Seminars on Overparametrized Machine Learning: Hand-in assignment 1

Antônio H. Ribeiro, Dave Zachariah, Per Mattsson

September 10, 2021

Due: 30th of September 2021, 23:59

The items in the assignment can be implemented in the programming language of choice. Nonetheless, we recommend the usage of Python as a programming language, since we might include suggestions of functions and code snippets in the exercise description.

Double-descent in random Feature regression

Follow the instructions to reproduce the double-descent behaviour in a regression problem. The setup resembles that used in Figure 2 in (Belkin et al., 2019): you will use random Fourier features and the minimum-norm solution. For simplicity, however, it will be used a smaller dataset and one that deals with regression (rather than classification).

Dataset. You will use the Boston house price dataset. The dataset is available in: https://archive.ics.uci.edu/ml/machine-learning-databases/housing/.

> **Note:** If you are using Python, scikit-learn library does have a function for loading the dataset, see sklearn.datasets.load_boston.

Divide the dataset into 60% / 40% splits. Using the 60% split for training and the 40% split for testing the model.

Model. Consider a linear-in-the-parameter model for predicting the output from the input:

$$\hat{y}_j = f(x_j) = \sum_{i=1}^{m} \theta_i \phi_i(x_j). \tag{1}$$

where ϕ_i, $i = 1, \cdots, m$ are nonlinear transformations that map the input to the feature space. As in Belkin et al. (2019) Figure 2, you will use random Fourier features (see a more detailed description next).

Given the training dataset $\{(x_j, y_j), j = 1, \cdots, n\}$, the model is estimated by finding the values θ_i that minimize

$$\frac{1}{T} \sum_{t=1}^{T} \left(y_t - \sum_{i=1}^{m} \theta_i \phi_i(x_j) \right)^2. \tag{2}$$

Or, equivalently, in matrix form, by finding the vector $\theta \in \mathbb{R}^m$ that minimizes

$$\frac{1}{T} \|y - \Phi\theta\|^2, \tag{3}$$

where $\Phi \in \mathbb{R}^{n \times m}$ is the matrix containing $\phi_j(x_i)$ at position (i, j) and $y \in \mathbb{R}^T$ is the vector of outputs. Indeed, finding the optimal parameter here is an ordinary least-squares problem and its analytical solution is

$$\hat{\theta} = (\Phi^\mathsf{T}\Phi)^{+}\Phi^\mathsf{T} y, \tag{4}$$

In the overparametrized regime, there are multiple solutions, and it is possible to prove that, in this case, using the above solution yields the minimum ℓ_2-norm solution to the problem (as used by Belkin et al. (2019)), i.e.:

$$\hat{\theta} = \arg\min_{\theta} \|\theta\|_2 \quad \text{subject to} \quad \Phi\theta = y, \tag{5}$$

where $(\Phi^\mathsf{T}\Phi)^{+}$ denotes the Moore-Penrose pseudo-inverse of $\Phi^\mathsf{T}\Phi$.

> **Note:** scipy.linalg.lstsq does yield the desired behaviour both in the underparametrized and overparametrized region.

Random Fourier features Use the feature map introduced by Rahimi and Recht (2008) to approximate the reproducing kernel Hilbert space (RKHS) defined by the Gaussian kernel $K(x, x') = \exp(-\gamma\|x - x'\|^2)$. More precisely, the features are generated as

$$\phi_i(x) = \sqrt{\frac{2}{m}} \cos(w_i^\mathsf{T} x + b_i), \tag{6}$$

where $w_i \in \mathbb{R}^n$ is a vector with each element sampled independently from $\mathcal{N}(0, 2\gamma)$ and $b_i \in \mathbb{R}$ is sampled from a uniform distribution $\mathcal{U}[0, 2\pi)$.

> **Note:** There is a implementation of random Fourier features in scikit-learn that you can use: sklearn.kernel_approximation.RBFSampler.

> You might notice that the formulation above is slightly different than the one presented in Belkin et al. (2019) for RFF. They are equivalent tough, and the one above has the advantage of do not be expressed using complex numbers.

Exercise

Let n be the number of training points, solve the above problem for m ranging from $0.1n$ to $10n$ (generate at least 100 configurations in this range using logspace). Plot as a function of the number of features m:

1. **Train** mean square error;
2. **Test** mean square error;
3. The **parameter** ℓ_2 **norm** $\|\theta\|_2$.

> Some tips for the plots to look nice:
> - Close to the interpolation point the test error takes very large values. We suggest to manually set the y-limits in the plot, so the region of interest is highlighted. The same also applies to the parameter norm.
> - Using logscale in the x-axis might also make the plot more clear.
> - Maybe add a vertical line in the threshold $m = n$ to show where the interpolation threshold is.

The Submission

Your submission should have a single page of content (a4paper, fontsize=10pt, margin=2cm, both single and double column are acceptable...). Include your name, the plots, a short description of experiment parameters that you used (i.e., which γ) and a paragraph of discussion/conclusion. You can assume that whoever will read your report has both read paper from (Belkin et al., 2019) and the entire description above, so there is no need for repeating it...

All requested plots should have proper figure captions, legends, and axis labels. You should submit two files, one pdf-file with the report and a standalone script (or jupyter notebook) that can be used to run the code and generate the plots (Write as comments the packages/libraries versions and additinal requirements as comments in the top of the script).

References

M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, Aug. 2019. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1903070116.

A. Rahimi and B. Recht. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems 20*, pages 1177–1184. 2008.