Improving Deep Neural Network Performance by Reusing Features Trained with Transductive Transference

Chetak Kandaswamy*, Luís Silva**, Luís Alexandre***, Jorge M. Santos†, Joaquim Marques de Sá‡

Instituto de Engenharia Biomédica (INEB), Porto, Portugal

Abstract. Transfer Learning is a paradigm in machine learning to solve a target problem by reusing the learning with minor modifications from a different but related source problem. In this paper we propose a novel feature transference approach, especially when the source and the target problems are drawn from different distributions. We use deep neural networks to transfer either low or middle or higher-layer features for a machine trained in either unsupervised or supervised way. Applying this feature transference approach on Convolutional Neural Network and Stacked Denoising Autoencoder on four different datasets, we achieve lower classification error rate with significant reduction in computation time with lower-layer features trained in supervised way and higher-layer features trained in unsupervised way for classifying images of uppercase and lowercase letters dataset.

Keywords: Feature Transference, Deep Neural Network.

1 Introduction

Machine learning can be broadly classified into Transductive or Inductive approach. In transductive learning, the objective is to learn from observed, specific (training) instances to specific (test) instances drawn from same distributions. In contrast, induction is to learn from observed training instances, a set of assumptions about the true distribution of the test cases (general rules). Transductive is preferable to inductive [1] since, induction requires solving a more general problem (assumptions about the true distribution) before solving a more specific problem. This distinction is most interesting in cases where the predictions of the transductive model are not achievable by any inductive model.

* Chetak Kandaswamy is with INEB and also with Dep. de Engenharia Electronica e de Computadores at FEUP, Porto, Portugal, e-mail: chetak.kand@gmail.com.

** Luís M. Silva is with INEB and also with Dep. de Matemática at Universidade de Aveiro, Portugal, e-mail: lmas@ua.pt

*** Luís A. Alexandre is with Universidade da Beira Interior and Instituto de Telecomunicações, Covilhã, Portugal, e-mail: lfbaa@di.ubi.pt

† Jorge M. Santos is with INEB and also with Dep. de Matemática at Instituto Superior de Engenharia do Instituto Politécnico do Porto, Portugal.

‡ Joaquim Marques de Sá is with INEB and also with Dep. de Engenharia Electronica e de Computadores at FEUP, Porto, Portugal.

Transfer Learning attempts to train a machine to solve a source problem and reuse it with minor modifications to solve a different but related target problem without having to train the machine from scratch. Thus transfer in transductive approach, a machine is trained on a specific problem to solve another specific problem, where the target problem distributions are not necessarily related to the source problem.

Deep Transfer Learning (DTL) is an alternative to transfer learning with shallow architectures [2]. The advantage of DTL is that it offers a far greater flexibility in extracting high-level features and transferring it from a source to a target problem, and unlike the classical approach, it is not affected by experts bias [2]. Despite the vast body of literature on the subject, there are still many contentious issues regarding avoiding negative transfer from source to target problems, especially when the source and target distributions are from different distributions. In this paper, we only consider transfer learning problems where the source and target distributions are different. Specially, in the case of negative transference from the source problem.

2 Deep Neural Network

We use state-of-the-art deep learning methods (see [3], [4]) that learn high-level features from large datasets and measure the classification performance of images.

Given an input space X, with a certain probability distribution $P(X)$, we draw a design data set $X_{ds} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_{ds}}\}$ which may be accompanied by a set of labels $Y = \{1, 2, \ldots, c\}$ with c distinct class labels. We define a classification problem as any function $g(\mathbf{x}) : X \to Y$ that maps n_{ds} instances of $\mathbf{x} \in X$ to labels. Thus, the classifier attempts to learn features (or filters), represented as a vector w^j of optimal weights and biases. For a classifier with k number of layers, the features w^j are represented as a set of vectors of each layer, i.e., $\mathbf{w} = (w^1, ..., w^k)$. We use error rate ε and computation time t to measure the classifier performance to predict on a test set $X_{ts} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_{ts}}\}$ with n_{ts} unlabeled instances drawn from the same distribution $P(X)$.

2.1 Stacked Denoising Autoencoder (SDA)

An autoencoder is a simple neural network with one hidden layer designed to reconstruct its own input, having, for that reason, an equal number of input and output neurons (tied weights). The reconstruction accuracy is obtained by minimizing the average reconstruction error between the original and the reconstructed instances. A denoising autoencoder is a variant of the autoencoder where now a corrupted version of the input is used to reconstruct the original instances. A SDA is made up of stacking multiple denoising autoencoder one on top of another. The training of SDA [3] comprises of two stages: an unsupervised pre-training stage followed by a supervised fine-tuning stage (see [5, Section 6.2]).

In the unsupervised pre-training stage $pretrain(\mathbf{w})$, the weights w^j of each hidden layer are trained in unsupervised way until the reconstruction cost of that layer reaches global minimum. Then we repeat the pre-training until the k^{th} hidden layer is completely pre-trained to obtain unsupervised features $U(\mathbf{w})$.

In the supervised fine-tuning stage $finetune(\mathbf{w}, \mathbf{c})$, a logistic regression layer with c neurons is added to the top of the pre-trained machine. Then, the entire classifier is trained (fine-tuned) using both X_{ds} and Y_{ds} in order to minimize a cross-entropy loss function [6] measuring the error between the classifier's predictions and the correct labels to obtain supervised features $S(\mathbf{w})$.

2.2 Convolutional Neural Network (CNN)

CNN [4] is a deep neural network whose convolutional layers alternate with subsampling layers. CNN is better explained in two stages. The alternating convolutional and subsampling stage and the classification stage. The convolution layer convolute the input with set of filters like Gabor filters or trained filters producing feature maps. These feature maps are further reduced by subsampling. Then the supervised feature $S(w^k)$ or kernels of the top convolution filters and subsampling are fed to classification stage. Then we fine-tune the filters, $finetune(\mathbf{w}, \mathbf{c})$ with labeled source data to obtain supervised features $S(\mathbf{w})$.

In this paper the term "Baseline approach" to refer to either a SDA or a CNN machine trained on a target problem with no transference from the source problem (trained from scratch).

3 Transfer Learning Method

Traditionally, the goal of the transfer learning is to transfer the learning (knowledge) from a input space X_S of source-problem S to one or more target-problems T, or distributions to efficiently develop an effective hypothesis for a new task, problem, or distribution [7]. In this framework of transfer learning, we address transfer learning problems, where the source and target problems are from different distributions and also the source Y_S and target Y_T labels may be equal or different. Thus, we address two important cases of transfer learning problems:

1. The distributions are different $P_S(X) \neq P_T(X)$ and the labels are equal $Y_S = Y_T$.
2. The distributions are different $P_S(X) \neq P_T(X)$ and the labels are not equal $Y_S \neq Y_T$.

Under such hypothesis, our goal is to obtain an accurate classification for target-problem instances by exploiting labeled training instances drawn from the source-problem. We use two types of feature transference: 1) unsupervised feature transference (UFT) only for SDA model, and 2) supervised layer based feature transference (SLFT) for both SDA and CNN model

In the *Unsupervised Feature Transference* (UFT) approach we transfer the unsupervised features of the SDA model from the source to the target problem,

Table 1: Lists SLFT, UFT and Baseline Approach

Approaches	Transference	Target problem
FT	$S(\mathbf{w}_S) \Rightarrow \mathbf{w}_T$	$finetune\,(\mathbf{w}_T, \mathbf{c}_T)$
L1+L2+L3	$S(w_S^1, w_S^2, w_S^3) \Rightarrow w_T^1, w_T^2, w_T^3$	$finetune\,(\mathbf{c}_T)$
L1+L3	$S(w_S^1, w_S^3) \Rightarrow w_T^1, w_T^3$	$finetune\,\left(w_T^2, \mathbf{c}_T\right)$
L2+L3	$S(w_S^2, w_S^3) \Rightarrow w_T^2, w_T^3$	$finetune\,\left(w_T^1, \mathbf{c}_T\right)$
L1+L2	$S(w_S^1, w_S^2) \Rightarrow w_T^1, w_T^2$	$finetune\,\left(w_T^3, \mathbf{c}_T\right)$
L3	$S(w_S^3) \Rightarrow w_T^3$	$finetune\,\left(w_T^1, {w_T}^3, \mathbf{c}_T\right)$
L2	$S(w_S^2) \Rightarrow w_T^2$	$finetune\,\left(w_T^1, w_T^3, \mathbf{c}_T\right)$
L1	$S(w_S^1) \Rightarrow w_T^1$	$finetune\,\left(w_T^2, w_T^3, \mathbf{c}_T\right)$
UFT	$U(\mathbf{w}_S) \Rightarrow \mathbf{w}_T$	$finetune\,(\mathbf{w}_T, \mathbf{c}_T)$
Baseline	-	$finetune\,(pretrain(\mathbf{w}_T), \mathbf{c}_T)$

that is, $U(\mathbf{w}_S) \Rightarrow \mathbf{w}_T$ as shown in Table 1. Once the features are transferred to the target problem, we add a logistic regression layer for the target Y_T with labels c_T on top of the transferred machine. We fine-tune this entire classifier $finetune(\mathbf{w}_T, c_T)$ as a multi-layer perceptron using back-propagation.

In the Supervised Layer Based Feature Transference (SLFT), lets consider the case of **L1** approach as listed in Table 1, we transfer $S(w_S^1) \Rightarrow w_T^1$ features from source to target problem. Then the rest of hidden and logistic regression layer of the target network is randomly initialized. Finally, we fine-tune the whole target network except w_T^1 feature set of the target network, $finetune\,\left(w_T^2, w_T^3, \mathbf{c}_T\right)$. Similarly, we can transfer the first and second layer features, that is, $S(w_S^1, w_S^2) \Rightarrow w_T^1, w_T^2$, listed as the **L1+L2** approach in Table 1. In the case of the **FT** approach we reuse the fully trained supervised features $S(\mathbf{w}_S) \Rightarrow \mathbf{w}_T$ of the source problem and then fine-tune again the entire classifier $S\,(\mathbf{w}_T, \mathbf{c}_T)$ for the target problem. In the case of $Y_S \neq Y_T$ transfer setting the FT approach cannot reuse the logistic regression layer. Thus the logistic regression layer is randomly initialized for the target problem.

4 Experiments and Results

In this work we used MNIST[1], MADbase[2] and Chars74k[3] [8] image datasets. The original Chars74k were split into two smaller datasets: *lowercase* with the a-to-z lowercase letters and *uppercase* with the A-to-Z uppercase letters. Then resized it to 28×28 pixels from original 128×128 pixels image. In our experiments we use Latin handwritten digits, Arabic handwritten digits, Lowercase synthetic letters and Uppercase synthetic letters datasets as shown in Table 2.

We performed all our experiments on a computer with i7-377 (3.50GHz) 16GB RAM using Theano [9] a GPU compatible machine learning library on a GTX 770 GPU. The GPU parallel processing allows training both CNN's

[1] http://yann.lecun.com/exdb/mnist/
[2] http://datacenter.aucegypt.edu/shazeem/
[3] We acknowledge Microsoft Research India for Chars74k dataset.

Table 2: Dataset characteristics, Average classification test error (%) ($\bar{\varepsilon}$), Average training times (seconds) ($\bar{t}$) with GTX 770 obtained for SDA and CNN baseline approach.

Data set		**Labels**			**Instances**			**SDA**		**CNN**	
Distribution		Y		c	**Train**	**Valid**	**Test**	$\bar{\varepsilon}$	$\bar{t}$	$\bar{\varepsilon}$	$\bar{t}$
Latin	P_L	0-to-9	Y_{09}	10	50,000	10,000	10,000	1.61±0.19	10698	0.93±0.06	1418
Arabic	P_A	•-to-*9*	$Y_{\bullet 9}$	10	50,000	10,000	10,000	1.37±0.07	8051	0.96±0.06	1209
Lowercase	P_{LC}	a-to-z	Y_{az}	26	13,208	6,604	6,604	4.95±0.16	2997	3.65±0.12	445
Uppercase	P_{UC}	A-to-Z	Y_{AZ}	26	13,208	6,604	6,604	5.01±0.27	2567	3.42±0.10	444

and SDA's deep neural networks with millions of neural connection faster than traditional CPUs. Each of these experiments are repeated 10 times to increase the confidence level of the results. The hyper parameters for CNN used kernel filter size of [20, 50] and max training epochs of 200. The learning rate of 0.1 is set with batch training of 500. The hyper parameters for SDA used pre-training and fine-tuning learning rates of 0.001 and 0.1, respectively. The stopping criteria for pre-training was fixed to 40 epochs; stopping criteria for fine-tuning was set to a maximum of 1000 epochs. The number of neurons in the three hidden layers and one output layer has a pyramidal structure with [576, 400, 256, c] neurons.

In the following experiments we compare baseline (BL) and transfer learning (TL) approach classification error ε using $X_{ts.target}$ dataset, for different amounts of instances per class, n_{ds}/c. We followed the procedure shown in Algorithm 1.

In step 1 of the experimental procedure, $X_{ds.source}$ for each n_{ds} samples were randomly picked from a set of different amounts of design samples per class $[100, 250, 500, 1000, 1320, 2500, 5000]$. Then in each of this iteration; step (a) we run the baseline approach from Algorithm 1, step (b) we build $X_{ds.target}$ by randomly picking n_{ts} samples, where $n_{ts} = n_{ds}$. Finally, in step (c), we apply the various layer based feature transference approaches as listed in Table 1.

Algorithm 1 Experimental procedure.

Given design sets $X_{ds.source}$, $X_{ds.target}$ and test set $X_{ts.source}$, $X_{ts.target}$,

For each *dataset* such that $(X_{ds}, X_{ts})\in$(Latin, Arabic, Lowercase, Uppercase)

1. For each n_{ds} such that $\frac{n_{ds}}{c}\in[100, 250, 500, 1000, 1320, 2500, 5000]$,
   (a) Run the baseline approach;
   (b) Obtain $X_{ds.target}$ by randomly picking n_{ds} samples from $X_{ds.target.full}$;
   (c) For each TL approach such that $L\in[L1, L1 + L2, ...]$ from Table 1,
      i. Fix L^{th} layer of the network trained on $X_{ds.source}$;
      ii. Retrain the network using $X_{ds.target}$ except the L^{th} layers;
      iii. Test the network using $X_{ts.target}$, obtaining classification error ε.

Table 3: Average classification test error (%) ($\overline{\varepsilon}$) obtained for different n_{ds}/c for SLFT approach on CNN model.

Approaches	$X_{ds.upper}$ reuse $X_{ds.latin}$		$X_{ds.lower}$ reuse $X_{ds.latin}$	
Source:	Latin		Latin	
Target:	Uppercase		Lowercase	
$n_{ds.source}/c$:	1320	5000	1320	5000
L1+L2+L3	5.96±0.13	5.32±0.18	6.13±0.13	5.63±0.15
L1+L3	4.49±0.14	4.24±0.10	4.75±0.13	4.57±0.09
L1+L2	3.61±0.12	**3.39±0.12**	3.83±0.06	**3.63±0.13**
L3	4.30±0.13	4.20±0.16	4.62±0.18	4.61±0.14
L2	3.54±0.14	3.43±0.06	3.72±0.11	3.58±0.15
L1	3.43±0.11	**3.35±0.09**	**3.64±0.06**	**3.56±0.11**
BL	**3.42±0.10**	3.42±0.10	3.65±0.12	3.65±0.12

4.1 Results for Transductive transfer: From digits to letters

Classifying images of lowercase from a-to-z by reusing supervised features of digits from 0-to-9. We train a CNN to solve Latin digits (specific source problem) and reuse it to solve a lowercase letters (different but related target problem) without having to train it from scratch.

Using Latin as source problem and either Lowercase or Uppercase letters as target problem. Table 3 presents the average classification error rate for SLFT approach on CNN model by applying Algorithm 1. We observe both **L1** and **L1+L2** approach performs better than the baseline approach for $n_{ds}/c = 5000$. In case of $n_{ds}/c = 1320$ we observe only **L1** approach performs better than the baseline approach. Reusing all three layers: L1+L2+L3 has degraded performs as the complete supervised features are well tuned for the source problem and training only the logistic regression layer has no improvement. Table 4 provides the summary of classification results for both CNN and SDA models.

Table 4: Summary: Average classification test error (%) ($\overline{\varepsilon}$), Average training times (seconds) ($\overline{t}$) by reusing Latin at $n_{ds}/c = 1320$

Approaches		Lowercase		Uppercase	
		$\overline{\varepsilon}$	$\overline{t}$	$\overline{\varepsilon}$	$\overline{t}$
SDA	BL	4.95±0.16	2997	5.01±0.27	2567
SDA	SLFT: L1	4.72±0.17	2261	4.72±0.18	2515
SDA	UFT	4.67±0.38	1148	4.65±0.19	1498
SDA	SLFT: FT	**4.57±0.08**	1020	**4.58±0.19**	1180
CNN	SLFT: L1+L2	3.83±0.06	196	3.61±0.12	197
CNN	BL	3.65±0.12	445	**3.42±0.10**	444
CNN	SLFT: L1	**3.64±0.06**	292	3.43±0.11	293

4.2 Transference From Arabic digits to Latin digits and Vice-versa

Utilizing previous conclusion that reusing L1 and L1+L2 perform better other approaches. We performed similar experiment with Latin and Arabic digits and also reverse the role of source and target datasets. To study the effect of negative transference, for example, *digit 0* in latin is represented as **0** where as *digit 5* in arabic is written as **0**. These labels may lead to negative transference during supervised learning. We observe SLFT L1 and L1+L2 approaches performs better than baseline approach as shown in Fig 1 and Fig 2.

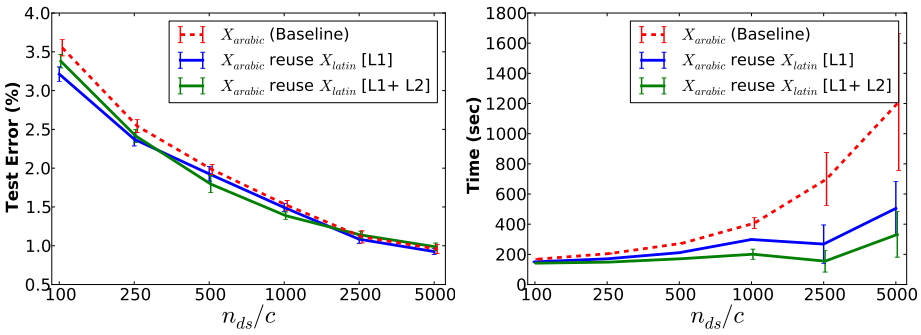

Fig. 1: Classification results on MAHDBase (Arabic digits) for SLFT: L1 and L1+L2 approach, for different numbers n_{ds}/c. **Left:** Average classification test error rate. **Right:** Average time taken for classification.

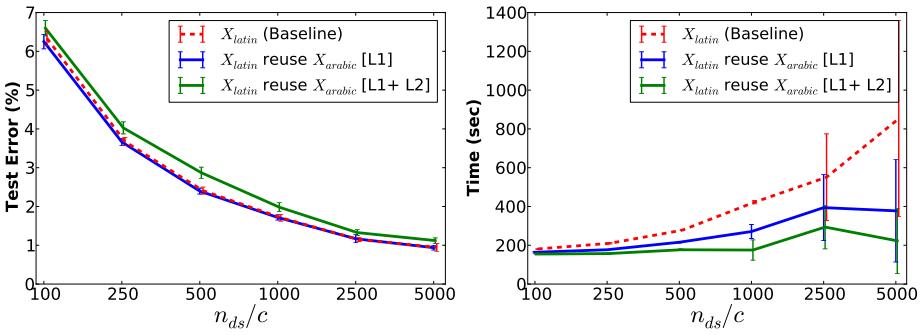

Fig. 2: Classification results on MNIST (Latin digits) for SLFT: L1 and L1+L2 approach, for different numbers n_{ds}/c. **Left:** Average classification test error rate. **Right:** Average time taken for classification.

5 Conclusions

We proposed a layer based feature transference approach that supports standard neural networks like CNN and SDA for solving transductive transfer learning problems. By transferring either low or high layer features on machines trained either unsupervised or supervised way. Using this approach we achieved performance improvement with significant reduction in computation time and also decreased classification error rate. We achieved significant performance by transferring learning from source to target problem, by using lower-layer features trained in supervised fashion in case of CNN's and unsupervised features trained in case of SDA's.

6 Acknowledgements

The authors would like to thank Dr Jaime S. Cardoso, Universidade do Porto, Dr Telmo Amaral and Dr Ricardo Sousa for their critical reviews. 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 and PEst-C/SAU/LA0002/2013.

References

1. Vapnik, V.N.: An overview of statistical learning theory. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council **10**(5) (January 1999) 988–99
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. (2013)
3. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11** (December 2010) 3371–3408 Cited by 0083.
4. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (1998) 2278–2324 Cited by 2008.
5. Bengio, Y.: Learning deep architectures for AI. Foundations and Trends in Machine Learning **2**(1) (2009) 1–127 Now Publishers, 2009.
6. Amaral, T., Silva, L.M., Alexandre, L.A., Kandaswamy, C., Santos, J.M., de Sá, J.M.: Using different cost functions to train stacked auto-encoders. In: Proceedings of the 12th Mexican International Conference on Artificial Intelligence, IEEE (2013)
7. Bruzzone, L., Marconcini, M.: Domain adaptation problems: A dasvm classification technique and a circular validation strategy. Pattern Analysis and Machine Intelligence, IEEE Transactions on **32**(5) (2010) 770–787
8. de Campos, T., Babu, B.R., Varma, M.: Character recognition in natural images. (2009)
9. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy). Volume 4. (2010)