Escuela Superior de Informática

Universidad de Castilla-La Mancha

UCLM

MAT

Enhancing the convergence speed of line search methods: Applications in Neural Network training

EURO 24 COPENHAGEN

José Ángel Martín-Baos
Ricardo García-Ródenas
Luis Rodriguez-Benitez
María Luz López-García

33rd European Conference on Operational Research

30 June - 3 July 2024

Introduction

Empirical Risk:

$$R(\omega) = \frac{1}{N}\sum_{n=1}^{N} f_n(\omega)$$

Introduction

Stochastic gradient descend (SGD):

$$\omega_{k+1} = \omega_k - \alpha_k \nabla f_n(\omega_k)$$

Batch gradient descend (GD):

$$\omega_{k+1} = \omega_k - \frac{\alpha_k}{N} \sum_{n=1}^{N} \nabla f_n(\omega_k)$$

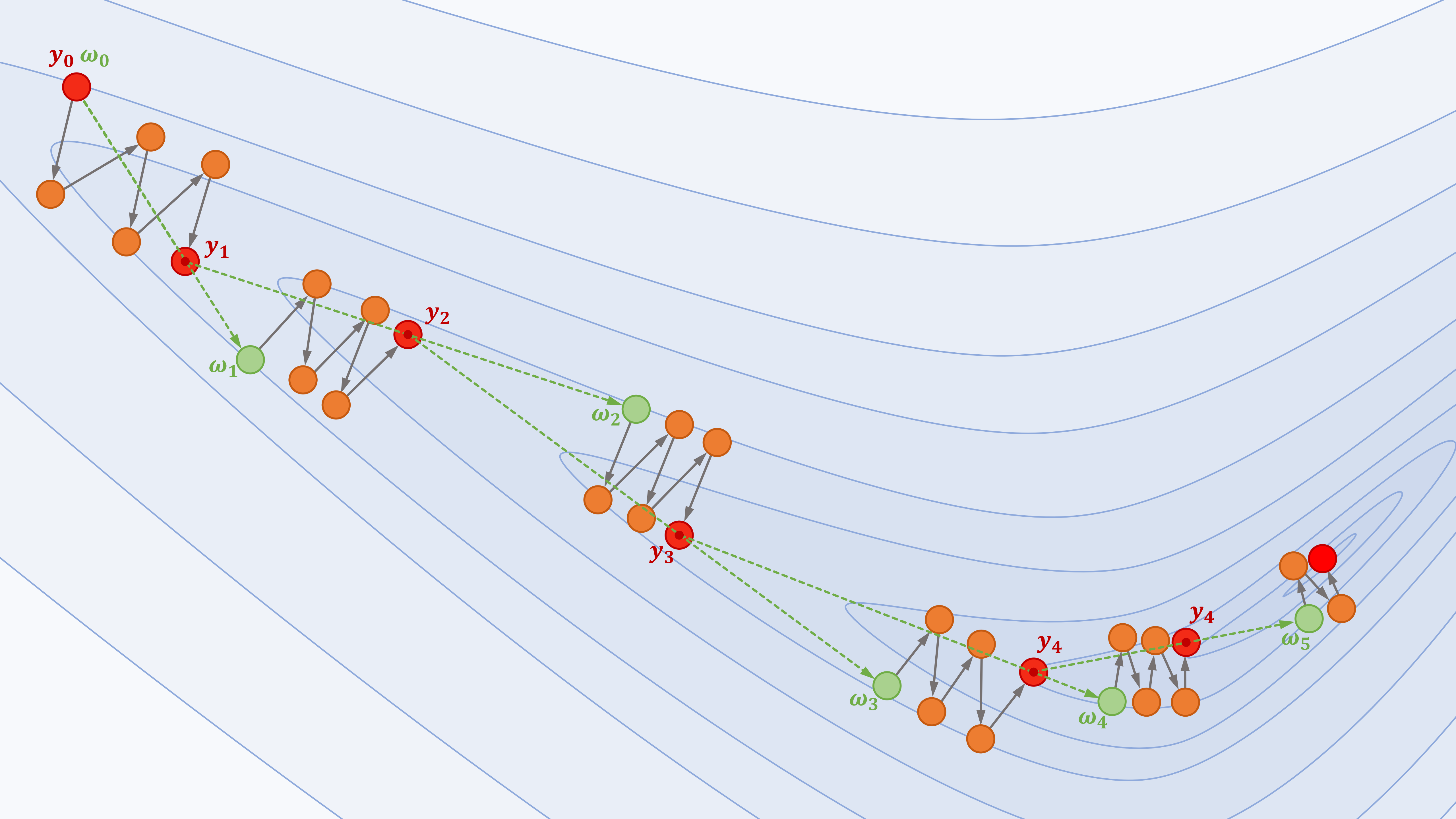
Mini-batch gradient descend:

$$\omega_{k+1} = \omega_k - \frac{\alpha_k}{|S_k|} \sum_{n \in S_k} \nabla f_n(\omega_k)$$

Introduction

Choose an initial iterate w_1.
for $k = 1, 2, \ldots$ **do**
    Generate a realization of the random variable ξ_k.
    Compute a stochastic vector $g(w_k, \xi_k)$.
    Choose a stepsize $\alpha_k > 0$.
    Set the new iterate as $w_{k+1} \leftarrow w_k - \alpha_k g(w_k, \xi_k)$.
end for

Proposed method



y_0 ω_0
y_1
y_2
ω_1
ω_2
y_3
y_4
y_4
ω_3
ω_4
ω_5

Input : A stochastic line search method $\mathcal{A}$
The hyperparameters of the optimisation method
The objective function F
Constants $\gamma > 1$, $\theta \in (0, 1)$, $\alpha_{\max}$, and n_a

Output: The parameter vector ω_t of the optimised model

1 Choose an initial guess ω_0 and set $y_0 = \omega_0$, $\alpha_0 = \gamma^{-1}\alpha_{\max}$ and $t = 1$
2 **Compute a search direction:** Complete $n_a \geq 1$ epochs using the optimisation algorithm $\mathcal{A}$, initialising it at the point ω_{t-1}. Store the optimised vector of parameters as y_t. Set the search direction by setting $d_t = y_t - y_{t-1}$
3 **Compute the function estimates:** Set $F_t^0 = F(y_{t-1})$
4 Set $\mu_0 = \alpha_{t-1}$. Set $\Delta = \theta \|d_t\|_2^2$ and iteration counter $j = 0$
5 **if** $F_t^0 - F\left(y_{t-1} + \mu_0 d_t\right) \geq \mu_0 \Delta$ **then**
/* Increase the step size while the Armijo condition is satisfied */
6 Armijo_satisfied = True
7 **while** *Armijo_satisfied* **and** $(\gamma \mu_j \leq \alpha_{max})$ **do**
8 $\mu_{j+1} = \gamma \mu_j$
9 **if** $F_t^0 - F\left(y_{t-1} + \mu_j d_t\right) < \mu_j \Delta$ **then**
10 Armijo_satisfied = False
11 **else**
12 $j = j + 1$
13 **else**
/* Decrease the step size until the Armijo condition is satisfied */
14 Armijo_satisfied = False
15 **while** *not Armijo_satisfied* **and** $(1 \leq \gamma^{-1}\mu_j)$ **do**
16 $\mu_{j+1} = \gamma^{-1}\mu_j$
17 **if** $(F_t^0 - F\left(y_{t-1} + \mu_j d_t\right) \geq \mu_j \Delta)$ **then**
18 Armijo_satisfied = True
19 **else**
20 $j = j + 1$
/* Update the parameters */
21 $\alpha_t = \mu_j$
22 $\omega_t = \omega_{t-1} + \alpha_t d_t$
23 If the stop criterion is not satisfied, set $t = t + 1$ and return to step 2

Experiments

Datasets

LPMC

- Single day travel diary data from 2012 to 2015.
- 81,096 surveys with 31 variables.
- After pre-processing, 20 variables selected.

 35%

 44%

 3%

 18%

NTS

- ML focused dataset:
  - Data from a Dutch transport survey from 2010 to 2012.
  - Environmental data.
- 230,608 surveys with 16 variables.

 4%

 55%

 24%

 17%

Kernel Logistic Regression (KLR)

[1] Martín-Baos et al (2021) *Revisiting kernel logistic regression under the random utility models perspective. An interpretable machine-learning approach*. Transportation Letters

$$U_{in} = V_{in} + \epsilon_{in}$$

$$U_{in} = \boxed{\text{KLR}} + \epsilon_{in}$$

KLR

$$V_i(\mathbf{x} \mid \boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_{in} k(\mathbf{x}_{in}, \mathbf{x})$$

$$[\mathbf{K}_i]_{n,n'} = k(\mathbf{x}_{in}, \mathbf{x}_{in'}) \quad \text{for } n, n' = 1, \ldots, N$$

$$V_i(\mathbf{x} \mid \boldsymbol{\alpha}) = \mathbf{K}_i^{(n)\top} \boldsymbol{\alpha}_i$$

$$\mathbf{K}_i$$

n	1	2	3	4	5	6
1	1.0	0.8	0.2	0.0	0.5	0.1
2	0.8	1.0	0.4	0.2	0.6	0.3
3	0.2	0.4	1.0	0.9	0.7	0.5
4	0.0	0.2	0.9	1.0	0.4	0.6
5	0.5	0.6	0.7	0.4	1.0	0.8
6	0.1	0.3	0.5	0.6	0.8	1.0

$\mathbf{K}_i$

n	1	2	3	4	5	6
1	1.0	0.8	0.2	0.0	0.5	0.1
2	0.8	1.0	0.4	0.2	0.6	0.3
3	0.2	0.4	1.0	0.9	0.7	0.5
4	0.0	0.2	0.9	1.0	0.4	0.6
5	0.5	0.6	0.7	0.4	1.0	0.8
6	0.1	0.3	0.5	0.6	0.8	1.0

$\mathbf{K}_i$

n	1	2	3	4	5	6
1	1.0	0.8	0.2	0.0	0.5	0.1
2	0.8	1.0	0.4	0.2	0.6	0.3
3	0.2	0.4	1.0	0.9	0.7	0.5
4	0.0	0.2	0.9	1.0	0.4	0.6
5	0.5	0.6	0.7	0.4	1.0	0.8
6	0.1	0.3	0.5	0.6	0.8	1.0

$\times$

$\boldsymbol{\alpha}_i$

0.82
0.14
-0.26
0.91
0.74
-0.6

$$\mathbb{P}(\, i \mid \mathbf{K}_n, \boldsymbol{\alpha} \,) = \frac{e^{V_i}}{\sum_{j=1}^{I} e^{V_j}} = \frac{e^{\mathbf{K}_i^{(n)\top} \boldsymbol{\alpha}_i}}{\sum_{j=1}^{I} e^{\mathbf{K}_j^{(n)\top} \boldsymbol{\alpha}_j}}$$

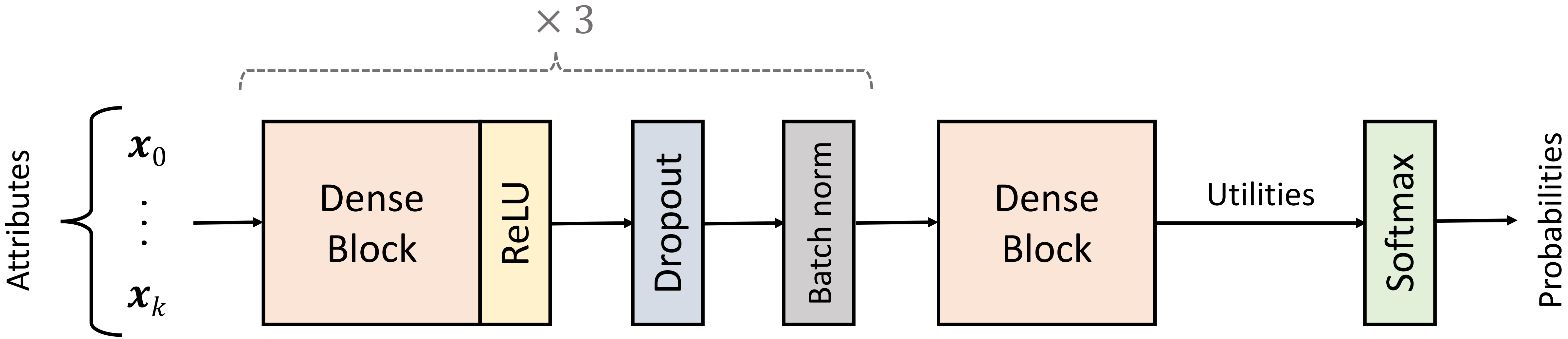
$$\log \mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \sum_{i=1}^{I} y_{in} \log \mathbb{P}(\, i \mid \mathbf{x}_n, \boldsymbol{\alpha}_i \,)$$

$$F = -\log \mathcal{L}(\boldsymbol{\alpha}) + \lambda \Omega(\boldsymbol{\alpha})$$

Goodness of fit

Penalisation term

Deep Neural Networks
(DNN)


× 3
Attributes
$\boldsymbol{x}_0$
$\boldsymbol{x}_k$
Dense Block
ReLU
Dropout
Batch norm
Dense Block
Utilities
Softmax
Probabilities

$$F = -\sum_{n=1}^{N} \sum_{i=1}^{I} y_{n,i} \log \hat{y}_{n,i}$$

Numerical results

GKLR Python package

https://github.com/JoseAngelMartinB/gklr

&

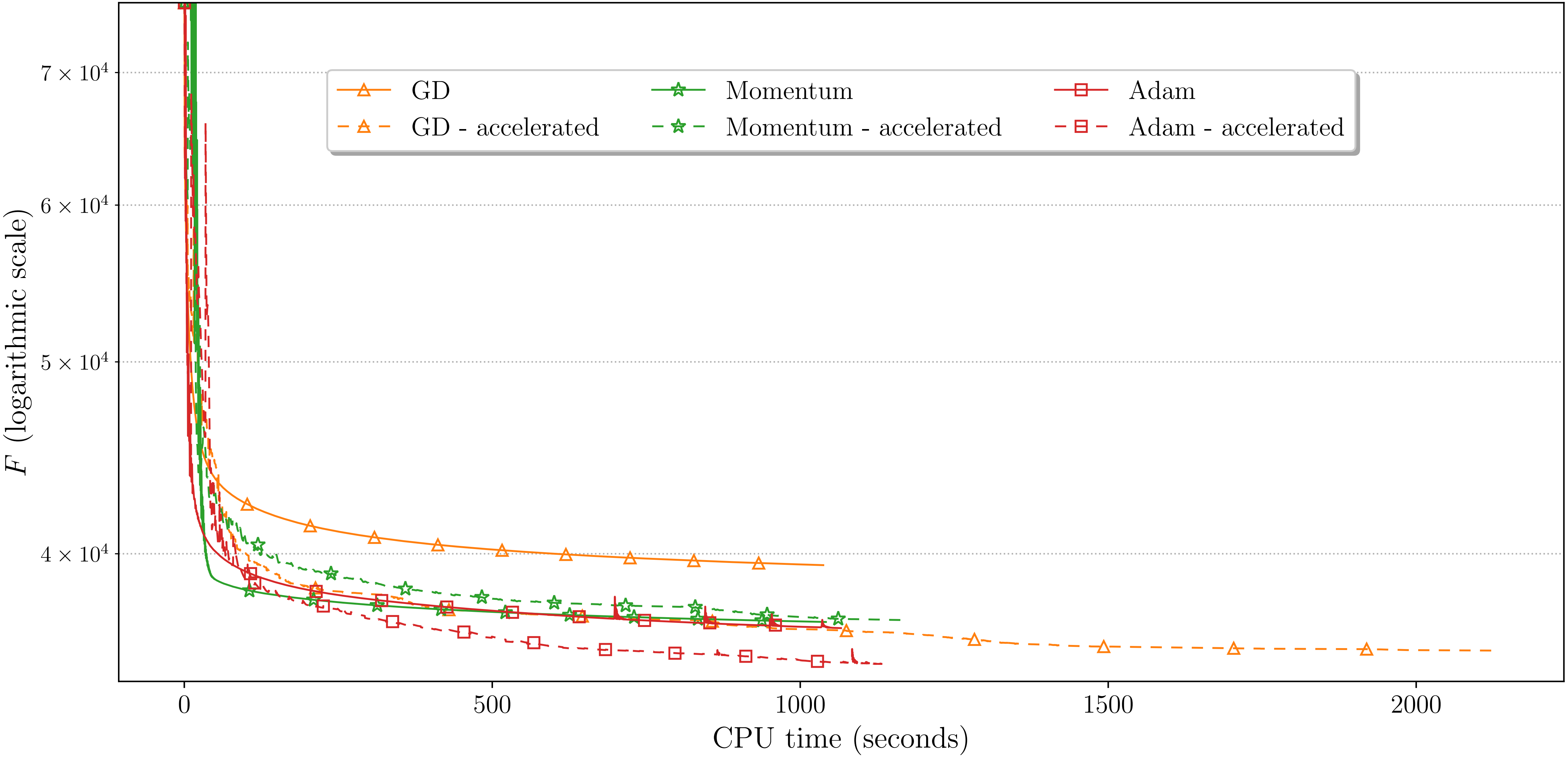
Keras with **tensorflow**

Experimental setup

- Ubuntu 20.04 LTS
- 3.8 GHz 12 core AMD Ryzen
- 32 GB of RAM

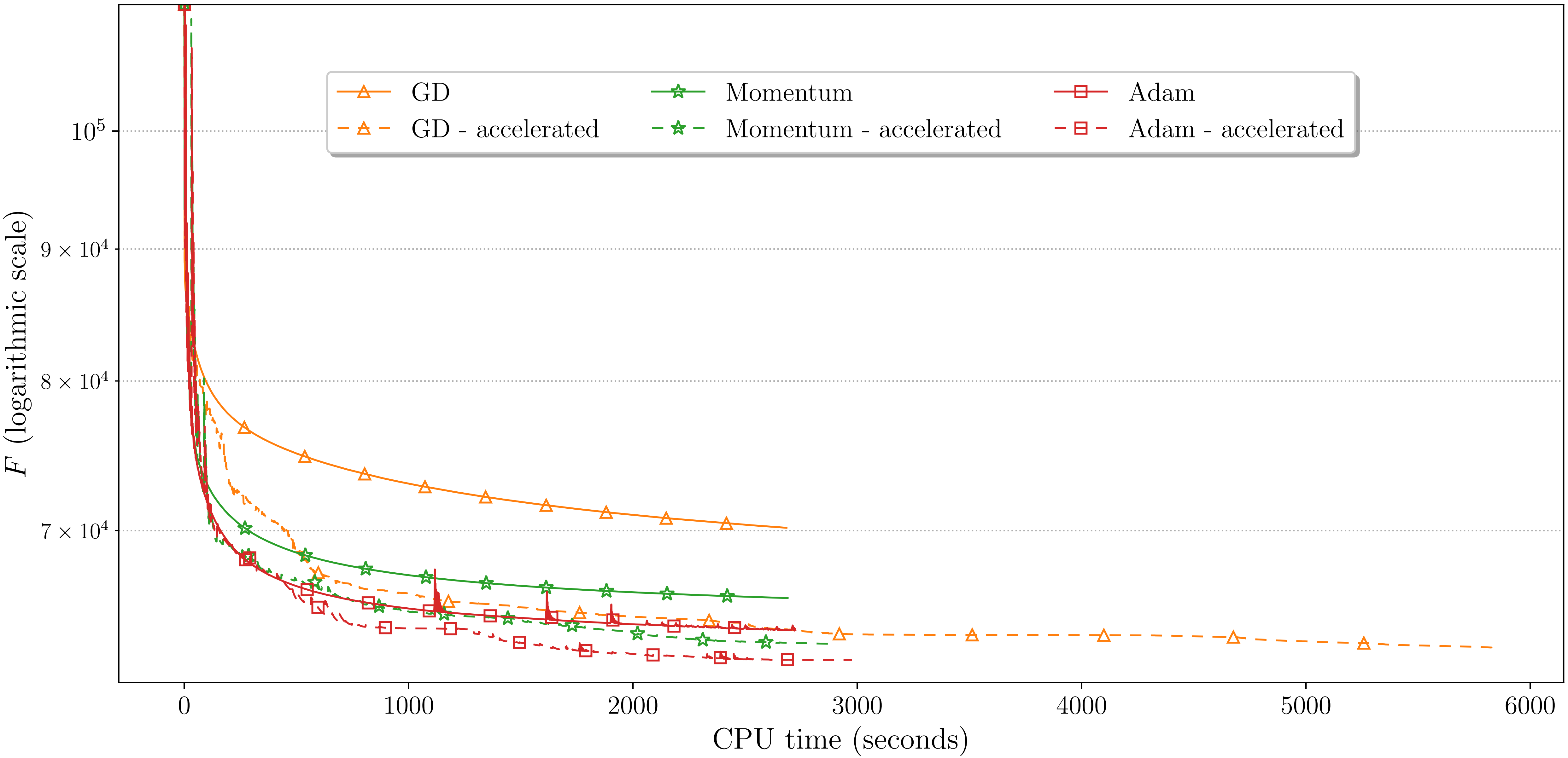
KLR – Estimation (log-likelihood)

LPMC



KLR – Estimation (log-likelihood)

NTS

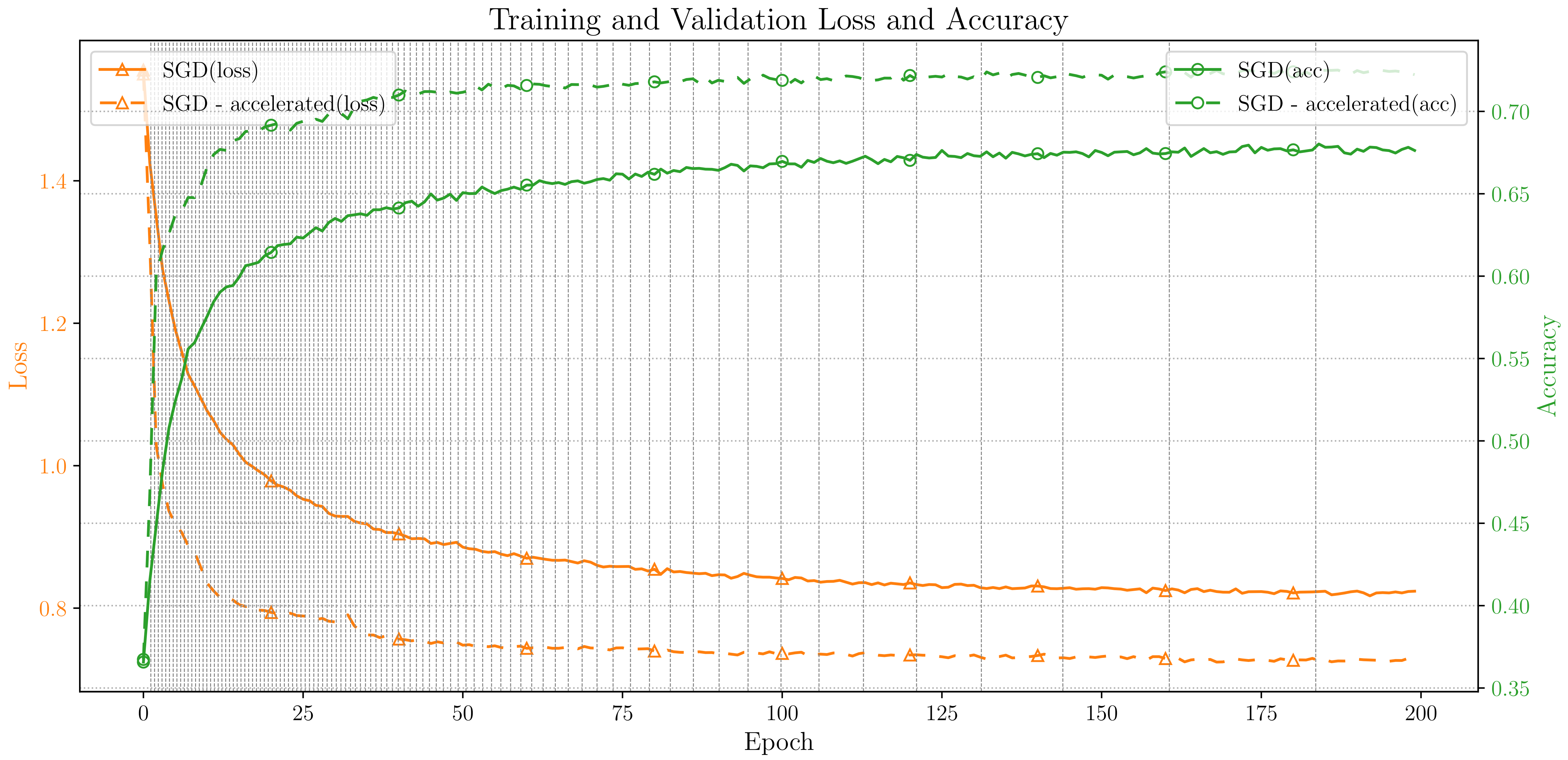


KLR – Estimation comparative

	LPMC		NTS	
	Δ iterations	Δ CPU time (s)	Δ iterations	Δ CPU time (s)
GD	4,711	913.71	4,643	2,249.93
Momentum	–	–	3,777	1,979.45
Adam	3,267	676.14	2,840	1,438.46

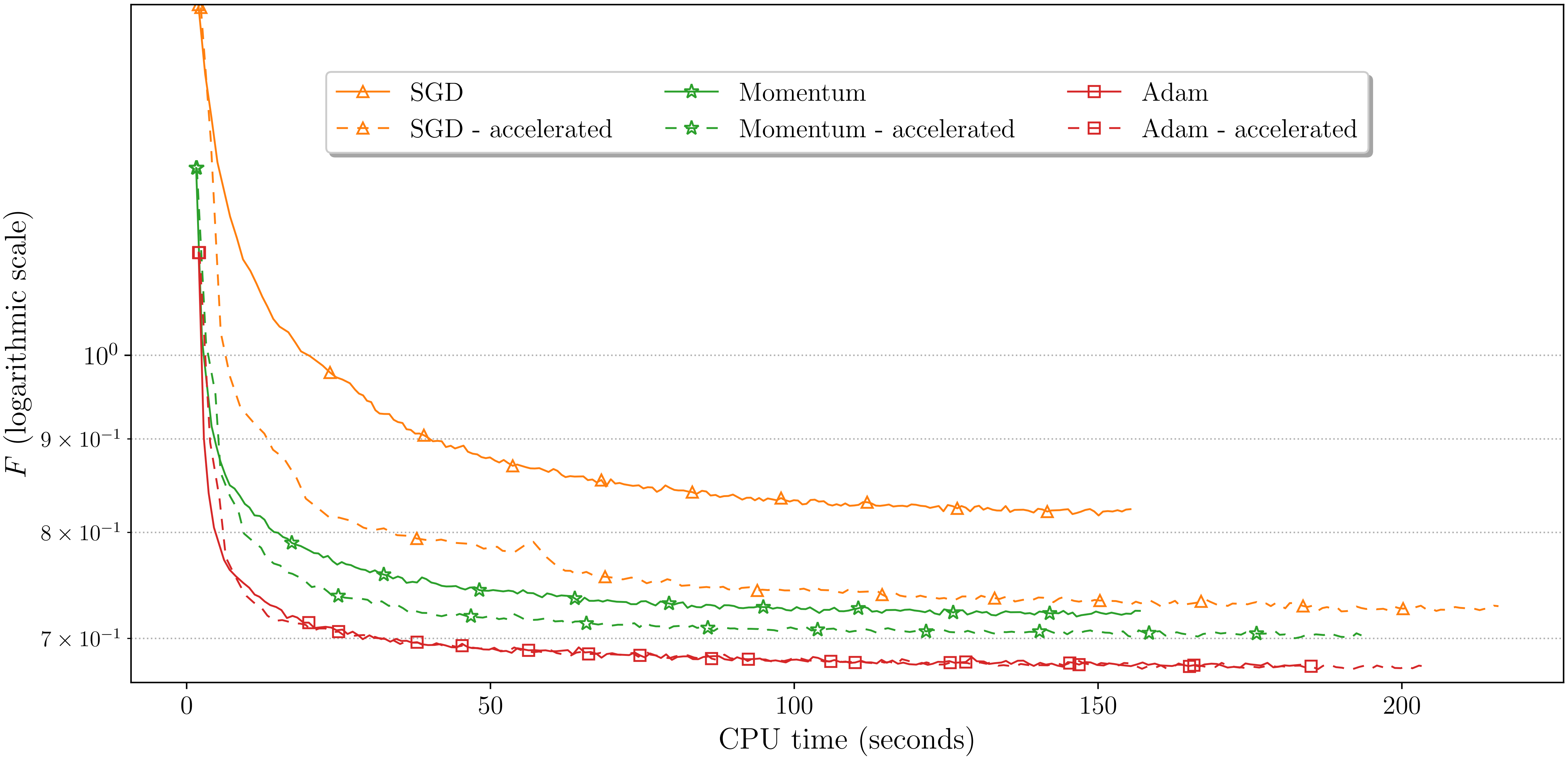
DNN – Estimation for SGD

LPMC



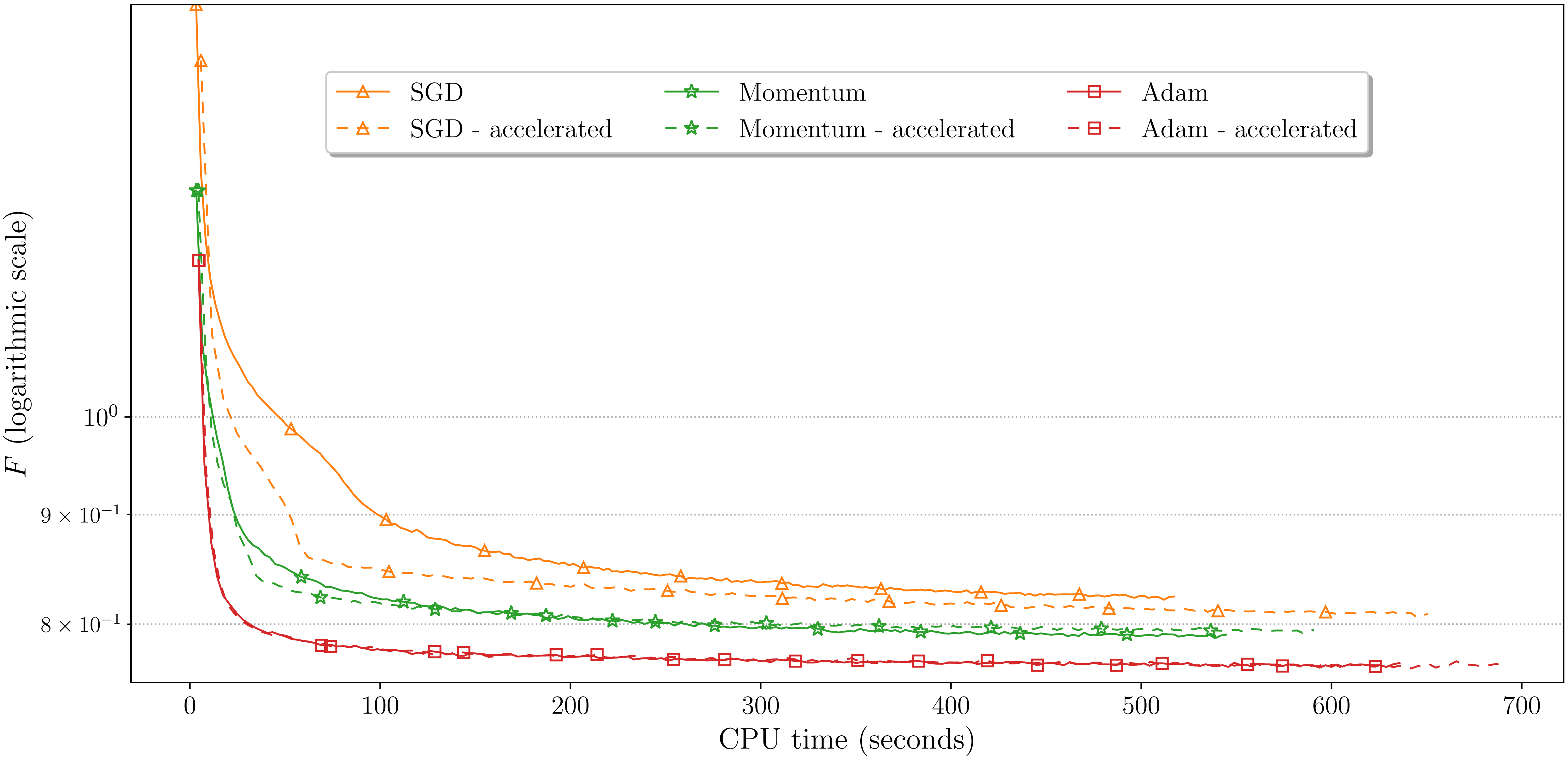
DNN – Estimation (categorical cross-entropy)

LPMC



DNN – Estimation (categorical cross-entropy)

NTS



DNN – Estimation comparative

	LPMC		NTS	
	Δ epochs	Δ CPU time (s)	Δ epochs	Δ CPU time (s)
SGD	188	132.14	120	206.08
Momentum	170	165.73	–	–
Adam	76	73.32	98	277.69

Conclusions

Conclusions

- We have implemeted this method in:
  - KLR
  - DNN
- It seems a promising method
  - Significantly speeds up the convergence of SGD
  - More subtly accelerates the convergence of complex methods such as Adam
- This is still a work in progress

Future work

- When modifying the weights, the internal moments in optimizers like Adam should be updated
- Test this technique on larger problems
  - Large DNNs with thousands or millions of data points, such as images.

Thanks for your attention!

For more information you can contact me at:

José Ángel Martín-Baos
Department of Mathematics
University of Castilla-La Mancha

joseangelmartin.com

JoseAngel.Martin@uclm.es

More info of this research:

UCLM Universidad de Castilla~La Mancha

33rd European Conference on Operational Research

EURO 24 COPENHAGEN