Assembly Output Codes for Learning Neural Networks

Philippe Tigréat*, Carlos Rosar Kos Lassance*, Xiaoran Jiang**, Vincent Gripon*, Claude Berrou*

*Electronics Department, Télécom Bretagne **INRIA Rennes

*name.surname@telecom-bretagne.eu , **name.surname@inria.fr

***Abstract*—Neural network-based classifiers usually encode the class labels of input data via a completely disjoint code, i.e. a binary vector with only one bit associated with each category. We use coding theory to propose assembly codes where each element is associated with several classes, making for better target vectors. These codes emulate the combination of several classifiers, which is a well-known method to improve decision accuracy. Our experiments on data-sets such as MNIST with a multi-layer neural network show that assembly output codes, which are characterized by a higher minimum Hamming distance, result in better classification performance. These codes are also well suited to the use of clustered clique-based networks in category representation.**

Keywords—Assembly coding; Clustered Clique Networks; ECOC; Deep Learning; Coding theory; Classification

I. INTRODUCTION

Automatic learning systems are different from storing systems in that they aim at generalizing to unknown inputs. This happens through the extraction of the core features of learned data, and works as long as the unknown data to extrapolate to follows a similar distribution. The system thus learns a dictionary of features that is targeted to be well suited for the task at hand, *e.g.* classification. These features are meant to correspond to the most relevant building blocks underlying the training inputs, that unseen data samples would likely also be made of.

In supervised Deep Learning networks, an output is calculated through a pipeline of vector-matrix products between input data and connection weights, intertwined with non-linear mathematical operators. Each level of the network is associated with a set of features that is more and more abstract as one moves towards the upper layers. During learning, an error is calculated from the difference between the resulting output and an objective vector specific to the class of the input. A gradient is then calculated from this error for the whole set of connection weights, and the hierarchy of features thus gets optimized through gradient descent for the task of classifying the data-set examples.

Combining classifiers has been an extensive area of research for a few decades [1] and several algorithms have been shown to bring improved decision by leveraging the diversity brought by an assembly of systems. Among these methods, boosting is a way to combine opinions of experts by weighting them based on their respective estimated accuracy. These base classifiers can be differentiated using various strategies, like training them on various subsets of the data or by providing them with different sub-parts of an ensemble of learned feature detectors.

Another way to combine classifiers is to split the problem into a set of binary problems. A base classifier will then focus on classifying inputs between two of the initial classes, as in One-Vs-One (OVO), or between one class and the rest as in One-Vs-All (OVA). A way to implement these strategies is to provide a classifier with objective vectors that are not always specific of a single class but can be associated with a set of classes. Much attention is paid here to the Error-Correcting Output Coding (ECOC) method, which splits a multi-class problem into several two-way classification problems between meta-classes. It is shown in [2] that ECOC can reach a better classification performance on an image data-set as compared to other multi-class methods.

The approaches presented here are derived from ECOC for multi-class problems. These methods allocate assemblies of output neurons to the different input classes, with potential overlap between the codes of two classes. A clustering of the output layer is also applied, with only one active neuron per cluster for each target vector, and a local soft-max [3] process applied in each cluster at test time. The soft-max operator has the effect of normalizing to 1 the sum of energies of output neurons in each cluster. Experimental results suggest that the number of classes sharing an output node impacts performance, and so does the minimal distance between class codes. This finding is maintained when output codes are repeated so as to ensure that the different tested networks have very similar numbers of parameters.

The outline of the paper is as follows. Section II provides theoretical considerations on the advantages of different output codes. Section III explains the methodology used in training the networks. Section IV presents experimental results.

II. CODING THEORY

Prior to experimenting with assembly codes as output for neural networks, a theoretical analysis can provide insights about which assemblies should perform better.

Consider a classifier with P classes to identify. The simplest way to make the classifier express its decision is to assign a single output node (the so-called *grandmother cell*) to each class. We propose to replace these P nodes with $n=\binom{P}{m}$ nodes representing all the combinations (or assemblies) of m classes among P. Let us define the

This work was supported by the European Research Council under Grant ERC-AdG2011 290901 NEUCOD

978-1-5090-3401-7/16/.00 ©2016 IEEE

coding rate R of the corresponding code as the ratio between $\log_2(P)$ and n:

$$R=\frac{\log_2(P)}{\binom{P}{m}} . \quad (1)$$

To calculate the minimum Hamming distance, let us consider any two classes among P. The number of assemblies that contain neither one nor the other is $\binom{P-2}{m}$ and the number of assemblies that contain both is $\binom{P-2}{m-2}$. The minimum Hamming distance of the code is given by n minus the latter two terms:

$$d_{\min}=\binom{P}{m}-\binom{P-2}{m}-\binom{P-2}{m-2} . \quad (2)$$

The product of R and $d_{\min}$, called the merit factor F, is deduced as:

$$F=Rd_{min}=\frac{2m(P-m)\log_2(P)}{(P-1)P} \quad (3)$$

and its maximal value is obtained for $m = P/2$:

$$F_{\max}=\frac{P\log_2(P)}{2(P-1)} . \quad (4)$$

The corresponding minimum distance may be expressed as:

$$(d_{\min})_{\max}=\frac{P}{(P-1)}\frac{n}{2} . \quad (5)$$

For instance, with $P = 10$ (*e.g.* for MNIST classification), the best code involves quintuplet assemblies and offers a minimum distance of 140 with $n = 252$. If quintuplets are replaced with couples, the parameters become: $d_{\min} = 16$ and $n = 45$. Note that the classical output code (one node per class), still with $P = 10$, has a minimum distance of 2 with $n = 10$.

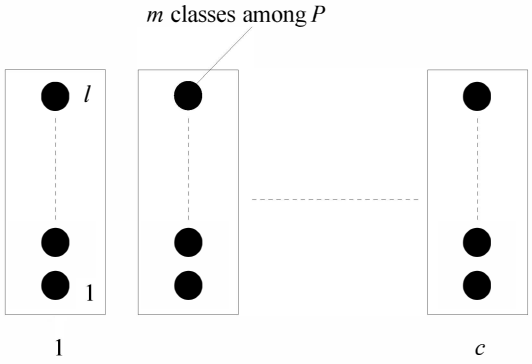

Figure 1. The output layer of the classifier is organized in c clusters, each one having l nodes. These l nodes represent disjoint combinations of m classes among P with l = P/m.

Now we propose that the n assemblies are distributed among c clusters such that each class appears once and only once in each cluster. Therefore, there are $l = P/m$ nodes in each cluster, assuming that P is a multiple of m (Fig. 1). This structure has two advantages. Firstly, it complies exactly with the clustered clique-based associative memory proposed in [4] which offers the possibility to store a number of patterns proportional to l^2 (for larger sought diversities, the sparse scheme proposed in [5] may be contemplated). However, the optimal value $l = 2$ (as deemed by the optimization of F) is too low for a clique-based implementation and a trade-off has then to be found. For instance, $l = 5$ (that is, $m = 2$) seems a good choice for $P = 10$. The second advantage is that a cluster containing disjoint assemblies and therefore probabilities, through the soft-max principle [6], may be rigorously used for both learning and testing.

III. METHODOLOGY

3.1. MNIST

MNIST [7] is a data-set of grey-scale images of handwritten digits that is widely used in deep learning. The data-set is made of 60000 images targeted for training and 10000 test examples. Many published works use the first 50000 examples from the training set to actually train the network, saving the last 10000 examples to perform cross-validation. Some papers however present networks trained on the whole set of 60000 training images, as is the case in [8]. Here the 50000/10000 split of the training set is used for the experiments of section 4.1, whereas in sections 4.2 and 4.3 the networks are trained with all 60000 examples.

3.2. SVHN

SVHN [9] is a data-set made of color images of digits captured in real–world situations, e.g. house numberings. It contains 73257 training images and 26032 test images. We use 60000 examples from the train set to actually train our networks, and the remaining 13257 serve for cross-validation.

3.3. Assembly codes

Couple cells are a method we introduce for encoding the class using a distributed code. Each cell no longer reacts to a single class but is specific of a couple of classes. Hence since the experiments are performed on data-sets with 10 classes, there are 45 possible couplings a given output neuron can be associated with. The whole set of 45 couplings is used here. Moreover, it is possible to partition these 45 couplings into 9 clusters of 5 couple cells where each individual class is represented exactly once in each cluster. This way the soft-max methodology can be applied inside of every cluster, between 5 competing hypotheses that are mutually exclusive.

Quintuplets are yet another code where each neuron is associated with a combination of classes, this time 5 among 10. The same approach as for the couples is used, by using all 252 possible quintuplets and partitioning them into 126 clusters each containing 2 complementary quintuplet cells.

The first two important factors considered to choose assembly codes are usability in a clique-based architecture and the merit factor. Alongside with the grandmother cell (our baseline), couples of classes (best trade-off cliques/merit factor) and quintuplets of classes (best merit factor, but larger output network) are tested. But only comparing these parameters is not enough, because one could argue that couples and quintuplets work better because they have a larger number of parameters for the last layer of the neural network. To avoid this we also train networks using repeated codes to reach output length of equal or comparable size. This repetition allows us to make a fair comparison between networks with virtually the same number of parameters in spite of using different output codes. Couples of classes have an interesting characteristic, in that there are plenty of different possible ways to partition 45 couple cells into 9 clusters, each featuring the 10 classes. Therefore it is possible to design an output layer of 1260 couple cells parted in 252 clusters where no cluster configuration is repeated twice.

During the training phase the categorical cross-entropy is used as the loss function. For classification a majority voting is done where each active output node votes for its associated set of classes. The assembly codes are summarized in table 1.

Assembly Code	n	m	l	c	F	$d_{\min}$
Grandmother cell (1G)	10	1	10	1	0.66	2
Couples (1C)	45	2	5	9	1.18	16
Quintuplets (1Q)	252	5	2	126	1.84	140
Grandmother cell * 25 (25G)	250	1	10	25	0.66	50
Couples * 6 (6C)	270	2	5	54	1.18	96
Grandmother cell * 126 (126G)	1260	1	10	126	0.66	252
Couples * 28 (28C)	1260	2	5	252	1.18	448
Quintuplets * 5 (5Q)	1260	5	2	630	1.84	700
Special Couples [1] (SC)	1260	2	5	252	1.18	448

*Table 1: Summary of the tested assembly codes. Assembly Code * number: Code repeated number times. [1]: 252 non-repeated clusters*

3.4 Neural network settings

For tests, the neural networks used are multi-layer perceptrons. Two architectures of network are used, one that is shallow and the other deep. The shallow network has only 1 hidden layer, while the deep network has 5 hidden layers. The results are presented by the mean and standard deviation over 10 executions with different weight initialization, noise and image order.

In sections 4.1 and 4.2 the neural network is shallow and its only hidden layer is composed of 2000 feature units. Training lasts for 200 epochs with a constant learning rate of 0.1. The model "*baseline + noise*" from [8] is chosen as a base for the deep network used in sections 4.3 and 4.4. This is a network that gets close to the current state-of-the-art in the task of permutation-invariant MNIST. It has 6 layers, where the first 5 of them are fully connected layers (with sizes 1000-500-250-250-250) and the last one is an output layer. At each connection layer, between the input and the activation, a batch normalization [11] and a Gaussian noise with mean 0 and standard deviation 0.3 are used. To finish the connection layer, a rectifier activation is used. The output layer has the length of the output code. A batch normalization is applied between the input and the activation and as in the case of the shallow network it uses a per-cluster soft-max activation. The number of parameters of each network is given by the formula: 1000*(*input length*) + 750000 + 250*(n). This means: 1534000+250*(n) for MNIST and 3822000 + 250*(n) for SVHN. This deep network has 150 epochs to learn where it optimizes the weights with the ADAM optimizer [10], using an initial learning rate of 0.002 that has an annealing phase of 50 epochs where the learning rate decays linearly to 0.

IV. RESULTS

4.1. Experimenting with codes

Assigning the same number of output nodes to the different classes may not be ideal for a real-world data-set. Following this idea, it may be interesting to allocate different amounts of the output material to the different combinations of classes, as for instance the distributions of examples can be more correlated in a subset of classes than on average in the whole data-set.

An experimental scheme inspired from the quintuplet configuration is tested, with 252 output neurons parted in 126 clusters of size 2 where a soft-max is applied. Instead of rigorously associating an output cell with each possible quintuplet however, 10 binary output codes are generated following a binomial distribution. This distribution is modulated to make vary the average number of ones in the output codes. Used as output for a shallow network, the average number of ones has an impact on classification performance, as shown by figure 2.

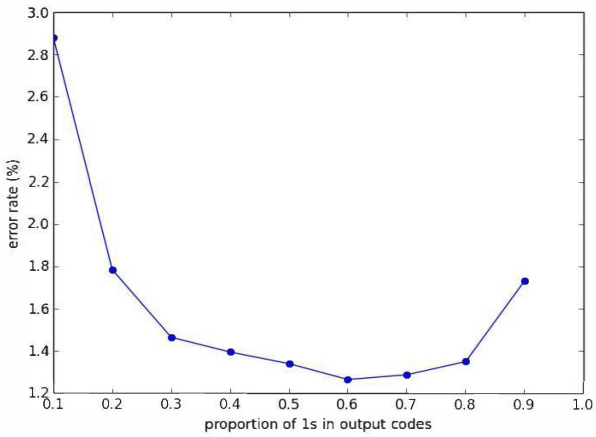

Figure 2: Influence of the proportion of ones in the output target vectors on classification error rates.

We see here that the error rate is maximal when there are less than 10% of ones in the output codes. It decays with higher values down to a minimum reached when there are around 60% of ones. Above that proportion, the error rate raises again.

The minimal distance is obtained when generating as many ones as zeros on output as shown on figure 3, whereas the classifier performs better with output codes made of 60% up to 80% of ones. The error rate is also about 1.7 times lower for 90% of ones as compared to the case with 10% of ones, while the minimal distance is the same in both cases. This asymmetry is due to the way the class label is selected at test time, where a majority voting procedure is applied in which each output unit getting a value of 1 increments the score of all classes it is associated with. In this setting, the higher the number of ones in the output targets, the more connection weights end up being involved in the decision process at test time. The classifier

thus makes use of a finer features-output mapping in this case. With too many ones however, e.g. 90%, the distance between codes becomes too low which affects performance.

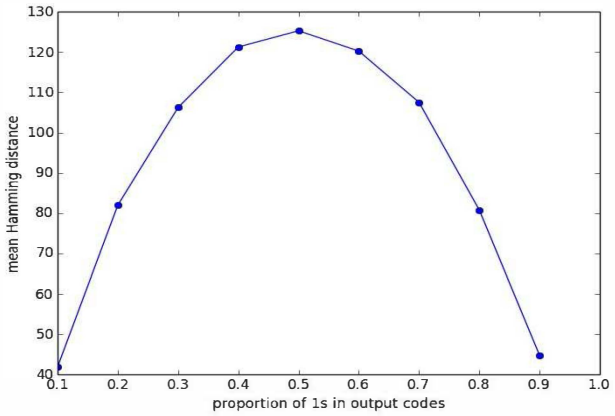

Figure 3: Measured mean Hamming distance between generated class codes depending on the proportion of ones.

4.2. MNIST – Shallow Network

The first test with the assembly codes defined in section 3.3, applies the shallow network to the MNIST data-set. The goal of this test is to compare the assemblies in similar settings, rather than achieving a competitive result to the state-of-the-art. The results indicate that Quintuplets are better than Grandmother cells and also that Couples are better than Grandmother cells, but it is inconclusive in the comparison between Couples and Quintuplets (less than two misclassified images of difference on average between the best networks of the two assemblies).

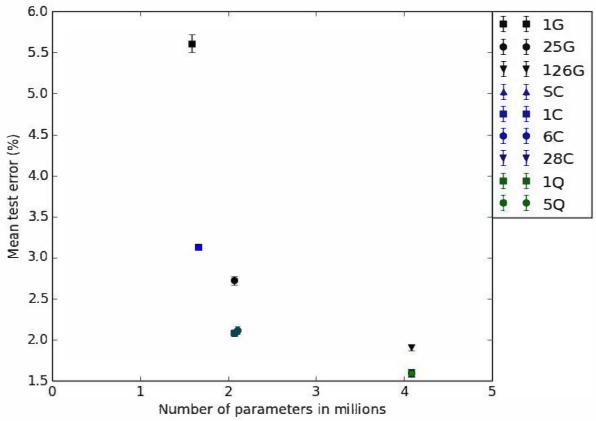

Figure 4: Results summary for a shallow network on the MNIST data-set

4.3. MNIST – Deep Network

Another test is conducted on the MNIST data-set, now trying to emulate the results from [8]. It is summarized in figure 5. Despite respecting the hierarchy of Quintuplets ≥ Couples ≥ Grandmother cells, the results are too close to take any conclusions (less than one image on average between 5Q and 126G).

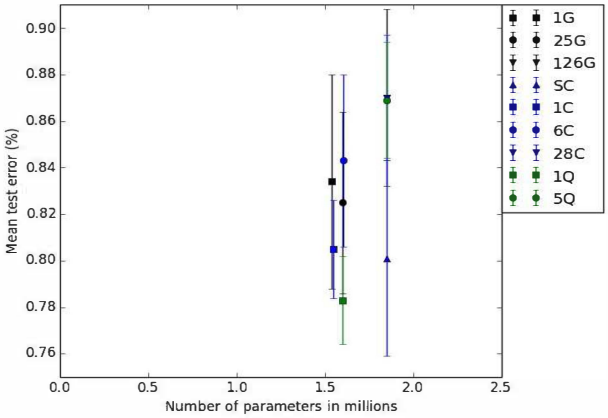

Figure 5: Results summary for a deep network on the MNIST data-set

4.4. SVHN – Deep Network

Finally, applying the deep network to the SVHN data-set allows us to obtain more significant results than the ones obtained over MNIST. SVHN is more difficult to classify and has been less extensively studied. The results respect the hierarchy drawn from coding theory in section 2 (Quintuplets ≥ Couples ≥ Grandmother Cells), with an average distance of 0.11% (~ 26 images) between the worst quintuplets network and the best couples network and 0.33% (~ 85 images) between the worst quintuplets network and the best grandmother cells.

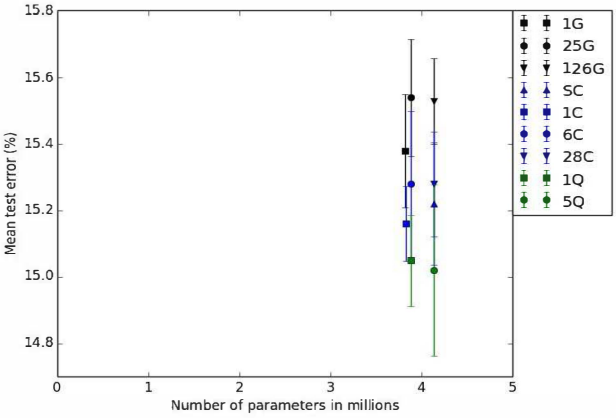

Figure 6: Results summary for a deep network on the SVHN data-set

On both MNIST and SVHN, these results show the interest of the quintuplets code which gives better performance than most other output codes for a comparable number of parameters. The only case where a code outperforms the quintuplets is the "special couples" (SC) code of non-repeated couple cells clusters, which beats 5Q on MNIST. It is also worth noting that the non-repeated couples perform better than the repeated 28C on both data-sets.

V. CONCLUSIONS AND PERSPECTIVES

A way to represent categories in multi-class problems is presented, that departs itself from the usual "grandmother cell" approach. Experimental results show that an assembly of neurons representing meta-classes can do a better job as output for a neural network. On the widely studied MNIST data-set, we use as starting point a multi-layer perceptron network that is close in performance to the state-of-the-art, and show that the proposed assembly codes can improve its accuracy. Furthermore, these results are in accordance with predictions drawn from coding theory in that a higher minimal Hamming distance between code words typically results in a better training of the classifier. The comparison also holds when code repetition is used to adjust the lengths of the outputs, so that all compared networks have about the same number of parameters.

However one of the goals underlying this work is to combine deep neural networks with clustered clique-based associative memories, and the quintuplet configuration which gives the best results here would not be a good fit for this use. Indeed, even if it has a good minimal distance, the resulting clique patterns would have too many nodes in common. In this 10-class case, couple cells would be a good trade-off with a high minimal Hamming distance and a good suitability with Clustered Cliques Networks. Similar trade-offs may also be found for higher-dimensional classification problems.

A prospect of our ongoing work is also to be able to classify data-sets with a high number of classes. This way the compression power of the clustered clique code may be exploited, which allows to support a big dictionary of code words with a limited amount of material while keeping a good word recognition ability. The assembly output deep learner presented is a first step in this direction. Sets of images of words are a very interesting application case as it can have tens of thousands of categories.

ACKOWLEDGEMENT

The authors would like to thank NVIDIA for providing us with a free graphics card allowing to speed up computations for the experiments performed during this work. The software developed to perform the simulations presented here was based on libraries Theano [12] and Keras [13].

REFERENCES

[1] S. Tulyakov, S. Jaeger, V. Govindaraju, and D. Doermann, "Review of classifier combination methods," In Machine Learning in Document Analysis and Recognition, Springer Berlin Heidelberg, pp. 361-386, 2008.

[2] M. A. Bagheri, G. A. Montazer, and S. Escalera, "Error correcting output codes for multiclass classification: application to two image vision problems," In *Artificial Intelligence and Signal Processing* (AISP), 2012 *16th CSI International Symposium on*, IEEE, pp. 508-513, 2012.

[3] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," In *Neurocomputing*, Springer Berlin Heidelberg, pp. 227-236, 1990.

[4] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity", *IEEE Trans. Neural Networks*, vol. 22, n° 7, pp. 1087-1096, July 2011.

[5] B. Kamary Aliabadi, C. Berrou, V. Gripon and X. Jiang, "Storing sparse messages in networks of neural cliques," *IEEE Trans. Neural Networks and Learning Systems*, vol. 25, n° 5, pp. 980-989, May 2014.

[6] J. I. Arribas et J. Cid-Sueiro, "A model selection algorithm for a posteriori probability estimation with neural networks" *IEEE Trans. Neural Networks*, vol. 16, n° 4, pp. 799-809, July 2005.

[7] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998.

[8] M. Pezeshki, L. Fan, P. Brakel, A. Courville, and Y. Bengio, "Deconstructing the Ladder Network Architecture," *arXiv preprint arXiv:1511.06430*, 2015.

[9] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," In *NIPS workshop on deep learning and unsupervised feature learning*, Vol. 2011, Granada, Spain, p.4, 2011.

[10] D. Kingma, and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[11] S. Ioffe, and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167.*, 2015.

[12] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, and Y. Bengio, "Theano: new features and speed improvements," *arXiv preprint arXiv:1211.5590*, 2012.

[13] F. Chollet, "Keras: Theano-based deep learning library," Code: https://github. com/fchollet. Documentation: http://keras. io., 2015.