Forward Noise Adjustment Scheme for Data Augmentation

Francisco J. Moreno-Barea¹, Fiammetta Strazzera¹, José M. Jerez¹, Daniel Urda², and Leonardo Franco Senior Member, IEEE.¹

¹Department of Computer Science, Universidad de Málaga, Málaga, Spain.
²Department of Computer Science Engineering, Universidad de Cádiz, Cádiz, Spain.

Abstract—Data augmentation has been proven particularly effective for image classification tasks where a significant boost of prediction accuracy can be obtained when the technique is combined with the use of Deep Learning architectures. Unfortunately, for non-image data the situation is quite different and the positive effect of augmenting the training set size is much smaller. In this work, we propose a method that creates new samples by adjusting the level of noise for individual input variables previously ranked by their relevance level. Results from several tests are analyzed using nine benchmark data sets when the augmented and original data are used for supervised training on Deep Learning architectures.

Keywords: Data Augmentation, Supervised learning, Deep Learning, Feature selection.

I. INTRODUCTION

Deep Learning (DL) models have become a key technology in the area of Machine Learning since 2012 when they showed outstanding performance on classification and prediction tasks [1], [2], [3]. Analyzing the main differences with previous neural network models, DL architectures essentially include several hidden layers with partial and full connectivity, including convolutional layers, and a series of modifications to accelerate the training process and prevent overfitting, like the use of ReLUs activation functions and the dropout technique. Another important aspect, not to be minimized that seems also necessary for the success of DL models is the amount of data needed to train these models. As DL architectures involves thousands of parameters (mainly synaptic weights) to be adjusted during training, a large data set with values of thousands of instances for every output category typically is needed. As the previous condition is not met in several problems of interest, the topic of Data Augmentation (DA) has become a very important issue in recent years. Indeed, most models that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [4] were trained with data augmentation techniques [5], [6], [7], [8]. Going back in time to previous work related to the topic of DA, it is worth mentioning techniques that were developed for dealing with unbalanced classes (another frequently scenario in classification and prediction problems), like the SMOTE technique that creates synthetic minority class examples [9]. Creating synthetic data is a challenging and active area of research, having the purpose of augmenting the size of the training set in order to improve the performance of classification and prediction methods, also having other potential uses. In this sense, it is worth mentioning the case of Generative Adversarial Networks (GANs) that have captured much attention in recent times [10], [11].

Data augmentation techniques can be applied directly to the input variables or can be generated from extracted features from the data, involving in both cases a modification of the original patterns. Successful cases of DA can be found in image data sets where original patterns can be transformed by applying geometrical modifications that do not alter the category of the samples, like rotation, enlargement, translation, etc. [12], [1]. Working with non-image data has been found to be much more complex, and apart from some applications of the SMOTE techniques mentioned above, the rest of the literature is mainly related to noise injection techniques as a way to prevent overfitting and improve the accuracy of the predictions [13].

In the current work we implement a forward scheme that modifies the original variables values through the addition of noise. The method is a greedy approach that adjusts the level of noise to be used in each of the input variables that may lead to an improvement on the prediction accuracy, and for doing this the input variables are ranked first according to its relevance regarding the prediction of the output categories. The work is structured as follows: Section II shows how data augmentation can be successfully implemented on images. Section III introduces the proposed method. Section IV describes the datasets, ranking method and pattern classification model used to then present the Results in Section V, and we finish the work with the discussion and conclusions in Section VI.

II. DATA AUGMENTATION ON IMAGES

As mentioned above, data augmentation is widely used when training neural networks in image classification tasks. This strategy consists essentially in increasing the amount of training samples by applying a transformation, such as reflection, rotation, shear and shift to the training images. In each epoch a transformation is applied to every input image,
creating distinct samples that are presented to the network. An example of a transformation is illustrated in Fig. 1, where an original image from the CIFAR-10 data set is shown (Left) together with a new transformed image created by a $15^\circ$ rotation (Right).

To clearly illustrate the effectiveness of data augmentation when dealing with image classification tasks, we train a 5-layer convolutional neural network on the CIFAR-10 data set, similarly to what has been done previously in Ref. [1]. The network architecture used consists of three convolutional layers, one fully connected layer, and the output layer. Each convolutional layer has 3x3 kernels and 32, 64 and 128 filters respectively and is followed by a ReLu activation function, batch normalization, a 2x2 max-pooling operation, and a dropout rate of 0.5. The fully connected layer use also ReLu activation functions and a dropout rate of 0.1. The network architecture used consists of three convolutional layers, one fully connected layer, and the output layer. Each convolutional layer has 3x3 kernels and 32, 64 and 128 filters respectively and is followed by a ReLu activation function, batch normalization, a 2x2 max-pooling operation, and a dropout rate of 0.5. The fully connected layer use also ReLu activation functions and a dropout rate of 0.1. The fully connected layer use also ReLu activation functions and a dropout rate of 0.5 was applied, whereas a softmax activation function is used in the output layer.

Data augmentation significantly improved the performance of the neural network as illustrated in Figure 2. Indeed, the final accuracy of the network increased of around 6.3% when applying data augmentation in comparison to training with the original data set.

III. FORWARD VARIABLE NOISE ADDITION METHOD FOR DATA AUGMENTATION

This section proposes and describes a general procedure for creating new samples (Data Augmentation) that works in a forward manner. The procedure called FNA is described in the pseudocode shown in the Algorithm 1 and works as follows: the inputs to the algorithm are a data set (dataset) and the different levels of noise to be used (noise). The data is split using a stratified 10-fold cross-validation procedure, such that 8 folds will be used for training, 1 fold will be used for choosing the best noise level (validation), and the remaining fold will be used for testing the accuracy ([Train, Val, Test] respectively). The cross-validation procedure was designed in such a way that the test set is not used for choosing the optimal level of noise, in order to avoid overfitting effects. The augmented data (Aug) to be created from the original samples Train through the addition of noise is first initialized in line 3 of the algorithm, to then applied a method for ranking the input variables according to their relevance (sorted in descending order). Then, for each attribute and level of noise, the algorithm loops over the folds, adding noise in the attribute $Attr_a$ considering the noise level $noise_a$, concatenating this new noisy data to the original data $Train_Y$ to finally obtaining the F1-measure ($F1_{Val_a(V)}$) for $Val_a$ and $Test_a$ sets. When all folds are tested for $noise_a$, the F1-measure associated with it is the average of all the folds, and when all levels of noise are tested for $Attr_a$, the F1-measure associated with it ($F1_{Val_a(V)}$) is the F1-measure associated with the level of noise that maximizes this value at $F1_{Val_a(V)}$. Recorded. If $F1_{Val_a(V)}$ is higher or equal that the best value obtained until this moment, then the algorithm adds noise to $Aug$ (note that $Aug$ does not changes inside the inner loops) considering the $Attr_a$ and $noise_{max}$. The results stored for this attribute $a$ are $noise_{max}$ and the test F1-measure associated with it. When all ranked attributes are tested, the algorithm returns at the end the results obtained for each attribute: the optimal level of noise found using the validation folds and the F1 measure averaged across its corresponding test folds.

A scheme of the noise adjustment procedure is also depicted graphically in Fig. 3. The figure represents using $p$ boxes the adjustment of the noise level corresponding to each of the $p$ input variables (features) of the data set to be augmented. As mentioned above, first the variables are ordered using a rank relevance method, and the adequate level of noise is adjusted in a greedy forward procedure starting from the most relevant variable. Within each box, a ten fold cross validation procedure is used, in which 8 folds are used for training, one fold for validation, i.e., adjustment of the noise level, and the remaining one is for obtaining the accuracy estimation of the whole procedure (test). Thus, in the first box of the figure, indicated by $X_1$, the different levels of noise are tested (noise, $i = 1, \ldots, N$) only for the first input variable ranked as the most relevant. Using the validation fold, the adequate level of noise is chosen according to the best validation accuracy obtained (process indicated by the equation shown out of the box on the right). After the right level of noise is chosen for the first variable, the process continues with the remaining ones.
Algorithm 1 Forward noise adjustment algorithm (FNA)

1: procedure FNA(dataset, noise\{1..N\})
2: \{Train, Val, Test\}\{1..10\} ⇒ StratifiedSplit(dataset)
3: Aug \leftarrow Train
4: \{Attr\}\{1..P\} ⇒ VariablesRank(dataset)
5: for \(a = 1 \rightarrow P\) do \(\triangleright\) loop over attributes
6: for \(n = 1 \rightarrow N\) do \(\triangleright\) loop over noise levels
7: for \(i = 1 \rightarrow 10\) do \(\triangleright\) loop over folds
8: \(T_i \leftarrow\) AddNoise(Aug, Attr\(_a\), noise\(_n\))
9: \(T_i \leftarrow\) Concatenate(Train\(_i\), T\(_i\))
10: \([f_1 V, f_1 T]\) ⇒ Classifier(T\(_i\), Val\(_i\), Test\(_i\))
11: end do
12: \(f_1 V\) \leftarrow mean(\(f_1 V\))
13: \(f_1 T\) \leftarrow mean(\(f_1 T\))
14: end do
15: \(f_1 \_a\) \leftarrow \text{max}(\(f_1 V\)\{1..N\})
16: noise\(_\text{max}\) \leftarrow arg max(\(f_1 V\)\{1..N\})
17: if \(f_1 \_a\) \(\geq\) \(f_1 \text{best}\) then
18: Aug \leftarrow AddNoise(Aug, Attr\(_a\), noise\(_\text{max}\))
19: \(f_1 \text{best}\) \leftarrow \(f_1 \_a\)
20: end if
21: Results\(_a\) \leftarrow [noise\(_\text{max}\), \(f_1 T\)\(_{\text{noise\(_\text{max}\)}}\)]
22: end do
23: return Results

Fig. 3. Scheme of the forward variable procedure proposed for estimating the optimal level of noise (see text for details).

Testing again different levels of noise for the current variable but using the chosen noise level value obtained previously for variables considered more relevant.

The proposed method requires the use of two methods: one for ranking the individual variables according to its importance (named VariablesRank in the pseudocode of the algorithm) and a second one to be used as the classification algorithm Classifier. In the present work, we have used Random Forest for ranking the importance of the individual variables and Deep Learning models as the pattern classification algorithm. We describe both methods, the data set used and the procedure to augment instances by noise modification in the next section.

IV. METHODS AND DATA SETS

A. Random forest for feature importance sets

Random forest is a supervised learning algorithm developed by Leo Breiman [14] that uses an ensemble (called forest) of decision trees. The algorithm combines Breiman’s own idea of bagging [15] with the random variable selection [16] in order to build an ensemble of decision trees with controlled variance. To build each tree, a randomly selected data set with replacement is used (bootstrap), and for each node split, a random subset of the variables is select, giving both methods low correlation to the individual trees. As a supervised learning algorithm, the main uses of random forests are classification and regression, but random forests can also be used for feature selection [17], [18], since as a classifier, the random forest model produces an implicit feature selection while the model is being built.

Random forest provides different feature importance measures. One of them directly measures the impact of each feature on the model accuracy. For this, permutations are made in the features values and it is measured how much this affects the accuracy of the model, giving greater importance to features whose permutation produces a greater variation of the accuracy. The other feature important measure, called Gini importance, is calculated during the individual trees construction, and it is based on the Gini impurity index used for the calculation of node splits. Every time a split is made on a variable, the Gini impurity criterion for the two descendant nodes is less than the parent node, so a higher mean value of this measure over all trees indicates a higher importance of the feature at tree nodes divisions. In this work we use the Gini importance measure for ranking the input variables in order to apply the forward variable noise adjustment scheme. We choose the Gini importance measure instead of the other measure described, because although both measures correlate reasonably well [19], Gini importance is slightly better when we dealing with continuous features [20]. For the numerical simulations carried out, the number of trees used in the random forest model was set to 1000.

B. Noise addition procedure

A standard random uniform noise procedure was implemented in our experiments in order to augment the original data. Eq. 1 mathematically describes the process of obtaining a new variable value \(\tilde{x}\) from the original one \(x\).

\[\tilde{x} = x + \text{RND}(-0.5, 0.5) \times \text{Noise}_i\]  \(\text{(1)}\)

The original \([0, 1]\) normalized values of a variable for an instance of the problem is modified by the addition of noise using a random uniform number (denoted ‘RND’ in the Eq. 1) in the interval \((-0.5, 0.5)\) multiplied by a level of noise (a parameter of the model, indicated by ‘Noise’ in Eq. 1). Ten levels of noise were tested in the numerical simulations with values from 0.1 to 1 in steps of 0.1. Alternative tests were also carried out using Gaussian random noise but no significant changes were observed.
C. Benchmark data

Table I shows some characteristics of the benchmark data used for the experiments. These benchmark data, taken from the UCI database [21] and PROBEN1 benchmark set [22] frequently used in the literature, consist in real world data that represent problems which could be called diagnosis tasks. In Table I, the columns show the name of the benchmark data set, the number of features and instances, and the percentage of instances that belong to the majority class with the purpose of showing the balance of both output classes of the benchmark data.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Features</th>
<th>Instances</th>
<th>Majority Class (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pima diabetes</td>
<td>8</td>
<td>768</td>
<td>65.10</td>
</tr>
<tr>
<td>breast cancer</td>
<td>9</td>
<td>699</td>
<td>65.52</td>
</tr>
<tr>
<td>waveform*</td>
<td>12</td>
<td>3353</td>
<td>50.58</td>
</tr>
<tr>
<td>heart cleveland</td>
<td>13</td>
<td>303</td>
<td>54.12</td>
</tr>
<tr>
<td>card</td>
<td>15</td>
<td>690</td>
<td>55.51</td>
</tr>
<tr>
<td>thyroid*</td>
<td>21</td>
<td>7200</td>
<td>92.58</td>
</tr>
<tr>
<td>wdbc</td>
<td>30</td>
<td>569</td>
<td>62.74</td>
</tr>
<tr>
<td>horse colic*</td>
<td>58</td>
<td>364</td>
<td>61.54</td>
</tr>
<tr>
<td>gene*</td>
<td>120</td>
<td>3175</td>
<td>51.90</td>
</tr>
</tbody>
</table>

Note that some of the data sets from table I are marked with an asterisk (*) to indicate the fact that the original problem data contained three output categories and that they have been converted into binary problems (two classes). Specifically, for the the data sets ‘thyroid’, ‘horse colic’ and ‘gene’ the less populated category has been joined with the second less one thus balancing the problem. In the case of the ‘waveform’ data set, the less represented class was deleted, because the percentage of all the three classes was quite similar. Further, all input variables have been normalized for its subsequent treatment in the augmentation process. Further, we note that all the experiments and methods can be applied to multi-class classification straightforward, just by using additional output neurons for the different categories.

D. Classification model architecture

This section details the learning model used to test the effect of data augmentation on the benchmark data sets considered. The model implemented is a feedforward neural network (FFNN). Fig. 4 shows a scheme of the architecture with 3 hidden layers, 32 neurons at first and third hidden layers, and 64 in the second hidden layer. Our intention when using a FFNN with this number of layers and neurons is to experiment with a model of neural networks that could be considered simple, without having to deal with dozens of intermediate layers and hundreds of neurons in each layer, but which could be considered within the category of Deep Learning models. A leaky ReLU [23] activation function was used in the hidden neurons as it is more stable, is less affected from the ‘dying ReLu’ effect and leads to slightly better results that the standard ReLU function [24]. The model also have 2 output neurons to classify the patterns in controls and cases, and they use a sigmoid as activation function.

As regularization technique to avoid the network overfitting, we use the combination of dropout [25] and L2 norm, also known as weight decay, a combination that is frequently used in the literature. Specifically, dropout is the most common regularization technique in deep learning. This technique randomly set to zero (‘dropped out’) the activation of some neurons during training, giving rise to a set of different networks whose predictions are averaged. The dropout rate used at this work is 0.4, which corresponds to the probability that a neuron will be dropped out.

Finally, as optimization algorithm with adaptive learning rate, the Adam algorithm [26] is used. Adam can be seen as a variant of the combination of RMSProp [27] and the momentum algorithm [28], while maintaining some differences with both, such as the incorporation of momentum and bias corrections, which help avoid the high biases present in the training of RMSProp. The values used in the parametrization of the Adam algorithm are, 2e-4 as learning rate, 0.5 for the exponential decay rate for the 1st moment estimates, 0.9 for the exponential decay rate for the 2nd moment, and 1e-8 for $\epsilon$ ($\epsilon$ in the original Kigma and Ba paper [26]).

V. RESULTS

The experiments executed in order to test the proposal have been executed on a PC Intel Core 7 under Windows 10 using a NVIDIA® Titan Xp GPU. For the implementation of the experiments, the programming language used was Python 3.5.2v [29], together with TensorFlow 1.8v with GPU support [30], CUDA® Toolkit 9.0 from NVIDIA and cuDNN v7.0. After preliminary experiments in which different metrics were tested (F1-measure, accuracy and area under the ROC curve (AUC)), the decision was to only use the F1-measure, because it works as well as the AUC for data sets with unbalanced classes and needs lower computational resources. The F1-measure (F1) is defined as the harmonic mean of precision (positive predictive value) and recall (sensitivity), and is a derivation of the F-measure [31] in which 1 is taken as value of the parameter that controls the balance between precision and recall.
Table II shows the results obtained for the 9 binary data sets used as benchmarks, the F1-measure when no data augmentation is applied, the number of features for which the procedure selected a level of noise different from zero, the F1-measure using the 100% augmented training data set, and the percentage relative difference Eq. 2 (RD%) between the F1-measure when data augmentation is and is not applied. Noting that validation and testing are done on the original data (no augmentation). Tables III and IV show similar results using 200% and 500% augmented training data set, and Table V summarizes the results for the different levels of data augmentation (DA) shown in the previous tables (boldface fonts indicate the best values).

\[
RD\% = \left(\frac{F1_{Aug} - F1_{Ref}}{F1_{Ref}}\right) \times 100
\]  

(2)

In order to compare the effectiveness of the DL models, an additional experiment was carried out by using a traditional MLP with 32 neurons in the single hidden layer of a fully connected architecture. In average, the obtained results showed an F1 measure of 0.854 when no data augmentation is applied and 0.86 when data augmentation is applied, and a value of 0.729 for the relative percentage difference. As expected, the results obtained are worse than for the case of using DL architectures but the effect of DA can also be observed in this case.

Fig. 5 shows the relationship between the F1-measure for the augmented training data sets (500 % DA) and the reference case for the test sets (the continuous line is the identity function).
We also analyzed the dependence between the results obtained (measuring the relative F1 accuracy (rF1)) for each data set according to the number of features, number of instances, and degree of unbalancedness (measured by the majority class %), and the results are shown in Fig. 6. The results might suggest some tendency towards better results for the DA process when low values of the three aspects of the data sets are considered, but further experiment should be needed in order to verify them.

Data augmentation not only leads to better prediction accuracies as shown before but also accelerate the training performance of the models as illustrated in Fig. 7, which shows the loss obtained when fitting the model to the *wdbc* data set without data augmentation and with data augmentation for the three levels considered: 100%, 200% and 500%. The data augmentation process for the *wdbc* data set (average results shown in Table V) corresponds to a noise level of 0.4, 0.8 and 0.7 to attributes 22, 27 and 20 respectively. Further, we mentioned that several other tests were performed considering different parameters and conditions, like for example using Gaussian noise instead of uniform distributed, different dropout rates, different number of neurons in the architectures, etc. One limitation for the number of tests carried out was the computational time involved in the numerical simulations, that approximately takes 5 days with the hardware mentioned before for each set of parameters for the whole benchmark data set.

**VI. DISCUSSION AND CONCLUSIONS**

In this work, a forward noise adjustment scheme for the data augmentation task with the aim of improving the prediction accuracy in classification problems was introduced. The proposal consists essentially in setting the adequate level of noise for individual variables in a forward process in which most relevant variables are considered first. The proposed procedure needs two internal methods for its operation: one for ranking the input variables according to its relevance and a second one to act as a classifier. An extensive bunch of numerical simulations was performed using a benchmark of nine data set problems from the well known UCI repository, considering binary class problems with a number of inputs between 8 and 120. Further, authors considered Random Forest as the method for ranking input variables and Neural Networks Deep Learning models as classifiers, obtaining in the best case a 1.648% average relative increase in the prediction accuracy (F1 metric was used), corresponding to the situation in which the dataset was increased six times (500% of DA). The behavior of the proposed method was analyzed as a function of the number of input variables, number of instances and degree of class unbalance (cf. Fig. 6) of the problems considered and noticed no clear tendency with relationship to these important parameters so it seems that the method could be in principle extended and applied to other and probably larger data sets, task that authors expect to carry in the near future using gene expression data containing large number of input features. As an overall conclusion, the results obtained so far with the application of the proposed method are satisfactory, although authors are aware that more extensive tests with several larger data sets should be performed in order to validate the efficacy of the presented procedure.
Further, considering that 6% or more increase in prediction accuracy can be obtained with DA methods using image data (cf. Section II), the current results give room for alternative approaches for DA, also indicating that noise addition might not be enough. In this sense, the use of GANs seems a novel and interesting approach to be considered in the near future.

VII. ACKNOWLEDGEMENTS

The authors acknowledge the support from MINECO (Spain) through grants TIN2014-58516-C2-1-R and TIN2017-88728-C2-1-R, and from Universidad de Málaga (all including FEDER funds). The authors thank NVIDIA Corporation for the donation of a Titan Xp GPU used for this research.

REFERENCES


