Speeding up the Training of Neural Networks with the One-Step Procedure

Wajd Meskini, Alexandre Brouste, Nicolas Dugué

Neural Processing Letters (2024) 56:178
https://doi.org/10.1007/s11063-024-11637-6

Speeding up the Training of Neural Networks with the One-Step Procedure

Wajd Meskini[1] · **Alexandre Brouste**[1] · **Nicolas Dugué**[2]

Accepted: 5 May 2024 / Published online: 21 May 2024

Abstract
In the last decade, research and corporate have shown a dramatically growing interest in the field of machine learning, mostly due to the performances of deep neural networks. These increasingly complex architectures solved a wide range of problems. However, training these sophisticated architectures require many computation on advanced hardware. With this paper, we introduce a new approach based on the One-Step procedure that may fasten their training. In this procedure, an initial guess estimator is computed on a subsample that is then improved with only one step of the Newton gradient descent on the whole dataset. To show the efficiency of this framework, we consider regression and classification tasks using simulated and real datasets. We consider classic architectures, namely multi-layer perceptrons and show, on our examples, that the One-Step procedure is often halving the computation time to train the neural networks while preserving the performances.

Keywords Multi-layer perceptron · One-step procedure · Binary classification · Regression

1 Introduction

As shown by Cardon et al. [5], neural networks took revenge since 2010. Snubbed during the winter of artificial intelligence, the fast increase in computing capacity of the last decade allowed to exploit their potential, leading to their supremacy when considering machine learning. Indeed, they were successfully operated to solve a wide range of problems: image recognition [26], machine translation and automatic speech recognition [1], etc. However, these networks became deeper and deeper and increasingly complex. Architectures to deal with sequential data are symptomatic of this growing complexity. Recurrent neural networks led to convolutional networks and Long short-term memory architectures [11]. Furthermore, for data requiring bidirectional contexts, transformers and their efficient attention mecha-

✉ Wajd Meskini
wajd.meskini@gmail.com

✉ Alexandre Brouste
Alexandre.Brouste@univ-lemans.fr

Nicolas Dugué
Nicolas.Dugue@univ-lemans.fr

[1] Laboratoire Manceau de Mathématiques, Le Mans Université, Le Mans, France

[2] Laboratoire d'Informatique de l'Université du Mans, Le Mans Université, Le Mans, France

nisms were widely adopted [25]. The Bert language model which is transformer-based is, in its large version, made of 24 layers with 16 attention heads each [8]. With an embedding size of 1028, it leads to 340 millions parameters to train. The training time of such huge architectures has to be done on sophisticated hardware and still requires a large amount of computation and time. That is why we focus on speeding up the training of neural networks. It would help to generalize their use while reducing the high carbon footprint of such use [15, 23]. To do so, we bring from statistics to machine learning the One-Step procedure [16]. The main contribution of this paper is extending and evaluating this procedure for simple neural networks architecture, paving the way to its use to speed up the training of more complex architectures.

One-Step procedure. This procedure was initially considered for the estimation of finite-dimensional parameters with an observation sample composed by independent and identically distributed (i.i.d.) variables [16]. In such procedure, an initial guess estimator is proposed which is fast to be computed but not asymptotically efficient. Then, a single step (hence its name One-Step) of the gradient descent method is done on the log-likelihood function in order to correct the initial estimation and reach asymptotic efficiency. With some recent developments, the One-Step procedure has been successfully generalized to more sophisticated statistical experiments as diffusion processes [9, 13], ergodic Markov chains [14], inhomogeneous Poisson counting processes [6], fractional Gaussian and stable noises observed at high frequency [2, 3].

In order to make it work, the initial sequence of guess estimator is generally supposed to be $\sqrt{n}$-consistent (see Sect. 2.2 for the definition) and the Fisher information matrix uniformly continuous. But it had been recently shown [13, 14] that for a $n^{\frac{\delta}{2}}$-consistent initial sequence of guess estimators (with $\frac{1}{2} < \delta \leq 1$) and a Lipschitz Fisher information matrix, the sequence of One-Step estimators is still consistent, asymptotically normal and efficient. In this setting, for a initial sequence which is neither asymptotically rate nor variance efficient, the new sequence is asymptotically rate and variance efficient. This result allows to use the numerical computation of the Maximum Likelihood estimation (MLE) on a subsample (of size n^{δ}, $\frac{1}{2} < \delta \leq 1$) as a fast initial sequence of guess estimators (see [4] for the practical application in the i.i.d. setting).

In the usual One-Step procedure, a single step of the gradient descent is done on the log-likelihood function. In this paper, we generalize this procedure to allow its use with neural networks optimizing the least-square functional in the regression setting and the cross-entropy in the binary classification setting. Our experiments show that this procedure allows to fasten the training of multi-layer perceptron neural networks, which may thus be of use for a wide panel of applications.

Outline. The notations and the One-Step estimation procedure we introduce for neural networks are presented in Sect. 2. The experimental setup to evaluate the performance of our procedure on multi-layer perceptrons neural networks is described Sect. 3. In this setup, we consider both simulated and real dataset describing classification and regression tasks. Results are detailed Sect. 5 and show that our extended One-Step procedure is at least halving the training time while preserving the performances. Conclusions and perspectives of this method for real applications are given in Sect. 6.

2 Extending the One-Step Estimation Procedure

2.1 Notations

Let $(y_i)_{i=1,\dots,n}$ be the response variable belonging to $\mathbb{R}$ for the regression setting or to $\{-1, 1\}$ for the binary classification setting. Let $(x_i)_{i=1,\dots,n}$ be the dataset of explanatory variables belonging to $\mathbb{R}^d$.

Let us consider a loss functional $\ell_n(\vartheta)$ for the model f (linear model, multi-layer perceptrons neural networks, etc.) characterized by the finite-dimensional parameter $\vartheta \in \Theta \subset \mathbb{R}^d$. For instance, in the regression setting,

$$\ell_n(\vartheta) = \frac{1}{n}\sum_{i=1}^{n} (y_i - f(\vartheta, x_i))^2 \tag{1}$$

and, for the binary classification,

$$\ell_n(\vartheta) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + e^{-y_i f(\vartheta, x_i)}\right). \tag{2}$$

Generally, the training of the model is done by simply minimizing the loss function

$$\widehat{\vartheta}_n = \arg\min_{\vartheta \in \Theta} \ell_n(\vartheta). \tag{3}$$

For multi-layer perceptrons neural networks, the back-propagation equations allows to compute the gradient $\frac{\partial}{\partial \vartheta}\ell_n$ and the Hessian $\frac{\partial^2}{\partial \vartheta^2}\ell_n$ of the function ℓ_n defined in (1) and (2). In the most simple form, without any regularization method, the gradient descent method could be proposed to minimize the loss function and train the model. In this paper, the Newton method for optimization allows to avoid considering the setup of hyper-parameters such as the learning rate, or the mini-batch size in the case of stochastic gradient descent and to focus on the comparison in terms of performance and computation time. This method is not so commonly used in machine learning due to the fact that calculating the inverse of the Hessian can be expansive in computing resources. But the Newton type gradient descent, or its approximations, are efficient to train physically-inspired neural networks [17]. Fastening this method of optimization is yet of great interest in order to allow its use.

2.2 Our One-Step Estimation Procedure

Let us denote $\vartheta \in \Theta \subset \mathbb{R}^d$ the finite-dimensional parameter to be estimated. We propose in this paper the following One-Step procedure to speed up the training of multi-layer perceptrons neural networks: we first train a neural network on a subsample of the data to get a guess estimator that we drastically improve with one step of Newton gradient descent on the whole dataset.

Guess estimator. Firstly, we propose an initial guess estimator $(\widetilde{\vartheta}_n, n \geq 1)$. Here the initial sequence of guess estimator $(\widehat{\vartheta}_n, n \geq 1)$ is the estimator (3) on a subsample of size $\lceil n^\delta \rceil$, $\frac{1}{2} < \delta < 1$. In other words, the parameter δ is a key ingredient in the proposed One-Step method which characterizes the subsample size considered in the initial estimation.

One-Step **estimators.** Then the sequence of One-Step estimators $(\overline{\vartheta}_n, n \geq 1)$ is defined by

$$\overline{\vartheta}_n = \widetilde{\vartheta}_n - \left(\frac{\partial^2}{\partial \vartheta^2} \ell_n(\widetilde{\vartheta}_n) \right)^{-1} \cdot \frac{\partial}{\partial \vartheta} \ell_n(\widetilde{\vartheta}_n), \quad n \geq 1. \tag{4}$$

Heuristically, in the simplest statistical experiments (i.i.d. or generalized linear models with controlled covariates variables as in [18]), if the initial guess estimator $\widetilde{\vartheta}_n$ is $n^{\frac{\delta}{2}}$-consistent (with $\frac{1}{2} < \delta \leq 1$), i.e. for any $\epsilon > 0$ there exists a constant C such that

$$P_\vartheta \left(n^{\frac{\delta}{2}} \left\| \widetilde{\vartheta}_n - \vartheta \right\| > C \right) \leq \epsilon,$$

and the Fisher information matrix (the mean of the opposite of the Hessian) is Lipschitz regular, then the sequence of One-Step estimators defined by (4) is asymptotically equivalent to $\widehat{\vartheta}_n$ defined in (3). The proof in this setting is postponed in Appendix A. In other words, if the initial guess is statistically close enough to the true parameter and the Hessian is sufficiently regular, then a single step in the Newton gradient descent is sufficient to reach the asymptotic efficiency in the calibration procedure.

In the following, we show on simulations and real datasets that the sequential experiment setup described in Sect. 3 which mimicks the One-Step procedure (4) can speed up the training of the multi-layer perceptrons neural networks with the same asymptotic performance.

3 Experimental Setup

In this paper, we consider speeding up the training of neural networks architectures with a sequential experiment setup which mimicks the One-Step estimation procedure. To evaluate the efficiency of our approach, we consider regression and classification tasks with simulated and real datasets. However, we do not compare the results of our systems to the state-of-the-art nor do we claim our models to be better. The aim is to demonstrate that the One-Step procedure that we extended has the potential to speed up the training while preserving performances. We thus compare models trained using a classic Newton type gradient descent with models trained with the One-Step procedure we described in Sect. 2.2, and calculate runtime in addition to performance indicators.

Therefore, in this paper, we aim to demonstrate empirically that the adaptation of One-Step's to neural networks can speed up their training.

3.1 Infrastructure

All results in the paper are computed using Python 3.7.13, scikit-learn 1.0.2, keras 2.8.0 and tensorflow 2.8.2 on Google Colab with a GPU accelerator. In addition, we use the Kormos implementation of a Newton optimizer [12].

3.2 Simulations

Multiple datasets were simulated for regression and classification using randomly generated but fixed random states. At the beginning of the experiment, a random state was fixed arbitrarily and used to generate 10 other random states for dataset simulations, making sure the

method is robust but allowing other users to easily reproduce the results. Results that are presented in Sect. 5 are averaged results of these 10 runs.

In adddition, we vary the sample size throughout the experiment to display the impact of dataset size on model training speed. The values tested were 5.10^4 and 5.10^5.

3.2.1 Simulated Regression Dataset

The method used was scikit-learn's **make_regression** (for more information on the method, see [20]). Here are the data generation parameters:

- n_features: 50, total number of features
- n_informative: 95, total number of features that will be used to generate the target. This is to better simulate real data where some features have no impact on the target
- noise: 0.5, standard deviation of the gaussian noise applied to the target. This is to make predicting the target a little harder

3.2.2 Simulated Classification Dataset

The method used was scikit-learn's **make_classification** (for more information on the method, see [20]) with the generation parameters as follows:

- n_features: 50, total number of features
- n_redundant: 5, total number of features generated as random linear combinations of the informative features. This is to help better simulate a real dataset with some level of feature colinearity
- weights: [0.7], class weights. 0.7 corresponds to 30–70% split between positive and negative values. The imbalance is to simuate a real dataset with some data imbalance.

3.3 Real Datasets

In addition to simulations, real datasets were used to validate our approach. Yolanda and Credit Card Fraud Detection datasets were chosen in order to eliminate the influence of different preprocessing techniques and having enough volume to necessitate the speedup of the training. We described further these datasets in this section.

3.3.1 Regression Dataset: Yolanda

Yolanda is a dataset introduced in Guyon et al. [10] as part of an automl challenge. This dataset is a subsample of the Million song dataset and the regression task consists in predicting the year a song was released from 90 audio features extracted with Echo Nest API. Most of the songs are western, commercial tracks ranging from 1922 to 2011. Yolanda is made of 460.000 songs divided in 400.000 as a training set, 30.000 as validation set and 30.000 as a test set.

3.3.2 Classification Dataset: Credit Card Fraud Detection

The Credit Card Fraud Detection dataset contains transactions made by credit cards in september 2013 by European cardholders. The dataset's target is binary and imbalanced, with the

positive class accounting for only 0.172% of all transactions. In addition, its variables are the result of PCA transformation, making it perfect for our purpose. The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group of Université Libre de Bruxelles on big data mining and fraud detection [7].

4 Models and Training

4.1 Preprocessing

The variables of all datasets are somewhat similar in format, allowing for homogeneous preprocessing across the board - barring some exceptions. All datasets have no missing values and have purely quantitative explicative variables. All variables are normalized in order to have a mean of 0 and standard deviation of 1. Extra care is done during normalization to avoid data leakage.

The one exception is the handling of the "Amount" variable in the Credit Card Fraud Detection dataset, which is replaced by its logarithm in order to reduce its range and limit the effect of outlying values. These steps are inspired from tensorflow's tutorial on that dataset [19].

After preprocessing, the datasets go through a train test split. If the task is classification, the split is stratified according to the target variable. In all cases except Fraud Detection, 10% of the data is dedicated to testing. For Fraud Detection, we use 30% of the data given the small number of positive values.

4.2 Model Architectures

Models used change based on dataset difficulty, but they share the same overall simple design. All models features two layers and a batch size equal to the size of the dataset (Newton optimization). Such simple architectures are commonly used in physically-inspired neural networks, for instance to solve differential equations [22, 24]. For such networks, because of the use of the Hessian to guide the convergence with Newton, the training may actually be better than with Adam. Furthermore, there is no such hyper-parameters as batch size and learning rate that requires much computation to tune. In addition, we used Michael B Hynes's Python implementation of a Newton-cg optimizer for keras in the Kormos Python library [12].

For all models, sigmoid is used as the activation function for all layers (except in the prediction layer for regression models). The sigmoid's structure has been chosen in order to meet the usual regularity assumptions to make the One-Step procedure works, although other activation functions (such as ReLU) may give better empirical results.

Finally, classification models were initialized in order to have them predict the percentage of positive cases at initialization - effectively initializing the bias to $\log(pos/neg)$, pos being the number of positive values, and neg being the number of negative values. This helps convergence in imbalanced datasets, as described in the tensorflow imbalanced datasets tutorial [19]. In addition, class weights with a factor of 2 are introduced during model training in order to make learning to predict the positive classes easier.

4.3 The Experiment

After preprocessing and train/test splitting, we begin by training the One-Step model (see One-Step experiments below). We save the training time and the performances for a specific metric on the test set (see Sect. 4.4). For simulated datasets, we calculate the average training time and performance (on the test set) after training separately on every dataset. For real datasets, a 10-fold cross-validation is done in order to get the average training time.

Then we train the control model with a callback stopping training when it overtakes the performance (on the test set) of the One-Step model trained on the same sample.

Finally we compare the training time, one trained using the One-Step method, and the other being the control model with a classic Newton gradient descent. The comparison is made in terms of average training time on the 10 simulations (or 10-fold cross-validation). We also compare the performance of the two models in terms of performances on the test set.

4.3.1 One-Step Experiments

The One-Step training experiment starts with a data split based on a δ value that allows us to take a subset of the train set for training. Larger values of δ allowing for larger portions of the dataset in the initial training. In addition, this subset would be stratified by the target value in case of classification.

Therefore, we initially train the model on p epochs of the same subsample and then perform one final epoch on the whole dataset (One-Step procedure). The parameter p may vary, depending on the dataset.

4.3.2 Control Experiment

The control training experiment (written Ctrl in tables of results) goes through the training of model on m epochs with a callback stopping it when it reaches the One-Step model's performance on the same sample in terms of loss function on the test set. The m epochs are never effectively reached due to the callback - model training stops when the evaluation metric is overtaken.

The callback depends on the score attained by the One-Step experiment relative to the dataset sample in question. For simulated datasets, this means that we compare the control score at iteration level with the score attained by the One-Step procedure on the respective dataset simulation in question. For real datasets, this means comparing the score attained at the respective cross-validation fold.

4.4 Metrics

For regression tasks, we use the Mean Average Error (MAE) metric to evaluate our models. For classification tasks, given the data imbalance, we use the Area Under the Precision-Recall Curve (AUPRC).

5 Results

The One-Step procedure outperforms the classical approach in terms of computation time with the same asymptotic performances in terms of precision. However, this is only true for very large datasets, as when dataset size is small, differences in execution time are minimal. Detailed results are described in the section below.

5.1 Regression on Simulation Data

The specific parameters of this experiment are as follows:

- $p = 5$, number of epochs trained on a δ dependant subsample pre-One-Step;
- $\delta = 0.9$, allows us to compute what share of the dataset to use in the pre-One-Step iterations;
- Test set size: 10%;
- Model Architecture: Two fully-connected sigmoid layers ($256 \rightarrow 128$) followed by one prediction neuron.

In this first experiment, we vary the sample size and observe the average training time with the control approach, and with the extended One-Step that we introduced. We can observe Table 1 that for larger datasets, the One-Step method is faster then the control on (Ctrl) for a similar performance: roughly 2, 3 times faster for 500.000 data.

In our second experiment, we aim to understand the impact of the hyper-parameter δ of our One-Step procedure introduced Sect. 2.2. This parameter controls the size of the subsample used to compute the guess estimator, having thus an influence on the quality of this guess estimator. In particular, we monitor the average training time on this parameter on the dataset made of 500.000 observations. As one can see on Fig. 1, the lower the δ, the lower the training time. Furthermore, with this sample size, even with low δ, the guess estimators are good enough: one step of Newton gradient (One-Step) on the whole data allows to converge to the same results as the control.

Training time remains somewhat similar for models trained using the control procedure, i.e. the classic Newton gradient descent. Their variation is due to the variations of model performance of their respective One-Step training iterations, since the iteration score is used in the callback.

5.2 Classification on Simulated Data

The specific parameters of this experiment are as follows:

- $p = 5$, number of epochs trained on a δ dependant subsample pre-One-Step;
- $\delta = 0.85$, allows us to compute what share of the dataset to use in the pre-One-Step iterations;

Table 1 Results of the One-Step and control approaches (Ctrl) on simulated data for regression

n_samples	500.000		50.000	
Method	One-Step	Ctrl	One-Step	Ctrl
Avg training time (s)	12.29	27.86	4.63	3.47
Avg MAE	220.4	220.4	243.4	241.2

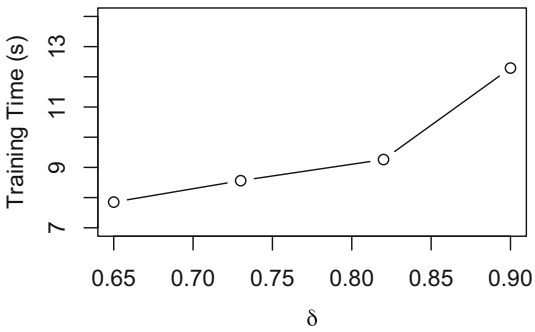

Fig. 1 Influence of the δ parameter regarding the average time to train the systems (in seconds) on a regression dataset made of 500.000 observations. Mean time for Ctrl is 26.785 s for comparison

Table 2 Results of the One-Step and control approaches on simulated data for classification

n_samples	500.000		50.000	
Method	One-Step	Ctrl	One-Step	Ctrl
Avg training time (s)	4.35	6.32	3.14	3.27
Avg AUPRC	0.74	0.74	0.81	0.81

- Test set size: 10%;
- Model Architecture: Three fully-connected sigmoid layers ($256 \rightarrow 128 \rightarrow 1$);
- A bias initializer element depending on target imbalance as per the Tensorflow imbalanced classification tutorial [19].

Results of Table 2 show a similar trend to the regression experiment, in the sense that average training times are significantly lower for the One-Step experiment than in the control (Ctrl) in the larger dataset while preserving performances of classification given by the AUPRC. These resultats are thus very encourageing, which is why we aim to confirm these trends on real datasets in the rest of this section.

5.3 Regression Dataset: Yolanda

The specific parameters of this experiment are as follows:

- $p = 8$, number of epochs trained on a δ dependant subsample pre-One-Step;
- $\delta = 0.9$, allows us to compute what share of the dataset to use in the pre-One-Step iterations;
- Test set size: 10%;
- Model Architecture: Two fully-connected sigmoid layers ($64 \rightarrow 16$) and one prediction neuron;
- Number of folds: 10, number of cross-validation folds.

The gap in training time in this experiment is lower compared to the regression experiments on simulated data as one can see in Table 3, especially regarding the huge size of Yolanda

Table 3 Results on the Yolanda regression dataset

Method	One-Step	Ctrl
Avg training time (s)	4.30	6.07
Avg MAE	8.11	8.00

Table 4 Results on the Credit card fraud detection classification dataset

Method	One-Step	Ctrl
Avg training time (s)	5.07	27.00
Avg AUPRC	0.72	0.74

(400.000 observations in the train set). This is due to the fact that the model considered is significantly simpler with a lower number of neurons involved. Still, this displays a pattern that the size of the model has an influence on the time gained through the One-Step procedure in absolute terms.

5.4 Classification Dataset: Credit Card Fraud Detection

- $p = 4$, number of epochs trained on a δ dependant subsample pre-One-Step;
- $\delta = 0.9$, allows us to compute what share of the dataset to use in the pre-One-Step iterations;
- Test set size: 30%;
- Model Architecture: Three fully-connected sigmoid layers ($512 \rightarrow 256 \rightarrow 1$);
- Number of folds: 10, number of cross-validation folds;
- A bias initializer element depending on target imbalance as per the Tensorflow imbalanced classification tutorial [19].

As one can see Table 4, the gap in this experiment is significant, One-Step is roughly five times faster to train than with the control procedure (Ctrl). This is due both to the more complex model structure with more neurons involved, and to the dataset size (over 280.000 samples).

6 Conclusions and Perspectives

In this paper, we introduce a new extended One-Step procedure to deal with the training of neural networks. To prove the relevance of this approach, we consider classic multi-layer perceptrons that are trained using Newton optimization, which often suffers from its computing time. On simulated and real dataset, we have demonstrated that the One-Step methodology allows to speed up the training time while keeping the same asymptotic performance in terms of precision. Our One-Step based approach seems to be particularly efficient to deal with large dataset, speed up is also more impressive on bigger architectures as shown with the credit card fraud detection example.

It is worth mentioning that other experiments without early stopping were also conducted. They show that, on the aforementioned datasets, the performances of the One-Step and the control methods are similar.

To further demonstrate the relevance of this One-Step procedure, we aim at extending its application to a wider range of problems, and thus to bigger and more complex architectures. Other contributions could be made considering other optimizers for the initial guess. It would be for example interesting to experiment the relevance of such a One-Step approach when stochastic gradient descent is used. Furthermore, it would be interesting to develop a framework to efficiently setup the hyper-parameters of the One-Step procedure, i.e. δ, the size of the subsample, and the number of Newton gradient descents to run on the subsample to get

the most optimal speed up while preserving the performances. Eventually, we supposed here some regularity of the functionals using sigmoids helping us to understand the machinery. But simulations with ReLU activations functions are also encouraging and needs to be further studied theoretically.

7 Appendix A

7.1 Proof

The mean-value theorem gives, for the initial sequence of guess estimators $(\widetilde{\vartheta}_n, n \geq 1)$,

$$\begin{aligned}\frac{\partial}{\partial\vartheta}\ell_n\left(\widetilde{\vartheta}_n\right) &= \frac{\partial}{\partial\vartheta}\ell_n\left(\widehat{\vartheta}_n\right) + \int_0^1 \frac{\partial^2}{\partial\vartheta^2}\ell_n\left(\widehat{\vartheta}_n + v\left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right)\right) dv \cdot \left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right) \\ &= \int_0^1 \frac{\partial^2}{\partial\vartheta^2}\ell_n\left(\widehat{\vartheta}_n + v\left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right)\right) dv \cdot \left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right) \end{aligned} \tag{5}$$

since $\frac{\partial}{\partial\vartheta}\ell_n\left(\widehat{\vartheta}_n\right) = 0$ by definition. From (4), we have

$$\left(\overline{\vartheta}_n - \widehat{\vartheta}_n\right) = \left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right) - \left(\frac{\partial^2}{\partial\vartheta^2}\ell_n(\widetilde{\vartheta}_n)\right)^{-1} \cdot \frac{\partial}{\partial\vartheta}\ell_n(\widetilde{\vartheta}_n)$$

and

$$\left(\overline{\vartheta}_n - \widehat{\vartheta}_n\right) = \left(I_p - \left(\frac{\partial^2}{\partial\vartheta^2}\ell_n(\widetilde{\vartheta}_n)\right)^{-1} \int_0^1 \frac{\partial^2}{\partial\vartheta^2}\ell_n\left(\widehat{\vartheta}_n + v\left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right)\right) dv\right)\left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right)$$

where I_p is the $p \times p$ identity matrix.

In the regular setting, if the sequence of initial guess estimators is $\sqrt{n}$-consistent and the Hessian possess some kind of uniform continuity, we get the asymptotic equivalence by showing that

$$\begin{aligned}\sqrt{n}\left(\overline{\vartheta}_n - \widehat{\vartheta}_n\right) &= \underbrace{\left(I_p - \left(\frac{\partial^2}{\partial\vartheta^2}\ell_n(\widetilde{\vartheta}_n)\right)^{-1} \int_0^1 \frac{\partial^2}{\partial\vartheta^2}\ell_n\left(\widehat{\vartheta}_n + v\left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right)\right) dv\right)}_{\text{tending to 0}} \cdot \underbrace{\sqrt{n}\left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right)}_{\text{bounded in probability}} \\ &\longrightarrow 0 \quad \text{in probability}\end{aligned}$$

In the more recent result, the sequence of initial guess estimator is only $n^{\frac{\delta}{2}}$-consistent and we need more regularity on the loss function to make it work. Heuristically,

$$\begin{aligned}\sqrt{n}\left(\overline{\vartheta}_n - \widehat{\vartheta}_n\right) = \underbrace{n^{\frac{1}{2}-\delta}}_{\substack{\text{tending to 0 if } \frac{1}{2} < \delta \leq 1}} \underbrace{\left(\frac{\partial^2}{\partial\vartheta^2}\ell_n(\widetilde{\vartheta}_n)\right)^{-1}}_{\text{bounded with the LLN}} \cdot \underbrace{n^{\frac{\delta}{2}}\left(\widetilde{\vartheta}_n - \widehat{\vartheta}_n\right)}_{\text{bounded in probability}} \\ \underbrace{\cdot n^{\frac{\delta}{2}}\left(\left[\frac{\partial^2}{\partial\vartheta^2}\ell_n(\widetilde{\vartheta}_n) - \frac{\partial^2}{\partial\vartheta^2}\ell_n(\widehat{\vartheta}_n)\right] - \int_0^1 \left[\frac{\partial^2}{\partial\vartheta^2}\ell_n\left(\widehat{\vartheta}_n + v(\widetilde{\vartheta}_n - \widehat{\vartheta}_n)\right) - \frac{\partial^2}{\partial\vartheta^2}\ell_n(\widehat{\vartheta}_n)\right] dv\right)}_{\text{bounded in probability, see below}}.\end{aligned}$$

A Lipschitz condition on the Hessian function (obtained for instance when ℓ has three continuous derivatives (for smooth multi-layer perpcetrons neural networks)), gives for instance,

that

$$\left\| \frac{\partial^2}{\partial \vartheta^2} \ell_n \left(\widetilde{\vartheta}_n \right) - \frac{\partial^2}{\partial \vartheta^2} \ell_n (\widehat{\vartheta}_n) \right\| \leq L \| \widetilde{\vartheta}_n - \widehat{\vartheta}_n \|$$

and the boundedness of the last term in the r.h.s..

Acknowledgements This research benefited from the support of the ANR projects "Efficient Inference for large and high-frequency data" (ANR-21-CE40-0021) and "Dynamic and Interpretable Graph-based word embeddINGs" (ANR-21-CE23-0010) and the "Chair Efficience et Sobriété Numériques", under the aegis of Fondation du Risque, a joint initiative by Le Mans University and EREN Groupe.

References

1. Boito MZ, Ortega J, Riguidel H, Laurent A, Barrault L, Bougares F, Estéve Y (2022). ON-TRAC consortium systems for the IWSLT 2022 dialect and low-resource speech translation tasks. arXiv preprint arXiv:2205.01987
2. Brouste A, Masuda H (2018) Efficient estimation of stable Lévy process with symmetric jumps. Stat Inference Stoch Process 21:289–307
3. Brouste A, Soltane M, Votsi E (2020) One-step estimation for the fractional Gaussian noise model at high-frequency. ESAIM Probab Stat 24:827–841
4. Brouste A, Dutang C, Noutsa Mieniedou D (2021) OneStep - Le Cam's onestep estimation procedure. R Journal 13(1):366–377
5. Cardon D, Cointet JP, Maziéres A (2018) Neurons spike back. Réseaux 211(5):173–220
6. Dabye A, Gounoung A, Kutoyants Y (2018) Method of moments estimators and multi-step MLE for Poisson processes. J Contemp Math Anal 53(4):187–196
7. Dal Pozzolo A, Caelen O, Johnson RA, Bontempi G (2015) Calibrating probability with undersampling for unbalanced classification. In: IEEE, symposium on computational intelligence and data mining (CIDM)
8. Devlin J, Chang MW, Lee K, Toutanova K (2018) Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805
9. Gloter A, Yoshida N (2021) Adaptive estimation for degenerate diffusion processes. Electron J Stat 15(1):1424–1472
10. Guyon I, Sun-Hosoya L, Boullé M, Escalante HJ, Escalera S, Liu Z, Viegas E (2019) Analysis of the automl challenge series. Autom Mach Learn 177:177–219
11. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
12. Hynes MB, Kormos, (2022) Software available on pypi.org/project/kormos/
13. Kamatani K, Uchida M (2015) Hybrid multi-step estimators for stochastic differential equations based on sampled data. Stat Inference Stoch Process 18:177–204
14. Kutoyants Y, Motrunich A (2016) On multi-step MLE-process for Markov sequences. Metrika 79:705–724
15. Lannelongue L, Grealey J, Inouye M (2021) Green algorithms: quantifying the carbon footprint of computation. Adv Sci 8(12):2100707
16. Le Cam L (1956) On the asymptotic theory of estimation and testing hypothesis. In: Proceedings of the 3rd Berkeley Symposium, 1, 355–368
17. Lu L, Meng X, Mao Z, Karniadakis GE (2021) DeepXDE: a deep learning library for solving differential equations. SIAM Rev 63(1):208–228
18. Lumley T (2019) Fast generalized linear models by database sampling and one-step polishing. J Comput Graph Stat 28(4):1007–1010. https://doi.org/10.1080/10618600.2019.1610312

19. Martín A, Ashish A, Paul B, Eugene B, Zhifeng C, Craig C, Greg SC, Andy D, Jeffrey D, Matthieu D, Sanjay G, Ian G, Andrew H, Geoffrey I, Michael I, Rafal J, Yangqing J, Lukasz K, Manjunath K, Josh L, Dan M, Mike S, Rajat M, Sherry M, Derek M, Chris O, Jonathon S, Benoit S, Ilya S, Kunal T, Paul T, Vincent V, Vijay V, Fernanda V, Oriol V, Pete W, Martin W, Martin W, Yuan Y, Xiaoqiang Z (2015) TensorFlow: large-scale machine learning on heterogeneous systems, Software available from tensorflow.org
20. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830
21. R Core Team (2022) R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/
22. Shen Z, Yang H, Zhang S (2021) Neural network approximation: three hidden layers are enough. Neural Netw 141:160–173
23. Strubell E, Ganesh, A, McCallum, A (2019). Energy and policy considerations for deep learning in NLP. arXiv preprint arXiv:1906.02243
24. Taylor J, Wang W, Bala B, Bednarz T (2022) Optimizing the optimizer for data driven deep neural networks and physics informed neural networks. arXiv preprint arXiv:2205.07430
25. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Polosukhin I (2017) Attention is all you need. Adv Neural Inf Process Syst 30
26. Zhai X, Kolesnikov A, Houlsby N, Beyer L (2022) Scaling vision transformers. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition pp 12104–12113