Tuning Neural network hyperparameters through Bayesian optimization and Application to cosmetic formulation data

Mathilde Guillemot[1] Catherine Heusèle[1] Rodolphe Korichi[1]
Syvianne Schnebert[1] Maxime Petit[2] Liming Chen[2]

[1] LVMH Recherche - Parfums et Cosmétiques
[2] LIRIS UMR 5205 CNRS - Ecole centrale de Lyon

mguillemot@research.lvmh-pc.com

Abstract

A major issue in machine learning is to select the best hyperparameters of a predictive model without over-fitting. In this paper, we propose to study through a principled way the hyperparameter optimization in a neural network designed for a classification problem on cosmetic formulation data. Specifically, we propose to make use of Bayesian optimization (BO) to automatically choose the next hyperparameter set to try based on previous observations and a surrogate function of the model. This BO-based hyperparameter selection method is compared to the popular grid search method. A 2-hidden-layer fully-connected neural network is trained on cosmetic formulation data. Extensive experiments show that hyperparameters found with Bayesian optimization outperform the grid search hyperparameter set. Moreover, Bayesian optimization only needs 80 evaluations while 140 are used for the grid search.

Keywords

Hyperparameters optimization; Bayesian optimization; cosmetic formulation data; neural network

1 Introduction

Neural networks are the gold standard for machine learning and big data problems. One advantage of those models is their high flexibility and capacity to fit almost every data. The major drawback is the difficulty to find the good architecture and to tune hyperparameters. Determining those parameters is a major concern to reach the best possible results. Classical techniques using grid search or random search need many different experiments and are time-consuming. Using Bayesian optimization is a solution widely used in recent years to help to find good hyperparameters while using as less calculation as possible. Unlike grid search, it uses former calculations to find the next hyperparameter set to evaluate. In case where data are known not to be qualitative, it is difficult to know if the mixed results are due to the low data quality or the model fine tuning. The issue is to calibrate the time to spend on the model.

This is the case we are studying while dealing with some cosmetic formulation data. Cosmetic products are formulated to provide care such as moisturizing, anti-aging or anti-oxidant effect to the skin. A good cosmetic product must not only be efficient but also have a good sensoriality, a nice perfume, a good consistency, should be stable and reproducible. Behind all those properties, scientific procedures are set to measure quantitative efficiency. Therefore, cosmetic formulation is a complex task and involves many ingredients. Effects are measured on volunteers' skin, a complex and dynamic biological system, introducing uncertainty in the data set. The aim of the model is to predict short term moisturizing effect of a skincare product based on its chemical formulation. The main contribution of this work is the application of Bayesian optimization (BO) to tune the hyperparameters of a neural network designed to deal with cosmetic formulation data. Through extensive experiments, we have shown the interest and superiority of using BO for a principled hyperparameter tuning in comparison with the popular grid based search.

In this work, the performances achieved by a neural network tuned with Bayesian optimization on the data set are compared with the one delivered by the same neural network tuned with a classical grid approach.

The paper is organized as follows. Section 2 is a state of the art on the related work. Section 3 states the problem. Section 4 details the proposed method. Section 5 analyzes the experimental results. Section 6 concludes the paper.

2 State of the art

Bayesian optimization has been widely used for few years. Tutorials have been written for instance for proving the importance of the Gaussian process choice during optimization[7], and explaining theoretical basis of Bayesian optimization[2], [5], [6]. In literature, articles also give practical examples where Bayesian optimization outperforms other techniques. In [7] Bayesian optimization is compared to grid search to tune hyperparameters of LDA (Latent Dirichlet allocation) applied to a collec-

tion of Wikipedia articles. It has been found that BO finds the best parameters in significantly less time. BO was also used to find the hyperparameters of latent structured SVMs for a problem of binary classification of protein DNA sequences. BO proved to be much better than grid search. The last model studied in this paper is a neural network on CIFAR-10 where Bayesian optimization also surpasses human expert level tuning.
Further results are reported in paper [1] where tree Parzen estimator is analyzed against random search for LFW and pubFid data sets. The same algorithm is used on the CIFAR-10 data set where Bayesian optimization finds almost the same parameters as a human expert.

3 Problem Statement

In our applicative context, we aim to build a model able to predict the effect of a cosmetic product on skin from the formulation. In this work a formulation means a list of chemical ingredients with the associated concentrations. The moisturizing effect is measured with a device named corneometer which measures the skin capacitance. Final result, expressed in percentage, translates a difference computed before and after application of the product on the arms of a volunteer panel. Capacitance is directly linked with the moisturizing level of the skin. Data were collected anonymously.
Based on raw data, predicted effects are then filed in three classes (see Table 1).

Class	Raw data	Interpretation
0	data<23%	No moisturizing power or dehydrating formulas
1	23%<data<54%	Low moisturizing power
2	54%<data	Good moisturizing power

Table 1 – Class description for moisturizing effect

Data set is composed of 4015 formulas described by 2039 features. Data set is imbalanced (see Table 2).

Train set		Test set	
Class 0	1329	Class 0	373
Class 1	1260	Class 1	522
Class 2	391	Class 2	140

Table 2 – Class description for moisturizing effect

To split the data set in test and training sets, the Kennar-Stone algorithm is used. It allows to select samples with a uniform distribution over the predictor space. Stratified k-fold is also used to build the cross-validation folds inside the training set. Each cross-validation fold should have the same distribution as the whole training set.
A 5-fold cross validation is exploited to pick the best hyperparameter set either on grid search or Bayesian optimization. Another set is kept to test the efficiency of the hyperparameter set.

In both cases, a 2-hidden-layer neural network is studied. A batch normalization is used before every hidden layer and reLu function is used as neuron activation function. We choose to set some parameters as:

- the mini batch size of train and test phase is set to 75
- the dropout rate is maintained to 0.5
- a learning rate schedule is also programmed dividing learning rate by 10 every 30 epochs

The same cost function adapted to an ordinal-regression problem is employed, it is based on the Cohen's Kappa coefficient [3]. Adam optimization is used to update weights and biases after each iteration. The weights are initialized with the He et al weight initialization[4]. The learning rate η is a hyperparameter that must be tuned. The dataset with which we are working is imbalanced. All minibatches are artificially balanced adding samples until the class are balanced.
L_1-regularization is used for sparsity and L_2-regularization to avoid infinite weights, so the loss function is written as equation 1.

$$loss = loss_{Cohen's\ Kappa} + \lambda_1 \|w\| + \lambda_2 \|w\|^2 \qquad (1)$$

The above loss function thus introduces two other hyperparameters (see Table 3), namely the parameter for the lasso regularization λ_1 and the parameter for the ridge regularization λ_2.

Hyperparameter symbol	Hyperparameter name
η	Learning rate
λ_1	Lasso regularization parameter
λ_2	Ridge regularization parameter

Table 3 – Hyperparameters to tune

4 The method

In this section, a first part explains in detail the practical Bayesian optimization method used in this paper to optimize neural network hyperparameters. A second part takes in account on a generalization of Area Under ROC Curve (AUC) for multiclass problems.

4.1 Bayesian optimization

The idea behind Bayesian optimization is to build a probabilistic model of an objective function and use it to select the most promising hyperparameters to evaluate a real objective function.
When the goal is to find the best hyperparameters for a neural network automatically, the objective function is a performance indicator such as accuracy or AUC. Let be X the space of possible hyperparameters, the objective function is marked f. We want to find x^* such as $x^* = argmax_{x \in X}\ f(x)$.

The objective function is a black box, its expression is unknown, it cannot be analyzed, and its derivatives are unknown either. In addition to find extrema of a complex function, BO does it using the less function evaluation as possible.
Let x_i be the ith hyperparameter set and $f(x_i)$ the evaluation of the objective function at this point. Data are aggregated as : $D_{(1:t)} = \{(x_{(1:t)}, f(x_{(1:t)})\} = \{x_1, x_2, \dots, x_t), (f(x_1), f(x_2), \dots, f(x_t))\}$. BO is an iterative process, from known observations $D_{(1:t)}$ it decides which point, hyperparameter set, x_{t+1} to try next.

Bayes' theorem. The principle of BO belongs on Bayes' theorem, three probabilities must be introduced:

- The posterior probability of a model knowing the evidence : $P(M \mid E)$
- The likelihood of evidence E knowing model M : $P(E \mid M)$
- Prior probability of M : $P(M)$

Bayes' theorem indicates that the posterior probability is proportional to the likelihood times the prior probability.

$$P(M \mid E) \propto P(M \mid E) * P(M) \tag{2}$$

Transposing equation 2 to our case, Bayes' theorem is written :

$$P(f \mid D_{1:t}) \propto P(D_{1:t} \mid f) * P(f) \tag{3}$$

It uses the evidence already computed and the prior knowledge to maximize the posterior probability on each point of X. The posterior captures the update knowledge about the unknown function. Gaussian processes are common functions to surrogate f.

Gaussian process. The goal is to find an expression of $P(f_{t+1} \mid D_{1:t}, x_{t+1})$. With this expression it is then possible to get the next hyperparameter set x_{t+1} that should be evaluated.
The same way a Gaussian distribution is completely described by its mean and variance, a Gaussian process is completely described by a mean function m and a covariance function k.

$$f(x) \sim GP(m(x), k(x, x')) \tag{4}$$

Gaussian process returns the mean and variance of a normal distribution over the possible values of f on x.
According to [2], if mean is set to 0, the posterior distribution can be written as 5

$$P(f_{t+1} \mid D_{1:t}, x_{t+1}) = \mathcal{N}(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1})) \tag{5}$$

where :

$$\mu_t(x_{t+1}) = \mathbf{k}^T \mathbf{K}^{-1} \mathbf{f}_{1:t}$$
$$\sigma_t^2(x_{t+1}) = k(x_{t+1}, x_{t+1}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k}$$

and

$$\mathbf{k} = \begin{bmatrix} k(x_{t+1}, x_1) & k(x_{t+1}, x_2) & \dots & k(x_{t+1}, x_t) \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} k(x_1, x_1) & \dots & k(x_1, x_t) \\ \vdots & \ddots & \vdots \\ k(x_t, x_1) & \dots & k(x_t, x_t) \end{bmatrix}$$

The covariance function is a kernel controlling the smoothness and amplitude of the Gaussian process samples. The mean provides a possible offset, in practice the mean is kept constant: $\mu_0(x) = \mu_0$.
The interesting part is the choice of the covariance function. If two points (two hyperparameter sets) are close, they must have a high influence on each other whereas if they are far away, the influence of a point on the other is barely nonexistent. Widely used kernel are Matérn's ones, they are parametrized by a smoothness parameter $\varsigma > 0$ (see equation 6).

$$k(x_i, x_j) = \frac{1}{2^{\varsigma-1}\Gamma(\varsigma)} (2\sqrt{\varsigma}\|x_i - x_j)\|^{\varsigma} H_{\varsigma}(2\sqrt{\varsigma}\|x_i - x_j\|) \tag{6}$$

With Γ the gamma function and H_{ς} the Bessel function of order ς. It is convenient to use $\varsigma = p + \frac{1}{2}$ with p a natural number. In that particular case, the covariance function is the product of an exponential and a p-order polynomial. For its application to machine learning, the most interesting ς are $\varsigma = \frac{3}{2}$ or $\varsigma = \frac{5}{2}$ [6].

Acquisition function. Once $P(f \mid D_{1:t})$ is estimated, an acquisition function is used to find the point (a hyperparameter set) where f is supposed to be maximal. This decision always relies on a trade-off between exploration (sets where acquisition function is not known at all) and exploitation (values where acquisition function should have high value). During exploration, points with high variance are evaluated, during exploitation those points have high mean.
Classically, three acquisition functions are used: probability of improvement, expected improvement or GP Upper Confidence Bound (GP-UCB). We choose to work with expected improvement which behaves better that probability of improvement and does not require hyperparameter tuning as it is the case with GB-UCB [7].
Intuitively, the expected improvement is the non-negative improvement expected from the previous best observed value.
Writing $x^+ = argmax_{x_n} f(x_n)$, expected improvement can be analytically evaluated as shown in equation 7

$$EI(x) = \begin{cases} (\mu(x) - f(x^+))\Phi(\frac{\mu(x) - f(x^+)}{\sigma(x)}) \\ +\sigma(x)\phi(\frac{\mu(x) - f(x^+)}{\sigma(x)}), \text{if } \sigma(x) > 0. \\ 0, \text{if } \sigma(x) = 0. \end{cases} \tag{7}$$

where ϕ is the normal distribution probability density function and Φ its cumulative density function.

Summary and example. Bayesian optimization works, building a posterior function distribution, using Gaussian processes and evidences, permitting to describe the function we are trying to optimize. When the number of evidence grows, the posterior distribution is more precise, and the algorithm is more certain of hyperparameter region it should exploit. In Fig. 1 there is an example of Bayesian optimization search for maxima for a one-dimension function.
The Bayesian algorithm is described in algorithm 1.

for *t=1,2,...* **do**
  select new $_{t+1}$ by optimizing acquisition function EI;

$$x_{t+1} = argmax_x \; EI(\mathbf{x}; \mathbf{D}_{1:t})$$

  ;
  Find the objective value data at x_{t+1};
  Augment data $\mathbf{D}_{1:t+1} = \{\mathbf{D}_{1:t}, (x_{t+1}, f(x_{t+1})\}$;
  Update statistical model;
end

Algorithme 1 : Bayesian optimization algorithm

An example of the first steps of a BO algorithm applied to a function with one variable are shown in figure 1.
There are some drawbacks using Bayesian optimization:

- Finding the hyperparameters and specifically the kernel of the gaussian process as a surrogate function for prior distribution is a major concern. There are many different kernels more or less complex possible
- This method can turn out to be very long when working with many parameters
- The choice of surrogate function might be critical, gaussian processes are not always the best solution [2].
- It is unclear how to handle trade-off between exploration and exploitation in the acquisition function. Too much exploration leads to many iterations without seeing any improvement, while with extreme exploitation, we might fall into a local extremum [2].

4.2 AUC for multiclass

A function must be chosen to be the objective function optimized by the Bayesian optimization procedure. A classical choice is the global accuracy of the model. In our cosmetic formulation data set, classes are highly imbalanced, see Table 2, particularly formulas with very very high moisturizing power are underrepresented. This is the reason why we choose to study AUC as objective function of BO.
AUC is a measure designed for binary problems. It computes the area under ROC curve representing true positive rate as a function of false positive rate. AUC can be adapted for a multiclass classification problem, using a one-versus-all strategy. It will produce one curve for each class and so one measure for each class. In order to plot those curves, for all class c, the number of true positives (TP_c), false positives (FP_c), false negatives (FN_c) and false negatives (FN_c) must be known.
Besides a AUC value for each class, we need to compute a global AUC measure able to provide information on the whole model. There are two ways to compute a mean of those AUC per class and then obtain a performance measure for the model:

- A macro-average (8) computes the metric independently for each class and then take the average. It treated all classes equally.
- On the contrary micro-average aggregates the contributions of all classes to compute the average metric (9).

When classes are imbalanced, micro-average is preferably used.

$$AUC_{macro} = \frac{1}{C} \sum_{c=0}^{C-1} AUC(TP_c, FP_c, TN_c, FN_c). \quad (8)$$

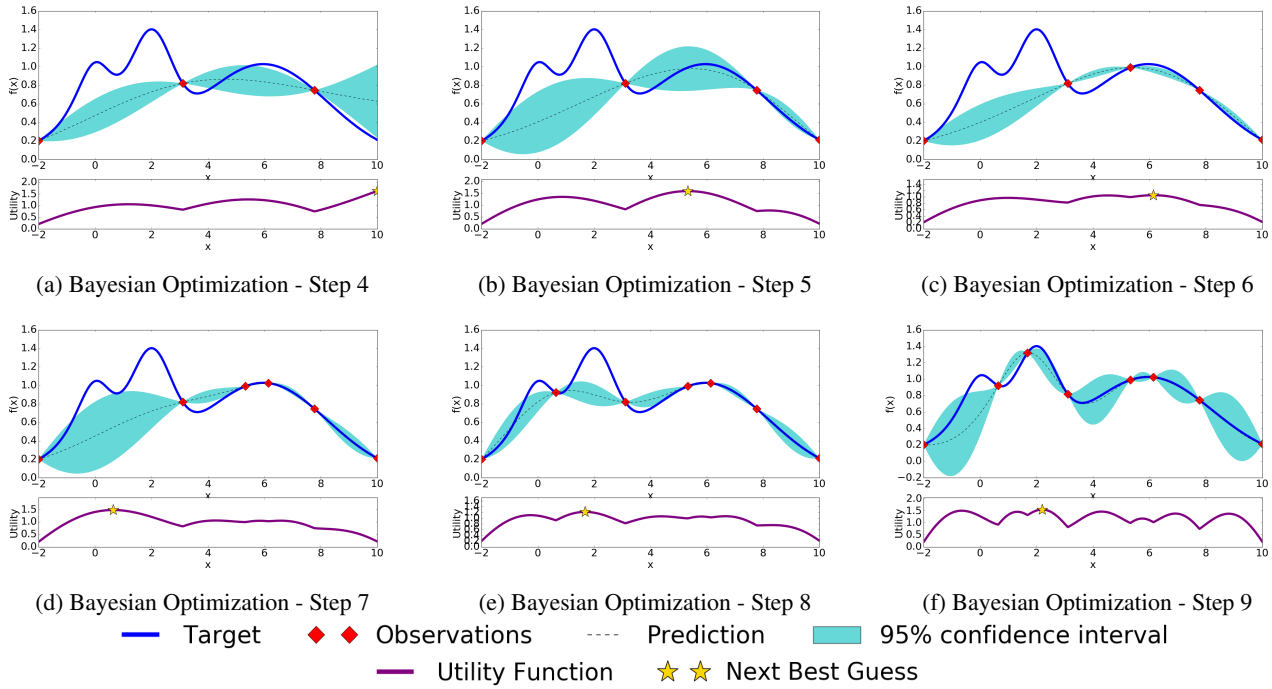
$$AUC_{micro} = AUC(\sum_{c=0}^{C-1} TP_c, \sum_{c=0}^{C-1} FP_c, \sum_{c=0}^{C-1} TN_c, \sum_{c=0}^{C-1} FN_c). \quad (9)$$

5 Experiments

In this work, performances of two hyperparameter sets on the cosmetic formulation dataset are compared. One set is found using a grid search, the other using Bayesian optimization. For each method, the evaluation criterion chosen is AUC, which is expected to be as high as possible. For both hyperparameters search, trials are limited to 100 attempts, because evaluations are time consuming. Consequently, grid search is restricted to three hyperparameters while, this number can be increased working with Bayesian optimization. A fourth hyperparameter is tuned with the Bayesian optimization procedure.

5.1 Grid search

For this procedure, the number of neurons is set to 100 for each hidden layer. There are three hyperparameters to set (learning rate η, lasso regularization parameter λ_1 and ridge regularization parameter λ_2). Those parameters can only take some value previously defined : $\eta \in [10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}]$, $\lambda_1 \in [10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}, 10^{-9}]$ and $\lambda_2 \in [10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}, 10^{-9}]$.
All possible combinations are tried, analyzing which ranges give higher AUC, another run is set up to obtain more precise results. At the end, only the combination giving best performances is kept. This mean that 100 runs are needed just for the first phase. Based on the results of the



(a) Bayesian Optimization - Step 4 (b) Bayesian Optimization - Step 5 (c) Bayesian Optimization - Step 6

(d) Bayesian Optimization - Step 7 (e) Bayesian Optimization - Step 8 (f) Bayesian Optimization - Step 9

Figure 1 – Evolution of Bayesian optimization over 6 steps. Every figure present two graphs, on the first: blue line is the target function, black dotted line the mean prediction function, surround blue area is the 95% confidence interval and red points the evaluated observation. The second graph represent the variation acquisition function along each value

first phase, results are refined. This leads to another 40 runs trying values with $\eta \in [10^{-1}, 5.10^{-2}, 10^{-3}, 10^{-2}, 10^{-3}]$, $\lambda_1 \in [10^{-2}, 5.10^{-3}, 10^{-3}, 510^{-4}, 10^{-4}]$ and $\lambda_2 \in [5.10^{-2}, 5.10^{-4}]$. Best results are achieved for $\eta = 10^{-2}$, $\lambda_1 = 5.10^{-3}$ and $\lambda_2 = 5.10^{-4}$. For this hyperparameter set, results are showed on figure 2 and the confusion matrix is given in table 4

It is a choice to stop the grid search here, it would be possible to go deeper and continue refining parameters. Moreover, some other parameters can vary such as the number of neuron for instance. The cost of grid search increases exponentially as hyperparameters are added, therefore, this technique can only be used is there are a few parameters in the set to optimize.

		Predicted Class		
		Class 0	Class 1	Class 2
Ground truth	Class 0	256	110	7
	Class 1	107	353	62
	Class 2	3	47	90

Table 4 – Matrix confusion obtained with grid-search hyperparameter set

The final AUC reaches 0.817.

5.2 Bayesian optimization

For Bayesian optimization, objective function is chosen to be AUC. A Gaussian process with 5/2 Matérn kernel is used as surrogate function for prior knowledge over functions. The acquisition function chosen is the expected improvement.

Package BayesianOptimization already implemented for pytorch is used (https://github.com/fmfn/BayesianOptimization). Before starting Bayesian optimization, 20 hyperparameter sets are studied. During the procedure, 60 more are evaluated.

For Bayesian optimization, there are four hyperparameters to determine. As for grid search, learning rate and regularization parameters must be tuned, the number of neurons (the same in each layer) is added to this list. Those parameters do not have to take specific value, they must be in intervals, $\eta \in [10^{-5}\ 10^{-1}]$, $\lambda_1 \in [10^{-15}\ 10^{-1}]$, $\lambda_2 \in [10^{-15}\ 10^{-1}]$ and $nb_{neurons} \in [25\ 1000]$

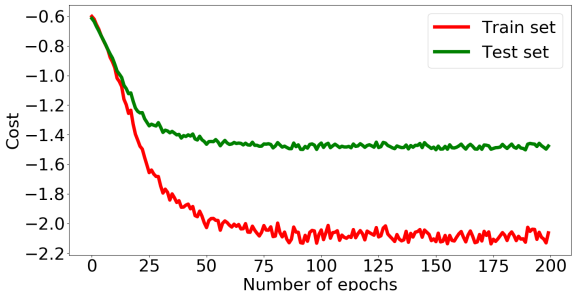
Hyperparameters are uncorrelated and independent. Logarithmic scale is used for η, λ_1 and λ_2 parameters.

Best results are achieved for $\eta = 10^{-2.41}$, $\lambda_1 = 10^{-2.03}$, $\lambda_2 = 10^{-9.41}$, $nb_{neurones} = 229$. The evolution of cost, and AUC is shown on figure 3. Table 5 depicts the confusion matrix of this model.

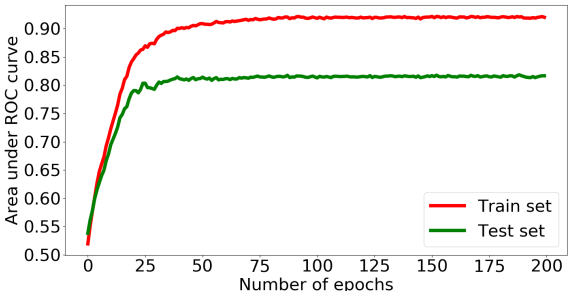
With this second model, AUC reaches 0.856.

5.3 Interpretation

Comparing the models studied, the most different hyperparameters are the number of neurons and the value of λ_2. Cost curves and AUC curves (Fig. 2 and Fig. 3) for both grid search and Bayesian optimization models, are similar.

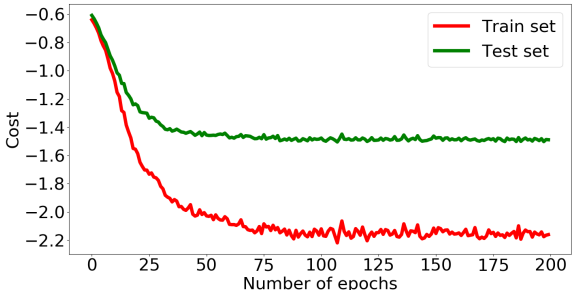


(a) Evolution of the cost function

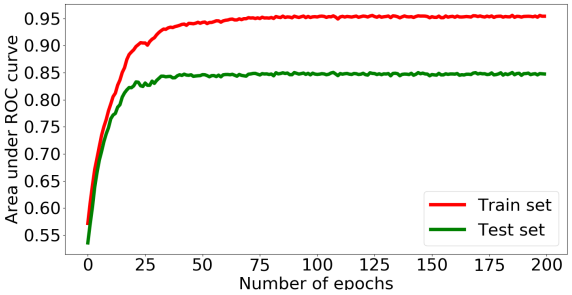


(b) Evolution of the AUC criterion

Figure 2 – Graph a) and b) respectively represent the evolution of the cost function and AUC as a function of the number of epoch for the network tuned with grid search. Red line are results on training set, green line on testing set



(a) Evolution of the cost function



(b) Evolution of the AUC criterion

Figure 3 – Graph a) and b) respectively represents the evolution of the cost function and AUC as a function of the number of epoch for the network tuned with Bayesian optimization. Red line shows the results on training set, green line on testing set

		Predicted Class		
		Class 0	Class 1	Class 2
Ground truth	Class 0	249	120	4
	Class 1	79	404	39
	Class 2	1	44	95

Table 5 – Matrix confusion obtained with BO hyperparameter set

Curves evolve as they supposed to: cost function decreases as the number of epoch increases, on the contrary AUC increases reaching an asymptote after some iterations. The value of this asymptote differs in both model and is higher for the Bayesian optimization method. Despite using regularization terms and dropout, both suffer from overfitting on the train set. In each case, AUC on train set reaches 0.92-0.93 while values for test set are around 0.85. Considering the confusion tables, Table 4 and Table 5, we can notice that there are few mistakes made between class 0 and 2 for both model. This is the result expected using a loss function adapted to ordinal regression problem. The model was trained learning that a mistake between a class 0 and a class 1 is less severe, than a mistake between classes 0 and 2. Analyzing Table 6, results on classes 0 and 2 are similar for both models. The model with grid search even achieved slightly better results than model using hyperparameters tuned with Bayesian optimization for class 0. The most noteworthy difference is found watching class 1 results. In fact, accuracy on class 1 with B0 outperforms grid search method by almost 10 points. With the second model using BO, individuals from class 1 are less confused with either class 0 and class 2.

6 Conclusion

Being able to quickly find the hyperparameter sets giving the best results on the data is a real issue since it gives an indication on the need to spend more time on finding the best model or working differently on data. In this work two methods to select neural network hyperparameters are compared. Our results confirmed what can be found in literature, Bayesian optimization outperforms grid search to find the best hyperparameter set.

Indeed, based on the chosen criterion, AUC, both model respectively reaches 0.817 for the grid search and 0.856 for Bayesian optimization hyperparameter set, representing an improvement of 5%. Computing accuracy for each classes and watching for the global accuracy, it appears that the model tuned with Bayesian optimization surpasses model tuned with grid search on every criterion (see Table 6).

	Grid search	Bayesian optimization
AUC	0.817	0.856
Accuracy for class 0	68.3%	66.8%
Accuracy for class 1	67.6%	77.3%
Accuracy for class 2	64.3%	67.3 %
Accuracy	67.5%	72.3%

Table 6 – Performance comparison between models tuned with grid search or Bayesian optimization

Because each evaluation is time consuming, we limited our trials to around 100 attempts for each technique, as a consequence grid search was restricted to only three hyperparameters. This number can be increased working with Bayesian optimization, we take advantage of it introducing a fourth hyperparameter to tune (i.e. the number of neurons in each hidden layer). Furthermore, using Bayesian optimization also allowed us to stretch the hyperparameter range of search. For instance, regularization parameters are restricted to $[10^{-1}\ 10^{-9}]$ for the grid search while using Bayesian optimization, it is possible to try element in range $[10^{-1}\ 10^{-15}]$. Using Bayesian optimization leads to find better hyperparameter sets using less iterations. In a further step, a work will be done to increase the data set with new features and rearrange the existing features.

References

[1] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[2] E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 12 2010.

[3] J. de la Torre Gallart, D. Puig, and A. Valls. Weighted kappa loss function for multi-class classification of ordinal data in deep learning. *Pattern Recognition Letters*, 05 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision (ICCV 2015)*, 1502, 02 2015.

[5] P. I. Frazier. A tutorial on bayesian optimization. 07 2018.

[6] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[7] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 4, 06 2012.