Corentin Ambroise
Luis Montero
Margaux Zaffran

Kernel Methods
Generalization Properties - 2020

02/26/20

Contents

1 Project: new algorithms for large scale learning with kernels

In this first section we will focus on kernel methods for large scale learning. We will first introduce the Kernel Ridge Regression (KRR) framework, then try to explain our understanding of recent papers on the subject ([5], [3], [7] and [6]) and finally propose a numerical study of theses papers (all implementations are available in python at `https://github.com/fd0r/rkhs_map670`, theoretical results presented in the papers are not further explained). The following tries to follow the reproducibility checklist: `https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf`.

1.1 Learning with kernels

Let's first describe the usual framework for kernel learning, with a focus on kernel ridge regression (KRR).
Framework: We have n observations in $(\mathcal{X}, \mathcal{Y})$, in the case of regression $\mathcal{Y} = \mathbb{R}$ but we could consider others supervised learning frameworks, (possibly with $\dim(\mathcal{X}) = \infty$, in fact when $n << \dim(\mathcal{X})$) we can consider that $\dim(\mathcal{X}) = \infty$), and $K : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ a kernel on $\mathcal{X}$.
Let $\mathcal{H}$ be a Hilbert space, with $\mathcal{H} \subset \mathbb{R}^{\mathcal{X}}$ and let $f \in \mathbb{R}^{\mathcal{X}}$. We choose K such that $\mathcal{H}$ is a RKHS associated to K, i.e. :

- $\mathcal{H} = \overline{\text{Vect}(\{K(\cdot, x); \forall x \in \mathcal{X}\})}$ (closure)
- $\forall f \in \mathcal{H}; \langle K(\cdot, x), f\rangle = f(x)$ (we also note $K(\cdot, x)$ as K_x)

Our goal is to find the solution to

$$\min_f \mathbb{E}[(f(x) - y)^2]$$

In fact we only have access to a sampling (x_i, y_i) of our distribution so what we are minimizing is actually the empirical risk defined as:

$$\hat{R_n}(f) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

Finding the global solution is hard so we constrain our solution to belong to $\mathcal{H}$. Assuming that we want to find a solution in this sub-space we can reformulate the equation with the RKHS formalism as:

$$\hat{R_n}(f) = \frac{1}{n}\sum_{i=1}^{n}(\langle f(\cdot), K_{x_i}\rangle - y_i)^2$$

But our space $\mathcal{H}$ that can be described as $\mathcal{H} = \text{Vect}(K_{x_i}) \bigoplus \text{Orthognal}$ (so that $\forall f \in \text{Orthognal}, \forall i \in [1, \dots, n]$; $\langle f, K_{x_i}\rangle = 0$), thus $\forall f \in \mathcal{H}, f = f^{\perp} + f'$ with $f' \in \text{Vect}(\{K_{x_i}\}_{i\in[1,\dots,n]}) = H_n$, i.e. we can write $f'(x)$ as $\sum_{i=1}^{n} \alpha_i K_{x_i}(x)$.

1.1.1 KRR Exact Resolution

We are looking for the exact minimizer $\alpha \in \mathbb{R}^n$ of $\hat{R_n}(f) = \hat{R_n}(\alpha)$ with $f \in \mathcal{H}$ i.e. $f = \sum \alpha_i K_{x_i}$

$$\hat{R_n}(f) = \frac{1}{n}\sum_{j=1}^{n}\left(\left(\sum \alpha_i K_{X_i}\right)(X_j) - y_j\right)^2$$

We can then introduce the kernel matrix $\hat{K} = (K(x_i, x_j))_{(i,j)\in[1,\dots,n]^2}$ and write the problem as

$$\frac{1}{n}\left\|\hat{K}\alpha - y\right\|^2$$

By adding a ridge penalty we have:

$$\frac{1}{n}\left\|y - \hat{K}\alpha\right\|^2 + \lambda\|f\|^2 = \frac{1}{n}\left\|y - \hat{K}\alpha\right\|^2 + \lambda\alpha^T\hat{K}\alpha$$

As sum of convex functions this is still convex so the solution is unique and can be found by setting the gradient to 0, i.e. :

$$-\frac{1}{n}\hat{K}(y - \hat{K}\alpha) + \lambda\hat{K}\alpha = 0$$
$$\implies \hat{K}(\hat{K} + \lambda nI)\alpha = \hat{K}y$$

$$\implies \alpha = (\hat{K} + \lambda n I)^{-1} y$$

Since the exact minimization of the empirical risk requires the computation of the inverse of $\hat{K} \in \mathbb{R}^{n\times n}$ it can be very expansive computationally $(0(n^3))$. One solution could be to try to find the optimal solution by other optimization methods.

1.1.2 KRR gradient descent

To solve this issue we can optimize the $\hat{\alpha}$ by gradient descent. With the regularization λ and the learning rate γ we have:

$$\alpha^t = \alpha^{t-1} + \gamma[(\hat{K}\alpha^{t-1} - y) + \lambda n \alpha^{t-1}]$$

1.1.3 KRR stochastic gradient descent

Another issue is that $\hat{K}$ might not fit in memory, to solve that we can do stochastic gradient descent. Instead of considering $\hat{K} = K_{nn}$ and y we can consider the sub-sampled K_{mn} and y_m and do the same computation as before:

$$\alpha^t = \alpha^{t-1} + \gamma[(K_{mn}\alpha^{t-1} - y_m) + \lambda n \alpha^{t-1}]$$

1.1.4 KRR incremental

1.2 New algorithms

Other solutions to solve the memory and computational issues for large scale learning have been proposed. We are going to focus on three papers published at NeurIPS; [5] [6] and [3], but we could also cite [1].

1.2.1 Nyström approximation

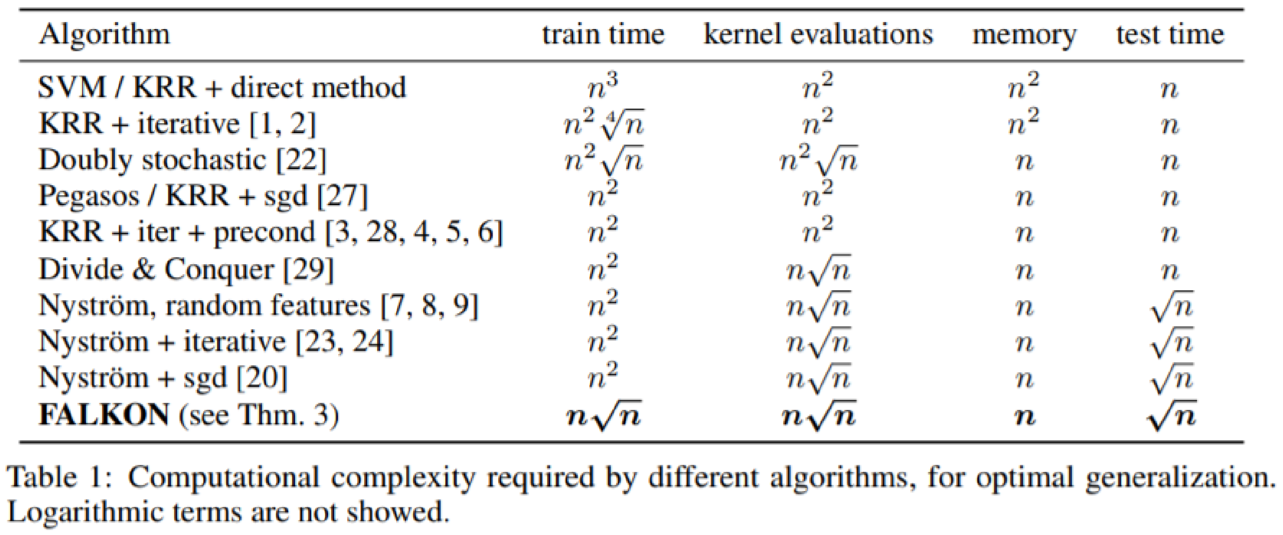
When n grows the kernel matrix, of size $n \times n$ might not fit in memory anymore. The Nyström approximation tries to solve this problem by computing the kernel matrix only on a sub-sample of the data. We define as $\{\tilde{x}_i\}_{i\in[1,\ldots,m]}$, with $m < n$ the sampled data. The kernel matrix $\hat{K} = (K(x_i, x_j))_{i,j\in[1,\ldots,n]^2}$ becomes $\tilde{K} = (K(x_i, \tilde{x}_j))_{i,j\in[1,\ldots,n]\times[1,\ldots,m]}$. That means that we are approximating $\mathcal{H}_n = Vect(\{K(x_i,\cdot)\}_{i\in[1,\ldots,n]})$ by $\mathcal{H}_m = Vect(\{K(\tilde{x}_i,\cdot)\}_{i\in[1,\ldots,m]})$ so we can write our new functions as $f(x) = \sum_{i=1}^{m} \tilde{\alpha}_i K(\tilde{x}_i, x)$.
Our previous solution $\hat{\alpha}$ becomes $\tilde{\alpha} = (K_{nm}^T K_{nm} + \lambda n K_{mm})^{\dagger} K_{nm}^T y$ where $K_{mn} = (K(x_i, \tilde{x}_j))_{i,j\in[1,\ldots,n]\times[1,\ldots,m]})$ and $K_{mm} = (K(\tilde{x}_i, \tilde{x}_j))_{i,j\in[1,\ldots,m]^2})$ (and $(\cdot)^{\dagger}$ the Moore-Penrose inverse operator). The matrix we are (pseudo-)inverting is of size $m \times m$ (and not $n \times n$ as previously), and we can control this parameter based on our problem/computational/memory capabilities.
They are two ways explained in the paper "Less is More: Nyström Computational Regularization" [5] to sample the data:

- Plain Nystrom: the data is just sampled uniformly without replacement
- Approximate leverage scores Nystrom: in this case the data is sampled with replacement according to an approximation of each element leverage score. The leverage score being a parametric metric defined as $l_i(t) = (K_n(K_n + tnI)^{-1})_{ii}$

The main goal of the paper is to show that besides solving computational issues, the Nystrom approximation also helps regularize the model. They give theoretical bounds on the approximated solution (details in the paper) for both the uniform sampling and ALS sampling.
Let us note that the theoretical optimal bounds values differ from the values taken in the numerical approach. We will comment the numerical results in the last sections **??** and **??**.

1.2.2 FALKON

In FALKON [6] the authors improve the Nystrom approach by using an iterative solver with preconditioning. The main interest of this paper is the comparison of the different approaches in term of convergence rates and optimality. They summarize that in the table 1



Algorithm	train time	kernel evaluations	memory	test time
SVM / KRR + direct method	n^3	n^2	n^2	n
KRR + iterative [1, 2]	$n^2\sqrt[4]{n}$	n^2	n^2	n
Doubly stochastic [22]	$n^2\sqrt{n}$	$n^2\sqrt{n}$	n	n
Pegasos / KRR + sgd [27]	n^2	n^2	n	n
KRR + iter + precond [3, 28, 4, 5, 6]	n^2	n^2	n	n
Divide & Conquer [29]	n^2	$n\sqrt{n}$	n	n
Nyström, random features [7, 8, 9]	n^2	$n\sqrt{n}$	n	$\sqrt{n}$
Nyström + iterative [23, 24]	n^2	$n\sqrt{n}$	n	$\sqrt{n}$
Nyström + sgd [20]	n^2	$n\sqrt{n}$	n	$\sqrt{n}$
FALKON (see Thm. 3)	$\boldsymbol{n\sqrt{n}}$	$\boldsymbol{n\sqrt{n}}$	$\boldsymbol{n}$	$\boldsymbol{\sqrt{n}}$

Table 1: Computational complexity required by different kernel algorithms, for optimal generalization. Logarithmic terms are not showed.

Figure 1: Summary from [6]

1.2.3 Random Fourier Features

The idea of random Fourier Features, as explained in [3], is to lower the dimensional-space of the features, from d to a certain D and thus to lower the computation cost.

This idea is theoretically based on the Bochner's theorem:

Theorem 1 (Bochner). *A continuous kernel $k(x, y) = k(x - y)$ on $\mathbb{R}^d$ is positive definite if and only if $k(\delta)$ is the Fourier transform of a non-negative measure.*

Thus, we can use the Fourier transform of our kernel to simulate values from the obtained distribution (if the non-negative measure is indeed a probability measure) and estimate our kernel. More precisely, we can write:

$$k(x - y) = \int_{\mathbb{R}^d} p(w)e^{jw'(x-y)}dw = \mathbb{E}_w[\xi_w(x)\xi_w(y)^*]$$

with $\xi_w(x) = e^{jwx}$ as defined in [3]. Thus, the estimator $\xi_w(x)\xi_w(y)^*$ of $k(x, y)$ is unbiased.

If p is scaled correctly, we then can generate w from p, and with $\varphi_w(x) = [\cos(wx)\sin(wx)]$, we have $\varphi_w(x)^T\varphi_w(y) \approx k(x - y)$. To approximate correctly this kernel, we can generate D such w, and use $\frac{1}{D}\sum_{i=1}^{D}\varphi_{w_i}(x)^T\varphi_{w_j}(y)$. Then, taking quite a small D in comparison to d, can lower considerably the computation costs.

This approximation actually converges to the kernel thanks to the Hoeffding's inequality under some hypothesis. Details of this claim can be found in [3].

We can wonder for which kernel we obtain such a p and this approximation can thus be used. It is for example the case with the gaussian kernel.

Example 1 (gaussian kernel). *Let $G(x) = e^{-\frac{||x||^2}{2\sigma^2}}$, with $x \in \mathbb{R}^d$. Its Fourier tranform is given by $\widehat{G}(\xi) = (2\pi\sigma^2)^{\frac{d}{2}}e^{-\frac{\sigma^2||\xi||^2}{2}}$ (see [4] for details of the computation).*

As explained with detail in [7], this approximation can be used within the framework of ridge regression, as did the authors of [3] in their work. Indeed, we've obtained an approximation of the kernel K and we can use it to find the best α_i for a regression function of the form: $\hat{f}(x) = \sum_{i=1}^{n}\alpha_i K(x_i, x)$.

For this, we would need to compute our estimation of the kernel matrix, with a dot product or the pairwise_kernel function of scikit-learn. In fact, this computation is quite long (more than 3 minutes on our synthetic data with $D = 300$). In comparison with the LinearSVC it is very long (and this execution time is only for the kernel matrix, not for the resolution!), thus we didn't go further on this approach.

Convergence bounds of this method have been given and proved in [7].

Instead, since what we approximate is our chosen kernel, and this is done by a dot product of the transformed data, we decided to run a linear SVM on the transformed data.

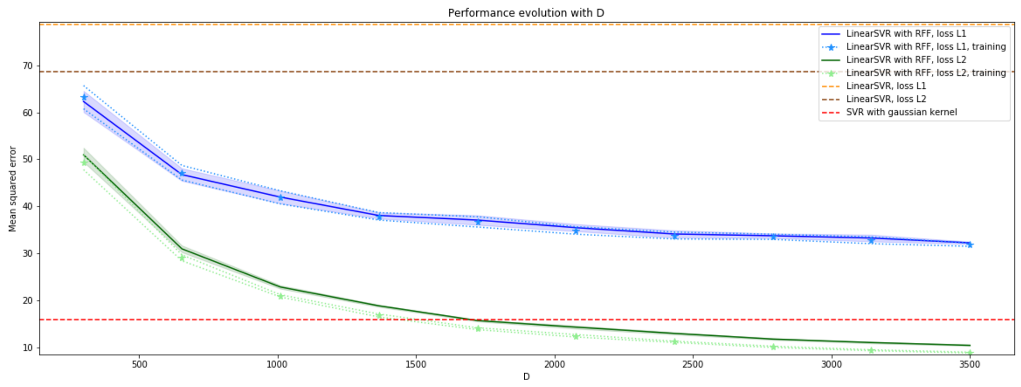
1.3 Applications

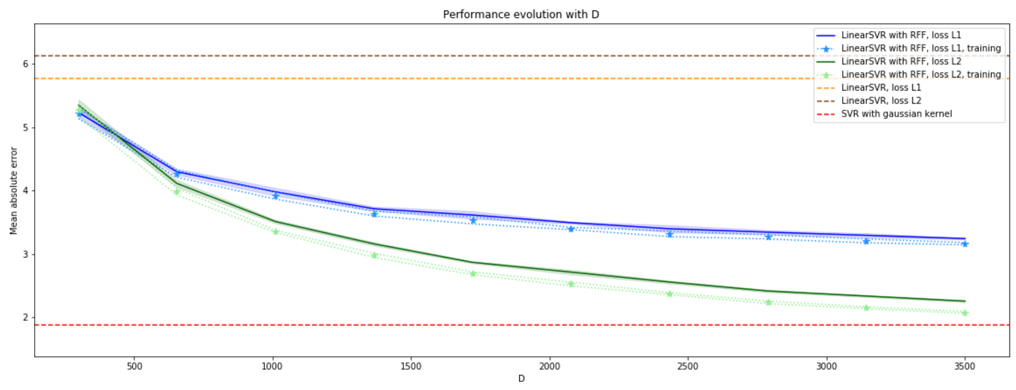
For the numerical study, we chose to benchmark the different methods, on two datasets. There is a regression problem and a classification problem.

For each method, we studied what was their best parameter values for different type of regularization, for different losses according to the problem, while comparing with other algorithms.
The experiments were run under macOs Catalina 10.15.2, on a MacBook Pro 13-inch 2018, with 8GB of RAM and a quadra core processor 2.3GHz Intel Core i5.

1.4 Comparison on real regression data

The regression problem is a dataset of 53500 observations, each one of them has 385 features. The features are CT-scan acquisition parameters, the goal is to predict the height coordinate of the CT-Scan slice on the patient. The data was generated by computing CT-scans of 74 different patients. This dataset can be downloaded following this link. Again, the dataset has been been splitted using `train_test_split` from scikit learn, with a random_state of 24, and default parameters (thus the test set is 25% of the initial one). The presented results are in fact aggregated performances on 5 independent runs, with mean and standard deviation.





We can see that even for a really large D (much bigger than d), our model is fast and has a better performance. The approximation of the kernel improves with D, which is consistent with the claim in [3].

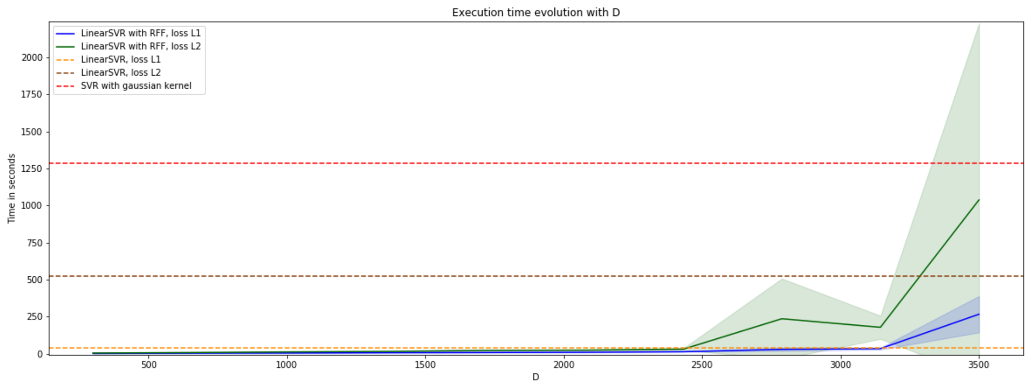
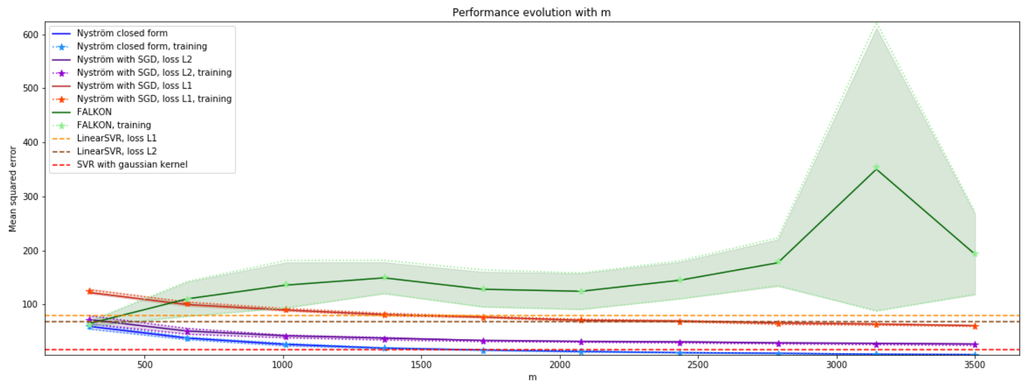


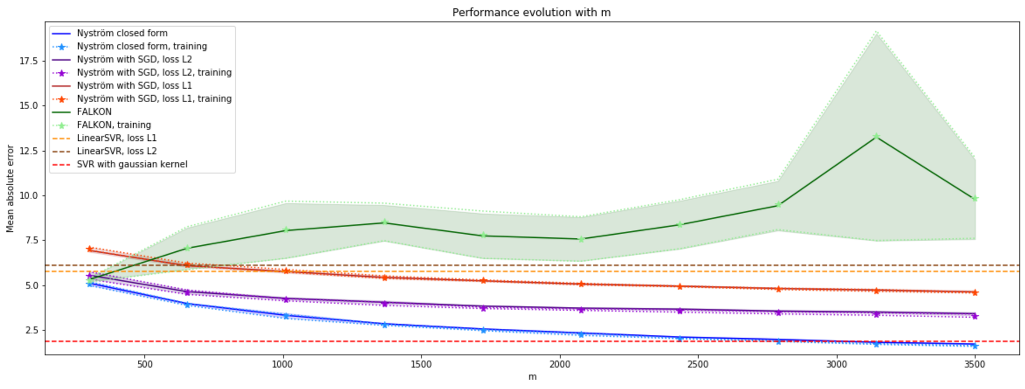
Figure 2: Training time against the number of feature

The training and testing scores are extremely close, in mean and variance, which tend to tell us that the models generalize well and don't overfit.

We can also see that a model with a loss L2 performs better in MAE than the one trained with a loss L1.

With respect to the execution times, we obtained a training time of 1285 seconds for the SVR with a gaussian kernel, thus we considerably reduced the time.





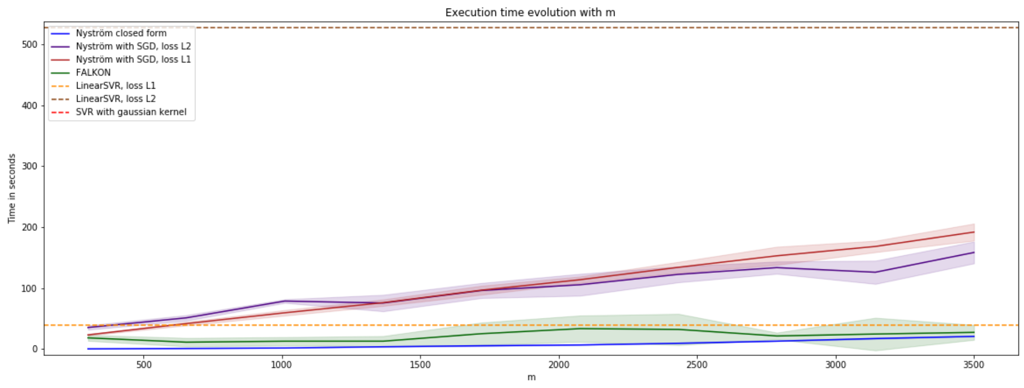
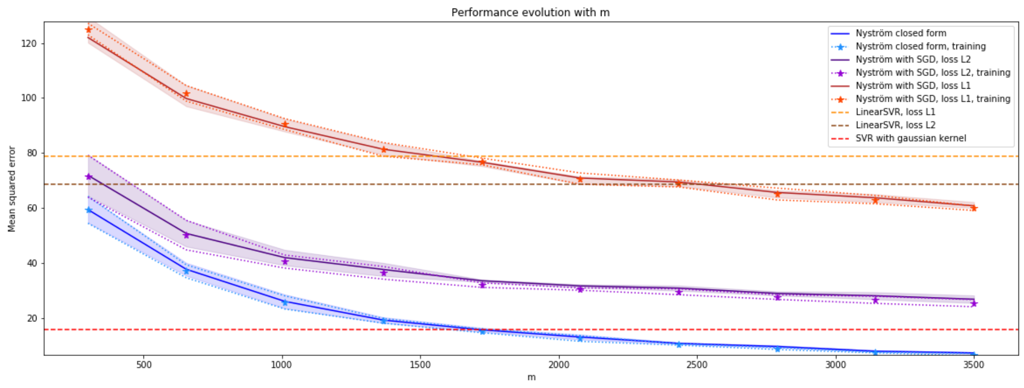


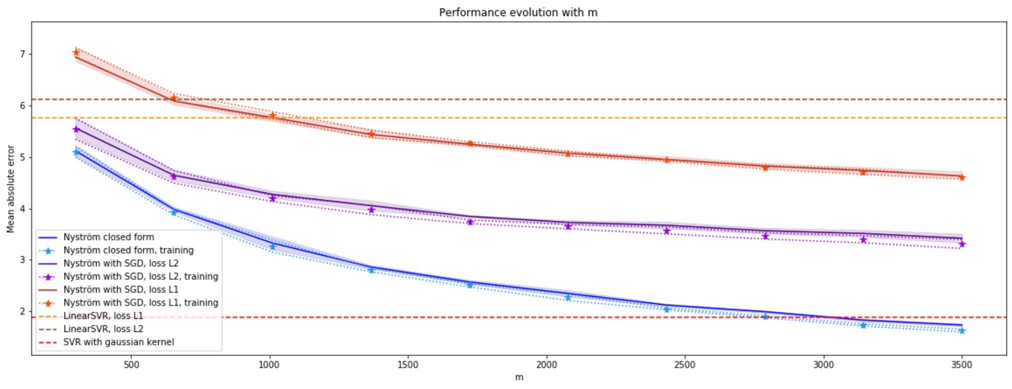
Figure 3: Training time against the number of feature

Falkon doesn't seem to perform well, at least when m increases. As explained in [5], this parameter play the role of a regularization parameter and a trade-off between m and C, regularization parameter, needs to be found. With a small C like we did here, m seems to need to be small: this sounds similar to the results they obtained using grid-search on other datasets. It could explain our result.

We observe a reduction of the execution time again.

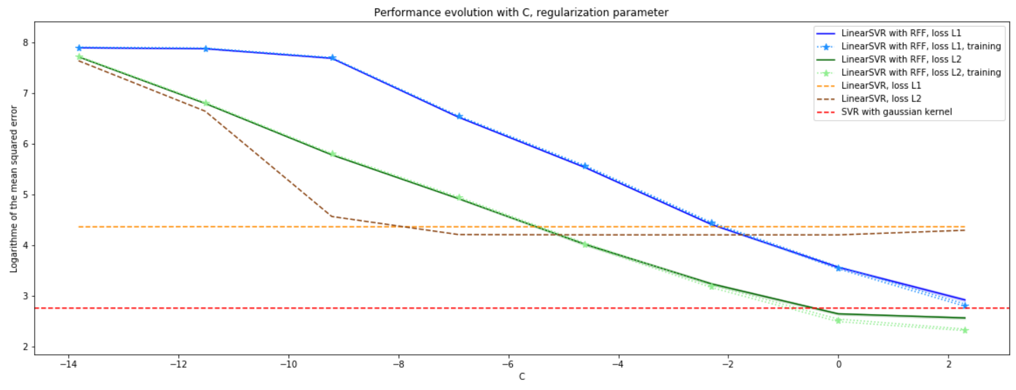
We now delete Falkon of the plots to zoom on the Nyström methods.

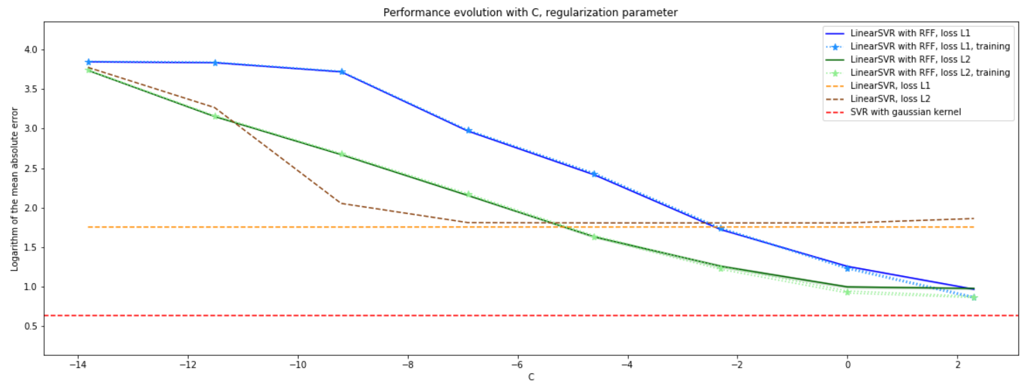




We observe that the exact resolution of Nyström isn't that long, despite the matrix inversions, and at the same time performs extremely well (probably thanks to the exactness).

As with the random fourier method, the loss L2 performs better in MSE and in MAE.





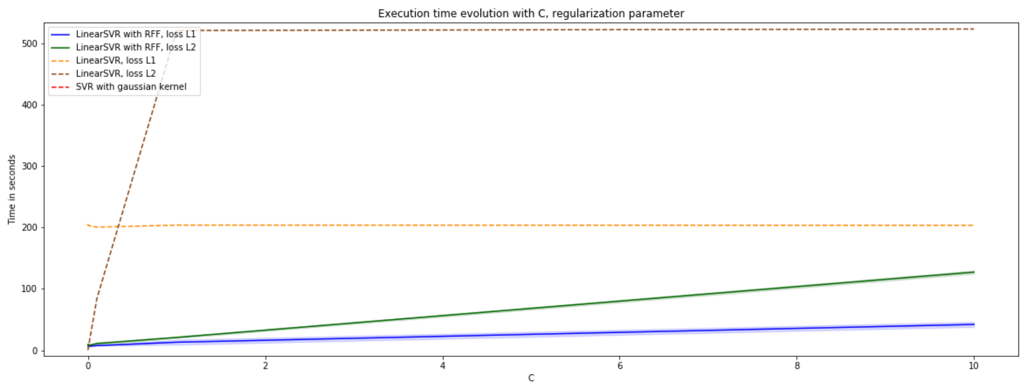
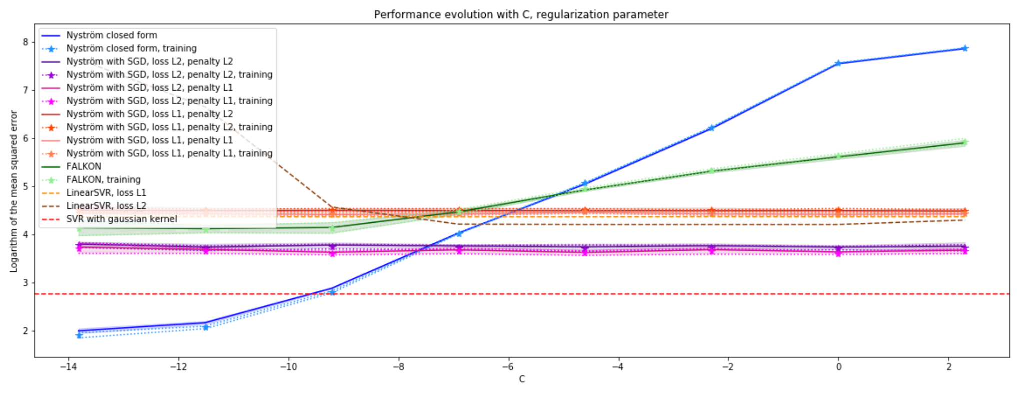
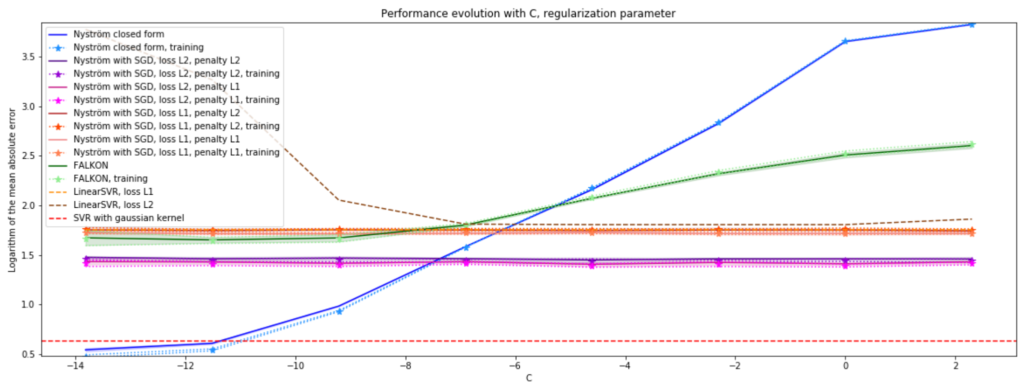


Figure 4: Training time against regularization strength

First, the training time increases really fast with the regularization.

Second, we observe that with a more important regularization the performances are increasing: indeed, the regularization can help to generalization and lead to simpler solutions.





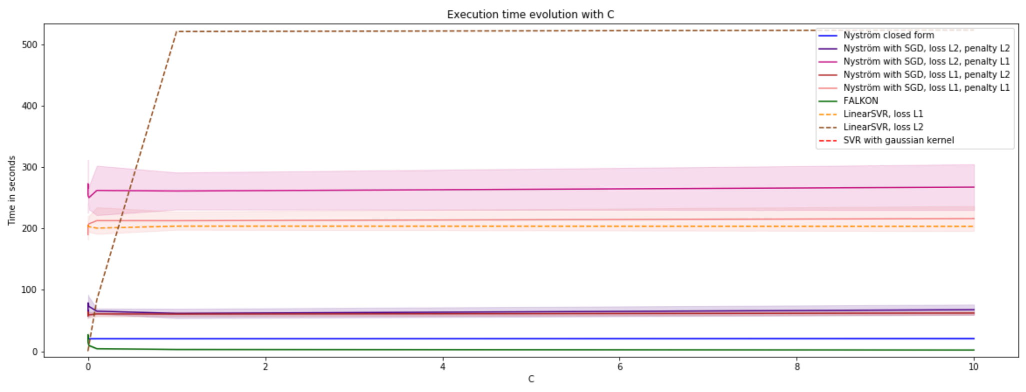


Figure 5: Training time against regularization strength

At the contrary, for Falkon, increasing the regularization decreases the performance: again, it may be due to the trade-off we were talking about, mentioned in [5].

In an exact setting (Nyström closed form), if we regularize too much, the penalty takes over the loss and we go too far away from the true solution. The regularization is surely more useful in a stochastic and non exact approach.

1.5 Comparison on simulated classification data

The classification problem is the one provided by the *dataset* API of scikit-learn with the function `make_classification` and a random_state of 17. We simulated this way 50 000 samples with 120 features, 15 of them being informative. It is a binary classification problem. The dataset has been been splitted using `train_test_split` from scikit learn, with a random_state of 24, and default parameters (thus the test set is 25% of the initial one). The presented results are aggregated perfomances on 10 independent runs.

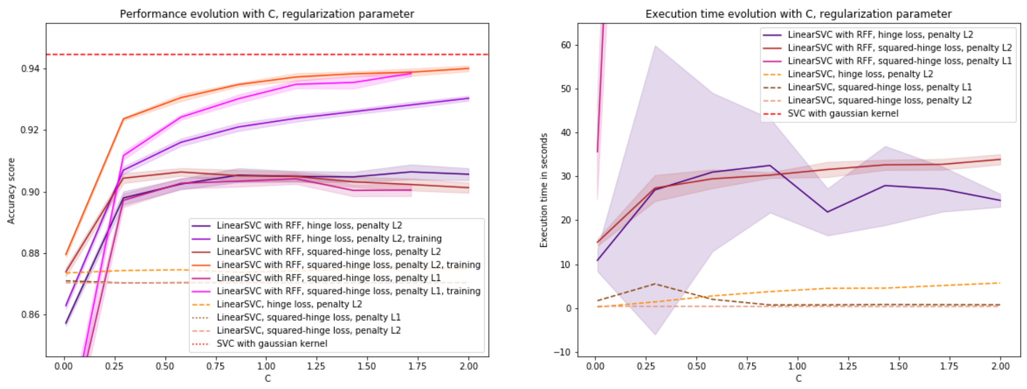


Figure 6: Performance and training time against the regularization strength

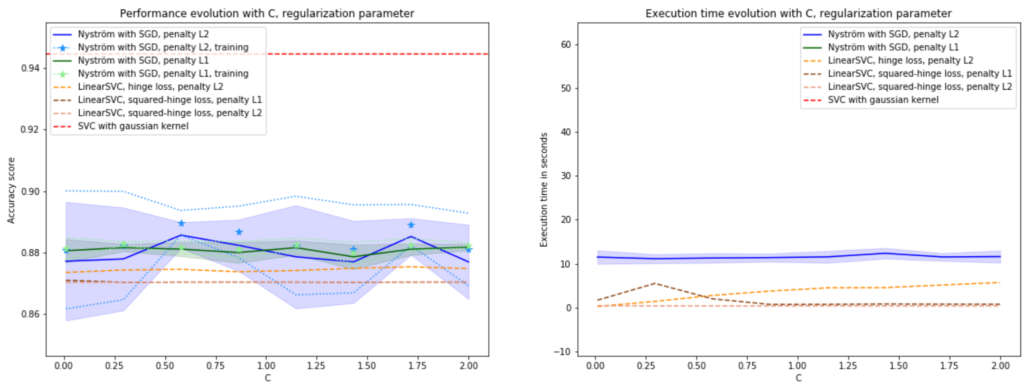


Figure 7: Performance and training time against regularization strength

Nystrom seems to yield unstable results for this classification tasks while it did not have this issue in our regression framework.

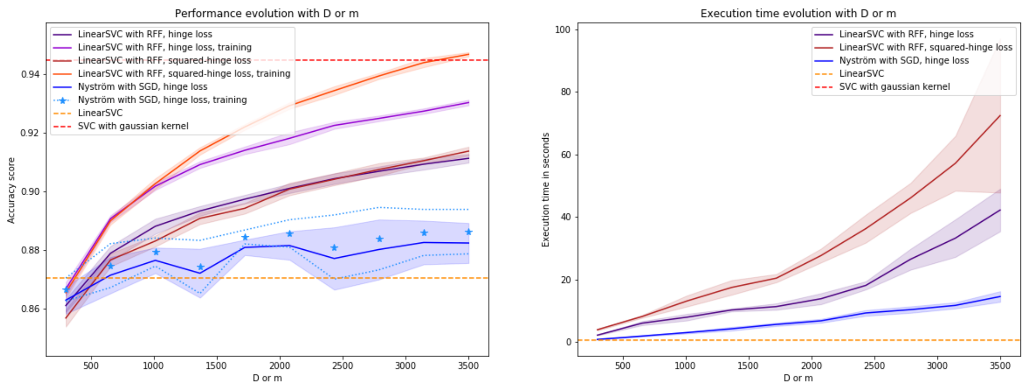


Figure 8: Performance and training time against the regularization strength

2 Paper: Understanding Deep Learning requires rethinking generalization [2]

2.1 Introduction

This paper starts by pointing out that neural networks achieve very small difference between test and train performances, eventhough they usually have a lot of parameters. Common knowledge would lead us to say it is due to properties of the function family generated by these networks, or regularization used during training.

In fact, in most of the well-known configurations, model have more parameters than the number of samples they are trained on. On the other hand, it is pretty easy to find a network architecture that generalizes poorly. So this leads us to an interesting question: what is it that makes a neural network model generalize well? Answering this could help us design future architectures in a more predefined and framed way, as well as improve today's model interpretation techniques.

This is why this article relates a number of experimentations showing that usual ways of evaluating model complexity in the statistical learning theory framework, namely the Vapnik-Chervonenkis dimension, the Rademacher complexity and the uniform stability, are not enough to qualify the generalization properties of a neural network.

2.2 Principal results and experimentations

The main idea of the paper is to empirically demonstrate that we do not have the technical means today to understand why neural networks generalize so well. It is then supported by a simple theoretical result which says that we can fit any function that maps the data to its labels with a 2-layers deep MLP with ReLu activations. Each one of the following subsections correspond more or less to sections 2, 3 and 4 of the article respectively.

2.2.1 Effective capacity

For this part, a few experiments were realised. The authors considered various neural network architecture that are known to work well for multi class and multi label classification tasks. They chose to work with Inception, Alexnet and a simple Multi Layer Perceptron, trained and evaluated on CIFAR10 mainly, but also some work was done on the dataset from ImageNet which offers a challenge quite different, since classes are not exclusive.

Their experimentations consisted mostly in randomization tests, inspired from non-parametric statistics. They biased the datasets in order to see how the networks learn and generalize under these constraints. They applied different corruptions to either the labels or the input data, notably by drawing some or all the labels from a uniform distribution between the valid labels, shuffling the pixels of the images or drawing them from a Gaussian distribution.

On the CIFAR10 dataset, the interesting finding was that whatever network was used and randomization was applied to the data, the model always achieved a perfect fit on the training data, without hyperparameter change from the case where the data was not modified. Even in the cases where the level of corruption of the data was the highest, the learning duration in term of number of training steps was not too far from when the data was untouched (a factor 3 in the worst corruption case, when all the labels are drawn at random).

Off course, the more a dataset is corrupted, the more the learning task will take time, and the generalization error (difference between test and training errors) will tend towards 0.9, since CIFAR10 contains images from 10 different classes. But this shows that any of these networks can easily ”brute force” the learning task by memorizing the whole dataset with their numerous parameters.

The results are less flagrant when they use the ImageNet dataset, but they still achieve 95.20% accuracy with random labels, with a version of the Inception model (for 99.53% with the true labels).

This leads to the fact that the Rademacher complexity associated to the family of function generated by neural networks is more or less 1, which is a trivial upper bound for this complexity, and so does not lead to any useful generalization bound. The same kind of reasoning works for the VC dimension, for which the paper refers to other related works. Different notions of stability of the learning algorithms are usually used to bound the generalization error. These bounds are ineffective in practice.

2.2.2 Regularization

The paper distinguishes two types of regularization. The explicit regularization is used when the model architecture is deemed unable to regularize enough by itself, and so specific techniques are used. In the case of neural networks, explicit regularization takes different forms, like weight decay, dropout or data augmentation.

The experimentations are done with the same network architectures on the same datasets. It seems that these regularization techniques act more in deep learning as tuning parameters whereas in convex learning, regularization is sometimes necessary to prevent convergence toward a trivial solution. The experiences consists in training these models by applying different type of regularizers for each run, choosing between weight decay, dropout and data augmentation, on both CIFAR10 and ImageNet datasets.

Results show that in some cases, regularizers will have quite a big impact on the generalization performance of a model. But in general, even without regularizer, the neural network models generalize well and even though regularizers have an impact on the test performances, usually bigger gains can be achieve by changing the model architecture.

Then we have implicit regularization techniques. While explicit regularizers may not be essential to generalization, it is a fact that not all model that fit the training data generalizes well. Early stopping is a well-known regularization technique. It is proven efficient for Inception applied to the ImageNet dataset, while it is not when applying it on CIFAR10 images. Another implicit regularizer is batch normalization. Even though it was not necessary designed for regularization, it was proven that it can improve generalization performance. For Inception on CIFAR10, it indeed improves the test error by some 3-4%, and stabilizes the learning dynamics.

Even if these regularization techniques are often usefull, they do not fully explain why some neural network generalizes so well, because they still perform well without them.

2.2.3 Theoretical result

Although most of the paper relies on empirical observations and summary analysis of the literature, they provide one simple and interesting theoretical result.

Until then, a lot of effort was put into characterizing the representation capabilities of certain types of neural networks. This paper focuses on finite-sample expressivity of neural networks, its representational capacities given a sample with n observations.
Given a neural network has more parameters than the number of observations in the sample, it can represent any function mapping the observations to their labels, with at least 2 layers.
More precisely, *there exists a two-layer neural network with ReLU activations and $2n+d$ weights that can represent any function on a sample of size n in d dimensions*.

2.3 A link with kernel learning

In the fifth section of the article, the authors decide to show that even with simple models (here linear models), it is not easy to understand the source of generalization either. They want to give us an intuition from parallel insights provided by the example of linear models.

In particular, they consider the empirical risk minimization problem. If the dimension d of the data points is superior to the number of data points n, then we can fit any labeling of the data. But how is it possible to generalize then with such a rich model and no explicit regularization?

They show that with an initialization to 0 of the weights with a first order descent algorithm to solve the optimization problem (they use the SGD in their paper), the problems is equivalent to solving a system involving the kernel matrix (kernel trick). The only problems to solve this are the memory space required to store all this information and the computation time, hence this is only doable for datasets with less than a hundred thousand observations, like MNIST or CIFAR10. But even for a small dataset like MNIST, it requires 30Gb of RAM to store this matrix. This is where our work on kernel matrix approximation like Nyström or FALKON can help us if we don't have Google's super computers.

Finally, this method achieves very good performances on the test datasets. For MNIST, applying a wavelet preprocessing reduces slightly the generalization error, but any kind of explicit regularization does not. However for CIFAR10 images, applying a convolutional neural network as preprocessing reduces the test error by a factor 3, and eplicit l_2 regularization only improves the test performance by 2%.

Then, SGD often converges towards the solution with minimal norm, which is a good generalization property. Usually solutions that generalize poorly have higher norms. This is how SGD acts as an implicit regularizer. But small norm does not always mean best generalization properties. For example, the solution without preprocessing of the problem on the MNIST dataset has l_2 norm about 220, whereas the best solution we found, with preprocessing, has norm 390. So while minimal norm is a good intuition to design our models, it does not solve the generalization problem by itself.

2.4 Main takeaways

The goal of this paper is not to solve the generalization issues in machine learning today. It is to provide some useful insights and ideas that can help the reader better understand how generalization works in neural networks, and what we know about it. It is an intermediate between a summary of the literature in 2016 of how generalization is characterized, and empirical evidences showing why and how this can, or more precisely can not be applied to neural network models.

It is an elegant paper that shows that the theoretical notions used until this day in machine learning to describe generalization properties of an algorithm are note applicable in deep learning, with only two neural network architectures and 3 different datasets, by conducting relevant and well defined experiments. In this way, the paper fulfilled his role and this is why it won the Best Paper Award of ICLR 2017. But after this reward was attributed to the paper, a lot of people started to question it and argue that it is not so revolutionary.

2.5 A highly criticized paper

We read some of the critics from notably a Open Review thread on the paper [8] (main source of review on machine learning papers?) and a Quora thread [9]. Here is what we retained from it.

Often qualified of a "though provoking" paper, it did shed light on some generalization properties of neural networks, but not by providing proofs or rigorous methodologies to solve these issues, which bothered a lot of people. They provide empirical intuition but did not actually find any result, except from their theoretical finding. They do not address questions like why does an architecture of neural network generalizes better than an other with the same number of parameters and don't really provide strong insight about how to proceed forward to solve the problem of generalization in deep learning.

Some said that they only stated a fact that was already well known then: we do not understand how neural networks work, and it applies also to their generalization properties. But others replied that is often necessary to point out paradoxes and empirical evidences that theoretical results are not sufficient to solve an issue in order to progress in science. In that sense, it is qualified as a valuable contribution to research in this machine learning.

Others has an other vision on deep learning. They think neural networks only work by memorization, and the fact that they can generalize maybe is only due to their numerous parameters that allow them to have locally hash function. So they would not they that neural networks generalize well, and question the notion of generalization used here. So this reasoning would make these model work uniquely by overfitting. Some argue this is the conclusion toward which the section on the effective capacity of neural networks with the randomized test should tend to.

Other sections of the paper were also challenged by many reviewers, but authors often replied calmly and with clarity. They never claimed that this paper would be groundbreaking, the ICLR 2017 committee decided so. The goal of the paper was not to answer the question ”How generalization works in deep learning” but was more of a warning saying ”Hey, techniques that we commonly use in machine learning to give good generalization properties to an algorithm mostly don’t apply to neural networks. It’s time to do some work! Here is what we know”. And in that sense, the authors proved properly their point with well-conducted experiments.

References

[1] Raffaello Camoriano, Tomás Angles, Alessandro Rudi, and Lorenzo Rosasco. Nytro: When subsampling meets early stopping. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1403–1411, Cadiz, Spain, 09–11 May 2016. PMLR.

[2] Moritz Hardt Benjamin Recht Oriol Vinyals Chiyuan Zhang, Samy Bengio. Understanding deep learning requires rethinking generalization. `https://arxiv.org/pdf/1611.03530.pdf`, 2016.

[3] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1177–1184. Curran Associates, Inc., 2008.

[4] Salim Rostam. Transformée de fourier de la gaussienne. `https://perso.ens-rennes.fr/math/people/salim.rostam/files/agreg/D%C3%A9veloppements/Transform%C3%A9e%20de%20Fourier%20de%20la%20gaussienne.pdf`, 2014. [Online; accessed 23-Febrary-2020].

[5] Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Less is more: Nyström computational regularization. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1657–1665. Curran Associates, Inc., 2015.

[6] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3888–3898. Curran Associates, Inc., 2017.

[7] Alessandro Rudi and Lorenzo Rosasco. Generalization properties of learning with random features. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3215–3225. Curran Associates, Inc., 2017.

[8] OpenReview thread. Understanding deep learning requires rethinking generalization reviews. `https://openreview.net/forum?id=Sy8gdB9xx`, 2016.

[9] Quora thread. Why is the paper “understanding deep learning requires rethinking generalization” important? `https://www.quora.com/Why-is-the-paper-%E2%80%9CUnderstanding-Deep-Learning-Requires-Rethinking-Generalization%E2%80%9D-important`, 2017.