Using different cost functions when pre-training stacked auto-encoders

Telmo Amaral, Luís A. Alexandre, Luís M. Silva

26th April 2013

NNIG Technical Report No. 1/2013

Project "Reusable Deep Neural Networks: Applications to Biomedical Data"
(PDTC/EIA-EIA/119004/2010)

Instituto de Engenharia Biomédica (INEB)
Rua Dr. Roberto Frias, 4200-465, Porto, Portugal

Contents

1 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 [1]. The organisation 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 [7], a specialised architecture for image processing, modelled after the structure of the visual cortex.

A breakthrough took place with the introduction by Hinton et al. of the deep belief network [4], a learning approach where the hidden layers of a deep network are initially treated as restricted Boltzmann machines (RBMs) [9] and pre-trained, one at a time, in an unsupervised greedy approach. Given that auto-encoders [3] are easier to train than RBMs, this unsupervised greedy procedure was soon generalised into algorithms that pre-train the hidden levels of a deep network by seeing them as a stack of auto-encoders [2, 6].

The auto-encoder (also called auto-associator or Diabolo network) is a type of neural network that is trained to output a reconstruction of its own input. Thus, during 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.

Marques de Sá et al. [8] have recently applied the information theoretical concept of minimum error entropy to data classification machines, providing evidence that risk functionals do not perform equally in their attainment of solutions that approximate the minimum probability of error. In their work, Marques de Sá et al. discuss the features of classic cost functions such as the mean squared error (MSE) cost and the so-called cross-entropy (CE) cost, and propose some approaches inspired by the error entropy concept. One such approach is a parameterised function called exponential (EXP) cost, sufficiently flexible to emulate the behaviour of classic costs, such as MSE 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 aim to compare the performances of MSE, CE, and EXP costs when employed in the pre-training of deep networks whose hidden layers are treated as stacks of auto-encoders. A variant of auto-encoders called denoising auto-encoders was also tested. The output layer of the networks we used was designed for classification learning, so the three cost functions could be compared in terms of their impact on classification test error, evaluated on five popular data sets.

The remainder of this report is organised as follows. Section 2 gives a brief overview of the principles behind auto-encoders and denoising auto-encoders, explaining how they can be used to pre-train the hidden layers of deep networks. Section 3 presents the empirical formulation of the MSE, CE, and EXP cost

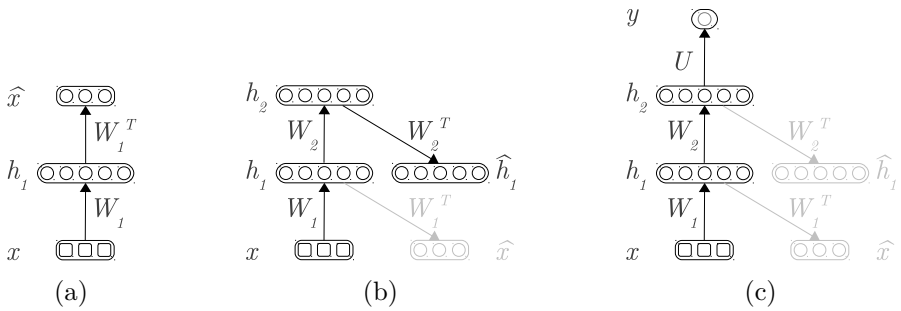

Figure 1: (a) An auto-encoder. (b) Pre-training of hidden layers of a deep network using auto-encoders. (c) A pre-trained deep network with an additional output layer. (Based on Larochelle et al. [6].)

functions to be used in the unsupervised training of auto-encoders. Section (4) presents the calculations for the gradients of the three cost functions. Section 5 presents some data sets that can be used in our experiments, describes the experimental setup and how hyper-parameters can be selected, and explains how the achieved classification results should be reported.

2 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 Figure 1a, the most basic AE is in fact a multi-layer perceptron that has one hidden layer and one output layer, with 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 with 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.

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

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

In a variant of AEs called denoising auto-encoders (DEAs) [10], the input values are corrupted with noise so that some are set to zero, with a probability denoted by ν. The goal is to force learning even with partial input. Unsupervised learning of the weights and biases of AEs and DAEs can be achieved by gradient descent, based on a training set of input vectors (if target vectors are available, they are ignored).

The deep networks we used for classification had an architecture similar to that shown in black in Figure 1c, with a layer of inputs, two or more hidden layers, and an output layer with as many units as classes (or a single output unit, if only two classes exist). 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 seen as belonging to an AE. Once a given AE has learned to reconstruct its own input, its output layer is no longer needed and its hidden layer values become the input to the next level, as shown in Figure 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, as in Figure 1c.

The goal of unsupervised pre-training is to bring the network's weights and biases to a region of the parameter space that constitutes a better starting point for a supervised training stage than a random initialisation. 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 U are not involved in the pre-training stage, so they are randomly initialised and learned only in the supervised stage.

3 Cost functions for auto-encoders

3.1 Conventional auto-encoders

Since the goal of AE training is to obtain at the output the same data values fed into the input, an adequate cost function should compare these two vectors. For real-valued inputs, the typical approach is to use an empirical version of the sum of squared errors (SSE) cost, as in Equation (3), where $\hat{x}_k$ and x_k denote the kth elements of the output and input, respectively, and n_x denotes the number of inputs and outputs. For binary inputs, the cross-entropy (CE) cost in Equation (4) can be used. Another possibility is to use the exponential (EXP) cost in Equation (5), which features an extra parameter τ.

$$C_{SSE}(\hat{\mathbf{x}}, \mathbf{x}) = \sum_{k=1}^{n_x} (\hat{x}_k - x_k)^2 \tag{3}$$

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

$$C_{EXP}(\hat{\mathbf{x}}, \mathbf{x}) = \tau \exp\left(\frac{1}{\tau} \sum_{k=1}^{n_x} (\hat{x}_k - x_k)^2 \right) \tag{5}$$

Given a training input vector, the change that it implies in the weight W_{ij} connecting hidden neuron i and input j can be expressed in a generic way by Equation (6), where n is the number of inputs and outputs, n_h is the number of hidden neurons, η is called the learning rate and, most importantly, Q is a term that changes according to the chosen cost function. The updates to the

biases of hidden neuron i and input j can be expressed as in Equations (7) and (8), respectively.

$$\Delta W_{ij} = -\eta \frac{\partial C}{\partial W_{ij}} \tag{6}$$

$$\Delta b_i = -\eta \frac{\partial C}{\partial b_i} \tag{7}$$

$$\Delta c_j = -\eta \frac{\partial C}{\partial c_j} \tag{8}$$

As detailed in the next section, the derivatives of the SSE, CE, and EXP costs in order to a weight or a bias take the form of the following expressions:

$$
\begin{array}{lll}
\frac{\partial C_{SSE}}{\partial W_{ij}} = & 2 & \left(\overbrace{\hat{x}_j(1-\hat{x}_j)}^{B_j} \quad (\hat{x}_j - x_j) \quad h_i + \quad x_j \quad h_i(1-h_i) \quad \sum_{k=1}^{n_x} \quad \overbrace{\hat{x}_k(1-\hat{x}_k)}^{B_k} \quad (\hat{x}_k - x_k)W_{ik} \right) \\
\frac{\partial C_{SSE}}{\partial b_i} = & 2 & h_i(1-h_i) \quad \sum_{k=1}^{n_x} \quad \hat{x}_k(1-\hat{x}_k) \quad (\hat{x}_k - x_k)W_{ik} \\
\frac{\partial C_{SSE}}{\partial c_j} = & 2 & \hat{x}_j(1-\hat{x}_j) \quad (\hat{x}_j - x_j)
\end{array}
$$

$$
\begin{array}{lll}
\frac{\partial C_{CE}}{\partial W_{ij}} = & & (\hat{x}_j - x_j) \quad h_i + \quad x_j \quad h_i(1-h_i) \quad \sum_{k=1}^{n_x} \quad (\hat{x}_k - x_k)W_{ik} \\
\frac{\partial C_{CE}}{\partial b_i} = & & h_i(1-h_i) \quad \sum_{k=1}^{n_x} \quad (\hat{x}_k - x_k)W_{ik} \\
\frac{\partial C_{CE}}{\partial b_{ji}} = & & \hat{x}_j - x_j
\end{array}
$$

$$
\begin{array}{lll}
\frac{\partial C_{EXP}}{\partial W_{ij}} = & 2 \quad \exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right) & \left(\hat{x}_j(1-\hat{x}_j) \quad (\hat{x}_j - x_j) \quad h_i + \quad x_j \quad h_i(1-h_i) \quad \sum_{k=1}^{n_x} \quad \hat{x}_k(1-\hat{x}_k) \quad (\hat{x}_k - x_k)W_{ik} \right) \\
\frac{\partial C_{EXP}}{\partial b_i} = & 2 \quad \exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right) & h_i(1-h_i) \quad \sum_{k=1}^{n_x} \quad \hat{x}_k(1-\hat{x}_k) \quad (\hat{x}_k - x_k)W_{ik} \\
\frac{\partial C_{EXP}}{\partial c_j} = & 2 \quad \exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right) & \hat{x}_j(1-\hat{x}_j) \quad (\hat{x}_j - x_j)
\end{array}
$$

So we can write more compactly

$$\frac{\partial C}{\partial W_{ij}} = A\Big(B_j(\hat{x}_j - x_j)h_i + x_j h_i(1-h_i)\sum_{k=1}^{n_x} B_k(\hat{x}_k - x_k)W_{ik}\Big) \tag{9}$$

$$\frac{\partial C}{\partial b_i} = Ah_i(1-h_i)\sum_{k=1}^{n_x} B_k(\hat{x}_k - x_k)W_{ik} \tag{10}$$

$$\frac{\partial C}{\partial c_j} = AB_j(\hat{x}_j - x_j) \tag{11}$$

where A and B_k take the values in the following table:

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

3.2 Denoising auto-encoders

In a denoising auto-encoder (DEA), given an input vector $\mathbf{x}$, each input x_j, $j = 1, ..., n_x$ can be set to zero (corrupted) with a given probability. So, from $\mathbf{x}$ we get a corrupted input vector $\tilde{\mathbf{x}}$, with elements $\tilde{x}_j$, $j = 1, ..., n_x$. This is reflected in the forward propagation:

$$a_i = b_i + \sum_{w=1}^{n_x} W_{iw}\tilde{x}_w \tag{12}$$

From this, the same way that we arrived at formula (43) (see detailed calculations in Section (4)), we arrive at

$$\frac{\partial a_i}{\partial W_{ij}} = \tilde{x}_j \tag{13}$$

Just like with normal auto-encoders, we are interested in cost $C(\hat{\mathbf{x}}, \mathbf{x})$ associated with *uncorrupted* input vector $\mathbf{x}$ and output vector $\hat{\mathbf{x}}$. So, corrupted input $\tilde{x}_j$ will appear only in the expression for $\partial C(\hat{\mathbf{x}}, \mathbf{x})/\partial W_{ij}$, as a term that results directly from the formula above:

$$\frac{\partial C(\hat{\mathbf{x}}, \mathbf{x})}{\partial W_{ij}} = A\Big(B_j(\hat{x}_j - x_j)h_i + \underbrace{\tilde{x}_j}\, h_i(1-h_i)\sum_{k=1}^{n_x} B_k(\hat{x}_k - x_k)W_{ik}\Big) \tag{14}$$

4 Detailed calculations

4.1 Notation

The following sections contain calculations for the partial derivatives of the sum of squared errors (SSE), cross-entropy (CE), and exponential (EXP) cost functions in order to the weights and biases of an auto-encoder. The following notation is used for the auto-encoder:

n_x	number of inputs and outputs
n_h	number of hidden units
$x_j, j \in \{1, 2, ..., n_x\}$	value of jth input
$h_i, i \in \{1, 2, ..., n_h\}$	value of ith hidden unit
$\hat{x}_j, j \in \{1, 2, ..., n_x\}$	value of jth output
W_{ij}	weight connecting ith hidden unit to jth input weight connecting ith hidden unit to jth output
b_i	bias of ith hidden unit
c_j	bias of jth output
θ	any individual weight or bias

Each $\sum$ or $\frac{\partial}{\partial\theta}$ symbol applies to all multiplicative terms to its right.

4.2 SSE

The SSE error between an $\hat{\mathbf{x}}$ vector of outputs and an $\mathbf{x}$ vector of inputs is expressed by

$$C_{SSE}(\hat{\mathbf{x}}, \mathbf{x}) = \sum_{k=1}^{n_x} (\hat{x}_k - x_k)^2 \tag{15}$$

So,

$$\frac{\partial C_{SSE}}{\partial\theta} = \sum_{k=1}^{n_x} \frac{\partial}{\partial\theta}(\hat{x}_k - x_k)^2 \tag{16}$$

$$= \sum_{k=1}^{n_x} 2(\hat{x}_k - x_k)\frac{\partial}{\partial\theta}(\hat{x}_k - x_k) \tag{17}$$

We note that

$$\frac{\partial x_k}{\partial\theta} = 0, \forall_k \tag{18}$$

because inputs are independent from weights and biases. So,

$$\frac{\partial C_{SSE}}{\partial \theta} = 2 \sum_{k=1}^{n_x} (\hat{x}_k - x_k) \frac{\partial \hat{x}_k}{\partial \theta} \tag{19}$$

4.2.1 Weights

In order to obtain $\frac{\partial \hat{x}_k}{\partial \theta}$, we note that

$$\hat{x}_k = s(\hat{a}_k) \tag{20}$$

where $s(.)$ denotes the sigmoid function. So,

$$\frac{\partial \hat{x}_k}{\partial \theta} = \frac{\partial}{\partial \theta} s(\hat{a}_k) \tag{21}$$

$$= s(\hat{a}_k)\Big(1 - s(\hat{a}_k)\Big) \frac{\partial \hat{a}_k}{\partial \theta} \tag{22}$$

$$= \hat{x}_k (1 - \hat{x}_k) \frac{\partial \hat{a}_k}{\partial \theta} \tag{23}$$

In order to obtain $\frac{\partial \hat{a}_k}{\partial W_{ij}}$, we note that

$$\hat{a}_k = c_k + \sum_{z=1}^{n_h} W_{zk} h_z \tag{24}$$

So,

$$\frac{\partial \hat{a}_k}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} (c_k + \sum_{z=1}^{n_h} W_{zk} h_z) \tag{25}$$

$$= \sum_{z=1}^{n_h} \frac{\partial}{\partial W_{ij}} W_{zk} h_z \tag{26}$$

From the architecture of an auto-encoder, it is clear that

$$\frac{\partial}{\partial W_{ij}} W_{zk} h_z = 0, \forall_{z \neq i} \tag{27}$$

because, if $z \neq i$, then W_{zk} is cannot be W_{ij} and h_z cannot be influenced by W_{ij}. (This is the first difference in relation to the incorrect calculations.) So,

$$\frac{\partial \hat{a}_k}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} W_{ik} h_i \tag{28}$$

$$= W_{ik} \frac{\partial h_i}{\partial W_{ij}} + \frac{\partial W_{ik}}{\partial W_{ij}} h_i \tag{29}$$

Now we note that

$$\frac{\partial W_{ik}}{\partial W_{ij}} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases} \tag{30}$$

that is, the h_i term in the expression for $\frac{\partial \hat{a}_k}{\partial W_{ij}}$ is "switched on" only for $k = j$. (This is the second difference in relation to the incorrect calculations.) So,

$$\frac{\partial \hat{a}_k}{\partial W_{ij}} = \begin{cases} W_{ij} \frac{\partial h_i}{\partial W_{ij}} + h_i & \text{if } k = j \\ W_{ik} \frac{\partial h_i}{\partial W_{ij}} & \text{if } k \neq j \end{cases} \tag{31}$$

Alternatively, we could denote $\frac{\partial W_{ik}}{\partial W_{ij}}$ as $1_{k=j}$ and keep $\frac{\partial \hat{a}_k}{\partial W_{ij}}$ in compact form:

$$\frac{\partial \hat{a}_k}{\partial W_{ij}} = W_{ik} \frac{\partial h_i}{\partial W_{ij}} + 1_{k=j} h_i \tag{32}$$

In order to obtain $\frac{\partial h_i}{\partial W_{ij}}$, we note that

$$h_i = s(a_i) \tag{33}$$

So,

$$\frac{\partial h_i}{\partial \theta} = \frac{\partial}{\partial \theta} s(a_i) \tag{34}$$

$$= s(a_i)\Big(1 - s(a_i)\Big) \frac{\partial a_i}{\partial \theta} \tag{35}$$

$$= h_i(1 - h_i) \frac{\partial a_i}{\partial \theta} \tag{36}$$

In order to obtain $\frac{\partial a_i}{\partial W_{ij}}$, we note that

$$a_i = b_i + \sum_{w=1}^{n_x} W_{iw} x_w \tag{37}$$

So,

$$\frac{\partial a_i}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}}(b_i + \sum_{w=1}^{n_x} W_{iw} x_w) \tag{38}$$

$$= \sum_{w=1}^{n_x} \frac{\partial}{\partial W_{ij}} W_{iw} x_w \tag{39}$$

$$= \sum_{w=1}^{n_x} (W_{iw} \frac{\partial x_w}{\partial W_{ij}} + \frac{\partial W_{iw}}{\partial W_{ij}} x_w) \tag{40}$$

We now note that

$$\frac{\partial x_w}{\partial \theta} = 0, \forall_w \tag{41}$$

because inputs are not influenced by weights or biases. We also note that

$$\frac{\partial W_{iw}}{\partial W_{ij}} \begin{cases} 1 & \text{if } w = j \\ 0 & \text{if } w \neq j \end{cases} \tag{42}$$

So the expression for $\frac{\partial a_i}{\partial W_{ij}}$ becomes simply

$$\frac{\partial a_i}{\partial W_{ij}} = x_j \tag{43}$$

From this and (36),

$$\frac{\partial h_i}{\partial W_{ij}} = h_i(1 - h_i) x_j \tag{44}$$

From this and (31),

$$\frac{\partial \hat{a}_k}{\partial W_{ij}} = \begin{cases} W_{ij} h_i (1 - h_i) x_j + h_i & \text{if } k = j \\ W_{ik} h_i (1 - h_i) x_j & \text{if } k \neq j \end{cases} \tag{45}$$

From this and (23),

$$\frac{\partial \hat{x}_k}{\partial W_{ij}} = \begin{cases} \hat{x}_k (1 - \hat{x}_k) \Big(W_{ij} h_i (1 - h_i) x_j + h_i \Big) & \text{if } k = j \\ \hat{x}_k (1 - \hat{x}_k) W_{ik} h_i (1 - h_i) x_j & \text{if } k \neq j \end{cases} \tag{46}$$

Finally, from this and (19),

$$\frac{\partial C_{SSE}}{\partial W_{ij}} = 2(\hat{x}_j - x_j)\hat{x}_j(1-\hat{x}_j)\Big(W_{ij}h_i(1-h_i)x_j + h_i\Big) + \tag{47}$$

$$2\sum_{k\neq j}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik}h_i(1-h_i)x_j \tag{48}$$

$$= 2(\hat{x}_j - x_j)\hat{x}_j(1-\hat{x}_j)h_i + \tag{49}$$

$$2(\hat{x}_j - x_j)\hat{x}_j(1-\hat{x}_j)W_{ij}h_i(1-h_i)x_j + \tag{50}$$

$$2\sum_{k\neq j}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik}h_i(1-h_i)x_j \tag{51}$$

$$= 2(\hat{x}_j - x_j)\hat{x}_j(1-\hat{x}_j)h_i + \tag{52}$$

$$2h_i(1-h_i)x_j\sum_{k=1}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik} \tag{53}$$

Alternatively, we could start from the compact form of $\frac{\partial \hat{a}_k}{\partial W_{ij}}$ shown in equation (32):

$$\frac{\partial \hat{a}_k}{\partial W_{ij}} = W_{ik}h_i(1-h_i)x_j + 1_{k=j}h_i \tag{54}$$

$$\frac{\partial \hat{x}_k}{\partial W_{ij}} = \hat{x}_k(1-\hat{x}_k)\Big(W_{ik}h_i(1-h_i)x_j + 1_{k=j}h_i\Big) \tag{55}$$

$$\frac{\partial C_{SSE}}{\partial W_{ij}} = 2\sum_{k=1}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)\Big(W_{ik}h_i(1-h_i)x_j + 1_{k=j}h_i\Big) \tag{56}$$

$$= 2\sum_{k=1}^{n_x}\Big((\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik}h_i(1-h_i)x_j + \tag{57}$$

$$(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)1_{k=j}h_i\Big) \tag{58}$$

$$= 2(\hat{x}_j - x_j)\hat{x}_j(1-\hat{x}_j)h_i + \tag{59}$$

$$2h_i(1-h_i)x_j\sum_{k=1}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik} \tag{60}$$

4.2.2 Hidden biases

From (24),

$$\frac{\partial \hat{a}_k}{\partial b_i} = \frac{\partial}{\partial b_i}(c_k + \sum_{z=1}^{n_h} W_{zk} h_z) \tag{61}$$

$$= \sum_{z=1}^{n_h} \frac{\partial}{\partial b_i} W_{zk} h_z \tag{62}$$

$$= \sum_{z=1}^{n_h} (W_{zk} \frac{\partial h_z}{\partial b_i} + \frac{\partial W_{zk}}{\partial b_i} h_z) \tag{63}$$

We note that

$$\frac{\partial W_{zk}}{\partial b_i} = 0, \forall_i \tag{64}$$

And also that

$$\frac{\partial h_z}{\partial b_i} = 0, \forall_{z \neq i} \tag{65}$$

because bias b_i influences only hidden unit h_i. So,

$$\frac{\partial \hat{a}_k}{\partial b_i} = W_{ik} \frac{\partial h_i}{\partial b_i} \tag{66}$$

From (37) and (41),

$$\frac{\partial a_i}{\partial b_i} = \frac{\partial}{\partial b_i}(b_i + \sum_{w=1}^{n_x} W_{iw} x_w) \tag{67}$$

$$= 1 \tag{68}$$

From this and (36),

$$\frac{\partial h_i}{\partial b_i} = h_i(1 - h_i) \tag{69}$$

From this and (66),

$$\frac{\partial \hat{a}_k}{\partial b_i} = W_{ik} h_i(1 - h_i) \tag{70}$$

From this and (23),

$$\frac{\partial \hat{x}_k}{\partial b_i} = \hat{x}_k(1-\hat{x}_k)W_{ik}h_i(1-h_i) \tag{71}$$

Finally, from this and (19),

$$\frac{\partial C_{SSE}}{\partial b_i} = 2h_i(1-h_i)\sum_{k=1}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik} \tag{72}$$

4.2.3 Output biases

From (24),

$$\frac{\partial \hat{a}_k}{\partial c_j} = \frac{\partial}{\partial c_j}(c_k + \sum_{z=1}^{n_h} W_{zk}h_z) \tag{73}$$

$$= \frac{\partial c_k}{\partial c_j} + \sum_{z=1}^{n_h} \frac{\partial}{\partial c_j} W_{zk}h_z \tag{74}$$

We note that

$$\frac{\partial c_k}{\partial c_j} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases} \tag{75}$$

And also

$$\frac{\partial}{\partial c_j} W_{zk}h_z = 0, \forall_j \tag{76}$$

because c_j doesn't influence weights or hidden units. So,

$$\frac{\partial \hat{a}_k}{\partial c_j} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases} \tag{77}$$

From this and (23),

$$\frac{\partial \hat{x}_k}{\partial c_j} = \begin{cases} \hat{x}_k(1-\hat{x}_k) & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases} \tag{78}$$

Finally, from this and (19),

$$\frac{\partial C_{SSE}}{\partial c_j} = 2\sum_{k=j}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k) \tag{79}$$

$$= 2(\hat{x}_j - x_j)\hat{x}_j(1-\hat{x}_j) \tag{80}$$

4.3 CE

The CE error between an $\hat{\mathbf{x}}$ vector of outputs and an $\mathbf{x}$ vector of inputs is expressed by

$$C_{CE}(\hat{\mathbf{x}}, \mathbf{x}) = -\sum_{k=1}^{n_x}\Big(x_k \ln(\hat{x}_k) + (1-x_k)\ln(1-\hat{x}_k)\Big) \tag{81}$$

From this and (41),

$$\frac{\partial C_{CE}}{\partial \theta} = -\sum_{k=1}^{n_x} \Big(\frac{\partial}{\partial \theta} x_k \ln(\hat{x}_k) + \frac{\partial}{\partial \theta}(1-x_k)\ln(1-\hat{x}_k) \Big) \tag{82}$$

$$= -\sum_{k=1}^{n_x} \Bigg(x_k \frac{\partial}{\partial \theta} \ln(\hat{x}_k) + \frac{\partial x_k}{\partial \theta} \ln(\hat{x}_k) + \tag{83}$$

$$(1-x_k)\frac{\partial}{\partial \theta} \ln(1-\hat{x}_k) + \Big(\frac{\partial}{\partial \theta}(1-x_k) \Big) \ln(1-\hat{x}_k) \Bigg) \tag{84}$$

$$= -\sum_{k=1}^{n_x} \Big(x_k \frac{1}{\hat{x}_k} \frac{\partial \hat{x}_k}{\partial \theta} + (1-x_k) \frac{1}{1-\hat{x}_k} \frac{\partial}{\partial \theta}(1-\hat{x}_k) \Big) \tag{85}$$

$$= -\sum_{k=1}^{n_x} \Big(\frac{x_k}{\hat{x}_k} \frac{\partial \hat{x}_k}{\partial \theta} - \frac{1-x_k}{1-\hat{x}_k} \frac{\partial \hat{x}_k}{\partial \theta} \Big) \tag{86}$$

$$= -\sum_{k=1}^{n_x} \frac{\partial \hat{x}_k}{\partial \theta} \Big(\frac{x_k}{\hat{x}_k} - \frac{1-x_k}{1-\hat{x}_k} \Big) \tag{87}$$

$$= -\sum_{k=1}^{n_x} \frac{\partial \hat{x}_k}{\partial \theta} \frac{x_k(1-\hat{x}_k) - (1-x_k)\hat{x}_k}{\hat{x}_k(1-\hat{x}_k)} \tag{88}$$

$$= -\sum_{k=1}^{n_x} \frac{\partial \hat{x}_k}{\partial \theta} \frac{x_k - x_k\hat{x}_k - (\hat{x}_k - x_k\hat{x}_k)}{\hat{x}_k(1-\hat{x}_k)} \tag{89}$$

$$= -\sum_{k=1}^{n_x} \frac{\partial \hat{x}_k}{\partial \theta} \frac{x_k - x_k\hat{x}_k - \hat{x}_k + x_k\hat{x}_k}{\hat{x}_k(1-\hat{x}_k)} \tag{90}$$

$$= -\sum_{k=1}^{n_x} \frac{\partial \hat{x}_k}{\partial \theta} \frac{x_k - \hat{x}_k}{\hat{x}_k(1-\hat{x}_k)} \tag{91}$$

$$= \sum_{k=1}^{n_x} \frac{\hat{x}_k - x_k}{\hat{x}_k(1-\hat{x}_k)} \frac{\partial \hat{x}_k}{\partial \theta} \tag{92}$$

4.3.1 Weights

From (92) and (55),

$$\frac{\partial C_{CE}}{\partial W_{ij}} = \sum_{k=1}^{n_x} \frac{\hat{x}_k - x_k}{\hat{x}_k(1-\hat{x}_k)} \hat{x}_k(1-\hat{x}_k) \Big(W_{ik} h_i (1-h_i) x_j + 1_{k=j} h_i \Big) \tag{93}$$

$$= \sum_{k=1}^{n_x} \Big((\hat{x}_k - x_k) W_{ik} h_i (1-h_i) x_j + (\hat{x}_k - x_k) 1_{k=j} h_i \Big) \tag{94}$$

$$= (\hat{x}_j - x_j) h_i + h_i(1-h_i) x_j \sum_{k=1}^{n_x} (\hat{x}_k - x_k) W_{ik} \tag{95}$$

4.3.2 Hidden biases

From (92) and (71),

$$\frac{\partial C_{CE}}{\partial b_i} = \sum_{k=1}^{n_x} \frac{\hat{x}_k - x_k}{\hat{x}_k(1-\hat{x}_k)} \hat{x}_k(1-\hat{x}_k) W_{ik} h_i (1-h_i) \tag{96}$$

$$= h_i(1-h_i) \sum_{k=1}^{n_x} (\hat{x}_k - x_k) W_{ik} \tag{97}$$

4.3.3 Output biases

From (92) and (78),

$$\frac{\partial C_{CE}}{\partial b_{ji}} = \frac{\hat{x}_j - x_j}{\hat{x}_j(1-\hat{x}_j)} \hat{x}_j(1-\hat{x}_j) + \sum_{k \neq j}^{n_x} \frac{\hat{x}_k - x_k}{\hat{x}_k(1-\hat{x}_k)} 0 \tag{98}$$

$$= \hat{x}_j - x_j \tag{99}$$

4.4 EXP

The EXP error between an $\hat{\mathbf{x}}$ vector of outputs and an $\mathbf{x}$ vector of inputs is expressed by

$$C_{EXP}(\hat{\mathbf{x}}, \mathbf{x}) = \tau \exp\left(\frac{1}{\tau} \sum_{k=1}^{n_x} (\hat{x}_k - x_k)^2\right) \tag{100}$$

From this and (41),

$$\frac{\partial C_{EXP}}{\partial \theta} = \tau \exp\left(\frac{1}{\tau} \sum_{k=1}^{n_x} (\hat{x}_k - x_k)^2\right) \frac{1}{\tau} \sum_{k=1}^{n_x} \frac{\partial}{\partial \theta} (\hat{x}_k - x_k)^2 \tag{101}$$

$$= \exp\left(\frac{1}{\tau} \sum_{k=1}^{n_x} (\hat{x}_k - x_k)^2\right) \sum_{k=1}^{n_x} 2(\hat{x}_k - x_k) \frac{\partial}{\partial \theta} (\hat{x}_k - x_k) \tag{102}$$

$$= 2 \exp\left(\frac{1}{\tau} \sum_{k=1}^{n_x} (\hat{x}_k - x_k)^2\right) \sum_{k=1}^{n_x} (\hat{x}_k - x_k) \frac{\partial \hat{x}_k}{\partial \theta} \tag{103}$$

It is interesting to note that the gradient of the EXP cost is the EXP cost itself multiplied by the gradient of the SSE cost (see formula (19)).

4.4.1 Weights

From (103) and (55),

$$\frac{\partial C_{EXP}}{\partial W_{ij}} = 2\exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right)\times \tag{104}$$

$$\sum_{k=1}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)\Big(W_{ik}h_i(1-h_i)x_j + 1_{k=j}h_i\Big) \tag{105}$$

$$= 2\exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right)\times \tag{106}$$

$$\sum_{k=1}^{n_x}\Big((\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik}h_i(1-h_i)x_j + \tag{107}$$

$$(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)1_{k=j}h_i\Big) \tag{108}$$

$$= 2\exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right)\times \tag{109}$$

$$\Big((\hat{x}_j - x_j)\hat{x}_j(1-\hat{x}_j)h_i + \tag{110}$$

$$h_i(1-h_i)x_j\sum_{k=1}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik}\Big) \tag{111}$$

4.4.2 Hidden biases

From (103) and (71),

$$\frac{\partial C_{EXP}}{\partial b_i} = 2\exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right)\sum_{k=1}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik}h_i(1-h_i) \tag{112}$$

$$= 2\exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right)h_i(1-h_i)\sum_{k=1}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k)W_{ik} \tag{113}$$

4.4.3 Output biases

From (103) and (78),

Table 1: Characteristics of the data sets used in the experiments.

data set	# features	# targets	# instances			type
			train	valid.	test	
adult	123	2	5000	1414	26147	binary
dna	180	3	1400	600	1186	binary
mnist-subset	784	10	5000	1000	1000	real-valued
mushrooms	112	2	2000	500	5624	binary
rectangles	784	2	1000	200	50000	binary

$$\frac{\partial C_{EXP}}{\partial c_j} = 2 \exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right)\sum_{k=j}^{n_x}(\hat{x}_k - x_k)\hat{x}_k(1-\hat{x}_k) \tag{114}$$

$$= 2 \exp\left(\frac{1}{\tau}\sum_{k=1}^{n_x}(\hat{x}_k - x_k)^2\right)(\hat{x}_j - x_j)\hat{x}_j(1-\hat{x}_j) \tag{115}$$

5 Experiments and results

Note: this work is still ongoing, therefore the selected data sets are not definitive and the tables for selected hyper-parameters and classification results are still empty, merely illustrating the way in which the information should be presented when final results are achieved. A number of preliminary experiments have been carried out by Chetak Kandaswamy and are described in a separate technical report [5].

5.1 Data sets

Our experimental evaluation was conducted using code based on the MLPython library[1]. We used five data sets for which MLPython has built-in support. Table 1 shows some characteristics of each data set, once prepared for classification learning . All data sets were obtained without modification from the MLPython data sets web page[2], except for *mnist-subset* (which is a subset of the *mnist* set, available from the same source). Of the data we used, only *mnist-subset* contained real-valued features, namely pixel intensities mapped between 0 and 1. The features in the *adult*, *dna*, and *mushrooms* sets resulted mostly from integer or categorical features that were expanded as sets of mutually exclusive binary features.

5.2 Setup and hyper-parameter selection

We trained and tested a deep architecture of the type exemplified in black in Figure (1)c with each of the five data sets, employing either AEs or DAEs as

[1] See http://www.dmi.usherb.ca/~larocheh/mlpython/.
[2] See http://www.dmi.usherb.ca/~larocheh/mlpython/datasets.html.

Table 2: Hyper-parameter values selected for each combination of data set and greedy module.

data set	greedy module									
	AE					DAE				
	l	n_h	τ	η_{PT}	η_{FT}	l	n_h	τ	η_{PT}	η_{FT}
adult	0	0000	0.00	0.00	0.00	0	0000	0.00	0.00	0.00
dna										
mnist_subset										
mushrooms										
rectangles										

pre-training greedy modules and using, for each greedy module, either CE, MSE, or EXP costs. Each network was first pre-trained without supervision using the training set (input features only), then fine-tuned with supervision using the training and validation sets, and finally tested on the test set. For comparison, an additional experiment with no pre-training was conducted.

We aimed to tune five hyper-parameters for each combination of data set and greedy module, namely: the number of hidden layers l; the hidden layer size n_h; the EXP cost parameter τ; the pre-training learning rate η_{PT} (shown as η in Equations (6) to (8)); and the fine-tuning learning rate η_{FT}. The same number of neurons was used for all hidden layers.

A grid search of the hyper-parameter space seemed prohibitive, as our experiments relied on CPUs and could take very long to run (especially when the image sets *mnist-subset* and *rectangles* were involved). Therefore we used an approximate selection procedure, starting from default values, then tuning each hyper-parameter individually to minimise the validation error. In most cases, averaging the error over a few repetitions helped to identify the best value for the parameter being explored. All hyper-parameters were tuned using CE pre-training costs, except for τ, which affects specifically the EXP cost function. Table 2 shows all the selected values.

A few other hyper-parameters were kept at fixed values, namely: five pre-training epochs were used, with no stop criteria; fine-tuning stopped if the validation error did not decrease for five consecutive epochs; the fine-tuning decrease constant was set to zero; and the noise probability ν of DAEs was set to 0.1.

5.3 Classification tests

For each combination of data set, pre-training greedy module, and pre-training cost function, the test stage was repeated 30 times. Table 3 shows the mean and standard deviation of the test errors obtained in each case. Also included for comparison are the results achieved without pre-training (using the same l, n_h, and η_{FT} values as with AE pre-training).

Table 3: Mean and standard deviation of test errors for each data set and pre-training cost function, using as greedy module (a) auto-encoders and (b) denoising auto-encoders.

(a) AE

data set	no pre-training	pre-training cost function		
		CE	MSE	EXP
adult	00.00±0.00%	00.00±0.00%	00.00±0.00%	00.00±0.00%
dna				
mnist_subset				
mushrooms				
rectangles				

(b) DAE

data set	no pre-training	pre-training cost function		
		CE	MSE	EXP
adult	00.00±0.00%	00.00±0.00%	00.00±0.00%	00.00±0.00%
dna				
mnist_subset				
mushrooms				
rectangles				

References

[1] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. 3

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

[3] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4-5):291–294, 1988. 3

[4] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. 3

[5] C. Kandaswamy. Tuning parameters of deep neural network algorithms for identifying best cost function. Technical Report 2/2013, Instituto de Engenharia Biomédica / NNIG, April 2013. 20

[6] 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*, pages 473–480, 2007. 3, 4

[7] 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. 3

[8] J. Marques de Sá, L. Silva, J. Santos, and L. Alexandre. *Minimum Error Entropy Classification*, volume 420 of *Studies in Computational Intelligence*. Springer, 2013. 3

[9] 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. 3

[10] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, pages 1096–1103, 2008. 4