Implementation of an Extreme Learning Machine optimized through Sthocastic Gradient Descendent (ADAM) and Cholesky factorization

Jacopo Bandoni
Luca Strazzera

j.bandoni@studenti.unipi.it
l.strazzera@studenti.unipi.it

Computational Mathematics for Learning and Data Analysis, A.Y. 2020-2021

Type of project: ML-17 project

Abstract

(M) Is a so-called extreme learning, i.e., a neural network with one hidden layer, $y = W_2\sigma(W_1x)$, where the weight matrix for the hidden layer W_1 is a fixed random matrix, $\sigma()$ is an elementwise activation function of your choice, and the output weight matrix W_2 is chosen by solving a linear least-squares problem (with L_2 regularization).

(A1) is an algorithm of the class of accelerated gradient methods applied to (M1).

(A2) is a closed-form solution with the normal equations and your own implementation of Cholesky (or LDL) factorization.

No off-the-shelf solvers allowed.

1 Introduction

1.1 Extreme Learning Machine

Let M be a Feed-Forward Neural Network with single hidden layer. Then the output of our network for a single pattern x_j is:

$$y_j = \sigma(x_j W_1 + b)W_2$$

where W_1 is a fixed random matrix, called weight matrix, connecting the input to the hidden layer, $\sigma(.)$ is an elementwise activation function that will be explained in next subsections and W_2 is another fixed random matrix, called weight matrix, connecting the hidden layer to the output layer, adjusted solving a linear least-squares problem.

Then for $\hat{n}$ hidden units and n arbitrary distinct samples (x_i, t_i) where $x_i = [x_{i_1}, ..., x_{i_v}] \in R^v$ and $t_i = [t_{i_1}, ..., t_{i_o}] \in R^o$ the M defined network can approximate these n samples with zero error $\sum_{j=1}^{n} \parallel y_j - t_j \parallel = 0$ if there exist W_1, W_2 and a bias for each neuron b_i such that:

$$\sigma(x_i W_1 + b)W_2 = t_i \qquad \forall i = 1, ..., n$$

or alternatively:

$$h(x_i) = \sigma(x_i W_1 + b)w_{2j} = t_{i_j} \qquad \forall i = 1, ..., n \ \ \& \ \ \forall j = 1, ..., o$$

Where w_{2j} represent the j-th column of the W_2 matrix. For the sake of simplicity in the next section we will analyze the functions of the network as if there was a single output, taking the form of the above equation without the j index.

1.2 Activation Function

The network implements the following activation functions:

- Sigmoid function:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$
$$f'(x) = \sigma'(x) = f(x)(1 - f(x))$$

- TanH:

$$f(x) = \sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$f'(x) = \sigma'(x) = 1 - f(x)^2$$

Both of them respect the following property:

- nonlinear;
- Finite range;
- Continuously differentiable;
- Monotonic;
- Smooth;
- Approximate identity near the origin;

1.3 Notation

For any vector $a \in \mathbb{R}^d$, we use $\sqrt{a}$ for element-wise square root and a^2 for element-wise square.

1.4 Solver

As solvers we will use two approach. *A1* is based on the class of adaptive accelerated gradient methods:

- Adam: is an optimization algorithm running averages of both the gradients and the second moments of the gradient, it's so defined as

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2}$$

  Where m_t is an estimation of the first moment of the gradient (mean) and v_t is an estimation of the second moment of the gradient (uncentered variance). Usually we have the following constant values: β_1 is $= 0.9$, β_2 is 0.999 and $\epsilon = 10^{-8}$ proposed by authors. Being biases toward zero at the start and when the decay rates β_1 and β_2 are small we will follow the paper applying a bias-correction rule:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4}$$

  Then they are finally used to update parameters:

$$w2_{t+1} = w2_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{5}$$

A2 is based on the class of closed-form solution with the normal equation:

- Cholesky: given a positive definite matrice we can rewrite it as

$$A = LDL^T = LD^{1/2}(D^{1/2^T} L^T) = CC^T \tag{6}$$

  Where $D^{1/2}$ is a diagonal matrix and C is lower triangular matrix.

1.5 Error Function

Mean Square Error (MSE) is the Error Function used to evaluate the performance of the model. It is obtained by the formula:

$$E(w_2) = MSE(w_2) = \frac{1}{n}\sum_{i=1}^{n}(h(x_i, w_2) - t_i)^2$$

Where n is the number of example, h_i is the output of our model for the i-th sample and t_i is the real target of our data for the i-th sample.

1.6 Regularization

The regularization term is used to control the complexity of the model. Is implemented adding to the Error Function a penalty term:

$$w_2 \in R^{\hat{n}} \quad \lambda\omega(w_2) \tag{7}$$

The λ is the hyperparameter that allows to take a fraction of the term.
In our implementation the specific regularization used is *Ridge Regression* or also called *Thikhonov regularization*. We are penalizing term with higher values. It has a form composed by the use of the norm two L_2:

$$\omega(w_2) = \sum_{i=1}^{\hat{n}} {w_{2_i}}^2 = \|w_2\|_2^2 \tag{8}$$

1.7 Loss function

The loss function is defined as the error function plus the regularization term:

$$L(w_2) = E(w_2) + \lambda\|w_2\|_2^2 \tag{9}$$

and generalizing to a neural network with multiple outputs it can be represented as:

$$L(W_2) = \sum_{j=1}^{o} E(w_{2_j}) + \lambda\|w_{2_j}\|_2^2 = \frac{1}{n}\sum_{j=1}^{o}\|Hw_{2_j} - t_j\|_2^2 + \lambda\|w_{2_j}\|_2^2 \tag{10}$$

where H, W_2 and T are the following matrices:

$$H = \begin{bmatrix} \sigma(x_1 w_{1_1} + b_1) & \dots & \sigma(x_1 w_{1_{\hat{n}}} + b_{\hat{n}}) \\ \vdots & \ddots & \vdots \\ \sigma(x_n w_{1_n} + b_1) & \dots & \sigma(x_n w_{1_{\hat{n}}} + b_{\hat{n}})) \end{bmatrix}_{n \times \hat{n}}$$

$$W_2 = \begin{bmatrix} | & & | \\ w_{2_1} & \dots & w_{2_o} \\ | & & | \end{bmatrix}_{\hat{n}\times o} \qquad T = \begin{bmatrix} | & & | \\ t_1 & \dots & t_o \\ | & & | \end{bmatrix}_{n\times o}$$

Note that we could also evaluate the model using o different loss functions, one for each output. As each output unit is independent of each other. In this case we would obtain:

$$\forall j = 1, ..., o \qquad L(w_{2_j}) = E(w_{2_j}) + \lambda \|w_{2_j}\|_2^2 = \frac{1}{n}\|Hw_{2_j} - t_j\|_2^2 + \lambda\|w_{2_j}\|_2^2 \quad (11)$$

1.7.1 Properties

To find an optimal value for the loss function it would be useful to evaluate the propriety of the function:

- **Differentiability**: the sum or composition of differentiable function is differentiable so the loss function is differentiable.
- **L-smoothness** The loss function L is *L-smooth* on its domain S if $\exists L > 0$ such that:
$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \ \forall x, y \in S,$$
Deriving the loss function for a single component of the vector we can observe that the loss function is a sum of Lipschitz continuous functions. This implies that the gradient is a Lipschitz continous function and so that the loss function is L-smooth.
- **Convex**: the square function is convex, the norm is convex and the composition and sum of convex function preserve convexity. This imply that the loss function is convex.

1.8 Problem

Finally, the problem is formalized as the least square problem:

$$min\frac{1}{n}\sum_{j=1}^{o}\|Hw_{2_j} - t_j\|_2^2 + \lambda\|w_{2_j}\|_2^2 \quad (12)$$

but it can also be represented as o different least square problems, one for each output unit:

$$\forall j = 1, ..., o \qquad min\frac{1}{n}\|Hw_{2_j} - t_j\|_2^2 + \lambda\|w_{2_j}\|_2^2 \quad (13)$$

and we will refer to $\hat{W}_2$ or $\hat{w_{2_j}}$ as the solutions to the previous problems.

2 Method

2.1 Cholesky

We would use this method to solve the several o regularized least square problems (11) and we will refer to w_j as a general column of the matrix W_2 which perform a linear combination of the values in the hidden units into a single output unit. So given the regularized least mean square problem:

$$min\frac{1}{n}\|Hw_j - t_j\|_2^2 + \lambda\|w_j\|^2$$

which is equivalent to

$$min\frac{1}{n}(Hw_j - t_j)^T(Hw_j - t_j) + \lambda w_j^{t_j} w_j$$

Can be solved computing the Jacobian of the function respect to w_j :

$$\frac{1}{n}\frac{\partial(Hw_j - t_j)^{t_j}(Hw_j - t_j)}{\partial w_j} = \frac{1}{n}2H^T(Hw_j - t_j); \qquad \frac{\partial\lambda w_j^T w_j}{\partial w_j} = 2\lambda w_j;$$

The minimum is then obtained putting the sum of the two partial derivatives to zero, so we get:

$$(H^T H + n\lambda I)w_j = H^T t_j \tag{14}$$

At this point we would like to know the value of w_j such that the equation above is valid. The matrix $H^T H$ is quadratic and positive semi definite while the matrix $n\lambda I$ is positive definite. So the sum of the two matrix is a positive definite matrix.

$$\forall x. \quad x^T(H^T H + n\lambda I)x = x^T H^T Hx + x^T n\lambda I x > 0$$

The matrix $H^T H$ is also symmetric and it keep being symmetric after the sum because $n\lambda I$ is diagonal. So $H^T H + n\lambda I$ is a symmetric positive definite matrix, it means we can always apply the Cholesky factorization such that:

$$H^T H + n\lambda I = CC^T$$

with C being lower triangular. We also need to do the multiplication in the second term: $H^T t_j = b_j$. So the equation (14) can be written as:

$$CC^T w_j = b_j \tag{15}$$

Here we solve via forward and then backwards substitution and we botain $\hat{w}_j$, we repeat for each $j \in \{1, ..., o\}$ and at the end we obtain $\hat{W}_2$.

2.1.1 Complexity

The complexity can be resumed into five operations:

- Multiplicating $H^T t_j \ \forall j \in \{1, ..., o\}$ with a total complexity of $O(\hat{n}no)$
- Multiplicating $H^T H$ with complexity $O(\hat{n}^2 n)$
- Cholesky decomposition with complexity $O(\frac{1}{3}\hat{n}^3)$
- Forward and Backwards decomposition for each w_j with a total complexity of $O(2\hat{n}^2 o)$

So considering the output unit number as a constant we will have a total complexity of:

$$O(\frac{1}{3}\hat{n}^3 + \hat{n}^2 n) \tag{16}$$

2.1.2 Stability

With normal equation the condition number is squared while ridge regression regularization can help to lower it, infact we have:

$$k(H^T H + n\lambda I) = \frac{\sigma_1^2 + n\lambda}{\sigma_m^2 + n\lambda}$$

Where σ_1 and σ_m are respectively the greatest and the smaller singular value of the matrix H. So for $H^T H$ not invertible or for high max singular value of H the condition number of the matrix is high and it would let to unstable results.
Because to solve the linear systems:

$$(H^T H + n\lambda I) w_{2_j} = H^T t_j \ \forall j = 1, ..., o$$

we would have an error of:

$$\frac{\|\tilde{w}_{2_j} - w_{2_j}\|}{\|w_{2_j}\|} \leqslant \frac{\sigma_1^2 + n\lambda}{\sigma_m^2 + n\lambda} \left(\frac{\|\tilde{A} - A\|}{\|A\|} + \frac{\|\tilde{t}_j - t_j\|}{\|t_j\|} \right) \ \forall j = 1, ..., o \tag{17}$$

with $A = H^T H + n\lambda$.

2.2 Adam

Adam is a stochastic gradient descendent algorithm that belongs to the adaptive methods which employs exponential moving average. So instead of the classic average, in Adam are used the following averaging functions:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) {g_t}^2$$

Where

1. g_t correspond to the computation of the gradient at iteration t.
2. m_t is the estimate of the first moment (the mean) of the gradient
3. v_t is the estimate of the second raw moment (the uncentered variance) of the gradient
4. the hyper-parameters β_1, β_2 control the exponential decay rates of these moving average

Note: being W_1 fixed in the following part we will refer to W_2 as W to simplify the notation.

Algorithm 1 Adam. Good default settings are $\beta_1 = 0.9$, $\beta_2 = 0.999$

Require: ϵ: Error threshold
Require: T: Max iteration
Require: η: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $L(W)$: Stochastic objective function with parameters W
Require: W: Initial random matrix
  procedure ADAM
    $m_0 = 0$ (initialize 1^{nd} moment vector)
    $v_0 = 0$ (initialize 2^{nd} moment vector)
    $t = 0$ (initialize iteration)
    while $t < T$ && $error < \epsilon$ **do**
      t = t+1
      $g_t = \nabla L(W_{t-1})$ (Get gradients at timestep t)
      $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (update biased first moment estimate)
      $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (update biased second raw moment estimate)
      $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ (Compute bias-correction first moment estimate)
      $\hat{v}_t = \frac{m_t}{1-\beta_2^t}$ (Compute bias-correction second raw moment estimate)
      $W_t = W_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
    end while
  end procedure

For the batch version we will show that the algorithm converge under reasonable assumption.
For the mini-batch version while in the original paper it was shown to converge in the minimization of convex function [2] it was later spotted an error and it was infact shown that Adam can fail to converge even in a simple convex case

[3]. We will cite the example, discuss the problem and we will indicate the new hypotheses under which it will be possible to prove the convergence of Adam even in the non-convex case [4].

2.2.1 Batch version convergence

For this discussion we refer to *Theoreme 1* in *Adaptive Method for Nonconvex Optimization* [4] adapting it to the batch version of the algorithm.
So we will assume the following points:

1. The loss function L is *L-smooth* i.e. there exists a constant K such that:
$$\|\nabla L(W_x) - \nabla L(W_y)\| \leq K\|W_x - W_y\|,\ \forall W_x, W_y \in \mathbb{R}^{\hat{n}\times o}$$
2. The loss function L has bounded gradient i.e. there exists a constant G such that:
$$\|\nabla L(W)\| \leq G,\ \forall W \in \mathbb{R}^{\hat{n}\times o}$$

Theorem 1. *Assume that ϵ, β_2 and η are chosen such that the following conditions are satisfied: $\eta \leq \frac{\epsilon}{2K}$ and $1 - \beta_2 \leq \frac{\epsilon^2}{16G^2}$. Then for x_t generated using ADAM, we have the following bound:*

$$\frac{1}{T}\sum_{t=1}^{T} \|\nabla L(W_t)\|_2^2 \leq O\left(\frac{L(W_1) - L(\hat{W})}{\eta T}\right)$$

where $\hat{W}$ is the optimal solution.

So on average the norm of the gradient will go to 0 as T increases.

2.2.2 Mini-batch version convergence

Consider the following sequence of linear functions for the loss function L in the domain $[-1, 1]$ corresponding to the loss function evaluated in different mini-batches:

$$f_t(x) = \begin{cases} Cx & \text{for t mod } 3 = 1 \\ -x & \text{else} \end{cases}$$

with $C > 2$. For this sequence of functions the point $x = -1$ is the one which minimize the function on the relative batch computation. But supposing $\beta_1 = 0$ and $\beta_2 = 1/(1 + C^2)$ ADAM converge to the suboptimal solution $x = +1$. The algorithm obtains the large gradient C ones every 3 steps while, in the other 2 steps, it obtains the gradient -1, which moves the algorithm in the wrong direction.
Intuitively the exponential moving average doesn't equally evaluate the influence of the different gradient, so when a large gradient it's rarely encountered, even if it would be informative, its influence dies out rather quickly due to the utilized averaging functions.
Further results can prove the following theorem:

Theorem 2. *For any constant β_1, $\beta_2 \in [0, 1)$ such that $\beta_1 < \beta_2$, there is a stochastic convex optimization problem for which ADAM does not converge to the optimal solution.*

The condition imposed to ensure the convergence will infact be raising min-batch size so as to reduce the variance in the gradients, preventing the loss of important information.

To discuss the conditions for the mini-batch case we will define the function $l(W_t, S)$ which corresponds to the function L evaluated on a mini-batch $S \subset \mathbb{R}^v$ of data from the dataset. We will also assume the following points:

1. that the function l is *L-smooth*, i.e. there exists a constant K such that:
$$\|\nabla l(W_x, S) - \nabla l(W_y, S) \leq K\|W_x - W_y\|,\ \forall W_x, W_y \in \mathbb{R}^{\hat{n}\times o} \text{ and } \forall S$$
2. that the function l has bounded gradient, i.e. there exists a constant G such that:
$$\|\nabla l(W, S)\| \leq G,\ \forall W \in \mathbb{R}^{\hat{n}\times o} \text{ and } \forall S$$
3. the existence of a bound on the variance with respect to the deterministic gradients:
$$\mathbb{E}[\|\nabla l(W, S) - \nabla L(W)\|]^2 \leq \sigma^2,\ \forall W \in \mathbb{R}^{\hat{n}\times o} \text{ and } \forall S$$

Given the following assumption we can refer back to the *Theorem 1* in *Adaptive Method for Nonconvex Optimization* [4]:

Theorem 3. *Assume that ϵ, β_1 β_2 and η are chosen such that the following conditions are satisfied: $\eta \leq \frac{\epsilon}{2K}$, $\beta_1 = 0$ and $1 - \beta_2 \leq \frac{\epsilon^2}{16G^2}$. Then for x_t generated using ADAM, we have the following bound:*

$$\mathbb{E}[\|\nabla L(W_a)\|^2] \leq O\left(\frac{L(W_1) - L(\hat{W})}{\eta T} + \sigma^2\right)$$

where $\hat{W}$ is the optimal solution and W_a is an iterate uniformly randomly chosen from $\{W_1, ..., W_T\}$

and it's declared that this results are possibly extendable also in the case of $\beta_1 \neq 0$. An immediate consequence of this result is that increasing mini-batch size reduce the variance improving convergence, up to removing the variance term in case of batch version, as shown in *Theorem 1.* So a consequence is the following corollary [4]:

Corollary 1. *ADAM with $b = \Theta(T)$ (and with the same assumption made above) lead to $\mathbb{E}[\|\nabla L(W_a)\|^2] \leq O(1/T)$ and a computational complexity of $O(1/\delta^2)$ for achieving a δ-accurate solution.*

In fact given b = $\Theta(T)$ the variance term σ goes to 0 as T increases, with the *Jensen's Inequality* we have:

$$\mathbb{E}[\|\nabla L(W_a)\|]^2 \leq \mathbb{E}[\|\nabla L(W_a)\|^2]$$

so we can rewrite the bound on the gradient of the function like that:

$$\mathbb{E}[\|\nabla L(W_a)\|]^2 \leq O\left(\frac{L(W_1) - L(\hat{W})}{\eta T}\right)$$

,we omit the constants

$$\mathbb{E}[\|\nabla L(W_a)\|] \leq \frac{1}{\sqrt{T}}$$

then we set $\frac{1}{\sqrt{T}} \leq \delta$ and we obtain $T \leq \frac{1}{\delta^2}$.

2.2.3 Discussion about convergence in batch case

Our loss function is L-smooth and it has bounded gradient (except when going to infinity). So in the batch case we can state that it will converge and being the function convex we would expect that the function will converge to the optimum.

2.2.4 Discussion about convergence in mini-batch case

Again we have that our loss function is L-smooth and it has bounded gradient (except when going to infinity). But for the mini-batch case we have to consider the σ term regulated by the mini-batch size parameter: b.
While the theoretical results about convergence are stated with batch size $b = \Theta(T)$ for the sake of simplicity, similar results can be obtained for increasing mini-batches $b_t = \Theta(t)$. And generally a much weaker increase in batch size is sufficient, infact when variance is not large a reasonably large batch size can work well [4] and in case of low variance a small batch size don't prevent convergence.
So in a pratical application we could start by launching ADAM with a low mini-batch size and then increase it in case of problems in convergence or we coud try to raise the batch-size as the iterations increase. In this way, being the function convex, we would expect to converge to the optimum.

3 Experiments

In this section, we will deal with input data. We will show at the beginning some details about how the provided dataset is composed. Next, we will show our experimental results on the methods discussing the relation with the underlying theory, and at the end, we will compare the two different optimization approaches.

3.1 Dataset

We use the well-known MONK's datasets, where the aim is to check the correctness of our implementation and get insight into how our methods behave. There are three classification problems (Monk1, Monk2, Monk3), each with a shape of (432,6). The difference between them is the increasing complexity in variables constraint. Each of them is split into a training set and a test set; however, the test set contains the training set. For this reason, we use the data provided in the test file.
To correctly test our models, we apply the *one-hot-encoding*. This technique results into increasing the number of features from 6 to 17.

3.2 Results

To compare the behaviour of algorithms, we move differently accordingly to the approach chosen. Besides (A1) accelerated gradient method supports output as non-linear activation, for fairy comparison versus (A2) closed-form solution, we move to the linear activation for both the output layer.

About non-linear activation, we figured out that *tanh* performs better, concerning the reconstruction error, compared to *sigmoid* in all scenario. So we will show results with only tanh, hiding it from the model parameters list.

Starting weights are drawn randomly from a uniform distribution with range +-0.7.

The following configurations and measurement are taken from a laptop with CPU i7-9750H and 16GB of ram, doing computation only in one core (normal 2.60GHz, turbo boost 4.50GHz).

3.2.1 Cholesky results

To measure Cholesky behaviour, we collect different values. Some are hyperparameters of the model (hidden units, penalty term), and other measurements aim to check if our theoretical results are verified.
As in the theoretical part, we will refer to the matrix A as:

$$A = H^T H + n\lambda I$$

Finally, following values will be traced:

- Loss: representing our loss function stated in equation 11.
- Residual: computed as $\frac{||(A)w - H^T t_j||}{||(A)w||}$
- k(A): condition number of matrix A.
- $\sigma_{max}/\sigma_{min}$: expressing maximum and minimum singular value of matrix H.
- $\gamma_{max}/\gamma_{min}$: expressing maximum and minimum eigenvalue of matrix A.
- Time: seconds elapsed to complete the computation. It will be a mean of 3 trials.

In Table 1 we have a summary taking into account the residual and eigenvalues, without specified the monk dataset that does not affect those values. We can see how a larger number of hidden units leads to to a greater γ_{max} while decreasing the penalty term leads to a lower γ_{min} how is supported by the theory.
About residual, we have a optimal accuracy solving all the minimization problems as we stated in theory in equation 15

Problem	**Units**	**Penalty**	$\gamma_{max}/\gamma_{min}$	**Residual**
Monk	200	9.0e-1	1.5e+4/3.9e+2	1.6e-16
Monk	200	9.0e-4	1.4e+4/4.1e-1	1.2e-15
Monk	200	9.0e-7	1.4e+4/2.6e-2	2.9e-15
Monk	1000	9.0e-1	6.1e+4/3.9e+2	3.1e-16
Monk	1000	9.0e-4	6.0e+4/3.9e-1	1.1e-15
Monk	1000	9.0e-7	6.0e+4/3.9e-4	1.3e-15

Table 1: Behaviour of maximum and minimum eigenvalues of A

In Table 2 we can see how $k(A)$ is proportionally quadratic with respect to σ_{max} and inversely quadratic with respect to σ_{min} and also how $k(A)$ is regulated by the number of hidden units and the penalty term.
From theoretical results we saw, that it brings an upper bound on the weight's relative error (equation 17) but for our experimental results a greater value is not necessarily a sign of greater error.
In fact the configuration with higher $k(A)$ results in the best error. However, in the field of machine learning this results generally imply a case of overfitting and it could be indeed beneficial improving the penalty term both for generalization capability and to lower the condition number improving stability.

Problem	**Units**	**Penalty**	**Loss**	**k(A)**	$\sigma_{max}/\sigma_{min}$
Monk1	200	9.0e-1	1.9e-1	3.8e+1	1.2e+2/1.6e-1
Monk1	200	9.0e-4	3.4e-2	3.5e+4	1.2e+2/1.6e-1
Monk1	200	9.0e-7	1.9e-2	5.6e+5	1.2e+2/1.6e-1
Monk1	1000	9.0e-1	1.5e-1	1.6e+2	2.5e+2/1.7e-1
Monk1	1000	9.0e-4	3.0e-3	1.6e+5	2.5e+2/1.7e-1
Monk1	1000	9.0e-7	3.8e-6	1.6e+8	2.5e+2/1.7e-1
Monk2	200	9.0e-1	2.1e-1	3.8e+1	1.2e+2/1.6e-1
Monk2	200	9.0e-4	1.4e-1	3.5e+4	1.2e+2/1.6e-1
Monk2	200	9.0e-7	1.2e-1	5.6e+5	1.2e+2/1.6e-1
Monk2	1000	9.0e-1	2.0e-1	1.6e+2	2.5e+2/1.7e-1
Monk2	1000	9.0e-4	6.4e-2	1.6e+5	2.5e+2/1.7e-1
Monk2	1000	9.0e-7	2.5e-4	1.6e+8	2.5e+2/1.7e-1
Monk3	200	9.0e-1	1.1e-1	3.8e+1	1.2e+2/1.6e-1
Monk3	200	9.0e-4	1.1e-2	3.5e+4	1.2e+2/1.6e-1
Monk3	200	9.0e-7	5.4e-3	5.6e+5	1.2e+2/1.6e-1
Monk3	1000	9.0e-1	5.9e-2	1.6e+2	2.5e+2/1.7e-1
Monk3	1000	9.0e-4	8.7e-4	1.6e+5	2.5e+2/1.7e-1
Monk3	1000	9.0e-7	1.2e-6	1.6e+8	2.5e+2/1.7e-1

Table 2: Monk's score

In the following images both for 200 neurons (Figures 1, 2 and 3) and 1000 neurons (Figures 4, 5 and 6) we can observe how raising the penalty term lead to a distribution of eigenvalues which is flattened in the the last lower values. It is better visible with an higher number of hidden unit.
For 200 neurons case, as prof of table Table 1 and 2 the lowering of lower eigenvalues distribution, most impacting factor on the condition number is not too much affected by decreasing of penalty term after the value 9.0e-4. In 1000 neurons case, as prof of table Table 1 and 2 the lowering of lower Eigenvalues, most impacting factor on the condition number is highly effected by decreasing of penalty term even after the value 9.0e-4.
All plots are in log scale.

Eigenvalues 200 neurons

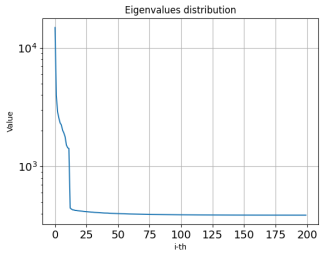


Figure 1: Penalty 9.0e-1

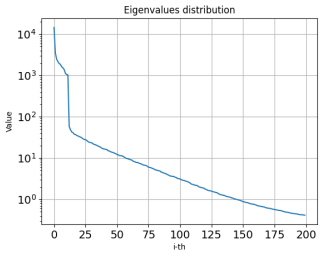


Figure 2: Penalty 9.0e-4

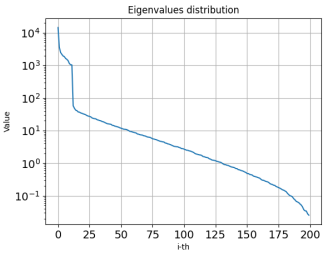


Figure 3: Penalty 9.0e-7

Eigenvalues 1000 neurons

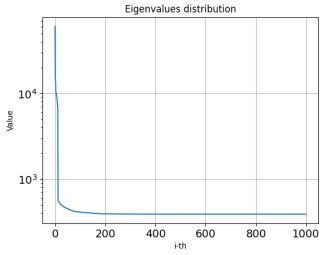


Figure 4: Penalty 9.0e-1

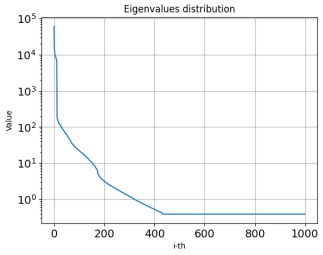


Figure 5: Penalty 9.0e-4

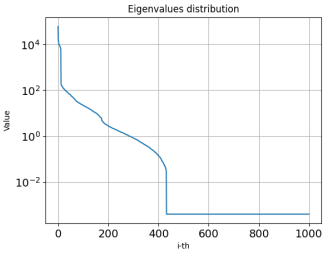


Figure 6: Penalty 9.0e-7

Next, we compare timing results versus the library (scipy) implementation of the Cholesky solver. Results are in table 3. We omitted Loss because it is always the same compute by us, and we omitted the specific monk used, giving the fact that it is independent by the timing results.

Problem	**Neuron**	**Time our**	**Time library**
Monk	100	0.0875	0.0054
Monk	200	0.491	0.0096
Monk	300	1.9455	0.0161
Monk	400	3.7467	0.0195
Monk	500	5.801	0.0257
Monk	600	11.8267	0.0353
Monk	700	17.9402	0.0395
Monk	800	25.6143	0.0487
Monk	900	40.3827	0.0555
Monk	1000	51.8683	0.0765

Table 3: Implementation: Our vs library performance

They implemented matrix multiplication trough parallelism, outperforming our implementation from timing point of view. The precision achieved is the same.

Order of time

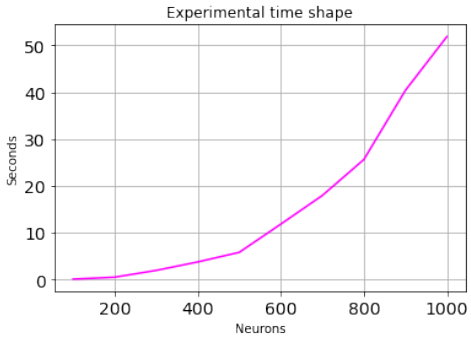


Figure 7: Experiment results

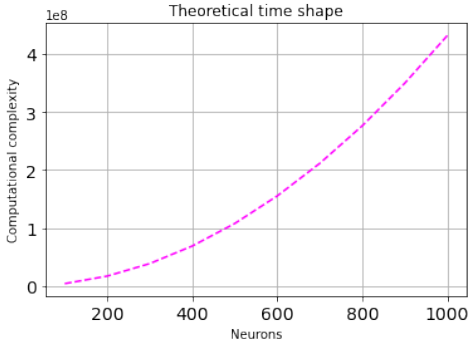


Figure 8: Theoretical results

As we can see from figure 7 and 8 the shape of timing at the increasing of neurons follow the order of complexity stated in theory in equation 16.

3.2.2 Adam results

To measure Adam behaviour, we collect different parameters. Some are hyperparameters of the model (hidden units, penalty term, step size), and other measurements aim to check if our theoretical results are verified.
Given the previously tested exact method, we have the "best" error found defined as f^*_{chol} and shows in Table 4.

Problem	Monk1	Monk1	Monk1	Monk1	Monk1	Monk1
Units	200	200	200	1000	1000	1000
Penalty	9.0e-1	9.0e-4	9.0e-7	9.0e-1	9.0e-4	9.0e-7
f^*_{chol}	**1.9e-1**	**3.4e-2**	**1.9e-2**	**1.5e-1**	**3.0e-3**	**3.8e-6**
Problem	Monk2	Monk2	Monk2	Monk2	Monk2	Monk2
Units	200	200	200	1000	1000	1000
Penalty	9.0e-1	9.0e-4	9.0e-7	9.0e-1	9.0e-4	9.0e-7
f^*_{chol}	**2.1e-1**	**1.4e-1**	**1.2e-1**	**2.0e-1**	**6.4e-2**	**2.5e-4**
Problem	Monk3	Monk3	Monk3	Monk3	Monk3	Monk3
Units	200	200	200	1000	1000	1000
Penalty	9.0e-1	9.0e-4	9.0e-7	9.0e-1	9.0e-4	9.0e-7
f^*_{chol}	**1.1e-1**	**1.1e-2**	**5.4e-3**	**5.9e-2**	**8.7e-4**	**1.2e-6**

Table 4: Cholesky f^*

Being f^*_{chol} the best accuracy we can achieve and the function convex, we will use it to define the stopping condition as either when $f^*_{adam} <= f^*_{chol}$ or when is achieved

the maximum number of iteration (20000).
Indeed, following values will be traced:

- Loss: representing our loss function stated in equation 11
- Iterations: number of iteration needed to reach the stopping condition.
- Norm of the gradient: useful to check bounds stated.
- Time: seconds elapsed to complete one iteration.

After shown some tables with a summary of parameters, we will test the error with respect to minimum:

$$\frac{f(x_i) - f^*}{f^*} \tag{18}$$

where f^* is again the optimal value obtained by Cholesky factorization and $f(x^i)$ is the value obtained for Adam algorithm for a given iteration. Both functions f consider inside them the penalty term. The plot will be in *log* scale.

A general note about step size: if it is set too high (above 0.009), the resulting results will be unstable, visible looking into the loss and the norm of the gradients. With this motivation, we set it to 9.0e-3 in batch (432) and mini-batch (144), and set to 9.0e-5 in mini-batch (16).

3.2.3 Batch version

Table 5 refers to the test over Monk's focusing on the hyperparameters configuration to reach the loss by the cholesky method. All results are from batch size equal to the number of samples available (432). Loss values with $*$ mean that they match the best optima found by cholesky.
Looking into corollary 1 from Adam theory we stated that:

$$\mathbb{E}[\|\nabla L(W_a)\|] \leq \frac{1}{\sqrt{T}}$$

So, for example, launching adam for 3000 iterations we would expect an approximate upper bound on the norm of the gradient of $\frac{1}{\sqrt{3000}} = 1.8\text{e-}2$. We can observe how this bound is validated in the experimental results.

problem	units	penalty	iteration	Loss	norm grad.
Monk1	200	9.0e-1	163	* 1.9e-1	7.3e-2
Monk1	200	9.0e-4	3133	* 3.4e-2	6.4e-5
Monk1	200	9.0e-7	5191	* 1.9e-2	4.6e-4
Monk1	1000	9.0e-1	200	* 1.5e-1	1.3e-1
Monk1	1000	9.0e-4	12356	* 3.0e-3	1.5e-3
Monk1	1000	9.0e-7	20000	1.5e-4	9.0e-5
Monk2	200	9.0e-1	153	* 2.1e-1	1.1e-1
Monk2	200	9.0e-4	1133	* 1.4e-1	7.5e-3
Monk2	200	9.0e-7	2223	* 1.2e-1	2.7e-3
Monk2	1000	9.0e-1	214	* 2.0e-1	9.1e-2
Monk2	1000	9.0e-4	20000	6.9e-2	5.6e-5
Monk2	1000	9.0e-7	20000	2.4e-3	1.3e-3
Monk3	200	9.0e-1	139	* 1.1e-1	1.7e-1
Monk3	200	9.0e-4	2491	* 1.1e-2	9.8e-4
Monk3	200	9.0e-7	10773	* 5.4e-3	4.9e-3
Monk3	1000	9.0e-1	246	* 5.9e-2	2.1e-2
Monk3	1000	9.0e-4	20000	9.0e-4	5.5e-4
Monk3	1000	9.0e-7	20000	1.4e-4	9.2e-5

Table 5: Monk's score

Figure 9 and 10 show behaviour of the norm of the gradient at the increasing of iterations.

Norm of gradients for 200 units

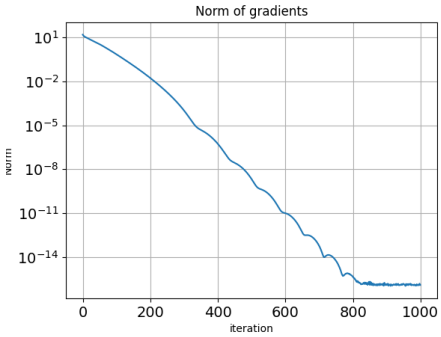


Figure 9: Penalty 9.0e-1

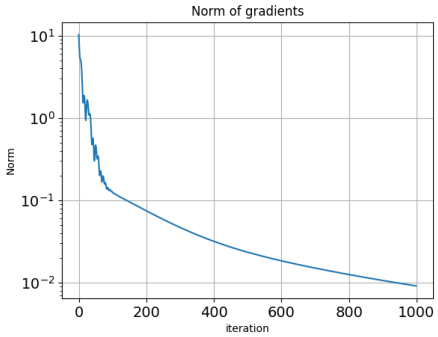
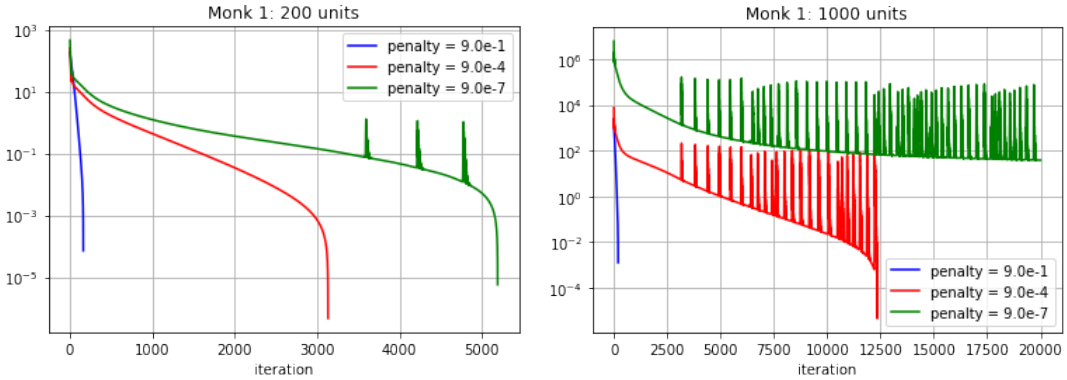
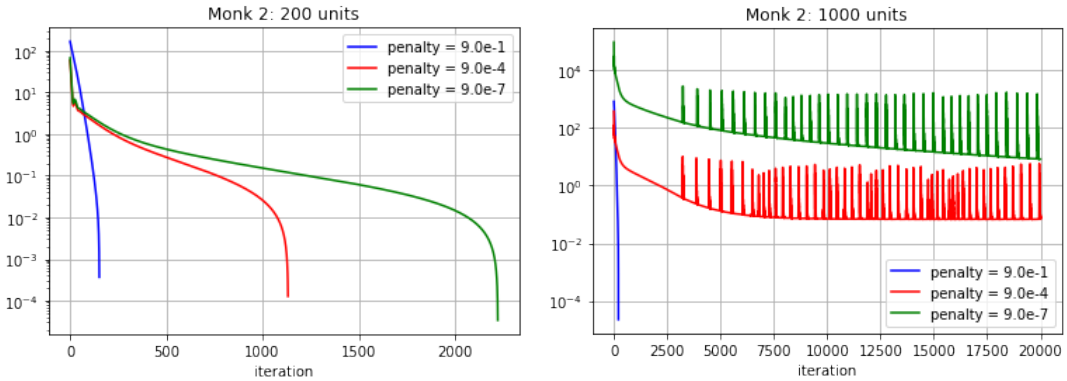


Figure 10: Penalty 9.0e-7



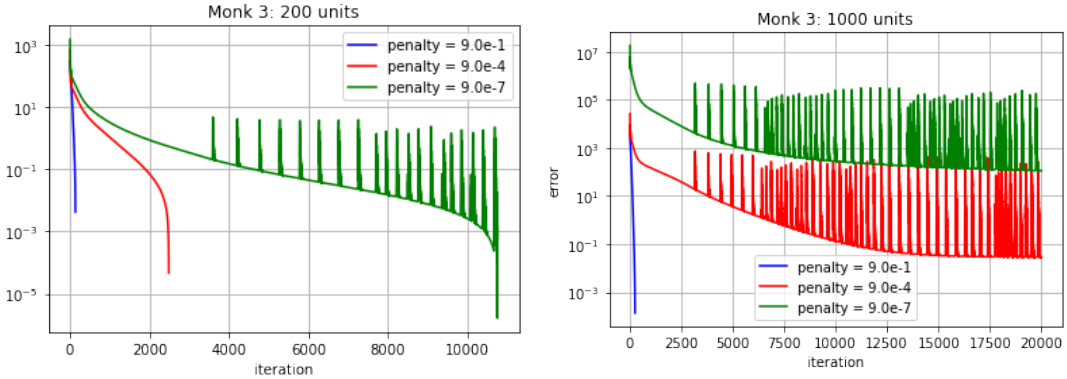
(a) 200 units (b) 1000 units

Figure 11: Error with respect to the minimum for Monk 1



(a) 200 units (b) 1000 units

Figure 12: Error with respect to the minimum for Monk 2



(a) 200 units (b) 1000 units

Figure 13: Error with respect to the minimum for Monk 3

Figure 14 show the behaviour of the norm of the gradient at the increasing of iterations for 1000 unit configuration. Even if this configuration seems to get no improvement at increasing iterations (figure 12b) we can see how the optimizer is consistently lowering the norm of the gradients getting closer to the optimum at increasing iterations.

Norm of gradients for 1000 units

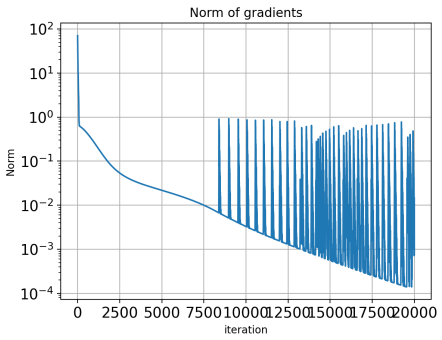


Figure 14: Penalty 9.0e-4

3.2.4 Mini-batch version

In this section we do another type of analysis with respect to Adam in Batch section. We will show only a table with Monk 1, motivated by the fact that it is sufficient to state our considerations.

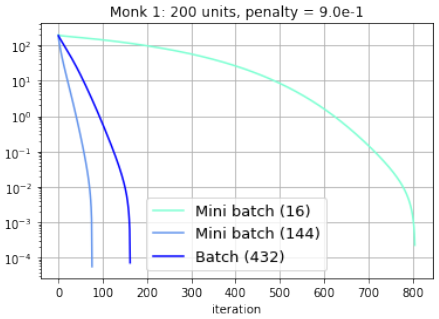
Norms and losses shown in this section are a mean over an iteration, being composed by $\frac{\#samples}{batchsize}$ batches. Therefore, we compute norm and loss at each batch and show the mean over iteration.

Table 6 refers to the test over Monk 1. We highlighted cases where a mini-batch size speed up optimum achievement with respect to the batch approach. Loss values with $*$ mean that they match the best optima found by Cholesky.

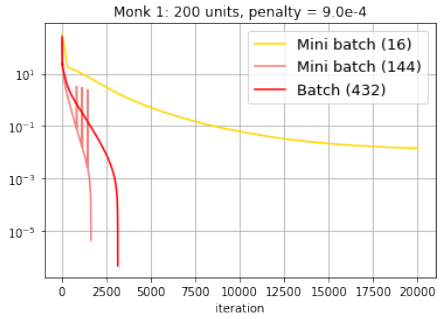
units	penalty	batch	iter.	Loss	norm	$iter._{batch}$	$Loss_{batch}$	$norm_{batch}$
200	9.0e-1	16	806	* 1.9e-1	2.5	163	* 1.9e-1	7.3e-2
200	9.0e-1	144	77	* 1.9e-1	7.3e-1	163	* 1.9e-1	7.3e-2
200	9.0e-4	16	20000	3.5e-2	4.5e-1	3133	* 3.4e-2	6.4e-5
200	9.0e-4	144	1624	* 3.4e-2	9.8e-2	3133	* 3.4e-2	6.4e-5
200	9.0e-7	16	20000	2.2e-2	3.1e-1	5191	* 1.9e-2	4.6e-4
200	9.0e-7	144	4804	* 1.9e-2	4.9e-2	5191	* 1.9e-2	4.6e-4
1000	9.0e-1	16	836	* 1.5e-1	5.1	200	* 1.5e-1	1.3e-1
1000	9.0e-1	144	114	* 1.5e-1	1.5	200	* 1.5e-1	1.3e-1
1000	9.0e-4	16	20000	1.1e-2	7.0e-1	12356	* 3.0e-3	1.5e-3
1000	9.0e-4	144	20000	3.1e-3	2.6e-2	12356	* 3.0e-3	1.5e-3
1000	9.0e-7	16	20000	8.2e-3	1.1	20000	1.5e-4	9.0e-5
1000	9.0e-7	144	20000	2.7e-4	9.7e-3	20000	1.5e-4	9.0e-5

Table 6: Monk 1 scores

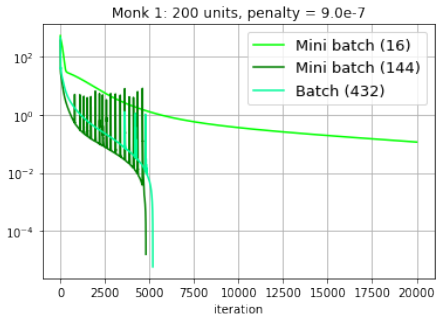
Below we will show the plots regarding the different configurations comparing mini-batch to the batch case.



(a) penalty 9.0e-1

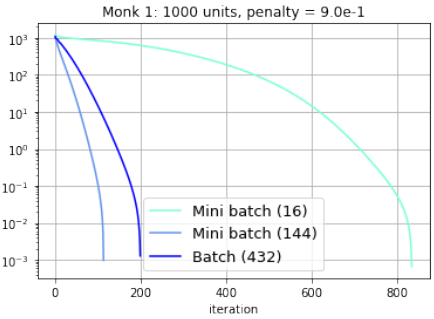


(b) penalty 9.0e-4

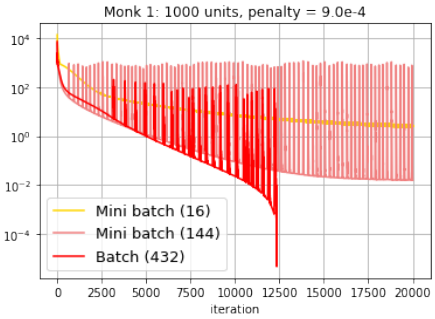


(c) penalty 9.0e-7

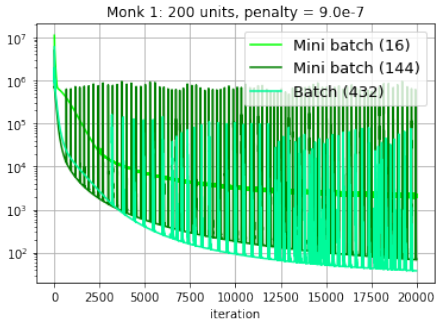
Figure 15: Error with respect to the minimum for Monk 1, 200 units



(a) penalty 9.0e-1



(b) penalty 9.0e-4



(c) penalty 9.0e-7

Figure 16: Error with respect to the minimum for Monk 1, 1000 units

We can see how batch-case can indeed be beneficial, using a batch size of 144 results in faster convergence except for the last two configurations.
For what it concern a smaller batch size equals to 16 we can observe how it degrades performance, this could be attributed to the oscillation of the gradient in several mini-batches. As stated in the theory [4] this could results into a loss of information for some informative mini-batches due to the fact that exponential moving average don't equally evaluate the influence of the different gradients.

3.2.5 Time

We can check whether if the batch size leads to speedup improvement with the same hyperparameters configuration. Figure 7 shows the time needed to do 10 iterations measured in seconds.

Units	Batch size	Time_{10}
200	16	**2.180**
200	144	**0.429**
200	batch	**0.281**
1000	16	**7.683**
1000	144	**1.550**
1000	batch	**1.023**

Table 7: Monk's time

Clearly, reducing the batch size leads to more matrix multiplication inside an iteration, resulting in more overhead and slower computation with respect to the iterations.

4 Conclusions

In this chapter, we are interested in comparing models: (A1) Adam as accelerated gradient method, and (A2) Cholesky closed-form solution. We can do considerations based on speed (seconds) over Monk 1.

In table 12 is shown time (seconds) needed to reach the the best optima accordingly to the problem we want to solve, defined by the penalty term.

Penalty	Batch size	Loss	Time
9.0e-1	-	1.9e-1	00,491
9.0e-4	-	3.4e-2	00,491
9.0e-7	-	1.9e-2	00,491

Table 8: Cholesky 200

Penalty	Batch size	Loss	Time
9.0e-1	16	1.9e-1	175,70
	144	1.9e-1	03,303
	batch	1.9e-1	04,580
9.0e-4	16	3.5e-2	>4360,0
	144	3.4e-2	69,669
	batch	3.4e-2	88,037
9.0e-7	16	2.2e-2	>4360,0
	144	1.9e-2	206,09
	batch	1.9e-2	145,87

Table 9: Adam 200

Penalty	Batch size	Loss	Time
9.0e-1	-	1.5e-1	51,858
9.0e-4	-	3.0e-3	51,858
9.0e-7	-	3.8e-6	51,858

Table 10: Cholesky 1000

Penalty	Batch size	Loss	Time
9.0e-1	16	1.5e-1	642,29
	144	1.5e-1	17,670
	batch	1.5e-1	20,460
9.0e-4	16	1.1e-2	>15366
	144	3.1e-3	>3100,0
	batch	3.0e-3	1264,0
9.0e-7	16	8.2e-3	>15366
	144	2.7e-4	>3100,0
	batch	1.5e-4	>2046,0

Table 11: Adam 1000

Table 12: Comparison by time

When the problem involves fewer parameters (figures: 8 and 9) Cholesky outperforms Adam. If the penalty stays high, the difference is negligible, else when the penalty is reduced (so we are more interested in the precision of solvers), timing differences are emphasized.

When the problem involves higher parameters (figures: 10 and 11) and the penalty stay high (9.0e-1), Adam reaches the optimum faster then Cholesky. However when the penalty decrease Cholesky outperforms Adam, needing much more time (more than 20000 iterations) to reach the best solution.

General considerations are: when the accuracy is fundamental a direct solution provide better performance, but in scenario where reaching an optimum, such as in Machine Learning, a gradients method could reduce the computational time. All this consideration are with our implementation that don't exploit parallel computing.

5 References

References

[1] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. *Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks.*

[2] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization. 2015*

[3] Sashank J. Reddi and Satyen Kale and Sanjiv Kumar. *On the Convergence of Adam and Beyond. 2018*

[4] Zaheer, Manzil and Reddi, Sashank J. and Sachan, Devendra and Kale, Satyen and Kumar, Sanjiv. *Adaptive Methods for Nonconvex Optimization. 2018*