McKernel: A Library for Approximate Kernel Expansions in Log-linear Time

J. D. Curtó*,1,2,3,4, I. C. Zarza*,1,2,3,4, F. Yang[2,6], A. Smola[2,5,6], F. Torre[2,7], C. W. Ngo[4], and L. Gool[1].
[1]Eidgenössische Technische Hochschule Zürich. [2]Carnegie Mellon. [3]The Chinese University of Hong Kong.
[4]City University of Hong Kong. [5]Amazon. [6]Google. [7]Facebook.
*Both authors contributed equally.

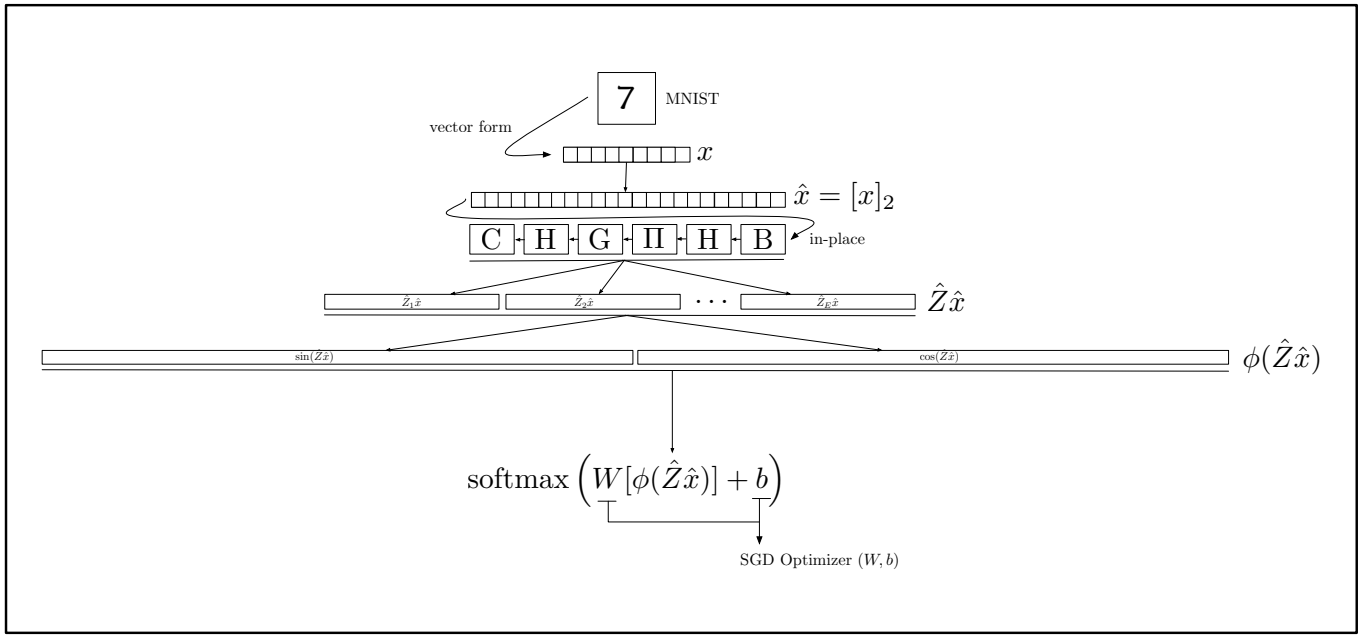

Figure 1. **Diagram of McKernel**. We visually describe $\text{softmax}(W\tilde{x} + b)$ where $\tilde{x} = \text{mckernel}(x)$. The original image is padded in form of long vector to the nearest power of 2, mapping $\hat{Z}$ is applied in-place. Calibration C defines the choice of Kernel. The tensor is expanded by the number of Kernel Expansions E building a network with high compositionality. Finally, use real feature map ϕ, Equation 9. SGD Optimizer finds appropriate weights W and bias b. Compute $\hat{Z}$ on-the-fly keeping same seed both for training and testing.

McKernel introduces a framework to use kernel approximates in the mini-batch setting with Stochastic Gradient Descent (SGD) as an alternative to Deep Learning. Based on Random Kitchen Sinks [Rahimi and Recht 2007], we provide a C++ library for Large-scale Machine Learning[1]. It contains a CPU optimized implementation of the algorithm in [Le et al. 2013], that allows the computation of approximated kernel expansions in log-linear time. The algorithm requires to compute the product of matrices Walsh Hadamard. A cache friendly Fast Walsh Hadamard that achieves compelling speed and outperforms current state-of-the-art methods has been developed. McKernel establishes the foundation of a new architecture of learning that allows to obtain large-scale non-linear classification combining lightning kernel expansions and a linear classifier. It travails in the mini-batch setting working analogously to Neural Networks. We show the validity of our method through extensive experiments on MNIST and FASHION MNIST [Xiao et al. 2017].

CCS Concepts: • **Neural Networks**; • **Kernel Methods**;

[1]McKernel is available at https://www.github.com/curto2/mckernel

{curto,zarza,vangool}@vision.ee.ethz.ch,fengyang@google.com,
{smola,ftorre}@cs.cmu.edu,cwngo@cs.cityu.edu.hk
decurto.tw dezarza.tw.

Additional Key Words and Phrases: Kernel Methods, Deep Learning, Hadamard.

1 Introduction

Kernel methods offer state-of-the-art estimation performance. They provide function classes that are flexible and easy to control in terms of regularization. However, the use of kernels in large-scale machine learning has been beset with difficulty. This is because using kernel expansions in large datasets is too expensive in terms of computation and storage. In order to solve this problem, [Le et al. 2013] proposed an approximation algorithm based on Random Kitchen Sinks by [Rahimi and Recht 2007], that speeds up the computation of a large range of kernel functions, allowing us to use them in big data. [Rudi and Rosasco 2017] describes the generalization of Random Features and potential effectiveness. Recent works on the topic build on it to propose state-of-the-art embeddings [Hong et al. 2017; Kawaguchi et al. 2018; Moczulski et al. 2016; Yang et al. 2015].

In this work, we go beyond former attempts [Al-Shedivat et al. 2017; Cho and Saul 2009; Wilson et al. 2016] and propose a general framework in lieu of Deep Learning. Our goal is to integrate current

advances in Neural Networks but at the same time propose a well established theoretically sound background.

[Wigner 1960] claims the unreasonable effectiveness of mathematics in the natural sciences. [Vapnik and Izmailov 2018] in the same way, states the unreasonable effectiveness of mathematics in machine learning. Kernel methods originate in rigorous mathematical treatment of the problem, while at the same time are incredibly effective.

At its heart, McKernel requires scalar multiplications, a permutation, access to trigonometric functions, and two Walsh Hadamard for implementation. The key computational bottleneck here is the Walsh Hadamard. We provide a fast, cache friendly SIMD (Single Instruction Multiple Data) oriented implementation that outperforms state-of-the-art codes such as Spiral [Johnson and Püschel 2000]. To allow for very compact distribution of models, we use hashing and a Pseudo-random Permutation for portability. In this way, for each feature dimension, we only need one floating point number.

In summary, our implementation serves as a drop-in generator of features for linear methods where attributes are generated on-the-fly [Chwialkowski et al. 2015; Li et al. 2016; Reddi et al. 2015; Sharmanska et al. 2013; Wang et al. 2017; Wang and Ji 2015], such as for regression, classification, or two-sample tests. This obviates the need for explicit kernel computations, particularly on large amounts of data.

Outline: Learning with Kernels is briefly introduced in Section 2. We begin then with a description of the feature construction McKernel in Section 3. Fast Walsh Hadamard is enunciated in Section 4. This is followed by a discussion of the computational issues for a SIMD implementation in Section 5. The concepts governing the API are described in Section 6. Large-scale Machine Learning by means of McKernel is discussed in Section 7. Concepts regarding TIKHONOV regulation are explained in Section 8. Experimental results can be found in Section 9. Section 10 gives a brief overall discussion.

2 Learning with Kernels

The problem of learning [Cortes and Vapnik 1995; Cucker and Smale 2001; Poggio and Smale 2003; Vapnik and Izmailov 2015; Vapnik and Vashist 2009] arises from the necessity to adapt a model $f : X \to Y$ to a given training set of data $S_n = (x_c, y_c)_{c=1}^n$, with $X \subset \mathbb{R}^n$ being closed and $Y \subset \mathbb{R}$, having f good properties of generalization.

Let $(x_c, y_c)_{c=1}^n$ be the data. Then, we pick a function $k_x(x') = k(x, x')$ symmetric, positive definite and continuous on $X \times X$. And set $f : X \to Y$ such that

$$f(x) = \sum_{z=0}^{n} t_z k_{x_z}(x), \tag{1}$$

where $t = (t_1, \ldots, t_n) \in \mathbb{R}^n$ and

$$(n\gamma I + K)t = y, \tag{2}$$

where matrix I is the identity, K is the matrix square positive definite with elements $k_{c,r} = k(x_c, x_r)$ and $\gamma > 0$ in $\mathbb{R}$.

It turns out this linear system of equations in n variables is well-posed as K is positive and $(n\gamma I + K)$ is strictly positive.

The intuition behind this algorithm, for instance given the Gaussian

$$k(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right), \tag{3}$$

is that we approximate the unknown function by a weighted superposition of Gaussians, each centered at location x_c of one of the n examples. The weight t_c of each Gaussian is chosen to minimize the error on the training set. The σ of the Gaussian, together with γ, controls the degree of smoothing, of noise tolerance and generalization.

[Vapnik and Izmailov 2018] proposes a generalization of this framework by the use of invariants. Equation 2 becomes

$$(n\gamma I + VK)t = Vy, \tag{4}$$

where V takes into account mutual positions of observed vectors and elements $V(c, z)$ can be computed for high-dimensional problems as follows

$$V(c, z) = \sum_{k=1}^{d} \left(t_k - \max(x_c^k, x_z^k)\right), \tag{5}$$

with $0 \leq x_k \leq t_k$. Matrix V is a symmetric non-negative matrix. The main idea is to take into account that the desired decision rule is related to the conditional probability function of the observations.

3 McKernel

Kernel methods work by defining a kernel function $k(x, x')$ on a domain X. We can write k as inner product between feature maps, as follows

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \tag{6}$$

for some suitably chosen ϕ. Random Kitchen Sinks [Rahimi and Recht 2007] approximate this mapping of features ϕ by a FOURIER expansion in the case of radial basis function (RBF), scilicet whenever $k(x, x') = \kappa(x - x')$. This is possible since the FOURIER transform diagonalizes the corresponding integral operator. This leads to

$$k(x, x') = \int \exp(i\langle w, x \rangle) \exp(-i\langle w, x' \rangle) d\rho(w) \tag{7}$$

for some L_2 measurable function $\rho(\omega) \geq 0$ that is given by the FOURIER transform of κ. Random Kitchen Sinks exploit this by replacing the integral by sampling $\omega \sim \rho(\omega)/\|\rho\|_1$. This allows for finite dimensional expansions but it is costly due to the large number of inner products required. [Le et al. 2013] resolves this for rotationally invariant κ by providing a fast approximation of the matrix $W := [\omega_1, \ldots \omega_n]$.

This is best seen for the RBF kernel, Equation 3. Since FOURIER transforms of Gaussians are Gaussians, albeit with inverse covariance, it follows that $\rho(\omega) \propto \exp\left(-\frac{\sigma^2}{2}\|\omega\|^2\right)$ and that W contains random variables independent and identically distributed (i.i.d.) Gaussian. It is this matrix that McKernel approximates via

$$\hat{Z} := \frac{1}{\sigma\sqrt{n}} CHG\Pi HB. \tag{8}$$

Here C, G and B are diagonal matrices, Π is a random permutation matrix and H is the Hadamard. Whenever the number of rows in W exceeds the dimensionality of the data, we can simply generate multiple instances of $\hat{Z}$, drawn i.i.d., until the required number of dimensions is obtained.

Binary B**.** This is a matrix with entries $B_{kk} \in \{\pm 1\}$, drawn from the uniform distribution. To avoid memory footprint, we simply use Murmurhash as function of hashing and extract bits from $h(k, x)$ with $x \in \{0, \dots N\}$.

Hadamard H**.** This matrix is iteratively composed of $H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix}$. It is fixed and matrix-vector products are carried out efficiently in $O(n \log n)$ time using the Fast Walsh Hadamard. We will discuss implementation details for a fast variant in Section 5.

Permutation Π**.** We generate a random permutation using the FISHER-YATES shuffle. That is, given a list $L = \{1, \dots n\}$ we generate permutations recursively as follows: pick a random element from L. Use this as the image of n and move n to the position where the element was removed. The algorithm runs in linear time and its coefficients can be stored in $O(n)$ space. Moreover, to obtain a deterministic mapping, replace the generator of random numbers with calls to the function of hashing.

Gaussian G**.** This is a matrix diagonal with entries i.i.d. Gaussian. We generate the random variates using the BOX-MULLER transform [Box and Muller 1958] while substituting the generator of random numbers by calls to the function of hashing to allow us to recompute the values at any time without the need to store random numbers.

Calibration C**.** This is a random scaling operator whose behavior depends on the type of kernel chosen, such as the RBF MATÉRN Kernel, the RBF Kernel or any other radial spectral distribution [Yang et al. 2014].

Ultimately, compute the feature pairs by assigning

$$[\cos(\hat{Z}x), \sin(\hat{Z}x)]. \tag{9}$$

In particular, McKernel computes the features by using the real version of the complex feature map ϕ in [Rahimi and Recht 2007]. SIMD vectorized instructions and cache locality are used to increase speed performance. These allow a speed improvement of 18x times for a 2^{24} dimension input matrix.

4 Fast Walsh Hadamard

Applications of the WALSH HADAMARD transform range across several areas, including Machine Learning [Andoni et al. 2015; Lu et al. 2013] and Computer Vision [Ben-Artzi et al. 2007; Ouyang and Cham 2010].

A naïve implementation results in complexity $O(n^2)$. A divide-and-conquer approach for this task that runs in time $O(n \log n)$ can be derived as follows.

We define the matrix Hadamard H_n

$$H_0 = [1] \tag{10}$$

$$H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix}, \tag{11}$$

with dimension $2^n \times 2^n$.

We want to compute the product of matrix vector $H_n \cdot c$, being $c = (c_o, c_1)$ where $|c_0| = |c_1| = \frac{|c|}{2}$,

$$H_n \cdot c = \begin{bmatrix} H_{n-1}c_0 + H_{n-1}c_1 \\ H_{n-1}c_0 - H_{n-1}c_1 \end{bmatrix}. \tag{12}$$

Hence, we only need to compute recursively $H_{n-1}c_0$ and $H_{n-1}c_1$ to obtain $H_n \cdot c$ via additions and subtractions. The running time is

$$T(n) = 2T(n/2) + O(n), \tag{13}$$

that gives $T(n) = O(n \log n)$.

In McKernel we generalize this approach to compute the resulting matrix Hadamard from a hard-coded specific-size routine.

5 Implementation of Fast Walsh Hadamard

A key part of the library is an efficient implementation of Fast Walsh Hadamard. In particular, McKernel offers considerable improvement over Spiral, due to automatic code generation, the use of SIMD intrinsics (SSE2 using 128 bit registers) and loop unrolling. This decreases the memory overhead.

McKernel proceeds with vectorized sums and subtractions iteratively for the first $\frac{n}{2^z}$ input vector positions (where n is the length of the input vector and z the iteration starting from 1), computing the intermediate operations of the COOLEY-TUKEY algorithm till a small routine Hadamard that fits in cache. Then the algorithm continues in the same way but starting from the smallest length and doubling on each iteration the input dimension until the whole Fast Walsh Hadamard is done in-place.

For instance, on an intel i5-4200 CPU @ 1.60GHz laptop the performance obtained can be observed in Figure 2.

Table 1. Numeric Comparison of Fast Walsh Hadamard.

$\|H_n\|$	McKernel $t(ms)$	Spiral $t(ms)$
1024	0	0.03
2048	0.03	0.07
4096	0.1	0.17
8192	0.07	0.2
16384	0.2	0.47
32768	0.2	0.9
65536	0.7	1.67
131072	1.3	3.5
262144	3.6	7.67
524288	7.86	15.97
1048576	15.97	35.7

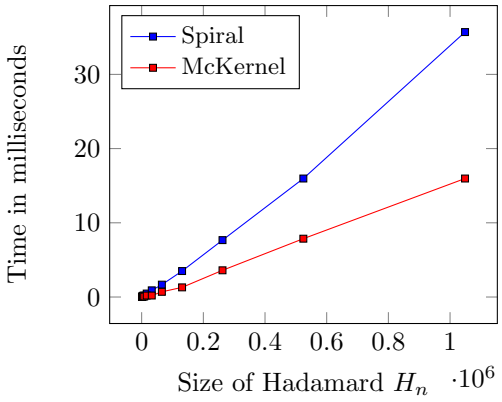

Figure 2. **Comparison of Fast Walsh Hadamard**. McKernel (red) outperforms Spiral (blue) across the range of arguments.

Our code outperforms Spiral consistently throughout the range of arguments, see Table 1. Furthermore, Spiral needs to precompute trees and by default can only perform the computation up to matrix size $n = 2^{20}$. On the other hand, our implementation works for any given size since it computes the high-level partitioning dynamically.

6 API Description

The API follows the design pattern in factory. That is, while the object McKernel is fairly generic in terms of computation, we have a factory that acts as a means of instantiating the parameters according to pre-specified sets of parameters, e.g. a RBF Kernel or a RBF MATÉRN Kernel. The so-chosen parameters are deterministic, given by the values of a function of hashing. The advantage of this approach is that there is no need to save the coefficients generated for McKernel when deploying the functions.

McKernel is integrated into a fully-fledged C++ DL framework that lets the user experiment, among other things, with dropout, convolutions, different activation functions, layer normalization, maxpooling, L1 and L2 regularization, gradient clipping, autoencoders, residual blocks, SGD optimization with momentum and dataset loading. That said, it also includes some classical algorithms for learning such as linear and logistic regression.

6.1 Customizing McKernel

For instance, to generate each C_{kk} entry for RBF MATÉRN Kernel we draw t i.i.d. samples from the n-dimensional unit ball S_n, add them and compute its Euclidean norm.

To draw efficiently samples from S_n we use the algorithm provided below.

Let $X = (X_1, \ldots, X_n)$ be a vector of i.i.d. random variables drawn from $N(0, 1)$, and let $||X||$ be the Euclidean norm of X, then $Y = \left(\frac{X_1}{||X||}, \ldots, \frac{X_n}{||X||}\right)$ is uniformly distributed over the n-sphere, where Y is the projection of X onto the surface of the n-dimensional sphere. To draw uniform random variables in the n-ball, we multiply Y by $U^{1/n}$ where $U \sim U(0, 1)$. This can be proved as follows: Let $Z = (Z_1, \ldots, Z_n)$ be a random vector uniformly distributed in the unit ball. Then, the radius $R = ||Z||$ satisfies $P(R \leq r) = r^n$. Now, by the inverse transform method we get $R = U^{1/n}$. Therefore to sample uniformly from the n-ball the following algorithm is used:

$$Z = rU^{1/n} \frac{X}{||X||}. \tag{14}$$

7 Large-scale Machine Learning

We introduce McKernel as an alternative to Deep Learning, where we argue that current techniques of Neural Networks are surrogates to very large kernel expansions, where optimization is done in a huge parameter space where the majority of learned weights are trivial to the actual problem statement.

We present here the idea that current developments in very deep neural networks can be achieved while drastically reducing the number of parameters learned. We build on the work in [Rahimi and Recht 2007] and [Le et al. 2013] to expand its scope to mini-batch training with SGD Optimizer. Our concern is to demonstrate that we are able to get the same gains obtained in Deep Learning by the use of McKernel and a linear classifier but with a behemoth kernel expansion.

Current research in Neural Networks is over-optimizing parameters that are indeed not informative for the problem to solve. That is, we pioneer the notion that Deep Learning is learning the inner parameters of a very large kernel expansion and, in the end, is doing a brute-force search of the appropriate kernel k thats fits well the data.

Observe that McKernel generates pseudo-random numbers by the use of hashing which is key for large-scale data. It allows to obtain a deterministic behavior and at the same time to load the weights on both training and testing without the need to actually store the matrices. As a further matter, it is crucial for distributed computation.

Notice here that the fact that we can increase the number of kernel expansions building highly hierarchical networks, see Equations 8 and 9, gives the property of compositionality to McKernel. Namely, the theoretical guarantee to avoid the curse of dimensionality [Mhaskar et al. 2017; Poggio et al. 2017].

We can behold that McKernel is corresponding to networks of the form

$$G(x) = \sum_{k=1}^{N} a_k \exp\left(-|x - x_k|^2\right), \qquad x \in \mathbb{R}^d. \tag{15}$$

In the following section we build on these ideas to propose some very simple examples to illustrate the essence of the problem and the solution proposed.

8 TIKHONOV Regularization

Let H be the hypothesis space of functions, in the problem of Empirical Risk Minimization (ERM) we want to find $f \in H$ that minimizes

$$\frac{1}{n}\sum_{c=1}^{n}(f(x_c) - y_c)^2. \tag{16}$$

This problem is in general ill-posed, depending on the choice of H. Following Tikhonov [Girosi 1998; Girosi et al. 1995] we minimize instead over the hypothesis space H_K, the regularized functional

$$\frac{1}{n}\sum_{c=1}^{n}(f(x_c) - y_c)^2 + \lambda||f||_K^2, \tag{17}$$

where $||f||_K^2$ is the norm in H_K - the REPRODUCING KERNEL HILBERT Space defined by the kernel K.

In general, under the TIKHONOV regularization scheme that follows,

$$\min_{w\in\mathbb{R}^D} \hat{E}(f_w) + \lambda||w||^2, \tag{18}$$

where $||w||^2$ is the regularizer and controls the stability of the solution and λ balances the error term and the regularizer. Different classes of methods are determined by the appropriate choice of loss function in Equation 18. Here we consider

$$\hat{E}(f_w) = \frac{1}{n}\sum_{c=1}^{n} l(y_c, f_w(x_c)) \tag{19}$$

with loss function l defined as

$$l(y, f_w(x)) = \log(1 + \exp(-y f_w(x))), \tag{20}$$

videlicet Logistic Regression.

Considering the logistic loss is differentiable and that we are in a large-scale setting a reasonable candidate to compute a minimizer is the Stochastic Gradient Descent (SGD),

$$w_{t+1} = w_t - \gamma \Delta g_{c_t}(w_t), \tag{21}$$

where c_t denotes a stochastic sequence of indices and γ is the learning rate.

In this line of argument, [Kawaguchi and Kaelbling 2019; Liang et al. 2018; Sohl-Dickstein and Kawaguchi 2019] state that local minimization is well posed in Deep Learning using SGD.

Augmenting the number of kernel expansions, and thus the representational power of the model, gives a degree of over-parametrization. That is to say, we increase the size of the network to fit the training data. Given these constraints, BÉZOUT theorem argues the existence of a large number of degenerate global minimizers with zero empirical error, that are very likely to be found by SGD that will in addition select with higher probability the most robust zero-minimizer [Poggio and Liao 2017].

9 Empirical Analysis and Experiments

We generalize the use of McKernel in mini-batch with SGD Optimizer, Figure 1, drastically reducing the number of parameters that need to be learned to achieve comparable state-of-the-art results to Deep Learning.

What is more, current breakthroughs in Neural Networks can be easily derived from Equation 8. Say for instance, Batch Normalization [Ioffe and Szegedy 2015] can be obtained from the normalizing factor. Or for example, the use of ensembles [Lakshminarayanan et al. 2017] and multi-branch architectures [Zhang et al. 2018] to improve performance can be seen on Figure 1, as it follows from the increase on Kernel Expansions. Not only that, but increasing the number of Kernel Expansions has another great property; data augmentation [Cubuk et al. 2019; Tran et al. 2017], which has recently seen a lot of interest. Its importance follows directly from the construction of McKernel, take the data, apply slightly (randomized) different functions to it to create new high-dimensional samples that will aid the process of learning. It also explains the need for backpropagation [Lecun et al. 1998]: certain types of data will work better for certain kernels, so it may be necessary to learn the appropriate Calibration C and G that fit well the data. Besides, learning B acts as mechanism of attention [Bahdanau et al. 2015; Luong et al. 2015; Vaswani et al. 2017]. Further, dropout [Srivastava et al. 2014] follows directly from the use of the Subsampled Randomized Hadamard. Additionally, research on finding alternate activation functions [Clevert et al. 2016; He et al. 2015; Maas et al. 2013; **?**] can be deduced from looking for different mappings in Equation 9.

Note here that the number of parameters to be estimated is of the order of thousands, proportional to the number of classes (depending on the size of the input image and the number of kernel expansions),

$$C \cdot (2 \cdot [S]_2 \cdot E + 1), \tag{22}$$

where C is the number of classes, $[\cdot]_2$ is an operator that returns the next power of 2, S is the size of the input samples and E is the number of Kernel Expansions. A drastic reduction compared to Neural Networks, while achieving comparable performance. Training time is therefore severely reduced and SVM kernel like learning can be achieved at scale.

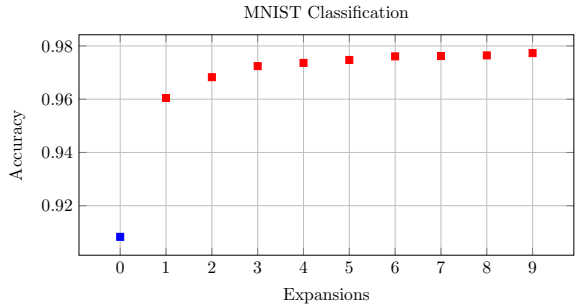

Figure 3. **MNIST Classification.** Logistic Regression (blue) and RBF MATÉRN (red) with increasing number of Kernel Expansions. 32768 samples of training data and 8192 samples of testing data are used in learning. RBF MATÉRN hyper-parameters, $\sigma = 1.0$, $t = 40$. Seed 1398239763, learning rate $\gamma = 0.001$ and batch size 10. LR learning rate 0.01. Number of epochs 20.

SGD Optimizer finds W and b in

$$\text{softmax}\left(W[\phi(\hat{Z}\hat{x})] + b\right), \tag{23}$$

where $\phi = (\sin(\cdot), \cos(\cdot))$, $\hat{x} = [x]_2$. Namely, it minimizes the loss l in Equation 20.

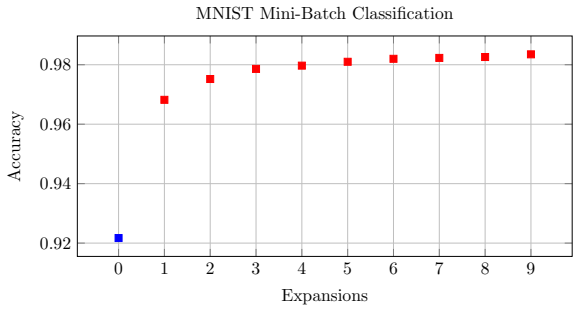

Figure 4. **MNIST Mini-Batch Classification.** Logistic Regression (LR) (blue) and RBF MATÉRN (red) with increasing number of Kernel Expansions. 60000 samples of training data and 10000 samples of testing data are used in learning. RBF MATÉRN hyper-parameters, $\sigma = 1.0$, $t = 40$. Seed 1398239763, learning rate $\gamma = 0.001$ and batch size 10. LR learning rate 0.01. Number of epochs 20.

Figures 3 and 4 show RBF MATÉRN, $\text{softmax}(W\tilde{x} + b)$ where $\tilde{x} = \text{mckernel}(x)$, performance compared to logistic regression, $\text{softmax}(Wx + b)$, in full-batch and mini-batch on MNIST, respectively. A fixed seed is used to obtain deterministic reproducible behavior. In full-batch, the number of samples for train and test is rounded to the nearest power of 2 due to algorithm constraint.

The same kind of intuition that applies to Neural Networks, where the deeper the network, the better the results, holds. But this time depending on the number of kernel expansions used.

FASHION MNIST [Xiao et al. 2017] is similar in scope to MNIST but relatively more difficult. Instead of classifying digits, we focus now on the task of fashion. It consists on ten classes of clothing; T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot.

Figure 5. **FASHION MNIST Mini-Batch Classification.** Logistic Regression (LR) (blue) and RBF MATÉRN (red) with increasing number of Kernel Expansions. 60000 samples of training data and 10000 samples of testing data are used in learning. RBF MATÉRN hyper-parameters, $\sigma = 1.0$, $t = 40$. Seed 1398239763, learning rate $\gamma = 0.001$ and batch size 10. LR learning rate 0.01. Number of epochs 20.

Figure 5 shows RBF MATÉRN performance compared to logistic regression in mini-batch. Comparable state-of-the-art performance to Deep Learning is achieved. The model presents a similar behavior to the one seen in MNIST dataset. McKernel performs analogously to modern techniques in Neural Networks in this highly non-linear problem of estimation.

10 Discussion

In this manuscript we provide a new framework of learning and illustrate with two examples that achieves comparable state-of-the-art performance to Neural Networks, proposing a new way to understand Deep Learning, as a huge kernel expansion where optimization is only performed over the parameters that are actually relevant to the problem at-hand. At the same time, a new methodology to build highly compositional networks for Large-scale Machine Learning is introduced.

We account for both the theoretical underpinnings and the practical implications to establish the building blocks of a unifying theory of learning.

References

M. Al-Shedivat, A. G. Wilson, Y. Saatchi, Z. Hu, and E. P. Xing. 2017. Learning Scalable Deep Kernels with Recurrent Structure. *JMLR* 18 (2017).
A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. 2015. Practical and Optimal LSH for Angular Distance. *NIPS* 1 (2015).
D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR* (2015).
G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or. 2007. The Gray-code Filter Kernels. *IEEE Transactions (TPAMI)* 29 (2007).
G. E. P. Box and M. E. Muller. 1958. A Note on the Generation of Random Normal Deviates. *The Annals of Mathematical Statistics* 29 (1958).
Y. Cho and L. Saul. 2009. Kernel Methods for Deep Learning. *NIPS* 22 (2009).
K. Chwialkowski, A. Ramdas, D. Sejdinovic, and A. Gretton. 2015. Fast Two-sample Testing with Analytic Representations of Probability Measures. *NIPS* 2 (2015).
D. Clevert, T. Unterthiner, and S. Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ICLR* (2016).
C. Cortes and V. Vapnik. 1995. Support-vector Networks. *Machine Language* 20 (1995).

E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le. 2019. AutoAugment: Learning Augmentation Strategies from Data. *IEEE Conference on Computer Vision and Pattern Recognition* (2019).
F. Cucker and S. Smale. 2001. On the Mathematical Foundations of Learning. *Bulletin of the AMERICAN Mathematical Society* 39 (2001).
F. Girosi. 1998. An equivalence between sparse approximation and Support Vector Machines. *Neural Computation* (1998).
F. Girosi, M. Jones, and T. Poggio. 1995. Regularization theory and neural networks architectures. *Neural Computation* (1995).
K. He, X. Zhang, S. Ren, and J. Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-level Performance on ImageNet Classification. *IEEE International Conference on Computer Vision* (2015).
W. Hong, J. Yuan, and S. Bhattacharjee. 2017. Fried Binary Embedding for High-dimensional Visual Features. *IEEE Conference on Computer Vision and Pattern Recognition* (2017).
S. Ioffe and C. Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML* 37 (2015).
J. Johnson and M. Püschel. 2000. In Search of the Optimal WALSH-HADAMARD Transform. *IEEE International Conference on Acoustics, Speech, and Signal Processing* (2000).
K. Kawaguchi and L. P. Kaelbling. 2019. Elimination of All Bad Local Minima in Deep Learning. *arXiv:1901.00279* (2019).
K. Kawaguchi, B. Xie, V. Verma, and L. Song. 2018. Deep Semi-random Features for Nonlinear Function Approximation. *AAAI Conference on Artificial Intelligence* (2018).
B. Lakshminarayanan, A. Pritzel, and C. Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles. *NIPS* (2017).
Q. Le, T. Sarlós, and Alexander Smola. 2013. Approximating Kernel Expansions in Loglinear Time. *ICML* (2013).
Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based Learning Applied to Document Recognition. *Proceedings of the Institute of Electrical and Electronics Engineers* 86 (1998).
W. Li, D. Dai, M. Tan, D. Xu, and L. Gool. 2016. Fast Algorithms for Linear and KERNEL SVM+. *IEEE Conference on Computer Vision and Pattern Recognition* (2016).
M. Liang, B. Yang, S. Wang, and R. Urtasun. 2018. Deep Continuous Fusion for Multi-sensor 3D Object Detection. *EUROPEAN Conference on Computer Vision* (2018).
Y. Lu, P. S. Dhillon, D. Foster, and L. Ungar. 2013. Faster Ridge Regression via the Subsampled Randomized HADAMARD Transform. *NIPS* 1 (2013).
M. Luong, H. Pham, and C. D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2015).
A. L. Maas, A. Y. Hannun, and A. Y. Ng. 2013. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *ICML* 30 (2013).
H. Mhaskar, Q. Liao, and T. Poggio. 2017. When and Why Are Deep Networks Better than Shallow Ones? *AAAI* (2017).
M. Moczulski, M. Denil, J. Appleyard, and N. Freitas. 2016. ACDC: a Structured Efficient Linear Layer. *ICLR* (2016).
W. Ouyang and W. K. Cham. 2010. Fast Algorithm for WALSH HADAMARD Transform on Sliding Windows. *IEEE Transactions (TPAMI)* 32 (2010).
T. Poggio and Q. Liao. 2017. Theory II: Landscape of the Empirical Risk in Deep Learning. *arXiv:1703.09833* (2017).
T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. 2017. Why and When Can Deep – but Not Shallow – Networks Avoid the Curse of Dimensionality: a Review. *International Journal of Automation and Computing* 14 (2017).
T. Poggio and S. Smale. 2003. The Mathematics of Learning: Dealing with Data. *Notices of the AMERICAN Mathematical Society* (2003).
A. Rahimi and B. Recht. 2007. Random Features for Large-scale Kernel Machines. *NIPS* 20 (2007).
S. Reddi, A. Ramdas, B. Póczos, A. Singh, and L. Wasserman. 2015. On the High-dimensional Power of a Linear-time Two Sample Test under Mean-shift Alternatives. *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2015).
A. Rudi and L. Rosasco. 2017. Generalization Properties of Learning with Random Features. *NIPS* 30 (2017).
V. Sharmanska, N. Quadrianto, and C. H. Lampert. 2013. Learning to Rank Using Privileged Information. *IEEE International Conference on Computer Vision* (2013).
J. Sohl-Dickstein and K. Kawaguchi. 2019. Eliminating All Bad Local Minima from Loss Landscapes without Even Adding an Extra Unit. *arXiv:1901.03909* (2019).
N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. Dropout: a Simple Way to Prevent Neural Networks from Overfitting. *JMLR* 15 (2014).
T. Tran, T. Pham, G. Carneiro, L. Palmer, and I. Reid. 2017. A Bayesian Data Augmentation Approach for Learning Deep Models. *NIPS* (2017).
V. Vapnik and R. Izmailov. 2015. Learning Using Privileged Information: Similarity Control and Knowledge Transfer. *JMLR* 16 (2015).
V. Vapnik and R. Izmailov. 2018. Rethinking Statistical Learning Theory: Learning Using Statistical Invariants. *Machine Learning* (2018).
V. Vapnik and A. Vashist. 2009. A New Learning Paradigm: Learning Using Privileged Information. *Neural Networks* 22 (2009).
A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. Attention Is All You Need. *NIPS* (2017).
M. Wang, Y. Liu, and Z. Huang. 2017. Large Margin Object Tracking with Circulant Feature Maps. *IEEE Conference on Computer Vision and Pattern Recognition* (2017).
Z. Wang and Q. Ji. 2015. Classifier Learning with Hidden Information. *IEEE Conference on Computer Vision and Pattern Recognition* (2015).
E. Wigner. 1960. The Unreasonable Effectiveness of Mathematics in the Natural Sciences. *Communications in Pure and Applied Mathematics* 13 (1960).
A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. 2016. Stochastic Variational Deep Kernel Learning. *NIPS* (2016).
H. Xiao, K. Rasul, and R. Vollgraf. 2017. FASHION-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747* (2017).
Z. Yang, M. Moczulski, M. Denil, N. Freitas, Alexander Smola, L. Song, and Z. Wang. 2015. Deep Fried Convnets. *IEEE International Conference on Computer Vision* (2015).
Z. Yang, Alexander Smola, L. Song, and A. G. Wilson. 2014. Learning Fast Kernels. *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2014).
H. Zhang, J. Shao, and R. Salakhutdinov. 2018. Deep Neural Networks with Multi-branch Architectures Are Less Non-convex. *arXiv:1806.01845* (2018).