running  $k$ -means?

To visualize the results, create the following plots for the RBF networks:

- Plot the dataset  $(y_n$  vs  $x_n)$  and the RBF network  $f(x)$ .
- Having trained RBF networks for a set of hyperparameter values  $(H, \sigma, \lambda)$ , plot the training and validation error for them.

Repeat for polynomials.

**MNIST dataset** This applies only to RBF networks.

- Select a small enough subset of MNIST as training inputs (using the whole dataset will be slow). Apply the ground-truth linear transformation to the data to generate the output labels  $\mathbf{y} = \mathbf{f}(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$ . Then proceed as with the toy dataset to compute the optimal solution exactly by picking a value for  $(H, \sigma, \lambda)$ , picking  $H$  training images as basis function centroids, and solving the linear system to get the weights.
- Same questions as with the toy dataset. Note that the appropriate values for  $\sigma$ , etc. may now be significantly different.

To visualize the results, create the following plots:

- Plot the training and validation error for several RBF networks (using different hyperparameter values).
- For each RBF network:
  - Plot each centroid  $\boldsymbol{\mu}_h$ ,  $h = 1, \dots, H$ , as a grayscale image.
  - If using as linear mapping the rotation/shift/scale/clip transformation, which produces as output a (possibly smaller) image  $\mathbf{y}_n$ , plot the following for a few sample images: the input image  $\mathbf{x}_n$ , and the outputs  $\mathbf{y}_n$ . Compare with the result produced by the true linear mapping.