Virtual Social Event

Open source tools and practices in state-of-the-art DL research

ICLR

neptune.ai/landings/iclr-virtual-social

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.

github.com/curto2/mckernel/

arxiv.org/pdf/1702.08159

Open source tools and practices in state-of-the-art DL research

ICLR

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.

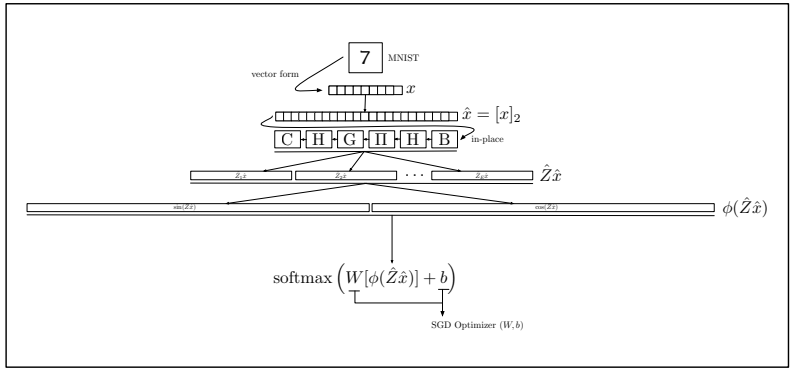


Figure 1. **Diagram of McKernel**. We visually describe softmax$(W\tilde{x} + b)$ where $\tilde{x}$ = mckernel(x). The original image is padded in form of long vector to the nearest power of 2, mapping $\hat{Z}$ is applied in-place. Calibration C defines the choice of Kernel. The tensor is expanded by 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

arxiv.org/pdf/1702.08159

decurto.tw

J. de Curtó i Díaz

I've had numerous research appointments, namely at the Department of Computer Science and Engineering at CUHK and at Carnegie Mellon. As well as at the EE and CS Departments at City University of Hong Kong.

I was a doctoral student in the Laboratory of Computer Vision at ETH Zürich. Previously I completed my master (with distinction) in Electrical Engineering at City University of Hong Kong, where I did an exchange at the School of Computer Science at Carnegie Mellon. I developed my master thesis at the ML Dept. and as part of the MS program, I also interned at the Robotics. At City University of Hong Kong, I received several awards: the Top Achiever 2015, MS Internship Sponsorship 2014 and MS Entrance Scholarship 2013/14.

I hold a 5-year degree in Engineering of Telecommunication from Universitat Autònoma de Barcelona and Universitat Politècnica de Catalunya. I also worked as Research Scientist at CELLS ALBA Synchrotron. I had near perfect top nationwide scores in the university entrance examinations and baccalaureate.

e-mail / cv / twitter

dezarza.tw

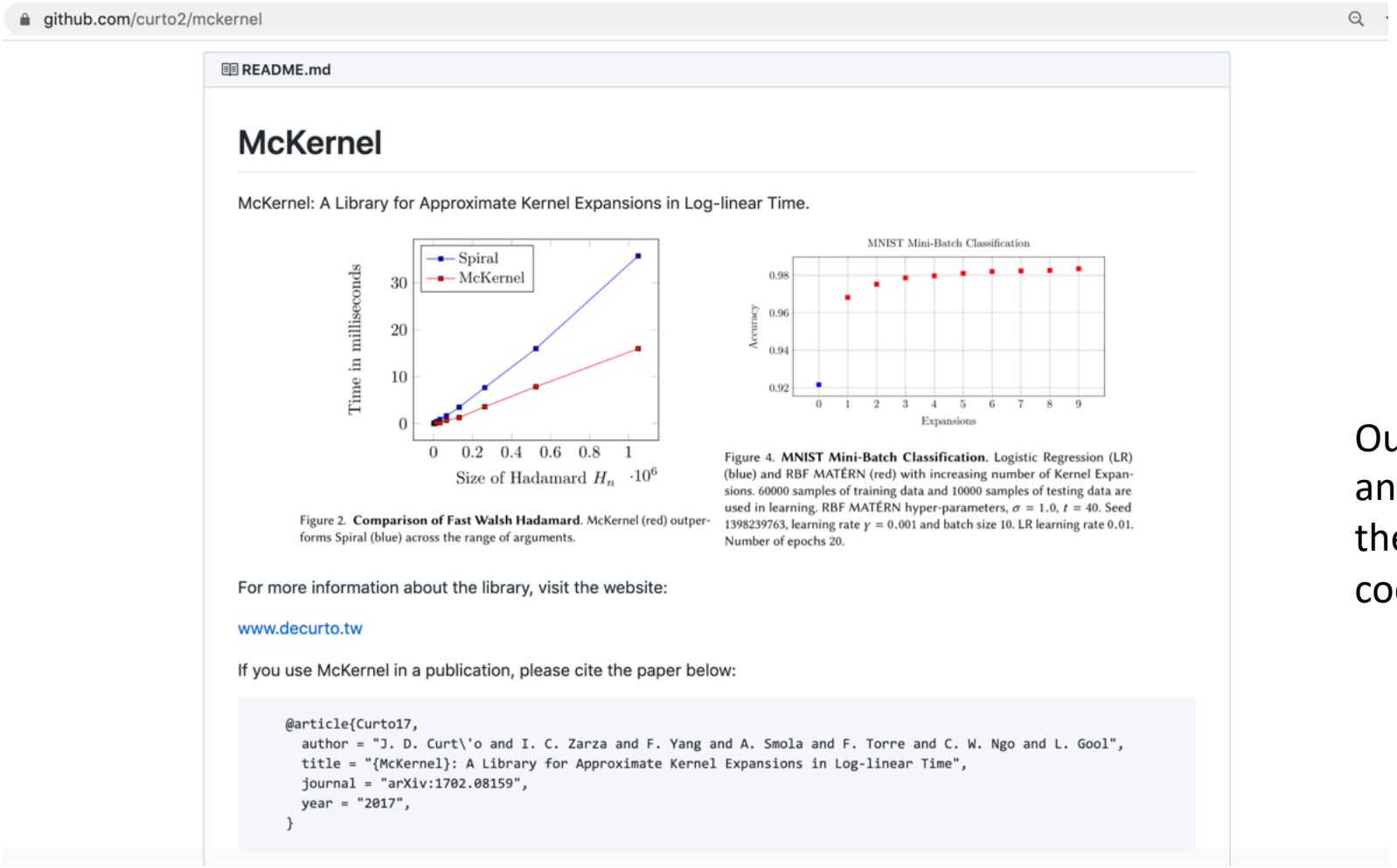
I. de Zarza i Cubero

I've had several research appointments, scilicet at the Department of Computer Science and Engineering at CUHK and at Carnegie Mellon. Forbye at the EE and CS Departments at City University of Hong Kong.

I was a doctoral student in the Laboratory of Computer Vision at ETH Zürich. Previously I completed my master (with distinction) in Electrical Engineering at City University of Hong Kong, where I did an exchange at the School of Computer Science at Carnegie Mellon. I developed my master thesis at the ML Dept. and as part of the MS program, I also interned at the Robotics.

I hold a degree in Mathematics from Universitat Autònoma de Barcelona and Universitat de Barcelona. I had perfect scores in the baccalaureate and top nationwide grades in the university entrance examinations.

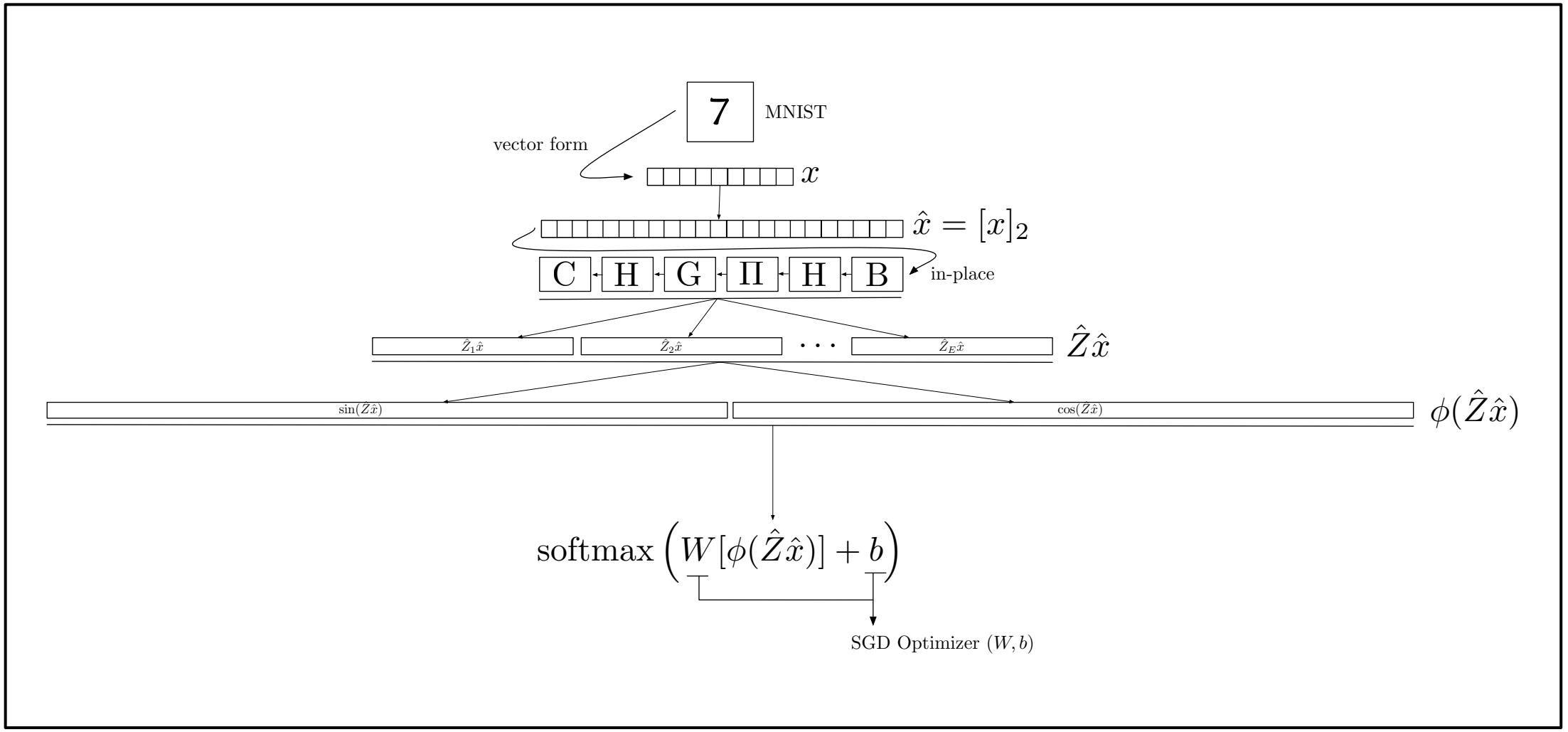
e-mail / cv / twitter



github.com/curto2/mckernel/

Our goal was to build a clear, concise and very well written code. Inspired by the principles of MapReduce: 50 lines of code and reusability at its core.

The big picture:



The core ingredient:

First use of McKernel: as a lighting-fast open-source Hadamard.

$$C \leftarrow H \leftarrow G \leftarrow \Pi \leftarrow H \leftarrow B$$

$$H$$

Works for any given input size!

Useful in any application where FOURIER Transform is applicable, such as compression (MPEG-4 AVC), encryption or quantum computing (Grover, Shor).

If you want to use FWH, add #include "hpp/McKernel.hpp" in your test file.

Kernel methods at the speed of light:

Second use: approximate kernel expansions in log-linear time.

$$\phi(\hat{Z}\hat{x})$$

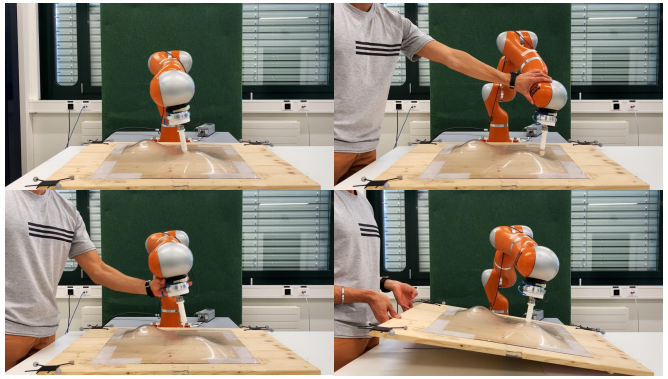


Image from Amanhoud, Khoramshahi, and Billard.
A Dynamical System Approach to Motion and Force Generation in Contact Tasks. RSS 2019.

Useful in large-scale setting to turn any linear classifier into non-linear.

Applications: wherever SVM is still useful over DL methods. For instance, in robotics and ML for healthcare when the number of samples to train on is relatively bounded.

A whole new perspective:

Third use: as a stepping stone to integrate DL methods and kernel expansions.

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

$$\text{SGD Optimizer } (W, b)$$

Useful to foster new DL architectures with better human-induced/mathematical priors.

Applications: Doctor of Crosswise: Reducing Over-parametrization in Neural Networks

J. D. Curtó *,1,2,3,4, I. C. Zarza *,1,2,3,4, K. Kitani[2], I. King[1], and M. R. Lyu[1].
[1]The Chinese University of Hong Kong. [2]Carnegie Mellon.
[3]Eidgenössische Technische Hochschule Zürich. [4]City University of Hong Kong.
*Both authors contributed equally.

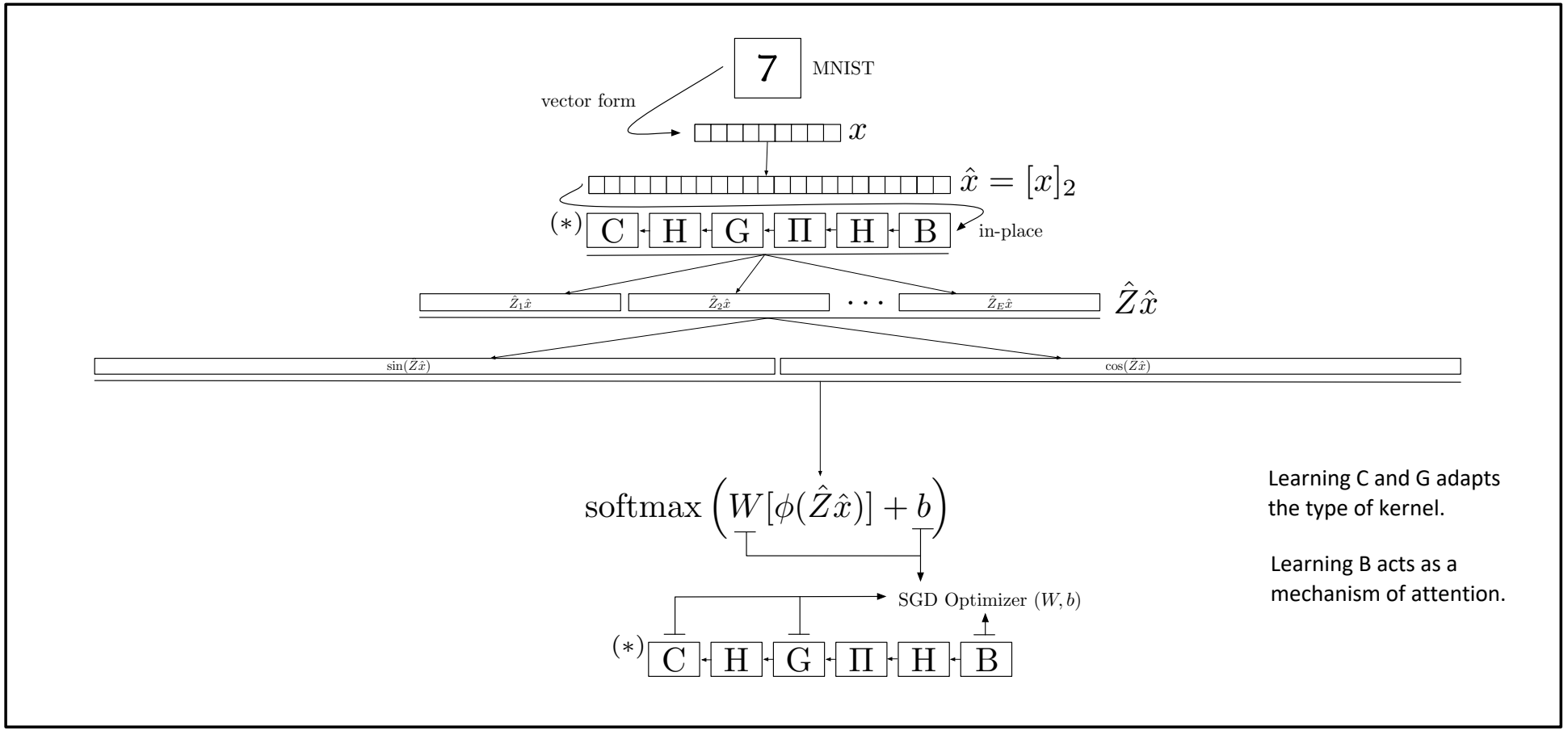
github.com/curto2/dr_of_crosswise
arxiv.org/pdf/1905.10324

We'd love to see:

- End-to-end training.
- Self-supervised Learning.
- Meta-learning.
- Integration with Evolution Strategies.
- NAS reducing substantially the search space.

...

Fourth use: as a DL research framework. It offers multiple open questions:



Learning C and G adapts the type of kernel.

Learning B acts as a mechanism of attention.

github.com/curto2/mckernel/

- Standard (mckernel/standard).
  - Library McKernel.
- Standard+ (mckernel/sdd+).
  - Library McKernel. Pseudo-random numbers generated with functions of hashing. Suitable for distributed applications. Recommended.
- Learning (mckernel/lg).
  - DL framework to reproduce experiments in the paper.

You'll like these features:

- No dependencies.
- Everything in C++.
- Runs on CPU using SIMD.
- MIT License.

Session of Meta-coding (a super-simple hands-on introduction):

```
$ git clone https://github.com/curto2/mckernel
$ cd mckernel
$ mkdir data && cd data
$ mkdir fashion_mnist && cd fashion_mnist
$ wget http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
$ wget http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
$ wget http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
$ wget http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
$ gunzip *.gz
$ cd ../../lg/examples/fashion_mnist
$ make
$ ./FASHION_MNIST_Classification
```

mckernel/lg/examples/fashion_mnist/FASHION_MNIST_Classification.cpp

Hyper-parameter tuning:

- # of kernel expansions.
- Parameters of RBF MATÉRN (t and sigma).
- # epochs.
- Learning rate.

…

$ gedit FASHION_MNIST_Classification.cpp

```
int main(int argc, const char *argv[])
{
        printf(R"EOF(


)EOF");

        //Load training set
        lg::Tensor_float trainingset, training_targets, testingset, testing_targets;
        lg::MNIST_Load_Binary(
                "../../../data/fashion_mnist/train-images-idx3-ubyte",
                "../../../data/fashion_mnist/t10k-images-idx3-ubyte",
                "../../../data/fashion_mnist/train-labels-idx1-ubyte",
                "../../../data/fashion_mnist/t10k-labels-idx1-ubyte",
                trainingset, training_targets, testingset, testing_targets);

        printf("# Samples Test: %d.\n", testingset.height());
        printf("Dimension Features: %d.\n", testingset.width());
        printf("# Samples Train: %d.\n", trainingset.height());
        printf("Dimension Features: %d.\n\n", trainingset.width());

        //Kernel Expansions
        int expansions = 3;

        //RBF MATÉRN
        unsigned long t = 40;
        float sigma = 1.0;
```

Always ensure same seed for training and testing!

mckernel/lg/examples/fashion_mnist/FASHION_MNIST_Classification.cpp

It provides a fully-fledged DL framework:

- Access to standard layers (BatchNorm, ReLU,…).
- SGD Optimization in mini-batches.
- Easy integration with kernel expansions (RBF Kernel, RBF MATÉRN).

…

$ gedit FASHION_MNIST_Classification.cpp

```
            //Initialize variables
            unsigned long nv = input.height();
            unsigned long dn = input.width();
            unsigned long D = expansions * dn;

            unsigned long tnv = tinput.height();
            unsigned long tdn = tinput.width();
            unsigned long tD = expansions * tdn;

            //Initialize McKernel
            McKernel* mckernel = FactoryMcKernel::createMcKernel(FactoryMcKernel::MRBF, input, nv, dn, D, seed, sigma, t);
            McKernel* tmckernel = FactoryMcKernel::createMcKernel(FactoryMcKernel::MRBF, tinput, tnv, tdn, tD, seed, sigma,
;


            //Initialize variables
            lg::Tensor_float trainingset_McKernel(2 * mckernel->M_dn_D, input.height());
            lg::Tensor_float testingset_McKernel(2 * tmckernel->M_dn_D, tinput.height());
            trainingset_McKernel.fill(0);
            testingset_McKernel.fill(0);

            //Linear Classifier (1-layer Neural Network)
            lg::Neural_Network nn;
            nn.push("INPUT",                "",                             lg::Variable::make(2 * tmckernel->M_dn_D));
            nn.push("LINEAR",               "INPUT",                        lg::Linear::make(10));
            nn.push("OUTPUT",               "LINEAR",                       lg::Sigmoid::make());

            //Print Network
            nn.printstack();

            for (int c = 0; c <= cycles; c++) {

                    //Update learning rate
                    sgd.setLearningrate(sgd.getLearningrate() - baselr / (double)cycles);

                    //Optimize neural network with random sample
                    int random_sample_id = rand() % samples;
```

Courtesy of https://amturing.acm.org/

”Greatness is a matter of style” R. W. Hamming.

Thank you!

Q&A (30 April 2020).

1) I was asked to provide some references to grasp the fundamentals of the construction, I highlighted these three works:

- The building block is Rahimi and Recht. NIPS 2007.
- The theoretical framework is provided in Le et al. ICML 2013. It describes in detail use 2.
- A follow-up work that gives a flavor of use 3 is Moczulski et al. ICLR 2016. Here they are using the FOURIER Transform instead of the Hadamard.

I also called attention to some first order methods like the Neural Tangent Kernel (NIPS 2018).

2) I was also asked about the # of users and acceptance of the library. I explained that we were growing in popularity and that we wanted McKernel to be the same that Theano or Caffe were in the early days of deep learning but for these new wave of learning representations. As well as a new way to come up with new architectures of neural networks that integrate properties of kernel methods.

3) I also noted the key takeaway from the talk: imagine the first law of Newton, and then think that we live in a world where we have an extremely good way to approximate gravity, but we don't know the real equation that describes the relationship between the variables. Then making progress in science would be severely difficult (up to a point), as we would never fully grasp the equation that lies behind the approximation. Imagine that the same is happening in DL and that using equation from slide 7, properties of neural networks can be derived by interpreting them as kernel methods.