17th International Conference on Hybrid Artificial Intelligence Systems. Salamanca, Spain, September 5-7, 2022. Lecture Notes in Computer Science, Volume 13469. Springer, 2022. 223-235

DOI: https://doi.org/10.1007/978-3-031-15471-3_20

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

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

1 Introduction

In the Machine Learning (ML) field it is often assumed that the data is independently identically distributed, and the empirical risk minimization principle [20], typically used in supervised learning, bases its generalization abilities in this claim. However, we often find problems that are similar but where this assumption might be too strong. Multi-Task Learning (MTL) [3] solves jointly similar problems, each one considered a task, with data sampled from possibly different distributions. An MTL empirical risk is minimized with the goal of improving the learning process.

Extending the taxonomy of [25], the MTL approaches can be divided in three main blocks: feature-learning, regularization-based and combination approaches. The feature-learning approach tries to learn a space of features useful for all tasks at the same time. The regularization-based approaches impose some soft constraints on the task-models so that there exist a connection across them. Finally, the combination approach combines task-specific models with a common one shared for all tasks. Recently a convex formulation was proposed in [17].

* 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 projects TIN2016-76406-P (AEI/FEDER, UE) and 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.

In ML we call Deep Models those where there exists a feature learning process that construct new features, not just a selection of the original ones. The Shallow Models, in contrast, use directly the original features or a fixed, non-learnable transformation of them. In MTL we can find examples of both Deep and Shallow Models either in feature-learning [7,14,16,2,13] and regularization-based approaches [1,8,9,18,24]. However, the combination-based approach has only been applied to Shallow Models [10,23,19].

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 using Deep Models. The convex formulation used enables an interpretable parametrization. More precisely, our main contributions are:

- Revise a taxonomy for MTL: we include a third category, the combination-based approaches, different from the original feature-learning and regularization-based approaches.
- Show a general formulation for combination-based MTL.
- 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 Section 2 we revise the Multi-Task Learning paradigm revising different views and propose a taxonomy. In Section 3 we show the general formulation for convex combination-based approaches and propose the application of this approach with Neural Networks. In 4 we show the experiments carried out to test our proposal and analyze the results. Finally, the paper ends with some conclusions and pointers to further work in Section 5.

2 Multi-Task Learning Approaches

Multi-Task Learning (MTL) tries to learn multiple tasks simultaneously with the goal of improving the learning process in each task. Given T tasks, with m examples each, a Multi-Task (MT) sample is $z = \{(\boldsymbol{x}_i^r, y_i^r); i = 1, \ldots, m; r = 1, \ldots, T\}$, where r indicates the 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 hypothesis h_r, that will be minimized, is defined as:

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

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, minimizing the risk independently for each task, without any transfer of information between them, is the Independent Task Learning (ITL) approach. Between these two trivial

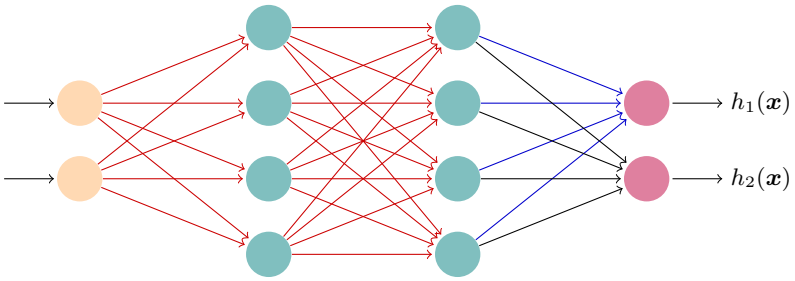

Fig. 1: *Hard Sharing* Neural Network for two tasks. Assuming a sample belonging to task 1 is used, the updated shared weights are represented in red, and in blue the updated specific weights. The input neurons are shown in yellow, the hidden ones in cyan and the output ones in magenta.

approaches lies the MTL. The coupling between tasks can be enforced using different strategies. The choice of the strategy is influenced by the properties of the underlying models performing the learning process. In this paper we will focus on deep models, but for completeness in this section we will also discuss details about shallow models.

2.1 Multi-Task Learning with a Feature-Learning Approach

The feature-based approaches implement the transfer learning by sharing a representation among tasks, that is $h_r(\boldsymbol{x}) = g_r(f(\boldsymbol{x}))$ where f is some feature transformation that can be learned. The first approach, *Hard Sharing*, is introduced in [7], 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 Figure 1 for an example. The corresponding regularized risk can be expressed as

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

where Ω_r and Ω are regularizers to penalize the complexity of the functions g_r and the function f, respectively; and 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 hypothesis $h_r(\boldsymbol{x}) = g_r(f_r(\boldsymbol{x}))$ where a coupling is enforced between the feature functions f_r. This is known as *Soft Sharing* approaches, where specific networks are used for each task and some feature sharing mechanism is implemented at each level of the networks, e.g. *cross-stich networks* [14] or *sluice networks* [16]. 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 [2,13]

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 $h_r(\boldsymbol{x}) = \boldsymbol{w_r^\intercal \phi(x)}$, where $\boldsymbol{\phi(x)}$ is a non-learnable transformation and $\boldsymbol{w_r}$ are the parameters of interest to establish a relation between tasks. The transformation $\boldsymbol{\phi(x)}$ can be either 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

$$\sum_{r=1}^{T}\sum_{i=1}^{m} \ell(\boldsymbol{w_r^\intercal \phi(x_i^r)}, y_i^r) + \mu\Omega(W), \tag{2}$$

where $\boldsymbol{\phi}$ is a non-learnable transformation that is common to all the tasks, $\Omega(W)$ is some regularizer of W to enforce a coupling between the columns and μ is a hyperparameter. For example, in [1,8] a low-rank constraint is imposed over W, i.e. $\Omega(W) = \operatorname{rank} W$; while in [9,18] a graph connecting the tasks is defined and a Laplacian regularization is used to penalize the distances between parameters, that is $\Omega(W) = \sum_{r,s=1}^{T} A_{rs} \left\|w_r - w_s\right\|^2$, where A is the adjacency matrix 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 [24].

2.3 Multi-Task Learning with a Combination Approach

Another strategy, different to both feature-learning and regularization-based approaches, is a combination of a shared common model and task-specific ones: $h_r(\boldsymbol{x}) = g(\boldsymbol{x}) + g_r(\boldsymbol{x})$. This approach was introduced in [10] where a combination of models is defined, i.e. $h_r(\boldsymbol{x}) = (\boldsymbol{w}+\boldsymbol{v_r})^\intercal \boldsymbol{\phi}(\boldsymbol{x})+b+b_r$, where $\boldsymbol{w}$ and $\boldsymbol{w_r}$ are the weights, and b and b_r the biases (common and task-specific, respectively). The corresponding regularized risk is

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

It can be shown that this regularized risk is equivalent to (2) with the regularizer

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

for some values $\rho_1(\mu_1, \mu_2)$ and $\rho_2(\mu_1, \mu_2)$, that is, it imposes a regularization that penalizes the complexity of 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 implicit transformation $\boldsymbol{\phi}$.

An extension shown in [5,6] uses $h_r(\boldsymbol{x}) = \boldsymbol{w^\intercal \phi(x)} + \boldsymbol{v_r^\intercal \phi_r(x)} + b + b_r$, 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 [5] it is also outlined the connection of this MT approach with the *Learning Under Privileged Information* paradigm [21].

A convex formulation for this approach is presented in [17], namely *ConvexMTL*, where

$$h_r(\boldsymbol{x}) = \lambda\{\boldsymbol{w}^\intercal \boldsymbol{\phi}(\boldsymbol{x}) + b\} + (1-\lambda)\{\boldsymbol{v_r}^\intercal \boldsymbol{\phi_r}(\boldsymbol{x}) + b_r\}$$

and λ is a hyperparameter in the $[0, 1]$ interval. The parameter λ 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

$$\sum_{r=1}^{T}\sum_{i=1}^{m} \ell(\lambda\{\boldsymbol{w}^\intercal \boldsymbol{\phi}(\boldsymbol{x}) + b\} + (1-\lambda)\{\boldsymbol{v_r}^\intercal \boldsymbol{\phi_r}(\boldsymbol{x}) + b_r\}, y_i^r) + \mu\left(\|\boldsymbol{w}\|^2 + \sum_{r=1}^{T}\|\boldsymbol{v}_r\|^2\right), \tag{3}$$

where μ_1 and μ_2 have been changed for λ and μ for a better interpretability. We can find the combination approach in the context of shallow models, e.g. [23,19].

3 Convex MTL Neural Networks

3.1 Definition

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

$$\sum_{r=1}^{T}\sum_{i=1}^{m} \ell(\lambda g(\boldsymbol{x}_i^r) + (1-\lambda) g_r(\boldsymbol{x}_i^r), y_i^r) + \mu\left(\Omega(g) + \sum_{r=1}^{T}\Omega_r(g_r)\right), \tag{4}$$

where Ω and Ω_r are regularizers and g and g_r are hypothesis. Observe that (4) 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 (3) each model is defined in a different space given by the implicit transformations ϕ and ϕ_r, that is

$$g(\boldsymbol{x}_i^r; \boldsymbol{w}) = \boldsymbol{w}^\intercal \boldsymbol{\phi}(\boldsymbol{x_i^r}) + b, \;\; g_r(\boldsymbol{x}_i^r; \boldsymbol{w}_r) = \boldsymbol{w_r}^\intercal \boldsymbol{\phi_r}(\boldsymbol{x_i^r}) + b_r.$$

This permits a great flexibility but also imposes the challenge of finding the optimal kernel width that implicitly defines the space for each model.

The *ConvexMTL* NN can be defined using a convex combination of a common model and task-specific models. The output of the model can be expressed as

$$h_r(\boldsymbol{x}_i^r) = \lambda\{\boldsymbol{w}^\intercal \boldsymbol{f}(\boldsymbol{x}_i^r; \Theta) + b\} + (1-\lambda)\{\boldsymbol{w_r}^\intercal \boldsymbol{f_r}(\boldsymbol{x}_i^r; \Theta_r) + b_r\}. \tag{5}$$

That is, we use Neural Networks as the models:

$$g(\boldsymbol{x}_i^r; \boldsymbol{w}, \Theta) = \boldsymbol{w}^\intercal \boldsymbol{f}(\boldsymbol{x}_i^r; \Theta) + b, \;\; g_r(\boldsymbol{x}_i^r; \boldsymbol{w}_r, \Theta_r) = \boldsymbol{w_r}^\intercal \boldsymbol{f_r}(\boldsymbol{x}_i^r; \Theta_r) + b_r,$$

where Θ and Θ_r are the sets of hidden parameters, and w, w_r, b and b_r are the weights and biases of the output layer (of the common and specific networks, respectively). In this formulation, the common and specific feature transformations $\boldsymbol{f}(\boldsymbol{x}_i^r;\Theta)$ and $\boldsymbol{f_r}(\boldsymbol{x}_i^r;\Theta_r)$, that represent the feature-building functions of the hidden layers, are automatically learned from data 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 [21], e.g. the common network can represent the privileged information for each of the tasks, since it can learn from more sources. To the best of our knowledge, MTL has been implemented in Neural Networks either with a feature-based or parameter-based approaches, so this is the first MTL with a joint-learning approach for deep models.

3.2 Training Procedure

The goal of the *ConvexMTL* NN is minimizing the regularized risk

$$\sum_{r=1}^{T}\sum_{i=1}^{m}\ell(h_r(\boldsymbol{x}_i^r),y_i^r)+\mu\left(\|\boldsymbol{w}\|^2+\sum_{r=1}^{T}\|\boldsymbol{w}_r\|^2+\Omega(\Theta)+\Omega(\Theta_r)\right). \tag{6}$$

Here, h_r is defined as in equation (5), 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 $\ell(\hat{y},y)$, the gradient for any set of parameters $\mathcal{P}$ is

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

Our set of parameters $\mathcal{P}$ is partitioned in the set of parameters of each network, so $\mathcal{P}=(\{\boldsymbol{w}\}\cup\Theta)\cup\bigcup_{r=1}^{T}(\{\boldsymbol{w}_r\}\cup\Theta_r)$, and the gradients are

$$\begin{aligned}
\nabla_{\boldsymbol{w}}h_t(\boldsymbol{x}_i^t)&=\lambda\{f(\boldsymbol{x}_i^t,\Theta)\};\\
\nabla_{\Theta}h_t(\boldsymbol{x}_i^t)&=\lambda\{\boldsymbol{w}^\intercal\nabla_{\Theta}f(\boldsymbol{x}_i^t,\Theta)\}:\\
\nabla_{\boldsymbol{w}_t}h_t(\boldsymbol{x}_i^t)&=(1-\lambda)\{f_t(\boldsymbol{x}_i^t,\Theta)\};\\
\nabla_{\Theta_t}h_t(\boldsymbol{x}_i^t)&=(1-\lambda)\{\boldsymbol{w}^\intercal\nabla_{\Theta_t}f_t(\boldsymbol{x}_i^t,\Theta_t)\};\\
\nabla_{\boldsymbol{w}_r}h_t(\boldsymbol{x}_i^t)&=0,\ \text{for } r\neq t;\\
\nabla_{\Theta_r}h_t(\boldsymbol{x}_i^t)&=0,\ \text{for } r\neq t.
\end{aligned} \tag{7}$$

The gradient of the loss function is scaled with λ in the common network and with $1-\lambda$ in the t-th specific network, while the rest of the task-specialized networks are not updated. The regularization is independent in each network,

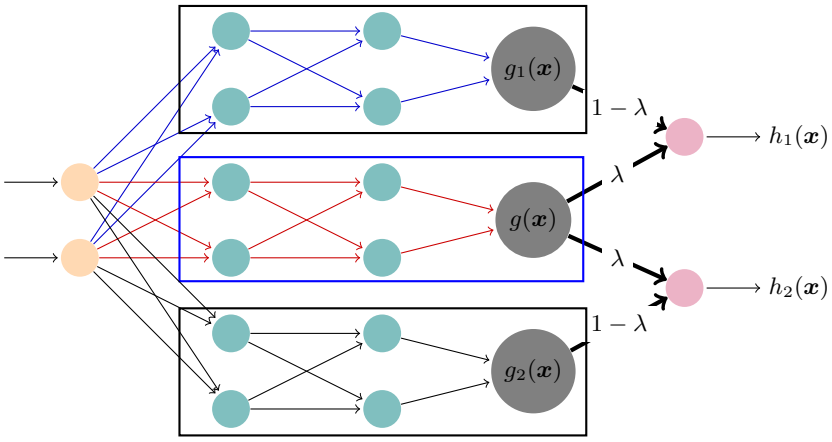

Fig. 2: *ConvexMTL* Neural Network for two tasks. Assuming a sample belonging to task 1 is used, the updated shared weights are represented in red, and in blue the updated specific weights. Specific networks are framed in black boxes and the common one in a blue box. The input neurons are shown in yellow, 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.

so the gradients of the regularizers are also computed independently. That is, no specific training algorithm has to be developed for the *ConvexMTL* NN, so (6) can be minimized with any stochastic gradient descent strategy using back propagation. In Figure 2, a *ConvexMTL* NN is shown in the gradient update step.

3.3 Implementation

Our implementation of the *ConvexMTL* Neural Network is based on `PyTorch` [15]. Although we include the gradients expressions in equation (7), the `PyTorch` package implements automatic differentiation, so no explicit gradient formulation is necessary. The *ConvexMTL* 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, 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 *ConvexMTL* forward pass.

Algorithm 1: Forward pass for *ConvexMTL* 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
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

4 Experimental Results

4.1 Problems Description

To test the performance of the ConvexMTL DeepNN approach we use four different image datasets: `var-MNIST`, `rot-MNIST`, `var-FMNIST` and `rot-FMNIST`. These datasets are generated using different transformations of other datasets: the first two using the MNIST dataset [12] as base, and the last two using the fashion-MNIST dataset [22].

Both MNIST and fashion-MNIST datasets are composed by 28×28 greyscale images, each belonging to one of 10 balanced classes, also both have 70 k examples. We define our datasets shuffling the original data and dividing it among the tasks considered, so each task has the same number of examples; then we apply the corresponding transformation to the images of each task.

The datasets `var-MNIST` and `var-FMNIST` are the result of applying two transformations described for the MNIST Variations datasets [4]. We consider only the transformations *background random*, adding random noise with the original image; and *background image*, adding with a random patch of a natural image. Using these transformations we define three tasks: `standard`, `random` and `image`, where no transformation, the *background random* and the *background image* transformations applied, respectively, to define each task. That is, two tasks have 23 333 examples each, and there are 23 334 in the remaining one.

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

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.
- A *hard sharing* Multi-Task Learning approach hsNN.

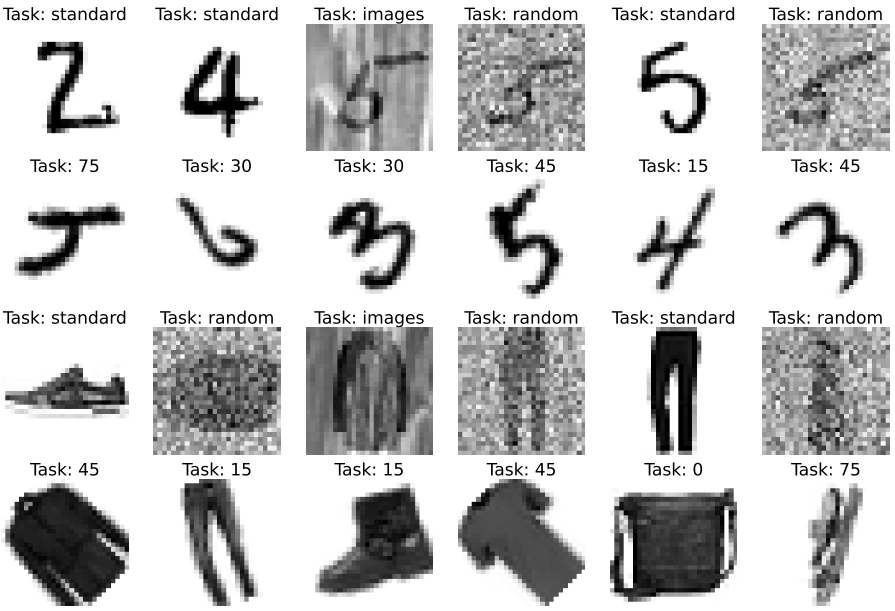

Fig. 3: Images of the four classification problems used. Each image has a title indicating the corresponding task. The rows correspond to `var-MNIST`, `rot-MNIST`, `var-FMNIST` and `rot-FMNIST` (from top to bottom).

The base architecture of every model is a convolutional NN that we will name *convNet*. This *convNet* has 2 convolutional layers of kernel size 5, the first one with 10 output channels and the second one with 20; then 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 output neurons, one for each class, is used. For the itlNN approach, an independent *convNet* with 10 output neurons is used for each task. In the cvxmtlNN, both the common and task-specific networks are modelled using a *convNet* with 10 output neurons. The 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 $\left\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\right\}$. The rest of the parameters corresponding to the algorithm are set to the default values: the dropout rate is 0.5 and a padding of 0 for the max pooling layer. Additionally, in the cvxmtlNN model the 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 % of the complete datasets. The cross-validation used in the grid-search is defined using 5 task-stratified folds.

Table 1: Test Accuracy with Majority Voting.

	var-MNIST	rot-MNIST	var-FMNIST	rot-FMNIST
ctINN	0.964	0.973	0.784	0.834
itINN	0.968	0.981	0.795	0.873
cvxmtINN	0.974	0.984	0.812	0.880
hsNN	0.971	0.980	0.770	0.852

Table 2: Test Mean Categorical Cross Entropy.

	var-MNIST	rot-MNIST	var-FMNIST	rot-FMNIST
ctINN	1.274 ± 0.143	1.145 ± 0.039	2.369 ± 0.183	1.757 ± 0.075
itINN	1.072 ± 0.029	0.873 ± 0.058	2.356 ± 0.130	1.598 ± 0.042
cvxmtINN	0.924 ± 0.024	0.831 ± 0.029	2.147 ± 0.090	1.482 ± 0.063
hsNN	1.087 ± 0.253	0.898 ± 0.073	3.067 ± 0.888	1.888 ± 0.075

4.3 Results Analysis

To show results less sensitive to randomness, the best models, with the optimal parameters selected in the cross-validation, are refitted using the whole training set and 5 different predictions are made. In classification problems, the final goal is typically the accuracy score, however it is not a differentiable loss, so the categorical cross entropy is used instead as the loss function to minimize. In this results we show both losses: accuracy and categorical cross entropy.

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. In all tables, the cvxmtINN obtains the best results in all four problems and the itINN 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 ctINN or hsNN models. Also, although the ctINN model obtains the worst results, the difference is not that large, so it induces the thought that the tasks are not very different, or that there exists information shared across tasks. The hsNN consistently outperforms the naive ctINN 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 cvxmtINN model has flexibility because it trains specific modules for each task, but it also captures the shared information in the common model. Moreover, we remark that in *ConvexMTL* the training of the common and specific models is made jointly, and since this results into a better model, we can conclude that the information learned in the common and specific parts is not totally overlapping, but they complement each other instead.

5 Discussion and Conclusions

In this paper we have proposed a combination-based MTL approach using deep models that combines a common and task-specific models using a convex formu-

lation. We have revised the taxonomy of MTL to include a distinct category for the combination-based models, and our proposal is, to the best of our knowledge , the first of this category based on Neural Networks.

The most popular approach to MTL with NN's has been *hard sharing*, which shares the hidden parameters and use a different output layer for each task. In our experiments we have observed that our model outperforms the *hard sharing* approach in four image problems. Moreover, our proposal also obtains better results than the baseline models based on common-task or independent-task learning. From this fact, we can infer that in our MTL approach the information learned by the common and task-specific parts is somehow complementary. The convex combination MTL approach can also be applied to shallow models, such as SVM's. However, due to their computational cost, we have not been able to apply these to our image classification problems.

As lines of further work, there are some interesting ideas to explore. In first place, our λ hyperparameter, that we currently select using CV, can be incorporated as another parameter of the networks to be learned using gradient descent. Also, it is interesting to fully exploit the flexibility of our approach by using different architectures for each module, common and task-specific ones.

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. Argyriou, A., Evgeniou, T., Pontil, M.: Convex multi-task feature learning. Mach. Learn. **73**(3), 243–272 (2008)
3. Baxter, J.: A model of inductive bias learning. Journal of artificial intelligence research **12**, 149–198 (2000)
4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
5. Cai, F., Cherkassky, V.: SVM+ regression and multi-task learning. In: International Joint Conference on Neural Networks, IJCNN 2009, Atlanta, Georgia, USA, 14-19 June 2009. pp. 418–424. IEEE Computer Society (2009)
6. Cai, F., Cherkassky, V.: Generalized SMO algorithm for svm-based multitask learning. IEEE Trans. Neural Networks Learn. Syst. **23**(6), 997–1003 (2012)
7. Caruana, R.: Multitask learning. Mach. Learn. **28**(1), 41–75 (1997)
8. Chen, J., Tang, L., Liu, J., Ye, J.: A convex formulation for learning shared structures from multiple tasks. In: Danyluk, A.P., Bottou, L., Littman, M.L. (eds.) Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009. ACM International Conference Proceeding Series, vol. 382, pp. 137–144. ACM (2009)
9. Evgeniou, T., Micchelli, C.A., Pontil, M.: Learning multiple tasks with kernel methods. J. Mach. Learn. Res. **6**, 615–637 (2005)
10. Evgeniou, T., Pontil, M.: Regularized multi–task learning. In: Kim, W., Kohavi, R., Gehrke, J., DuMouchel, W. (eds.) Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004. pp. 109–117. ACM (2004)

11. Ghifary, M., Kleijn, W.B., Zhang, M., Balduzzi, D.: Domain generalization for object recognition with multi-task autoencoders. In: 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015. pp. 2551–2559. IEEE Computer Society (2015)
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
13. 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, Atlanta, GA, USA, 16-21 June 2013. JMLR Workshop and Conference Proceedings, vol. 28, pp. 343–351. JMLR.org (2013)
14. 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, Las Vegas, NV, USA, June 27-30, 2016. pp. 3994–4003. IEEE Computer Society (2016)
15. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
16. Ruder, S.: An overview of multi-task learning in deep neural networks. CoRR **abs/1706.05098** (2017)
17. Ruiz, C., Alaíz, C.M., Dorronsoro, J.R.: A convex formulation of svm-based multi-task learning. vol. 11734, pp. 404–415. Springer (2019)
18. Ruiz, C., Alaíz, C.M., Dorronsoro, J.R.: Convex graph laplacian multi-task learning SVM. In: Farkas, I., Masulli, P., Wermter, S. (eds.) Artificial Neural Networks and Machine Learning - ICANN 2020 - 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15-18, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12397, pp. 142–154. Springer (2020)
19. Ruiz, C., Alaíz, C.M., Dorronsoro, J.R.: Convex formulation for multi-task l1-, l2-, and ls-svms. Neurocomputing **456**, 599–608 (2021)
20. Vapnik, V.: Estimation of dependences based on empirical data. Springer Science & Business Media (1982)
21. Vapnik, V., Izmailov, R.: Learning using privileged information: similarity control and knowledge transfer. J. Mach. Learn. Res. **16**, 2023–2049 (2015)
22. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017)
23. Xu, S., An, X., Qiao, X., Zhu, L.: Multi-task least-squares support vector machines. Multim. Tools Appl. **71**(2), 699–715 (2014)
24. Yang, Y., Hospedales, T.M.: Trace norm regularised deep multi-task learning. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings. OpenReview.net (2017)
25. Zhang, Y., Yang, Q.: An overview of multi-task learning. National Science Review **5**(1), 30–43 (2017)