Using Different Cost Functions to Train Stacked Auto-encoders

Telmo Amaral*, Luís M. Silva*†, Luís A. Alexandre‡,
Chetak Kandaswamy*, Jorge M. Santos*§, Joaquim Marques de Sá*¶
*Instituto de Engenharia Biomédica (INEB), Universidade do Porto, Portugal. Email: tga@fe.up.pt
†Departamento de Matemática, Universidade de Aveiro, Portugal. Email: lmas@ua.pt
‡Instituto de Telecomunicações, Universidade da Beira Interior, Covilhã, Portugal
§Departamento de Matemática, Instituto Superior de Engenharia do Instituto Politécnico do Porto, Portugal
¶Dep. de Engenharia Electrotécnica e de Computadores, Fac. de Engenharia da Univ. do Porto, Portugal

***Abstract*—Deep neural networks comprise several hidden layers of units, which can be pre-trained one at a time via an unsupervised greedy approach. A whole network can then be trained (fine-tuned) in a supervised fashion. One possible pre-training strategy is to regard each hidden layer in the network as the input layer of an auto-encoder. Since auto-encoders aim to reconstruct their own input, their training must be based on some cost function capable of measuring reconstruction performance. Similarly, the supervised fine-tuning of a deep network needs to be based on some cost function that reflects prediction performance. In this work we compare different combinations of cost functions in terms of their impact on layer-wise reconstruction performance and on supervised classification performance of deep networks. We employed two classic functions, namely the cross-entropy (CE) cost and the sum of squared errors (SSE), as well as the exponential (EXP) cost, inspired by the error entropy concept. Our results were based on a number of artificial and real-world data sets.**

***Keywords*-cost functions; stacked auto-encoders; deep neural networks**

I. Introduction

Deep architectures, such as neural networks with two or more hidden layers of units, are a class of machines that comprise several levels of non-linear operations, each expressed in terms of parameters that can be learned [2]. The organization of the mammal brain, as well as the apparent depth of cognitive processes, are among the main motivations for the use of such architectures. In spite of this, until 2006, attempts to train deep architectures resulted in poorer performance than that achieved by their shallow counterparts. The only exception to this difficulty was the convolutional neural network [8], a specialized architecture for image processing, modeled after the structure of the visual cortex.

A breakthrough took place with the introduction by Hinton *et al.* of the deep belief network [6], a learning approach where the hidden layers of a deep network are initially treated as restricted Boltzmann machines (RBMs) [10] and pre-trained, one at a time, in an unsupervised greedy approach. Given that auto-encoders [5] are easier to train than RBMs, this unsupervised greedy procedure was soon generalized into algorithms that pre-train the hidden levels of a deep network by treating them as a stack of auto-encoders [3], [7].

The auto-encoder (also called auto-associator or Diabolo network) is a type of neural network trained to output a reconstruction of its own input. Thus, in the training of auto-encoders, input vectors can themselves be interpreted as target vectors. This presents an opportunity for the comparison of various training criteria, namely different cost functions capable of reflecting the mismatch between inputs and targets.

The information theoretical concept of minimum error entropy has been recently applied by Marques de Sá *et al.* [9] to data classification machines, yielding evidence that risk functionals do not perform equally with respect to the attainment of solutions that approximate the minimum probability of error. In their work, the features of classic cost functions such as the sum of squared errors (SSE) and the so-called cross-entropy (CE) cost are discussed, and some approaches inspired by the error entropy concept are proposed. One such approach is a parameterized function called exponential (EXP) cost, sufficiently flexible to emulate the behavior of classic costs, namely SSE and CE, and to exhibit properties that are desirable in certain types of problems, such as good robustness to the presence of outliers.

In this work, we aimed to compare the performances of SSE, CE, and EXP costs when employed both in the unsupervised pre-training and in the supervised fine-tuning of deep networks whose hidden layers are regarded as a stack of auto-encoders. To the best of our knowledge, this type of comparison has not been done before in the context of deep learning. Using a number of artificial and real-world data sets, we first compared pre-training cost functions in terms of their impact on the reconstruction performance of hidden layers. Given that the output layer of our networks was designed for classification learning, we also compared various combinations of pre-training and fine-tuning costs in terms of their impact on classification performance.

978-1-4799-2605-3/13 $31.00 © 2013 IEEE
DOI 10.1109/MICAI.2013.20

II. Stacked Auto-encoders

The auto-encoder (AE) is a simple network that tries to produce at its output what is presented at the input. As exemplified in Fig. 1a, the basic AE is in fact a simple neural network with one hidden layer and one output layer, subject to two restrictions: the weight matrix of the output layer is the transposed of the weight matrix of the hidden layer (i.e. weights are clamped); and the number of output neurons is equal to the number of inputs.

The values of the hidden layer neurons, called the *encoding*, are obtained via Equation (1), where $\mathbf{x}$ is the input vector, s denotes the sigmoid function, $\mathbf{b}$ is the vector of hidden neuron biases, and $\mathbf{W}$ is the matrix of hidden weights. The values of the output neurons, called the *decoding*, are computed as in Equation (2), where $\mathbf{c}$ is the vector of output neuron biases. Unsupervised learning of the weights and biases of AEs can be achieved by gradient descent, based on a training set of input vectors.

$$\mathbf{h}(\mathbf{x}) = s\left(\mathbf{b} + \mathbf{W}\mathbf{x}\right) \tag{1}$$

$$\hat{\mathbf{x}}(\mathbf{h}(\mathbf{x})) = s\left(\mathbf{c} + \mathbf{W}^T\mathbf{h}(\mathbf{x})\right) \tag{2}$$

The deep networks we used for classification had an architecture similar to that shown in Fig. 1c, with a layer of inputs, two or more hidden layers, and an output layer L with as many units as target classes. The hidden layers of such a network can be *pre-trained* in an unsupervised way, one at a time, starting from the bottom layer. In order to be pre-trained, a hidden layer is "unfolded" to form an AE. Once a given AE has learned to reconstruct its own input, its output layer is no longer needed and its hidden layer becomes the input to the next level of the deep network, as shown in Fig. 1b. The next level is in turn pre-trained as an individual AE, and the process is repeated until there are no more hidden layers.

The goal of unsupervised pre-training is to bring the network's hidden weights and biases to a region of the parameter space that constitutes a better starting point than random initialization, for a subsequent supervised training stage. In this context, the supervised training stage is usually called *fine-tuning* and can be achieved by conventional gradient descent, based on a training set of paired input and target vectors. It should be noted that the output layer weights $\mathbf{W}^{(L)}$ are not involved in the pre-training stage, so they are randomly initialized and learned only in the fine-tuning stage.

III. Cost Functions

A. Pre-training

Since the goal of AE training is to obtain at the output the same data values fed into the input, an adequate cost function C should compare these two vectors. The classical approach is to use an empirical version of either the SSE cost, as in Equation (3), or the CE cost, as in Equation (4). Another possibility is to use the EXP cost shown in Equation (5), which features an extra parameter τ. In these expressions, $\hat{x}_k$ and x_k denote the kth elements of the output and input vectors, respectively.

$$C_{SSE}(\hat{\mathbf{x}}, \mathbf{x}) = \sum_k (\hat{x}_k - x_k)^2 \tag{3}$$

$$C_{CE}(\hat{\mathbf{x}}, \mathbf{x}) = -\sum_k \Big(x_k \ln(\hat{x}_k) + (1 - x_k)\ln(1 - \hat{x}_k)\Big) \tag{4}$$

$$C_{EXP}(\hat{\mathbf{x}}, \mathbf{x}) = \tau \exp\Big(\frac{1}{\tau}\sum_k (\hat{x}_k - x_k)^2\Big) \tag{5}$$

Given a training input vector, the change that it implies in weight W_{ji} connecting input i to hidden neuron j can be computed via gradient descent [4], in terms of the partial derivative of the cost with respect to that weight. This derivative can be expressed as in Equation (6), where A and B_k take the values shown in Table I [1]. The expression for the derivative with respect to a hidden bias b_j can be obtained from this equation by setting x_i and $\hat{x}_i$ to 1, and the derivative with respect to an output bias c_i can be obtained by setting h_j to 1.

$$\frac{\partial C}{\partial W_{ji}} = A\Big(B_i(\hat{x}_i - x_i)h_j + x_i h_j(1 - h_j)\sum_k B_k(\hat{x}_k - x_k)W_{jk}\Big) \tag{6}$$

Table I
Expressions for A and B_k in Equation (6), for each cost function.

Cost	A	B_k
SSE	2	$\hat{x}_k(1 - \hat{x}_k)$
CE	1	1
EXP	$2\exp\left(\frac{1}{\tau}\sum_k(\hat{x}_k - x_k)^2\right)$	$\hat{x}_k(1 - \hat{x}_k)$

B. Fine-tuning

The expressions for the SSE and EXP cost functions used in the supervised training stage are identical to Equations (3) and (5), respectively, keeping in mind that $\hat{\mathbf{x}}$ and $\mathbf{x}$ now become output vector $\mathbf{y}$ and target vector $\mathbf{t}$, respectively. As to the CE cost, it now assumes the multi-class expression shown in Equation (7), where index k iterates through the network's outputs.

$$C_{CE}(\mathbf{y}, \mathbf{t}) = -\sum_k t_k \ln y_k \tag{7}$$

As before, partial derivatives are needed to obtain weight changes via gradient descent. According to Bishop [4], the

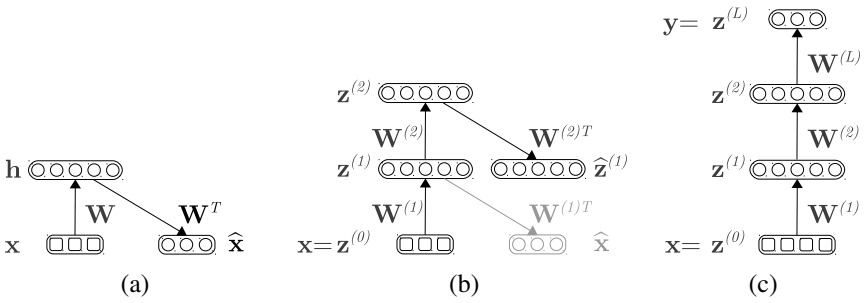


Figure 1. (a) An auto-encoder. (b) Pre-training of hidden layers of a deep network using auto-encoders. (c) A complete deep network with two hidden layers and an output layer. (Based on Larochelle *et al.* [7].)

partial derivative of the cost with respect to a weight W_{ji} connecting unit (or input) i to unit (or output) j can be expressed generically as in Equation (8), where δ_j is the so-called delta error associated with unit j. The expression for the derivative with respect to a hidden or output bias b_j can be obtained by setting z_i to 1.

$$\frac{\partial C}{\partial W_{ji}} = \delta_j z_i \tag{8}$$

If W_{ji} belongs to the output layer of the network, δ_j should be obtained from the definition in Equation (9), where a_j represents the jth argument of the output activation function (in our case the logistic softmax). For different cost functions, the expression of δ_j takes the forms shown in Table II, where I_{kj} is a selection variable corresponding to element kj of an identity matrix.

$$\delta_j \equiv \frac{\partial C}{\partial a_j} \tag{9}$$

Table II
EXPRESSION OF δ_j FOR A UNIT IN THE OUTPUT LAYER, AS DEFINED IN EQUATION (9), FOR EACH COST FUNCTION.

Cost	δ_j **associated with output unit** j
SSE	$\sum_k (y_k - t_k) y_k (I_{kj} - y_j)$
CE	$(y_j - t_j)$
EXP	$2 \exp\left(\frac{1}{\tau} \sum_k (y_k - t_k)^2\right) \sum_k (y_k - t_k) y_k (I_{kj} - y_j)$

If W_{ji} belongs to a hidden layer, δ_j can be obtained recursively from Equation (10), using the errors δ_k already computed for the layer above. In this expression, $h'(.)$ denotes the derivative of the hidden layer's activation function (in our case always a logistic sigmoid).

$$\delta_j = h'(a_j) \sum_k \delta_k W_{kj} \tag{10}$$

Table III
(A) CHARACTERISTICS OF EACH DATA SET. (B) HYPER-PARAMETER VALUES SELECTED FOR THE MODEL USED WITH EACH DATA SET.

(a)

Data set	**# Features**	**# Targets**	**# Instances**		
			Train.	***Valid.***	***Test***
`adult`	123	2	5000	1414	26147
`cb4x4(10)`	2	2	2000	1000	1000
`cb4x4(25)`	2	2	2000	1000	1000
`ocr_letters`	128	26	32152	10000	10000
`environ2cl`	7	2	1700	850	850
`environ4cl`	7	4	1700	850	850

(b)

Data set	**Arch.**	**Pre-training**		**Fine-tuning**		
		η_u	τ_u	η_s	τ_s	***L.a.***
`adult`	2, 2	0.01	10	0.1	10	25
`cb4x4(10)`	25,25	* 1.5	7	0.05	0.75	100
`cb4x4(25)`	25,25,25	* 0.01	0.31	0.05	2	160
`ocr_letters`	50,50,50	0.0007	10	0.1	4	50
`environ2cl`	10,10,10	0.35	-2.7	0.1	2	130
`environ4cl`	10,10,10	0.3500	-2.70	0.10	2.00	120

IV. EXPERIMENTS AND RESULTS

A. Data Sets and Selection of Hyper-parameters

We used six data sets appropriate for classification experiments, whose numbers of features, targets, and instances are shown in Table III. The `adult`[1] and `ocr_letters`[2] sets contain publicly available, real-world binarized data; `cb4x4(10)` and `cb4x4(25)` are artificially generated sets with a 4×4 checkerboard pattern, real-valued features, and proportions of the minority class of 0.10 and 0.25, respectively; `environ2cl` and `environ4cl` are private, real-world sets containing real-valued environmental data. Each data set was divided into three fixed subsets, for training, validation, and test purposes.

For each data set, we built a deep network of the type exemplified in Fig. 1c. Hyper-parameters were selected via a simplified procedure, starting from default values, then tuning each parameter individually to minimize the classification error on the validation set, averaged over 10 repetitions. Table III shows the values obtained for all

[1]See http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#a5a.
[2]See http://ai.stanford.edu/~btaskar/ocr/.

Table IV
RECONSTRUCTION MSE ACHIEVED FOR EACH DATA SET, HIDDEN LAYER, AND COST FUNCTION, COMPUTED OVER TEST DATA.

Data set	Hidden layer	Pre-training cost function		
		CE	*SSE*	*EXP*
adult	2	**0.0088**±0.0010	0.0090±**0.0008**	0.0305±0.0046
	1	**6.5199**±0.0088	6.7640±0.0269	6.5465±**0.0061**
cb4x4(10)	2	**0.0003±0.0001**	**0.0003±0.0001**	**0.0003±0.0001**
	1	**0.0005**±0.0003	0.0022±**0.0000**	0.0022±**0.0000**
cb4x4(25)	3	0.0046±0.0018	**0.0002±0.0000**	0.0011±0.0002
	2	0.0025±0.0007	**0.0003±0.0001**	0.0013±0.0002
	1	**0.0021±0.0000**	**0.0021±0.0000**	0.0066±0.0008
ocr_letters	3	0.4727±0.0124	**0.3378±0.0074**	0.5294±0.0143
	2	0.6687±0.0158	**0.5694±0.0125**	0.8502±0.0222
	1	**4.6354**±0.0456	4.7581±0.0367	4.8055±**0.0317**
environ2cl	3	0.0080±0.0038	0.0026±**0.0004**	**0.0023**±0.0007
	2	0.0054±0.0023	**0.0031±0.0005**	0.0107±0.0026
	1	0.0089±0.0013	**0.0071±0.0006**	0.0476±0.0052
environ4cl	3	0.0022±0.0007	0.0015±0.0011	**0.0012±0.0002**
	2	0.0027±**0.0007**	**0.0014**±0.0008	0.0054±0.0010
	1	0.0070±**0.0009**	**0.0034**±0.0010	0.0144±0.0015

parameters, namely: the architecture (e.g. three layers with 50 units each for the ocr_letters data set); the pre-training learning rate η_u and EXP parameter τ_u, the fine-tuning learning rate η_s and EXP parameter τ_s; and the so-called look-ahead, a parameter that controls the mechanism for early stopping of the fine-tuning. We chose to use the same number of units in all hidden layers. Values marked with an asterisk are not fixed learning rates, but initial values for an adaptive form of learning rate [9]. All parameters were selected for the CE/CE cost combination (i.e. pre-training CE cost and fine-tuning CE cost), except naturally for the two EXP τ parameters.

For each data set, we pre-trained the network's hidden layers, tested their reconstruction performances, fine-tuned the whole network, and tested its classification accuracy. All nine combinations of pre-training and fine-tuning cost functions were tried, plus a combination of no pre-training with CE cost. For each combination, 20 repetitions of full training and testing were executed. In order to incorporate the SSE and EXP cost functions, we adapted an implementation of stacked auto-encoders originally developed by Hugo Larochelle, based on the MLPython library[3].

B. Experimental Results

We compared the three pre-training cost functions in terms of their impact on the reconstruction performance achieved by each hidden layer, when "unfolded" as an auto-encoder. For that purpose, we chose to use the reconstruction mean squared error (MSE) as a common metric, computed over test data. Table IV shows the obtained results. The best MSE values and standard deviations computed over 20 repetitions are shown in bold.

The main conclusion that can be drawn from these results is that SSE is a good choice of pre-training cost, regardless of the data set. In all cases except one (cb4x4(10)), SSE yielded a reconstruction MSE that was either the lowest or close to the lowest. SSE also tended to yield the lowest variances. This superiority could arguably be attributed to the fact that the SSE cost, expressed in Equation (3), coincides in practice with the MSE metric being used.

[3]See http://www.dmi.usherb.ca/~larocheh/mlpython/.

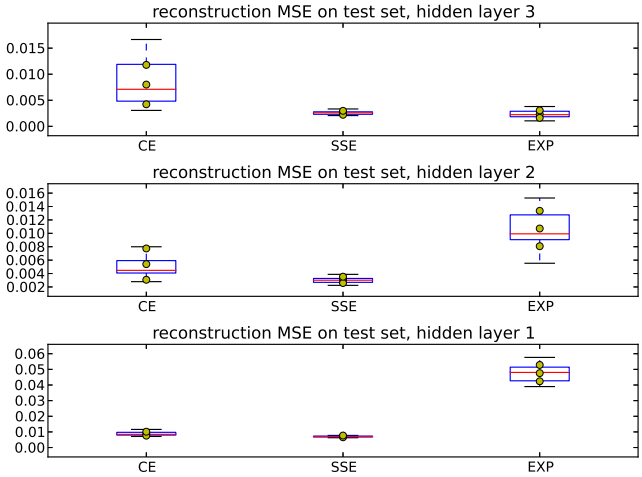


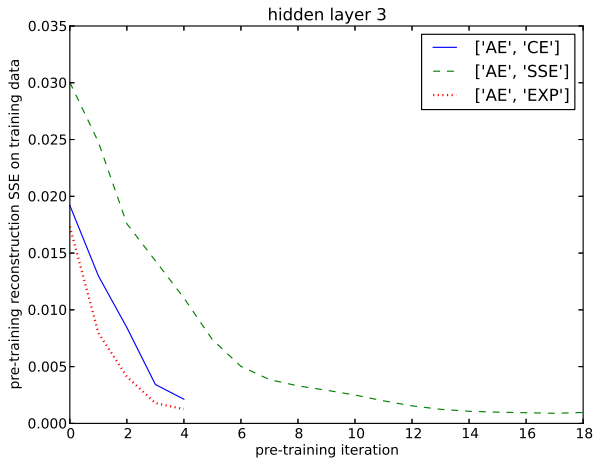
Figure 2. Box plots of reconstruction MSE for the environ2cl data set, for each hidden layer and cost function, computed over test data.

Nevertheless, with the binary data sets adult and ocr_letters, CE yielded the lowest mean errors for the first layer. These were the only two situations where the x_k values in Equation (4) were binary, and this equation in fact assumes binary x_k (though it produces meaningful cost values even with real-valued x_k). It appears that CE could be the best choice for pre-training the first hidden layer in the presence of binary data.

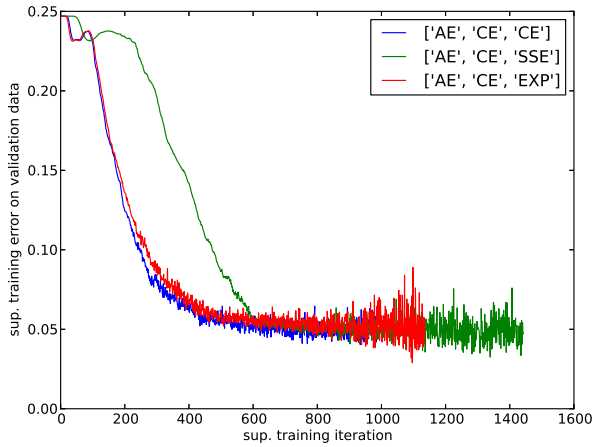
It is interesting to note that the MSE associated with each given cost function tended to decrease from the bottom hidden layer to higher layers, meaning that the learned representations became progressively easier to reconstruct. In the case of EXP, this happened with all data sets. This effect, as well as the tendency of SSE to yield the best results, can be observed e.g. in the box plots shown in Fig. 2 for the environ2cl data.

More detail on what happens during pre-training could be obtained by plotting the evolution of the reconstruction performance on training data, throughout the training of each hidden layer, as exemplified in Fig. 3a for the third layer of the architecture selected for the environ4cl data. These plots illustrate a problem observed with several data sets, in that the pre-training based on either CE or EXP appears to have prematurely ended due to the implemented early-stopping mechanism. The application of this mechanism [7] should therefore be improved in future work, possibly by selecting the value of an associated hyper-parameter for each data set individually (as was done with the look-ahead parameter, used to control early-stopping of fine-tuning).

We compared different combinations of pre-training and



(a)



(b)

Figure 3. (a) Evolution of reconstruction performance on training data during pre-training, for each pre-training cost, for the third layer of the model selected for `environ4cl` data. (b) Evolution of classification error on validation data during fine-tuning, for each fine-tuning cost, for `cb4x4(25)` data. (Best viewed in color.)

fine-tuning costs in terms of their impact on the classification error computed over test data. Table V shows the obtained results. Since all data sets were unbalanced, we chose to compute the balanced classification error (i.e. the average of intra-class errors). For each data set, the best mean error and error standard deviation computed over 20 repetitions are shown in bold. For each particular pre-training cost, the best mean error and error variance are marked with A and B, respectively. For each particular fine-tuning cost, the best mean error and error variance are marked with C and D, respectively.

For four of the data sets, the results achieved with no pre-training (and fine-tuning with CE) were clearly worse than those obtained with at least some of the pre-training cost functions, as would be expected. This was not the case, however, with the `adult` and `ocr_letters` sets.

At any rate, the main conclusion that can be drawn from the obtained results is that none of the combinations of pre-training and fine-tuning costs stood out as yielding either the lowest mean errors or the lowest error variances for most data sets.

For the `cb4x4(10)` data the training phase failed for several cost combinations, which yielded a balanced error of 50% or close. This failure happened necessarily at the fine-tuning stage, since no problems were observed at the pre-training stage (see Table IV). Interestingly, the only successful cost combinations were CE/CE and the three combinations involving the fine-tuning EXP cost. It should be noted that CE/CE was the combination for which the learning rate and look-ahead parameters were selected. This may indicate that, for heavily unbalanced data sets, the fine-tuning stage is not particularly robust to a departure from the cost function used to select parameters. Indeed, with the more balanced `cb4x4(25)` data, fine-tuning converged for all cost functions. On the other hand, this also suggests that the fine-tuning EXP cost may help to overcome this type of limitation.

It can be observed that, when the pre-training cost is CE, the lowest mean errors (marked with A) and the lowest error variances (marked with B) were achieved when the selected fine-tuning cost was either CE or EXP. In other words, when pre-training with CE, fine-tuning with SSE is not a good choice. This can be observed in more detail e.g. in the box plots for the `adult` and `ocr_letters` data shown in Fig. 4.

The C markers in Table V help to verify that, when fine-tuning with CE, three of the lowest mean errors were achieved in combination with SSE pre-training. Similarly, when fine-tuning with SSE, four lowest mean errors were achieved with SSE pre-training and, when fine-tuning with EXP, three lowest mean errors were achieved with SSE pre-training. Thus, given a particular fine-tuning cost, SSE appears to be a reasonable choice for the pre-training stage, as can be observed e.g. in Fig. 4a for the `adult` data. This suggests that the superiority of pre-training via SSE (previously observed in Table IV) is real and not merely a consequence of using MSE to measure reconstruction performance.

The D markers in Table V indicate that, curiously, when fine-tuning with CE, five lowest error variances were yielded in combination with CE pre-training, whereas, when fine-tuning with SSE, four lowest error variances were achieved with SSE pre-training and, when fine-tuning with EXP, four lowest error variances were achieved with EXP pre-training. In other words, given a particular fine-tuning cost, using the same pre-training cost appears to be beneficial if one wishes to achieve low error variance. This is exemplified also in Fig.

Table V
PERCENT BALANCED CLASSIFICATION ERROR FOR EACH DATA SET AND COST COMBINATION, COMPUTED OVER TEST DATA. (BEST VIEWED IN COLOR.)

Data set	**Pre-training / fine-tuning cost functions**									
	None	***CE***			***SSE***			***EXP***		
	CE	***CE***	***SSE***	***EXP***	***CE***	***SSE***	***EXP***	***CE***	***SSE***	***EXP***
adult	24.7±01.1	25.2±00.4 A B D	26.2±00.7	25.8±01.0	**24.4**±00.6 A C	25.6±**00.1** B C D	25.2±00.8 C	25.3±00.5 A	25.8±00.3 B	25.7±00.3 B D
cb4x4(10)	50.0±00.0	**13.6**±03.3 A B C D	50.0±00.0	20.2±06.1	45.4±11.0	50.0±00.0	19.6±03.8 A B	45.7±10.3	50.0±00.0	18.6±**02.0** A B C D
cb4x4(25)	50.0±00.0	05.8±01.2 D	23.8±20.4	05.4±01.1 A B	05.2±01.3 C	**04.7±00.8** A B C D	05.6±01.3	05.4±01.2 D	21.9±20.7	04.8±00.9 A B C D
ocr_letters	24.1±**00.5**	24.4±00.7 A B D	26.5±01.2	24.8±00.7 B D	23.6±00.9 A	24.9±00.9 C D	23.7±00.7 B C D	**23.4**±00.7 A B C D	25.1±01.1	23.8±00.8
environ2cl	50.0±00.0	37.5±02.6 B C D	37.3±03.2 C	**35.7**±04.6 A C	38.2±03.2 B	38.1±03.5	36.7±03.2 A B	39.5±02.6 D	38.8±03.0 A	39.9±**01.7** B D
environ4cl	75.0±00.0	58.9±03.4 B	58.2±04.5	58.0±04.0 A	57.9±03.2 C	**55.4**±03.6 A C D	57.7±**02.4** B C D	59.4±02.5 B D	58.5±03.8 A	60.1±04.2

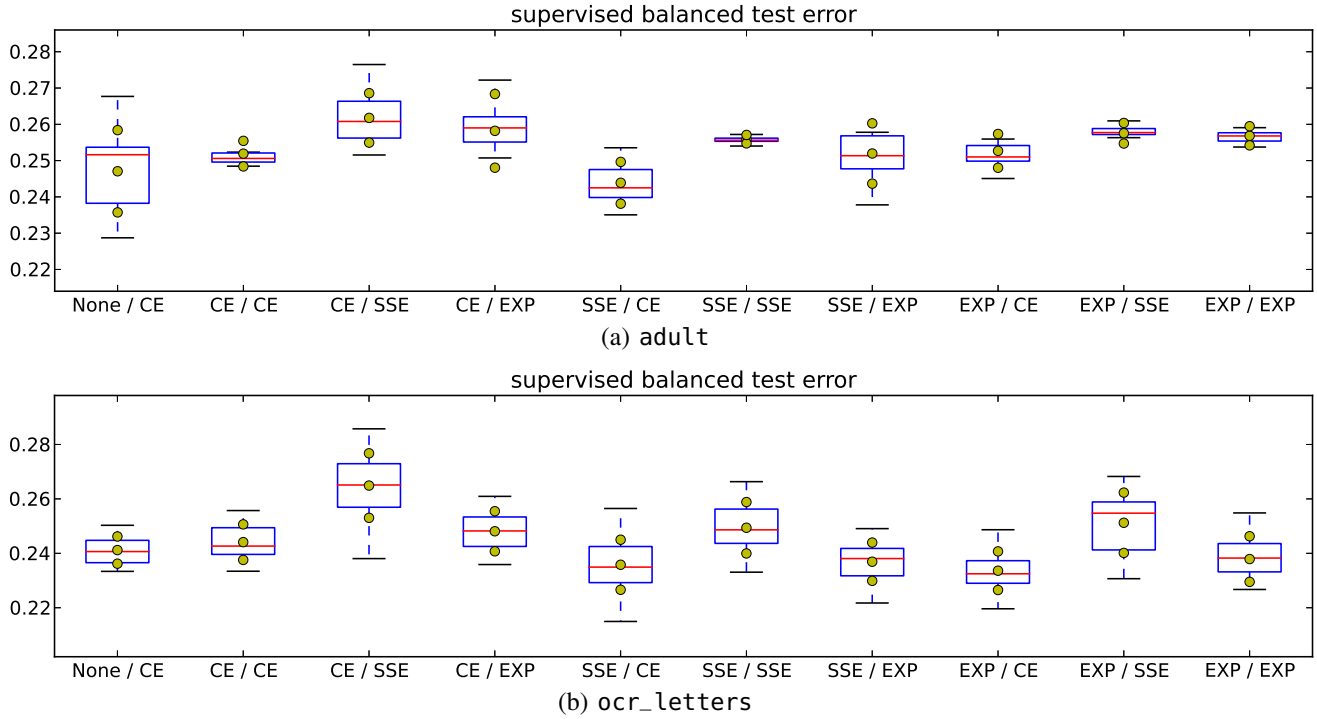


Figure 4. Box plots of balanced classification errors computed over test data, for each cost combination, for (a) `adult` data and (b) `ocr_letters` data.

4a, for the `adult` data.

Fig. 3b exemplifies the evolution of the classification error on validation data during training, for the `cb4x4(25)` data set. It can be seen that the errors associated with CE and EXP fine-tuning costs evolved in very similar ways. To an extent, this happened with all data sets and was an expected behavior, since positive values were always selected for the EXP τ_s parameter (see Table III), leading the EXP cost function to approximate the behavior of the CE cost [9]. The increasing instability of all three errors towards the end of training was observed also with the environmental data sets, and may be due to insufficient amounts of training and validation data. This effect should be further investigated.

Table VI shows the average times taken by pre-training and fine-tuning iterations, in seconds, for each cost function. In the case of pre-training, layer-wise times are shown. The last row shows the average ratios to the CE time, for both SSE and EXP. In terms of pre-training, it can be observed that SSE and EXP were actually slightly faster than CE, though it should be noted that each pre-training iteration involved the computation not only of cost derivatives, but also of the cost itself, used by the implemented early-stopping mechanism. In terms of fine-tuning, both SSE and EXP took approximately 60% more training time. All experiments were executed on Linux using a desktop computer equipped with an Intel Core i7-950 processor and enough physical

Table VI
AVERAGE ITERATION TIMES IN SECONDS, FOR EACH DATA SET AND COST FUNCTION: (A) DURING PRE-TRAINING OF EACH HIDDEN LAYER; AND (B) DURING FINE-TUNING.

(a)

Data set	**Pre-training**			
	Hidden layer	***CE***	***SSE***	***EXP***
adult	2	3.453	3.253	3.481
	1	3.135	2.772	3.106
cb4x4(10)	2	0.806	0.631	0.671
	1	0.614	0.450	0.489
cb4x4(25)	3	0.916	0.867	0.937
	2	0.788	0.718	0.806
	1	0.611	0.545	0.633
ocr_letters	3	16.331	14.657	15.377
	2	13.949	12.338	13.181
	1	13.399	12.503	13.362
environ2cl	3	0.833	0.812	0.862
	2	0.715	0.690	0.739
	1	0.585	0.562	0.613
environ4cl	3	0.833	0.812	0.445
	2	0.713	0.689	0.387
	1	0.589	0.560	0.318
Average ratios to CE:		1.000	0.912	0.895

(b)

Data set	**Fine-tuning**		
	CE	***SSE***	***EXP***
adult	0.476	0.839	0.956
cb4x4(10)	0.238	0.317	0.367
cb4x4(25)	0.291	0.434	0.518
ocr_letters	6.601	9.621	10.072
environ2cl	0.260	0.437	0.493
environ4cl	0.265	0.441	0.275
Average ratios to CE:	1.000	1.565	1.632

memory to prevent swapping.

V. CONCLUSIONS AND FUTURE DIRECTIONS

In this work, we compared different cost functions used in the pre-training and fine-tuning of deep networks, by performing experiments with a variety of data sets. In general, the best layer-wise reconstruction performance was achieved by SSE pre-training, though with binary data CE yielded the lowest errors for the first hidden layer. Classification performance was found to vary little with the combination of pre-training and fine-tuning costs. When pre-training with CE, fine-tuning via SSE was found not to be a good choice. In general, the choice of the same pre-training and fine-tuning costs yielded classification errors with lower variance.

With a heavily unbalanced artificial data set, fine-tuning failed except for the cost combination used to tune the model's hyper-parameters and for those cost combinations that involved EXP fine-tuning. This seeming robustness of EXP fine-tuning should be further investigated, using a wider variety data sets. Future work should also focus on improving both the pre-training early stopping mechanism and the stability of fine-tuning after large numbers of iterations. In future experiments we plan to adopt GPU-based processing, to allow the use of more computationally demanding data sets, such as variants of the popular MNIST character recognition set.

ACKNOWLEDGMENT

This work was financed by FEDER funds through the *Programa Operacional Factores de Competitividade* – COMPETE and by Portuguese funds through FCT – *Fundação para a Ciência e a Tecnologia* in the framework of the project PTDC/EIA-EIA/119004/2010.

REFERENCES

[1] T. Amaral, Luís M. Silva, and Luís A. Alexandre. Using different cost functions when pre-training stacked auto-encoders. Technical Report 1/2013, Instituto de Engenharia Biomédica / Neural Networks Interest Group (INEB/NNIG), April 2013.

[2] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[3] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Neural Information Processing Systems (NIPS)*, volume 19, pages 153–160, 2007.

[4] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4-5):291–294, 1988.

[6] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[7] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *International Conference on Machine Learning (ICML)*, pages 473–480, 2007.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] J. Marques de Sá, L. Silva, J. Santos, and L. Alexandre. *Minimum Error Entropy Classification*, volume 420 of *Studies in Computational Intelligence*. Springer, 2013.

[10] P. Smolensky. *Parallel distributed processing: explorations in the microstructure of cognition*, volume 1, chapter Information processing in dynamical systems: foundations of harmony theory, pages 194–281. University of Colorado, 1986.