On the dynamics of loss functions

Le Bich Phuong and Nguyen Tien Zung

Abstract

In differential machine learning, one uses a stochastic gradient flow with respect to a loss function on the parameter space to find an "almost minimal' point of the loss function, which would correspond to an "almost optimal" predictor. Somehow, a proper theory of loss functions is still missing, despite their importance. Our paper is a contribution towards the construction of such a theory.

1 Introduction

This paper is a brief report on our ongoing research on the properties and designs of loss functions in differential machine learning. We refer to [4, 5, 6, 15] for an introduction to machine learning. For simplicity of exposition, in this paper we will only consider binary decision problems, though most other problems can be treated similarly.

Loss functions play an extremely important role in differential learning. However, up until 2000, people didn't really care about them, thinking that they were just a computational issue, without much impact on the final results of machine learning models. (See, e.g., [3, 15]). Recently, people start paying more attention to properties of the loss functions which would help the stochastic gradient flows to converge to desired values of the parameters, see, e.g., [1, 8, 9, 10, 11, 12, 13, 16, 17]. Nevertheless, a full-fledged theory of loss functions is still missing.

The purpose of our work is to contribute to the development of such a theory of loss functions. In particular, after recalling a general setting of differential learning in Section 2, we show the following facts, based on both theoretical reasoning about stochastic flows and experiments:

1) The noise (stochasticity) prevents the gradient flows from converging to the minimal points (Section 3).

2) Asymmetric loss functions are better than symmetric loss functions, especially for problems with significant data imbalances (Section 4).

3) Nonlinear polynomial loss functions are more focal and thus can give better results than the usual cross entropy function (Section 5).

4) Even though most people (until now) automatically compose their loss functions with last-layer activation functions like sigmoid, one will get better results by not using such activation functions (Section 6).

5) Loss functions with a derivative jump at the threshold value creates a stochastic trap at this value for the gradient flow (Section 7).

2 A general differential learning setting

Let us recall here a general setting of differential machine learning for a binary classification problem, and fix some notations:

• Ω denotes the input space, consisting of all possible situations which may appear in the problem, together with a probability measure P (which depends on the context). For example, Ω is a set of images of skin lesions.

• $y_{true} : \Omega \to \{0, 1\}$ is the *ground truth* binary class function. For example, $y_{true}(x) = 1$ if and only the image is a melanoma (a dangerous skin cancer).

• A machine learning model is a map $M : \Omega \times \Theta \to \{0, 1\}$, where Θ denotes its corresponding learnable parameter space. For each choice of parameters $\theta \in \Theta$ the model M gives an output prediction function

$$y_{predict} = M_\theta : \Omega \to \{0, 1\}.$$

• In differential learning, one usually replaces the discrete-valued function $y_{predict}$ by a continuous almost everywhere smooth function

$$y = DM_\theta : \Omega \to [0, 1]$$

which may be interpreted as "probability", "likelyhood" or "level of confidence" in a binary prediction: one puts $y_{predict} = 1$ when $y > 0.5$ (or some other threshold), and the closer y is to 1 the more confident one is in this prediction.

When people talk about *deep learning*, it means that DM is constructed by composing together many layers of simple functions/operators. The theoretical basis for the possibility of approximating any function by composing simple functions is provided by the Kolmogorov superposition theorem (see, e.g., [2]). In deep learning, the number of independent numerical parameters is usually very high (tens of millions), so the parameter space Θ is very high-dimensional.

• The learning process with a given model M is a (stochastic, discretized, finite-time) dynamical system on the parameter space Θ:

$$\theta_0 \mapsto \theta_1 \mapsto \theta_2 \mapsto \ldots \mapsto \theta_n \mapsto \ldots$$

such that, hopefully, for some n, M_{θ_n} is a good approximation of y_{true}.

• The *binary accuracy* function

$$S(M_\theta, y_{true}) = P\{x \in \Omega \mid M_\theta(x) = y_{true}(x)\}$$

and similar functions, including *sensitivity* (true positive rate) and *specificity* (true negative rate), are used to measure the accuracy of the model. In practice, $S(M_\theta, y_{true})$ is calculated empirically by a random set of N instances $x_i \in \Omega, i = 1, \ldots, N$ which are not used in the learning process, called the *validation set* or the *test set* (depending on who tests it, the user or the developer):

$$\hat{S}(M_\theta, y_{true}) = \frac{|\{k = 1, \ldots, N;\ M_\theta(x_k) = y_{true}(x_k)\}|}{N}$$

• In *differential learning*, one replaces the error rate $1 - S(M_\theta, y_{true})$ by a proxy almost-everywhere *differentiable loss function*

$$L : \Theta \to \mathbb{R}$$

chosen in such a way that, intuitively, low values of L correspond to high accuracy rates. The loss L_θ is computed not directly from the binary model M but from the proxy differentiable model DM by some kind of summation (integral) formula, e.g.,

$$L(\theta) = \int_{x\in\Omega} \ell(DM_\theta(x), y_{true}(x))dP_\Omega$$

for some almost everywhere smooth *point-wise loss function* ℓ. Then one uses the method of *gradient descent* to find a parameter value θ_n which "almost minimizes" L.

• Roughly speaking, the differential learning process is defined as follows. Start with some $\theta_0 \in \Theta$ (either a random value, or a "pre-trained" one). At step i in the learning process, put

$$\theta_i \mapsto \theta_{i+1} = \theta_i - \alpha\nabla L(\theta_i) + m(\theta_i - \theta_{i-1})$$

where $\alpha > 0$ is a chosen small positive number, called the *learning rate*, ∇ denotes the gradient, and $m(\theta_i - \theta_{i-1})$ is a small momemtum term. "Just close your eyes and roll down, and hopefully you will reach the bottom".

• It is impossible to compute the exact gradient $\nabla L(\theta)$. One computes it empirically, using a small sample of data called a *batch* at each step, and so the flow is called a *stochastic gradient flow*.

• A true gradient flow can often get trapped at bad local minima (where the value is very high compared to the global minima) and saddle points. That's why a momentum term $m(\theta_i - \theta_{i-1})$ is added to the flow in order to get out of such situations, so in practice one uses a *stochastic gradient flow with momentum*.

3 Some issues affecting accuracy

A multitude of very accurate binary predictors could be constructed by using the above general differential learning method, despite many issues. In this section, we will just mention some of these issues.

•*Boundary cases*. It may happen that the space Ω is a kind of continuous space in which there is no clear-cut boundary between the two classes, and there are many "boundary points" which could belong to both classes at the same time. For example, the evolution of an actinic keratosis (AK) lesion into a squamous cell carcinoma (SCC) is a continuous process, at at some point in this process the lesion could be called AK and could also be called early-stage SCC. Likewise, one can write a number which looks like a 3 and a 5 at the same time, so just by looking at the image no one can tell with certainly which number is it. Due to such boundary cases, there is a hard limit on the level of accuracy that a binary predictor can achieve (which depends on the problem but does not depend on how the predictor is constructed). If one forces the model to make correct predictions on all boundary cases with known ground truth, it simply means overfitting, which does not help predictions in new cases.

•*Data imbalance*. It often happens that one class is much smaller (has much fewer data) than the other. Data imbalance makes the learning difficult: the stochastic gradient flow does not converge to the desired values of the parameters, because the parameters which give most accurate predictions are not at the minimum of the loss function. In Section 4 we will study this phenomenon in a very simple toy model.

•*Noise-induced uncertainty.* We remark that a stochastic gradient flow with momentum used in differential learning can also be viewed as a *damped stochastic Hamiltonian flow*: The momentum term makes it a Hamiltonian system, while the negative gradient term is the damping term. It is known (see, e.g., [14]) that a stochastic damped harmonic oscillator does not converge to the minimum energy point, but rather "converges stochastically" to an energy level higher than the minimum. For differential learning, it means that the stochastic gradient flow cannot be expected to reach the minimum of the loss function. Rather, it will hover around a certain loss level above the minimum. This is a phenomenon of noise-induced uncertainty, due to stochasticity.

•*Choice of the loss function*, which is the main topic of this paper. One can improve the accuracy results a lot by simply choosing a better loss function, more adapted to the problem.

4 Data imbalance in a simple toy model

In this model, the input space Ω is just an interval: $\Omega = [a, b[$. The ground truth binary function is piece-wise constant, i.e., there is a partition of Ω into a finite number of intervals,

$$\Omega = \cup_{i=0}^{n} [a_i, a_{i+1}[$$

with $a = a_0 < a_1 < \cdots < a_{n+1} = b$, and the ground truth is: $y_{true} = 1$ on $\Omega_+ = \cup [a_{2i}, a_{2i+1}[$ and $y_{true} = 0$ on $\Omega_- = \cup [a_{2i+1}, a_{2i+2}[$. The point-wise *gain function* g ($g = 1 - \ell$ where ℓ is the loss function) has n learnable parameters $\theta_1, \ldots, \theta_n$ and is of the type

$$g(\theta_1, \ldots, \theta_n, x) = \prod_i (-\phi(x - \theta_i))$$

where $\phi(x)$ is a increasing monotonous function on $\mathbb{R}$ such that $\phi(0) = 0$, and $\lim_{x \to \pm\infty} \phi(x) = \pm 1$. For example, we can take $\phi(x) = \frac{2}{\pi} \arctan(x)$, or $\phi(x) = \frac{x}{\sqrt{x^2 + \epsilon}}$ for some positive number ϵ. (We will not worry much about the exact formula of ϕ). Notice that the function $g(a_1, \ldots, a_n, x)$ (with fixed $\theta_1 = a_1, \ldots, \theta_n = a_n$) is positive on Ω_+ and negative on Ω_-. The prediction function of the model is:

$$M_\theta(\omega) = 1 \text{ if } g(\theta, \omega) \geq 0$$

and

$$M_\theta(\omega) = 0 \text{ if } g(\theta, \omega) < 0,$$

where $\theta = (\theta_1, \ldots, \theta_n)$. So if (and only if) $\theta = (\theta_1, \ldots, \theta_n) = (a_1, \ldots, a_n)$ then the prediction M_θ coincides with the ground truth, and we get 100% accuracy.

We do not know the value of $(a_1, \ldots, a_n)$, and want to find them by using the stochastic gradient flow of the gain function

$$G(\theta) = \int_a^b g(\theta, \omega) d\omega.$$

Unfortunately, in general the the maximal value of $G(\theta)$ is not at the point $\theta = (a_1, \ldots, a_n)$ in the parameter space, but at some nearby point at best.

In other words, in general, the differential learning method with this gain/loss function (or with any other differentiable loss function for that matter) will not give us a prediction model with 100% accuracy even if such a model exists. This fact is already clear in the case with just one learnable parameter ($n = 1$):

Proposition 4.1. *With the above notations, in the case when $n = 1$, $g(\theta, \omega) = -\phi(\omega - \theta)$, we have:*

i) Balanced case. If $b - a_1 = a_1 - a$, i.e., $a_1 = (a + b)/2$ then a_1 is the maximal point for $G(\theta)$

ii) Biased case. When $b - a_1 > a_1 - a$ but $|b + a - 2a_1|$ is small enough, then the maximal point of G is not at $\theta = a_1$, but at a nearby point in the interval $[a, a_1[$.

iii) Rare event case. If $b - a_1 >> a_1 - a$ so that $g(b - a) \geq 2g(a_1 - a)$, then the argmax of G on $[a, b]$ is a.

The proof of the above proposition follows directly from the following derivation formula for G,

$$\frac{dG(\theta)}{d\theta} = g(\theta, b) + g(a) - 2g(a_1) = -\phi(b - \theta) - \phi(a - \theta) + 2\phi(a_1 - \theta),$$

and the fact that in the third case this derivative is always negative, while in the first two cases the $\dfrac{dG(\theta)}{d\theta}$ vanishes at the maximal point of G on the interval $[a, b]$. A similar proposition holds for the case with many parameters ($n \geq 2$).

The interpretation of the above proposition is as follows:

In case i), when the two classes are balanced, i.e. they have equal weights in the total space, then the gradient flow of G (if it is not stochastic) will converge to the absolutely correct prediction function.

In case ii) there is a bias against the minority class in the differential learning method: if class 1 is minority then even less inputs will be predicted as of class 1 than should be.

In case iii) when one of the two classes is too small, then the machine cannot learning anything by the differential method.

The above very simple toy example already shows the impact of data imbalance on the results of differential learning. To remedy this situation, one has the following methods:

• *Data (re)balancing*: One artificially amplifies the minority class (e.g., change the probability measure on Ω by artificially adding points to the minority class, especially during the so-called "data augmentation" process), so that the two classes will have equal volume in Ω.

• *Asymmetric loss functions*: Most off-the-shelf loss functions are symmetric, i.e., they treat different classes in the same manner. But one may give different weights to different classes in the loss function. In the following sections, we will show some asymmetric loss functions, with an asymmetry parameter which can be tuned for each problem.

In practice, one may use both of the above methods together: depending on what one wants, some asymmetry in the loss function may help, even when the data are already balanced.

• *Sharp loss functions*: One may be tempted to increase the *sharpness of the loss function* in order to fight data imbalance problems. For example, if in the definition of the gain/loss function g mentioned earlier in this section we

use the formula $\phi(x) = \dfrac{x}{\sqrt{x^2+\epsilon}}$, then the ϵ is the sharpness coefficient: smaller ϵ corresponds to sharper loss functions, and when ϵ tends to 0 then the bias in case ii) of Proposition 4.1 also tends to 0.

However, there is a price to pay for the sharpness of the loss function. Namely, if the loss function is too sharp, then the noise-induced uncertainty discussed in Section 3 becomes too high, and so the end results will not be very good either. And of course, in the limit case, when $\epsilon = 0$ then the loss function becomes piece-wise constant and useless for differential learning. So, in each problem there is an optimal sharpness for the loss function.

5 Focality and polynomial loss functions

The idea of *focality* of loss functions (see, e.g., [1, 10]) for classification problems is as follows: If something is already correctly classified (its corresponding loss is already below a certain threshold) then we don't need to improve much its loss, while things which are wrongly classified must be given much higher attention. In differential learning, it means that the contribution to the derivative of the loss function from wrongly classified elements should be much higher than the contribution from correctly classified elements. Intuitively, this focality helps improve the convergence of the gradient flow to optimal accuracy results, and this also holds true in practice. One may call it a kind of *fast convergence method* in the spirit of Newton.

Popular loss functions like *binary cross entropy* (BCE) are not very focal, and so we can propose more focal loss functions which often work better in practice.

Recall that the formula for BCE is:

$$BCE(y) = -(y_{true} \ln y + (1 - y_{true}) \ln(1 - y)),$$

where $y \in]0, 1[$ is the output which is interpreted as the "probability" that the class is 1 (Yes), $1 - y$ is the "probability" that the class is 0 (No), and y_{true} is the ground truth (which is 0 or 1). When $y > 0.5$ (or another chosen cutoff number) then the predicted class is 1, otherwise it is 0.

In differential learning, it's not the loss function itself, but rather its derivative which counts. For BCE, the derivative of the loss function is:

$$F(y) = \frac{1}{1-y} \quad \text{or} \quad \frac{-1}{y},$$

depending on whether $y_{true} = 0$ or $y_{true} = 1$. The non-focality of BCE lies in the fact that the absolute value of its derivative is always above 1, which means that correctly predicted elements still play too important a role in the loss function. A simple way to improve this situation is to replace BCE by simple nonlinear *polynomial loss functions*.

As an example, we created the following two new loss functions:

$$\ell_{24}(y) = cy_{true}((1-y)^2 + (1-y)^4) + (1 - y_{true})(y^2 + y^4)$$

and

$$\ell_6(y) = cy_{true}(1-y)^6 + (1 - y_{true})y^6$$

(where c is the asymmetry coefficient).

When applied to Melanoma detection models, both of the above loss functions gave better results than BCE: while with BCE a team at Torus Actions SAS (a startup founded by the second author) could not make the average validation balanced accuracy to go above 82% after days of training, with the polynomial loss functions the team could get to 85% (before cross validation). (See the next section for more details).

6 The demerit of sigmoid

In most machine learning models, one finds a sigmoid type "activation function" in the last layer: $y = \text{sigmoid}(r) := 1/(1 + e^{-r})$. The loss function is then applied to the output y. The sigmoid function (or similar functions) has its values in the interval $]0, 1[$, so that the output y can be conveniently interpreted as the "probability" or "confidence level" of the answer "Yes" to the binary question.

Despite the convenience and popularity of the sigmoid activation function, we detected a serious demerit of this function in differential learning. Namely, the learning process tends to push y to 1 in the cases where the class is 1 (Yes) in the training set. It means that it is pushing x in those cases to infinity, by definition of sigmoid. But x is usually constructed using semi-algebraic formulas with parameters, so in order to push x to infinity, at least some of the parameters must be pushed to infinity too. In practice, it means that, after a certain number of epochs of learning, some of the parameters will become very high, making the model unstable (hyperbolic: a very small change in the input can too often lead to a large change in the output) and reducing its capability of generalization.

Our proposal is to *avoid using sigmoid* (and similar functions, e.g. softmax in n-ary classification problems) in the last layer of the model. In other words, it's better to construct the loss function as a function of r (the result of the last layer before sigmoid) rather than of y where $y = \text{sigmoid}(r)$. (Sigmoid/softmax can still be used for showing "probabilities", but not for composing with a loss function).

For example, we tested the following lost function:

$$l(r) = c(1 + y_{true})l_1(r) + (1 - y_{true})l_2(r),$$

where

$$l_1(r) = 0.5 \times \max(-r, 0) + (\max(\min(0.5 - r, 0.5), 0))^4,$$

$$l_2(r) = 0.5 \times \max(r, 0) + (\max(\min(0.5 + r, 0.5), 0))^4,$$

$c > 0$ is the asymmetry coefficient (if $c = 1$ then the loss function is symmetric), and the values of y_{true} are 1 (yes) and -1 (no).

A team at Torus Actions applied the above new function (without sigmoid) to the problem of Melanoma detection (among dermoscopic skin lesion images). The results are quite striking: after just 6 training epochs the validation binary accuracy got above 88% already (on a random validation set of more than 1000 images, half of which are Melanoma), better than what could be obtained with all the other loss functions with sigmoid activation, even after hundreds of epochs. (Each epoch runs through 10,000 images, half of which are Melanoma). After some fine tuning and more training, the validation accuracy (and both

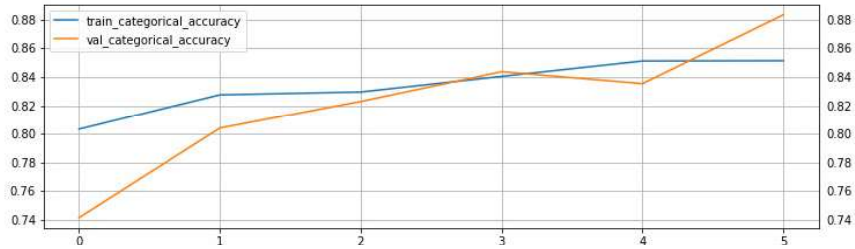

Figure 1: Training results on Melanoma detection with a new loss function without sigmoid activation, after 6 epochs.

sensitivity and specificity together) could get above 90%. Data are mainly from the ISIC 2019 challenge [7]. The model used is a pretrained Inception Resnet CNN with an additional last layer.

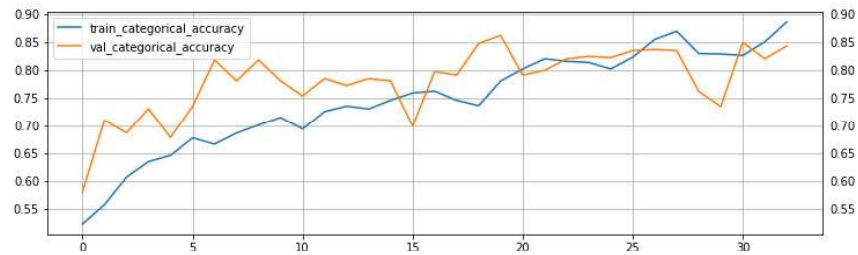

Figure 2: Training results on Melanoma detection, with the same model and same environment as in the previous picture, but with sigmoid activation, and a power six loss function.

7 How broken is a broken loss function?

In our study of focal loss functions, we also designed broken loss functions, i.e. continuous functions which are piece-wise smooth but with jumps in the derivative at a certain point. More concretely, we looked at the following loss function $\ell(y) = c(1-y)\ell_1(y) + y\ell_2(y)$ where c is the asymmetry coefficient and

$$\ell_1(y) = \begin{cases} y^4 \text{ for } y \leq 1/2 \\ y - 3/16 \text{ for } y \geq 1/2 \end{cases}$$

(and similarly for $\ell_2(y)$). The above continuous loss function has a jump in derivative, from 0.5 to 1, at $y = 0.5$

Surprisingly, contrary to our initial naive expectations, in our experiments the gradient flow of the above loss function *does not converge* well and gives erratic results.

It turns out that the jump in derivative at the threshold value $y = 0.5$ creates a *stochastic trap*: the stochastic flow cannot get out from a region where y is near 0.5 and where the prediction is very erratic.

To understand this phenomenon, imagine an input point x whose true class is 0 and whose y value $y_n(x)$ at some step n of the learning process is just a bit smaller 0.5. The class of x is correctly predicted at this step n, but near the threshold $y = 0.5$ the prediction is very erratic: there are many input points very near x whose class is still 0 but whose y-values at step n are just a bit more than 0.5, so that they are wrongly predicted. Let's say that, in a small neighborhood of x, the probability of being correctly predicted (true class is 0) at step n is p, where $p > 0.5$ but not by much. The contribution from the correctly predicted points in the loss function pushes the y-value of x in the right direction (the direction which diminishes y) and is approximately proportional to $0.5p$ (p is the density of correct predictions at x, and 0.5 is the derivative of the loss function for correctly predicted situations). The contribution from the wrongly predicted points in the loss function pushes x in the wrong direction (the direction which increases y) and is approximately proportional to $1-p$. So when $1-p > 0.5p$ (which is the case near the boundary erratic region) then the y-value of x is pushed in the wrong direction by the gradient flow. That's why we have a trap at $y = 0.5$: right things (which are correctly predicted) tend to become wrong by the flow (while wrong things tend to become right, and that's how the trap works: things are repeatedly changing their status from right to wrong to right to wrong again).

Our conclusion is that loss functions which are broken at threshold output values are really broken and should not be used.

References

[1] Nabila Abraham, Naimul Mefraz Khan, *A Novel Focal Tversky loss function with improved Attention U-Net for lesion segmentation*, arXiv:1810.07842 (2018)

[2] Jürgen Braun, On Kolmogorov's Superposition Theorem and Its Applications, SVH Verlag, 2010, 192 pp.

[3] Cristianini, N. and Shawe Taylor, J. (2000). An Introduction to Support Vector Machines. Cambridge University Press, Cambridge, UK.

[4] F. Cucker, S. Smale, *On the mathematical foundation of learning*, Bulletin A.M.S., 39 (2002), 1–49.

[5] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.

[6] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning. Springer, New York, 2001.

[7] ISIC 2019 Challenge (Skin cancer classification): https://challenge2019.isic-archive.com/

[8] Gareth M. James, *Variance and Bias for General Loss Functions*, Machine Learning, May 2003, Volume 51, Issue 2, pp 115–135.

[9] Hoel Kervadec, Jihene Bouchtiba, Christian Desrosiers, Eric Granger Jose Dolz, Ismail Ben Ayed, *Boundary loss for highly unbalanced segmentation* Proceedings of Machine Learning Research, 2019.

[10] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, *Focal Loss for Dense Object Detection*, arXiv:1708.02002 (2017).

[11] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana and Alessandro Verri, *Are Loss Functions All the Same?* Neural Computation, Volume 16, Issue 5, May 2004, p.1063-1076

[12] Chen Shen, Holger R. Roth, Hirohisa Oda, Masahiro Oda, Yuichiro Hayashi, Kazunari Misawa, Kensaku Mori, *On the influence of Dice loss function in multi-class organ segmentation of abdominal CT using 3D fully convolutional networks*, preprint arXiv:1801.05912v1, 2018.

[13] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sébastien Ourselin, M. Jorge Cardoso, *Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations*, arXiv:1707.03237v3 (2017)

[14] Nguyen Thanh Thien, Nguyen Tien Zung, *Reduction and Integrability of Stochastic Dynamical Systems*, Journal of Mathematical Sciences, 2017, 225 (4), pp.681-706.

[15] V. Vapnik, Statistical Learning Theory. Wiley, New York (1998).

[16] Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Jianhuang Lai, Tie-Yan Liu, *Learning to Teach with Dynamic Loss Functions*, 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

[17] Hang Zhao, Orazio Gallo, Iuri Frosio, Jan Kautz, *Loss Functions for Image Restoration With Neural Networks*, IEEE Transactions on Computational Imaging, Volume 3 , Issue 1, March 2017, pages 47 - 57.

L.B.P.: Department of Mathematics, Hanoi University of Mining and Geology, Hanoi, Vietnam. Email: lbphuong@sputnik.vn

N.T.Z.: Institut de Mathématiques, Université de Toulouse 3, Toulouse, France. E-mail: tienzung@math.univ-toulouse.fr