Convex Multi-Task Learning with Neural Networks

Carlos Ruiz[1(✉)], Carlos M. Alaíz[1], and José R. Dorronsoro[1,2]

[1] Dept. Computer Engineering, Universidad Autónoma de Madrid, Madrid, Spain
carlos.ruizp@uam.es

[2] Inst. Ing. Conocimiento, Universidad Autónoma de Madrid, Madrid, Spain

Abstract. Multi-Task Learning aims at improving the learning process by solving different tasks simultaneously. The approaches to Multi-Task Learning can be categorized as feature-learning, regularization-based and combination strategies. Feature-learning approximations are more natural for deep models while regularization-based ones are usually designed for shallow ones, but we can see examples of both for shallow and deep models. However, the combination approach has been tested on shallow models exclusively. Here we propose a Multi-Task combination approach for Neural Networks, describe the training procedure, test it in four different multi-task image datasets and show improvements in the performance over other strategies.

Keywords: Multi-task learning · Deep learning · Convex combination

1 Introduction

In Machine Learning (ML) it is often assumed that the data is independently identically distributed, and the empirical risk minimization principle [19], typically used in supervised learning, bases its generalization abilities in this claim. However, we often find problems with different but possibly related data distributions. Multi-Task Learning (MTL) [2] solves jointly those similar problems, each of which is considered a task.

Extending the taxonomy of [24], the MTL approaches can be divided in three main blocks: feature-learning models, regularization-based methods and combination approaches. Feature-learning models try to learn a space of features useful for all tasks at the same time. The regularization-based methods impose some soft constraints on the task-models so that there exists a connection across them. Finally, the combination approach combines task-specific models with a

The authors acknowledge financial support from the European Regional Development Fund and the Spanish State Research Agency of the Ministry of Economy, Industry, and Competitiveness under the project PID2019-106827GB-I00. They also thank the UAM–ADIC Chair for Data Science and Machine Learning and gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM.

© Springer Nature Switzerland AG 2022
P. García Bringas et al. (Eds.): HAIS 2022, LNAI 13469, pp. 223–235, 2022.
https://doi.org/10.1007/978-3-031-15471-3_20

common one shared for all tasks. Recently a convex combination formulation was proposed in [16].

In ML we call deep models those who have a feature learning process that construct new features along the training process. The standard example are deep neural networks where the nonlinear process up to the last hidden layer builds new extended features presumably better than the original ones. The shallow models by which we essentially mean models different from deep NNs, in contrast, use directly the original features or a fixed transformation of them. Shallow models are not limited to linear approaches. Although kernel models are very expressive due to the implicit transformation of the original features in some Reproducing Kernel Hilbert Space (RKHS), this transformation is non-learnable and they can be considered shallow. In feature-learning-based MTL we can find examples of both deep models [5,13,15] and shallow ones [12]. Also in regularization-based approaches we have examples with deep [23] and shallow approaches [1,7,17]. However, the combination-based approach has only been applied to shallow models [8,18,22].

In this work we propose a convex formulation for a combination-based MTL approach based on deep models. To the best of our knowledge this is the first combination-based approach to MTL using deep models. The convex formulation we use enables an interpretable parametrization. The goal of this work is to define this approach and test its properties. More precisely, our main contributions are:

- Review the taxonomy for MTL, where we include a third category, the combination-based approaches, different from the original feature-learning and regularization-based approaches.
- Show a general formulation for convex combination-based MTL. Where the hypotheses that are combined can be taken from a wide range of models, not limited to kernel models.
- Propose a combination-based MTL with deep models and use a convex formulation for better interpretability.
- Implement this approach and test it with four image datasets.

This rest of the paper is organized as follows. In Sect. 2 we revise the Multi-Task Learning paradigm, reviewing different approaches and proposing a taxonomy. In Sect. 3 we present the general formulation for convex combination-based approaches and propose its application using Neural Networks. In Sect. 4 we show the experiments carried out to test our proposal and analyze their results. The paper ends with some conclusions and pointers to further work.

2 Multi-Task Learning Approaches

Multi-Task Learning (MTL) tries to learn multiple tasks simultaneously with the goal of improving the learning process of each task. Given T tasks, a Multi-Task (MT) sample is $z = \{(\boldsymbol{x}_i^r, y_i^r) \in \mathbb{R}^d \times \mathcal{Y}; i = 1, \ldots, m_r; r = 1, \ldots, T\}$, where $\mathcal{Y}$ can be $\mathbb{R}$ in the case of regression or $\{0, 1\}$ in the case of classification; the superindex $r \in \{1, \ldots, T\}$ indicates the task and m_r the number of examples in

each task. The pair $(\boldsymbol{x}_i^r, y_i^r)$ can be also expressed as the triplet $(\boldsymbol{x}_i, y_i, r_i)$. The MT regularized risk for hypotheses h_r, that will be minimized, is defined as:

$$\sum_{r=1}^{T} \sum_{i=1}^{m_r} \ell(h_r(\boldsymbol{x}_i^r), y_i^r) + R(h_1, \ldots, h_T), \tag{1}$$

where ℓ is some loss function and R some regularizer. One strategy to minimize this risk, denoted Common Task Learning (CTL), consists in using a common model for all the tasks, $h_1, \ldots, h_T = h$. On the other side, in the Independent Task Learning (ITL) approach we minimize the risk independently for each task, without any transfer of information between them. Between these two extreme approaches lies MTL. The coupling between tasks can be enforced using different strategies. The choice of a strategy is influenced by the properties of the underlying models performing the learning process. In this paper we will focus on deep models, but in this section we will also discuss shallow models.

2.1 Multi-Task Learning with a Feature-Learning Approach

The feature-based approaches implement transfer learning by sharing a representation among tasks; that is, $h_r(\boldsymbol{x}) = g_r(f(\boldsymbol{x}))$, where f is some common feature transformation that can be learned and g_r is a task-specific function over these features. The first approach, *Hard Sharing*, is introduced in [5], where a Neural Network with shared layers and multiple outputs is used. The hidden layers are common to all tasks and, using the representation from the last hidden layer, a linear model is learned for each task; see Fig. 1 for an illustration. In the figure, a sample belonging to task 1 is used, the updated shared weights are represented in red, and in blue the updated specific weights. The input neurons are shown in yellow, the hidden ones in cyan and the output ones in magenta. The regularized risk corresponding to feature-learning MTL can be expressed as

$$\sum_{r=1}^{T} \sum_{i=1}^{m_r} \ell(g_r(f(\boldsymbol{x}_i^r)), y_i^r) + \mu_1 \sum_{r=1}^{T} \Omega_r(g_r) + \mu_2 \Omega(f), \tag{2}$$

where Ω_r and Ω are regularizers that penalize the complexity of the functions g_r and the function f, respectively; μ_1 and μ_2 are hyperparameters. The regularization over the predictive functions g_r can be done independently because the coupling is enforced by sharing the feature-learning function f.

A relaxation of the *Hard Sharing* approach consists in using the hypotheses $h_r(\boldsymbol{x}) = g_r(f_r(\boldsymbol{x}))$ where a coupling is enforced between the feature functions f_r. This is known as *Soft Sharing*, where specific networks are used for each task and some feature sharing mechanism is implemented at each level of the networks; examples are *cross-stitch networks* [13] or *sluice networks* [15]. In deep models, where a good representation is learned in the training process, Feature-Learning MTL is the most natural approach; however some Feature-Learning MTL approaches for shallow models can be found [12].

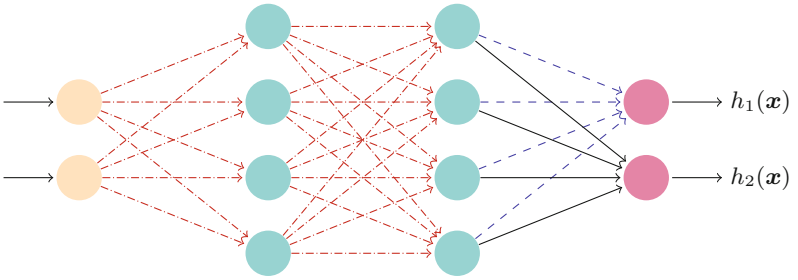

Fig. 1. *Hard sharing* neural network for two tasks and a two-dimensional input.

2.2 Multi-Task Learning with a Regularization-Based Approach

The regularization-based approaches are used when the hypothesis for each task can be expressed as $h_r(\boldsymbol{x}) = \boldsymbol{w_r}^\intercal \boldsymbol{\phi}(\boldsymbol{x})$, where $\boldsymbol{\phi}(\boldsymbol{x})$ is a non-learnable transformation which is the same for all tasks, and $\boldsymbol{w_r}$ are the parameters of interest to establish a relation between tasks. The transformation $\boldsymbol{\phi}(\boldsymbol{x})$ can be just the identity, using then the original features $\boldsymbol{x}$ in linear models, or some non-linear transformation of $\boldsymbol{x}$, explicit in deep models and implicit in kernel models. Here, the coupling is enforced by imposing some penalty over the matrix W whose columns are the vectors $\boldsymbol{w}_r$. The Multi-Task regularized risk is

$$\sum_{r=1}^{T} \sum_{i=1}^{m_r} \ell(\boldsymbol{w_r}^\intercal \boldsymbol{\phi}(\boldsymbol{x_i^r}), y_i^r) + \mu \Omega(W), \tag{3}$$

where $\Omega(W)$ is some regularizer of the matrix W and μ is a hyperparameter. For example, in [1,6] a low-rank constraint $\Omega(W) = \operatorname{rank} W$ is imposed over W, while in [7,17] a graph connecting the tasks is defined and a Laplacian regularization is used to penalize the distances between parameters, i.e., $\Omega(W) = \sum_{r,s=1}^{T} A_{rs} \left\| \boldsymbol{w}_r - \boldsymbol{w}_s \right\|^2$, where A is the adjacency matrix of the graph that encodes the pairwise task relations. These strategies can be more suitable for MTL with shallow models, but they are also applicable for deep ones [23]. In any case, in this work we use the standard L2 regularization common in deep networks; in particular, coupling is not necessarily enforced by the regularizer.

2.3 Multi-Task Learning with a Combination Approach

Another strategy, different to both the feature-learning and regularization-based approaches, is a combination $h_r(\boldsymbol{x}) = g(\boldsymbol{x}) + g_r(\boldsymbol{x})$ of a shared common model and task-specific ones. This approach was introduced in [8], where a combination of models $h_r(\boldsymbol{x}) = (\boldsymbol{w} + \boldsymbol{v_r})^\intercal \boldsymbol{\phi}(\boldsymbol{x}) + b + b_r$ is defined; here $\boldsymbol{w}$ and $\boldsymbol{v_r}$ are the common and task-specific weights, respectively, whereas b and b_r are the corresponding biases. The regularized risk is here

$$\sum_{r=1}^{T} \sum_{i=1}^{m_r} \ell((\boldsymbol{w} + \boldsymbol{v}_r)^\intercal \boldsymbol{\phi}(\boldsymbol{x_i^r}) + b + b_r, y_i^r) + \mu_1 \left\| \boldsymbol{w} \right\|^2 + \mu_2 \sum_{r=1}^{T} \left\| \boldsymbol{v}_r \right\|^2. \tag{4}$$

If $\boldsymbol{w}_r = \boldsymbol{w} + \boldsymbol{v}_r$, the risk in (4) is equivalent to that in (3) with the regularizer

$$\Omega(W) = \rho_1 \sum_{r=1}^{T} \left\| \boldsymbol{w}_r - \left(\sum_{r=1}^{T} \boldsymbol{w}_r \right) \right\|^2 + \rho_2 \sum_{r=1}^{T} \|\boldsymbol{w}_r\|^2$$

for some values of the hyperparameters $\rho_1(\mu_1, \mu_2)$ and $\rho_2(\mu_1, \mu_2)$. That is, it imposes a regularization that penalizes the complexity of the parameters w_r and the variance between these parameters. Observe that both the common and specific parts belong to the same RKHS defined by the transformation $\boldsymbol{\phi}$.

An extension proposed in [4] uses $h_r(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{\phi}(\boldsymbol{x}) + b + \boldsymbol{v}_r^\intercal \boldsymbol{\phi}_r(\boldsymbol{x}) + b_r$, where different transformations are used: $\boldsymbol{\phi}$ for the common and $\boldsymbol{\phi}_r$ for each of the specific parts. That is, the common part and each of the specific parts can belong to different spaces and, hence, capture distinct properties of the data. In [4] the connection of this MT approach with the *Learning Under Privileged Information* paradigm [20] is also outlined. A convex formulation for this approach, named Convex MTL, is presented in [16], where we have $h_r(\boldsymbol{x}) = \lambda\{\boldsymbol{w}^\intercal \boldsymbol{\phi}(\boldsymbol{x}) + b\} + (1-\lambda)\{\boldsymbol{v}_r^\intercal \boldsymbol{\phi}_r(\boldsymbol{x}) + b_r\}$ and λ is a hyperparameter in the $[0, 1]$ interval. This parameter controls how much to share among the tasks. When $\lambda = 1$, the model is equivalent to the CTL approach, whereas $\lambda = 0$ represents the ITL approach. The regularized risk corresponding to this convex formulation is

$$\sum_{r=1}^{T} \sum_{i=1}^{m_r} \ell(\lambda\{\boldsymbol{w}^\intercal \boldsymbol{\phi}(\boldsymbol{x}) + b\} + (1-\lambda)\{\boldsymbol{v}_r^\intercal \boldsymbol{\phi}_r(\boldsymbol{x}) + b_r\}, y_i^r) + \mu \left(\|\boldsymbol{w}\|^2 + \sum_{r=1}^{T} \|\boldsymbol{v}_r\|^2 \right), \tag{5}$$

where the hyperparameters μ_1 and μ_2 from (4) have been changed for λ and μ for a better interpretability: μ is the single regularization parameter and λ determines the specificity of our models. We can find the combination approach in the context of shallow models in [18,22].

3 Convex MTL Neural Networks

3.1 Definition

The Convex MTL formulation described above in terms of linear models in some RKHS, can be generalized as the problem of minimizing the regularized risk

$$\sum_{r=1}^{T} \sum_{i=1}^{m} \ell(\lambda g(\boldsymbol{x}_i^r) + (1-\lambda) g_r(\boldsymbol{x}_i^r), y_i^r) + \mu \left(\Omega(g) + \sum_{r=1}^{T} \Omega_r(g_r) \right), \tag{6}$$

where Ω and Ω_r are regularizers and g and g_r are functions. Observe that (6) is not an *a posteriori* combination of common and specific models, but the objective function is minimized jointly on g and the specific models $g_1, \ldots, g_T$. In (5) each model acts in a different space determined by the implicit transformations ϕ and ϕ_r, that is $g(\boldsymbol{x}_i^r; \boldsymbol{w}) = \boldsymbol{w}^\intercal \phi(x_i^r) + b$ and $g_r(\boldsymbol{x}_i^r; \boldsymbol{w}_r) = \boldsymbol{w}_r^\intercal \phi_r(x_i^r) + b_r$. This permits a great flexibility but also imposes, for instance, the challenge of finding

the optimal kernel width that implicitly defines the space for each model if kernels are used to define the underlying RKHS.

The Convex MTL neural network can be defined using a convex combination of common and task-specific models. The output of the overall model can be expressed as

$$h_r(\boldsymbol{x}_i^r) = \lambda\{\boldsymbol{w}^\intercal f(\boldsymbol{x}_i^r;\Theta) + b\} + (1-\lambda)\{\boldsymbol{w}_r^\intercal f_r(\boldsymbol{x}_i^r;\Theta_r) + b_r\}. \tag{7}$$

That is, we use neural networks as the models $g(\boldsymbol{x}_i^r;\boldsymbol{w},\Theta) = \boldsymbol{w}^\intercal f(\boldsymbol{x}_i^r;\Theta) + b$ and $g_r(\boldsymbol{x}_i^r;\boldsymbol{w}_r,\Theta_r) = \boldsymbol{w}_r^\intercal f_r(\boldsymbol{x}_i^r;\Theta_r) + b_r$. where Θ and Θ_r are the sets of hidden weights, and w, w_r are the output weights of the common and specific networks, respectively, and b and b_r the output biases. In this formulation, the common and specific feature transformations $f(\boldsymbol{x}_i^r;\Theta)$ and $f_r(\boldsymbol{x}_i^r;\Theta_r)$, the feature-building functions of the hidden layers, are automatically learned in the training process.

This formulation offers multiple combinations since we can model each common or independent function using different architectures. For example, we can use a larger network for the common part, since it will be fed with more data, and simpler networks for the specific parts. Even different types of neural networks, such as fully connected and convolutional, can be combined depending on the characteristics of each task. This combination of neural networks can also be interpreted as an implementation of the LUPI paradigm [20], i.e., the common network captures the privileged information for each of the tasks, since it can learn from more sources. To the best of our knowledge, this is the first joint-learning MTL approach for deep models, in contrast with previous feature-based or parameter-based approaches.

3.2 Training Procedure

The goal of the Convex MTL NN is to minimize the regularized risk

$$\sum_{r=1}^{T}\sum_{i=1}^{m} \ell(h_r(\boldsymbol{x}_i^r), y_i^r) + \mu\left(\|\boldsymbol{w}\|^2 + \sum_{r=1}^{T}\|\boldsymbol{w}_r\|^2 + \Omega(\Theta) + \Omega(\Theta_r)\right). \tag{8}$$

Here, h_r is defined as in equation (7), and $\Omega(\Theta)$ and $\Omega(\Theta_r)$ represents the L_2 regularization of the set of hidden weights of the common and specific networks, respectively. Given a loss $\ell(\hat{y}, y)$ and a pair $(\boldsymbol{x}_i^t, y_i^t)$ from task t, the gradient with respect to some parameters $\mathcal{P}$ is

$$\nabla_{\mathcal{P}}\ell(h_t(\boldsymbol{x}_i^t), y_i^t) = \frac{\partial}{\partial \hat{y}_i^t}\ell(\hat{y}_i^t, y_i^t)|_{\hat{y}_i^t = h_t(\boldsymbol{x}_i^t)}\nabla_{\mathcal{P}} h_t(\boldsymbol{x}_i^t). \tag{9}$$

Recall that we are using the formulation $h_t(\boldsymbol{x}_i^t) = \lambda\{\boldsymbol{w}^\intercal f(\boldsymbol{x}_i^t;\Theta) + b\} + (1-\lambda)\{\boldsymbol{w}_t^\intercal f_t(\boldsymbol{x}_i^t;\Theta_t) + b_t\}$, where we make a distinction between output weights $\boldsymbol{w}, \boldsymbol{w}_t$ and hidden parameters Θ, Θ_t. The corresponding gradients are

$$\begin{aligned}
&\nabla_{\boldsymbol{w}} h_t(\boldsymbol{x}_i^t) = \lambda\{f(\boldsymbol{x}_i^t,\Theta)\}, && \nabla_{\Theta} h_t(\boldsymbol{x}_i^t) = \lambda\{\boldsymbol{w}^\intercal \nabla_{\Theta} f(\boldsymbol{x}_i^t,\Theta)\} : \\
&\nabla_{\boldsymbol{w}_t} h_t(\boldsymbol{x}_i^t) = (1-\lambda)\{f_t(\boldsymbol{x}_i^t,\Theta)\}, && \nabla_{\Theta_t} h_t(\boldsymbol{x}_i^t) = (1-\lambda)\{\boldsymbol{w}^\intercal \nabla_{\Theta_t} f_t(\boldsymbol{x}_i^t,\Theta_t)\}; \\
&\nabla_{\boldsymbol{w}_r} h_t(\boldsymbol{x}_i^t) = 0, && \nabla_{\Theta_r} h_t(\boldsymbol{x}_i^t) = 0, \text{ for } r \neq t.
\end{aligned} \tag{10}$$

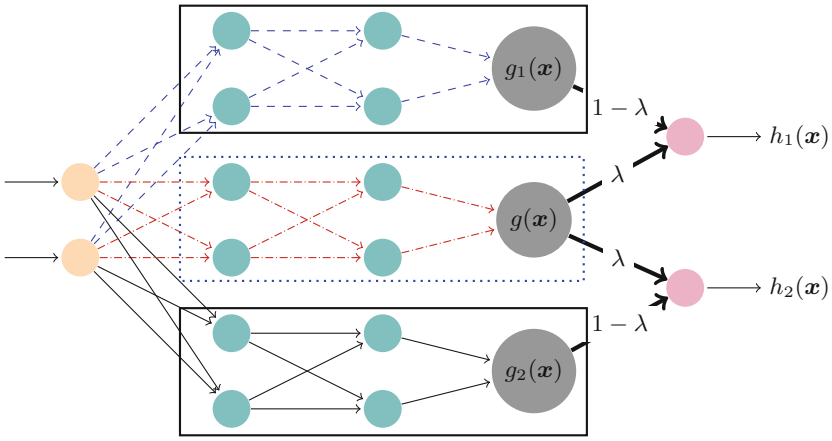

Fig. 2. Convex MTL neural network for two tasks and a two-dimensional input.

The convex combination information is transferred in the back propagation step as follows: the gradients of the loss function with respect to the common network parameters are scaled by λ, the gradients with respect to the t-th specific network parameters are scaled by $1-\lambda$, and the rest of the task-specialized networks parameters have null gradients, so they are not updated. The regularization is independent in each network, so the gradients of the regularizers are also computed independently. That is, no specific training algorithm has to be developed for the Convex MTL NN, so (8) can be minimized with any stochastic gradient descent strategy using back propagation. In Fig. 2, a Convex MTL NN is shown. In particular, the updated shared weights are represented in red, and in blue the updated specific weights. Specific networks are framed in black boxes and the common one in a blue box. The input neurons are shown in yellow, the hidden ones in cyan (except those in grey), and the output ones in magenta. We use the grey color for hidden neurons containing the intermediate functions that will be combined for the final output: $g_1(\boldsymbol{x})$, $g_2(\boldsymbol{x})$ and $g(\boldsymbol{x})$. The thick lines are the hyperparameters λ and $1-\lambda$ of the convex combination.

3.3 Implementation Details

Our implementation of the Convex MTL neural network is based on `PyTorch` [14]. Although we include the gradients expressions in equation (10), the `PyTorch` package implements automatic differentiation, so no explicit gradient formulation is necessary. The Convex MTL is implemented using (possibly different) `PyTorch` modules for the common model and each of the specific modules. In the forward pass of the network, the output for an example x from task r is computed using a forward pass of the common module and the specific module corresponding to task r, and the final output is simply the convex combination of both outputs. In the training phase, in which minibatches are used,

Algorithm 1: Forward pass for Convex MTL neural network.

Input: $X_{\text{mb}}, t_{\text{mb}}$ // Minibatch data and task labels
Output: f // Forward pass for the minibatch
Data: λ // Parameter of convex combination
Data: $g; g_1, \ldots, g_T$ // Modules of the common and specific networks
for $x_i, t_i \in (X_{mb}, t_{mb})$ **do**
  $f_i \leftarrow \lambda g(x_i) + (1 - \lambda) g_{t_i}(x_i)$ // Convex combination
end

the full minibatch is passed through the common model, but the minibatch is partitioned using only the corresponding examples for each task-specific modules. As mentioned above, with the adequate forward pass, the `PyTorch` package automatically computes the scaled gradients in the training phase.

In Algorithm 1 we show the pseudo-code of this Convex MTL forward pass, where g and $g_1, \ldots, g_T$ are the common and task-specific modules whose predictions are combined. As mentioned above, for the backward pass we rely on PyTorch automatic differentiation, so we do not need an explicit algorithm.

4 Experimental Results

4.1 Problems Description

To test the performance of the Convex MTL deep neural network approach we use four different image datasets: `var-MNIST`, `rot-MNIST`, `var-FMNIST` and `rot-FMNIST`. Datasets `var-MNIST` and `rot-MNIST` are the result of applying two different procedures, which will be detailed below, to the MNIST dataset [11], while for `var-FMNIST` and `rot-FMNIST` we apply the same procedure over the fashion-MNIST dataset [21]. Both MNIST and fashion-MNIST datasets are composed of 28×28 grey-scale images, with 10 balanced classes; also both problems have 70 000 examples. The procedures considered divide the original datasets and apply a different transformation to each resulting subset. To do this, we shuffle the original data and divide it equally among the tasks considered.

For datasets `var-MNIST` and `var-FMNIST` we consider two transformations described for the MNIST Variations datasets in [3]: *background random*, adding random noise to the original image, and *background image*, adding random patches of natural images. Using these transformations we define three tasks: `standard`, `random` and `images`, where either no transformation or one of the *background random* and the *background image* transformations is applied, respectively, to define each task. That is, two tasks have 23 333 examples each, and there are 23 334 in the third one.

The datasets `var-MNIST` and `var-FMNIST` are generated using the procedure specified in [9]. We define six different tasks, each one corresponding to a rotation of 0, 15, 30, 45, 60 and 75°, respectively; therefore, there are four tasks with 11 667 examples and two with 11 666. In Fig. 3, examples of the tasks for the four problems considered are shown.

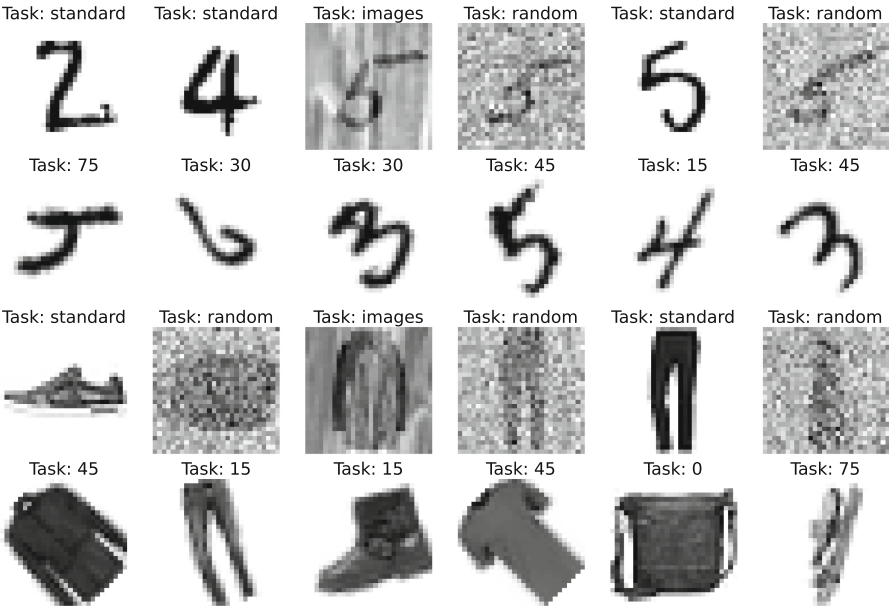

Fig. 3. Images of the four classification problems used. Each image has a title indicating the corresponding task. The rows correspond to `var-MNIST`, `rot-MNIST`, `var-FMNIST` and `rot-FMNIST` (from top to bottom).

4.2 Experimental Procedure

We compare four different models, all based on deep neural networks: a Common-Task Learning approach ctlNN, an Independent-Task learning approach itlNN, a Convex Multi-Task Learning approach cvxmtlNN and a *hard sharing* Multi-Task Learning approach hsNN. The base architecture of all models is a convolutional NN that we will name *convNet*. The architecture is based on the Spatial Transformer Network (STN) [10] architecture proposed in Pytorch[1], for further work using STN's. This *convNet* has 2 convolutional layers of kernel size 5, the first one with 10 output channels and the second one with 20; then we add a dropout layer, a max pooling layer and two hidden linear layers with 320 and 50 neurons each. In the ctlNN approach, a single *convNet* with 10 outputs, one for each class, is used. For the itlNN approach, an independent *convNet* with 10 outputs is used for each task. In cvxmtlNN both the common and task-specific networks are modelled using a *convNet* with 10 outputs; hsNN uses a *convNet* and a group of 10 outputs for each task.

All the models considered are trained using the *AdamW* algorithm and the optimal weight decay parameter μ is selected using a cross-validation grid search over the values $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$. The rest of the parameters corresponding to the algorithm are set to the default values: the dropout rate is 0.5

[1] www.pytorch.org/tutorials/intermediate/spatial_transformer_tutorial.html.

and a stride of 2 for the 2×2 max pooling layer. Additionally, in the cvxmtlNN model the mixing parameter λ is also included in the grid search using the values $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. The training and test sets are generated using a task-stratified 70 % and 30 % random split of the complete datasets. Grid-search is done by cross-validation using 5 folds over the training subset. We use task-stratified folds, that is, we divide the training set in 5 differents subsets where the task proportions are kept constant. Notice that the problems are class-balanced, so no further stratification is needed.

4.3 Analysis of the Results

To obtain results less sensitive to the randomness in deep networks, once the hyperparameters have been selected by cross-validation, we refit the models with optimal hyperparameters five times using the entire training set and the predictions are combined as described below. The final goal in classification problems is typically to maximize accuracy; however it cannot be used as a loss, so we will minimize the categorical cross entropy. We show both scores in the results.

In Table 1 we compute a single accuracy score for each model using the majority voting prediction of the 5 refitted models. In Table 2 we show the average cross entropy loss of the 5 different models. We also show in the tables the optimal values for hyperparameter λ^* selected in CV for cvxmtlNN. In both tables, cvxmtlNN obtains the best results in all four problems and the itlNN comes second except for the `var-MNIST` problem using majority voting. That is, training a specific model for each task obtains better results than the more rigid ctlNN or hsNN models. Also, although the ctlNN model obtains the worst results, the difference is not that large, so it suggests that the tasks are not very different, or that there exists information shared across tasks. The hsNN model consistently outperforms the ctlNN model and it seems to capture some shared information; however this hard sharing approach seems too rigid to fully exploit this common knowledge. Our proposal, the cvxmtlNN model, has the adequate flexibility because it trains specific modules for each task, but it also captures the shared information through the common model. Moreover, in cvxmtlNN the training of the common and specific models is made jointly and since this results in better models, we can conclude that, although the common and specific parts do not learn totally overlapping information, they complement each other's learning. This is supported by the fact that the λ values selected in CV are away from the extremes 0 and 1 of the independent and common models.

Table 1. Test accuracy with majority voting.

	var-MNIST	rot-MNIST	var-FMNIST	rot-FMNIST
ctlNN	0.964	0.973	0.784	0.834
itlNN	0.968	0.981	0.795	0.873
hsNN	0.971	0.980	0.770	0.852
cvxmtlNN	**0.974**	**0.984**	**0.812**	**0.880**
	($\lambda^* = 0.6$)	($\lambda^* = 0.8$)	($\lambda^* = 0.6$)	($\lambda^* = 0.6$)

Table 2. Test mean categorical cross entropy.

	var-MNIST	rot-MNIST	var-FMNIST	rot-FMNIST
ctlNN	1.274 ± 0.143	1.145 ± 0.039	2.369 ± 0.183	1.757 ± 0.075
itlNN	1.072 ± 0.029	0.873 ± 0.058	2.356 ± 0.130	1.598 ± 0.042
hsNN	1.087 ± 0.253	0.898 ± 0.073	3.067 ± 0.888	1.888 ± 0.075
cvxmtlNN	$\mathbf{0.924 \pm 0.024}$	$\mathbf{0.831 \pm 0.029}$	$\mathbf{2.147 \pm 0.090}$	$\mathbf{1.482 \pm 0.063}$
	($\lambda^* = 0.6$)	($\lambda^* = 0.8$)	($\lambda^* = 0.6$)	($\lambda^* = 0.6$)

5 Conclusions and Further Work

Here we have proposed a combination-based MTL approach using deep networks to define common and task-specific models that work together through a convex formulation. We have revised a taxonomy of previous MTL proposals adding the combination-based models as a distinct category; to the best of our knowledge, ours is the first proposal in this new category which uses neural networks.

The most popular NN based approach to MTL has been *hard sharing*, where tasks share the hidden parameters and different outputs are used for each task. In our experiments we have observed that our model outperforms the *hard sharing* approach in the four image problems considered. Moreover, our proposal also obtains better results than the baseline neural models for common- or independent-task learning. From this fact, we can infer that our MTL approach is able to extract and jointly exploit the information learned by the common and task-specific parts. We point out that the convex combination approach to MTL can also be applied to other models, such as SVMs. However, their potentially very high computational cost is well known and, in fact, we have not been able to apply them to our image classification problems.

As lines of further work, we point out that the mixing λ coefficient selected here as a hyperparameter by CV, can be alternatively seen as another network weight to be learned. It is also interesting to fully exploit the flexibility of our approach by using different architectures for each one of the common and task-specific modules. We are currently pursuing these and other ideas.

References

1. Ando, R.K., Zhang, T.: A framework for learning predictive structures from multiple tasks and unlabeled data. J. Mach. Learn. Res. **6**, 1817–1853 (2005)
2. Baxter, J.: A model of inductive bias learning. J. Artif. Intell. Res. **12**, 149–198 (2000)
3. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
4. Cai, F., Cherkassky, V.: SVM+ regression and multi-task learning. In: International Joint Conference on Neural Networks, IJCNN 2009, pp. 418–424. IEEE Computer Society (2009)
5. Caruana, R.: Multitask learning. Mach. Learn. **28**(1), 41–75 (1997)
6. Chen, J., Tang, L., Liu, J., Ye, J.: A convex formulation for learning shared structures from multiple tasks. In: ACM International Conference Proceeding Series, ICML 2009, vol. 382, pp. 137–144 (2009)
7. Evgeniou, T., Micchelli, C.A., Pontil, M.: Learning multiple tasks with kernel methods. J. Mach. Learn. Res. **6**, 615–637 (2005)
8. Evgeniou, T., Pontil, M.: Regularized multi-task learning. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 109–117. ACM (2004)
9. Ghifary, M., Kleijn, W.B., Zhang, M., Balduzzi, D.: Domain generalization for object recognition with multi-task autoencoders. In: IEEE International Conference on Computer Vision, ICCV, pp. 2551–2559. IEEE Computer Society (2015)
10. Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K.: Spatial transformer networks (2015). https://doi.org/10.48550/ARXIV.1506.02025. https://arxiv.org/abs/1506.02025
11. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
12. Maurer, A., Pontil, M., Romera-Paredes, B.: Sparse coding for multitask and transfer learning. In: Proceedings of the 30th International Conference on Machine Learning, ICML 2013, vol. 28, pp. 343–351. JMLR.org (2013)
13. Misra, I., Shrivastava, A., Gupta, A., Hebert, M.: Cross-stitch networks for multi-task learning. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, pp. 3994–4003. IEEE Computer Society (2016)
14. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035 (2019)
15. Ruder, S.: An overview of multi-task learning in deep neural networks. CoRR abs/1706.05098 (2017)
16. Ruiz, C., Alaíz, C.M., Dorronsoro, J.R.: A convex formulation of SVM-based multi-task learning. In: Pérez García, H., Sánchez González, L., Castejón Limas, M., Quintián Pardo, H., Corchado Rodríguez, E. (eds.) HAIS 2019. LNCS (LNAI), vol. 11734, pp. 404–415. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29859-3_35
17. Ruiz, C., Alaíz, C.M., Dorronsoro, J.R.: Convex graph Laplacian multi-task learning SVM. In: Farkaš, I., Masulli, P., Wermter, S. (eds.) ICANN 2020. LNCS, vol. 12397, pp. 142–154. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61616-8_12
18. Ruiz, C., Alaíz, C.M., Dorronsoro, J.R.: Convex formulation for multi-task L1-, L2-, and LS-SVMs. Neurocomputing **456**, 599–608 (2021)

19. Vapnik, V.: Estimation of Dependences Based on Empirical Data. Springer, New York (1982)
20. Vapnik, V., Izmailov, R.: Learning using privileged information: similarity control and knowledge transfer. J. Mach. Learn. Res. **16**, 2023–2049 (2015)
21. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms (2017)
22. Xu, S., An, X., Qiao, X., Zhu, L.: Multi-task least-squares support vector machines. Multimedia Tools Appl. **71**(2), 699–715 (2013). https://doi.org/10.1007/s11042-013-1526-5
23. Yang, Y., Hospedales, T.M.: Trace norm regularised deep multi-task learning. In: 5th International Conference on Learning Representations, ICLR 2017. OpenReview.net (2017)
24. Zhang, Y., Yang, Q.: An overview of multi-task learning. Natl. Sci. Rev. **5**(1), 30–43 (2017)