29 May 2017

Note: If you are looking for a review paper, this blog post is also available as an article on arXiv.

Table of contents:

# Introduction

In Machine Learning (ML), we typically care about optimizing for a particular metric, whether this is a score on a certain benchmark or a business KPI. In order to do this, we generally train a single model or an ensemble of models to perform our desired task. We then fine-tune and tweak these models until their performance no longer increases. While we can generally achieve acceptable performance this way, by being laser-focused on our single task, we ignore information that might help us do even better on the metric we care about. Specifically, this information comes from the training signals of related tasks. By sharing representations between related tasks, we can enable our model to generalize better on our original task. This approach is called Multi-Task Learning (MTL) and will be the topic of this blog post.

Multi-task learning has been used successfully across all applications of machine learning, from natural language processing [ 1 ] and speech recognition [ 2 ] to computer vision [ 3 ] and drug discovery [ 4 ]. MTL comes in many guises: joint learning, learning to learn, and learning with auxiliary tasks are only some names that have been used to refer to it. Generally, as soon as you find yourself optimizing more than one loss function, you are effectively doing multi-task learning (in contrast to single-task learning). In those scenarios, it helps to think about what you are trying to do explicitly in terms of MTL and to draw insights from it.

Even if you're only optimizing one loss as is the typical case, chances are there is an auxiliary task that will help you improve upon your main task. Rich Caruana [ 5 ] summarizes the goal of MTL succinctly: "MTL improves generalization by leveraging the domain-specific information contained in the training signals of related tasks".

Over the course of this blog post, I will try to give a general overview of the current state of multi-task learning, in particular when it comes to MTL with deep neural networks. I will first motivate MTL from different perspectives. I will then introduce the two most frequently employed methods for MTL in Deep Learning. Subsequently, I will describe mechanisms that together illustrate why MTL works in practice. Before looking at more advanced neural network-based MTL methods, I will provide some context by discussing the literature in MTL. I will then introduce some more powerful recently proposed methods for MTL in deep neural networks. Finally, I will talk about commonly used types of auxiliary tasks and discuss what makes a good auxiliary task for MTL.

# Motivation

We can motivate multi-task learning in different ways: Biologically, we can see multi-task learning as being inspired by human learning. For learning new tasks, we often apply the knowledge we have acquired by learning related tasks. For instance, a baby first learns to recognize faces and can then apply this knowledge to recognize other objects.

From a pedagogical perspective, we often learn tasks first that provide us with the necessary skills to master more complex techniques. This is true for learning the proper way of falling in martial arts, e.g. Judo as much as learning to program.

Taking an example out of pop culture, we can also consider *The Karate Kid* (1984) (thanks to Margaret Mitchell and Adrian Benton for the inspiration). In the movie, *sensei* Mr Miyagi teaches the karate kid seemingly unrelated tasks such as sanding the floor and waxing a car. In hindsight, these, however, turn out to equip him with invaluable skills that are relevant for learning karate .

MTL as inductive bias : l2 regulariation prefers sparse solution

Finally, we can motivate multi-task learning from a machine learning point of view: We can view multi-task learning as a form of inductive transfer. Inductive transfer can help improve a model by introducing an inductive bias, which causes a model to prefer some hypotheses over others. For instance, a common form of inductive bias is $\ell_1$ regularization, which leads to a preference for sparse solutions. In the case of MTL, the inductive bias is provided by the auxiliary tasks, which cause the model to prefer hypotheses that explain more than one task. As we will see shortly, this generally leads to solutions that generalize better.

# Two MTL methods for Deep Learning

So far, we have focused on theoretical motivations for MTL. To make the ideas of MTL more concrete, we will now look at the two most commonly used ways to perform multi-task learning in deep neural networks. In the context of Deep Learning, multi-task learning is typically done with either *hard* or *soft parameter sharing* of hidden layers.

## Hard parameter sharing

Hard parameter sharing is the most commonly used approach to MTL in neural networks and goes back to [ 6 ]. It is generally applied by sharing the hidden layers between all tasks, while keeping several task-specific output layers.

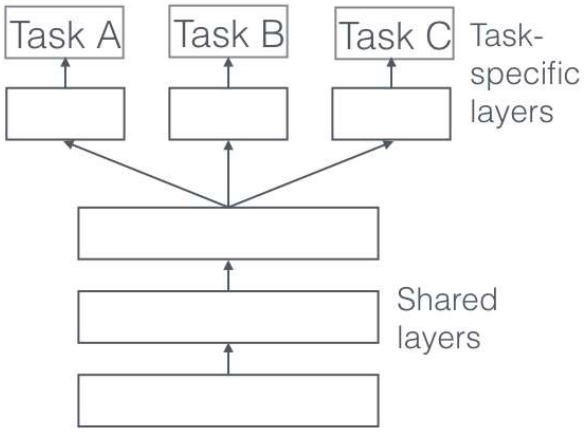


Figure 1: Hard parameter sharing for multi-task learning in deep neural networks

Hard parameter sharing greatly reduces the risk of overfitting. In fact, [ 7 ] showed that the risk of overfitting the shared parameters is an order N -- where N is the number of tasks -- smaller than overfitting the task-specific parameters, i.e. the output layers. This makes sense intuitively: The more tasks we are learning simultaneously, the more our model has to find a representation that captures all of the tasks and the less is our chance of overfitting on our original task.

## Soft parameter sharing

In soft parameter sharing on the other hand, each task has its own model with its own parameters. The distance between the parameters of the model is then regularized in order to encourage the parameters to be similar. [ 8 ] for instance use the $\ell_2$ norm for regularization, while [ 9 ] use the trace norm.

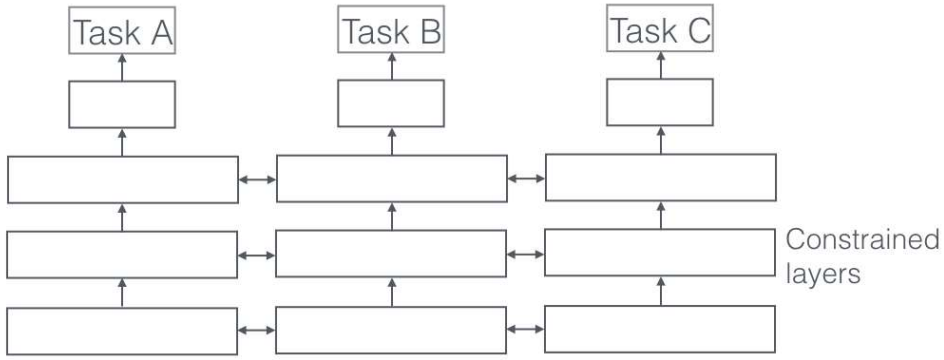


Figure 2: Soft parameter sharing for multi-task learning in deep neural networks

The constraints used for soft parameter sharing in deep neural networks have been greatly inspired by regularization techniques for MTL that have been developed for other models, which we will soon discuss.

# Why does MTL work?

Even though an inductive bias obtained through multi-task learning seems intuitively plausible, in order to understand MTL better, we need to look at the mechanisms that underlie it. Most of these have first been proposed by Caruana (1998). For all examples, we will assume that we have two related tasks $A$ and $B$, which rely on a common hidden layer representation $F$.

### Implicit data augmentation

MTL effectively increases the sample size that we are using for training our model. As all tasks are at least somewhat noisy, when training a model on some task $A$, our aim is to learn a good representation for task $A$ that ideally ignores the data-dependent noise and generalizes well. As different tasks have different noise patterns, a model that learns two tasks simultaneously is able to learn a more general representation. Learning just task $A$ bears the risk of overfitting to task $A$, while learning $A$ and $B$ jointly enables the model to obtain a better representation $F$ through averaging the noise patterns.

### Attention focusing

If a task is very noisy or data is limited and high-dimensional, it can be difficult for a model to differentiate between relevant and irrelevant features. MTL can help the model focus its attention on those features that actually matter as other tasks will provide additional evidence for the relevance or irrelevance of those features.

### Eavesdropping

Some features $G$ are easy to learn for some task $B$, while being difficult to learn for another task $A$. This might either be because $A$ interacts with the features in a more complex way or because other features are impeding the model's ability to learn $G$. Through MTL, we can allow the model to *eavesdrop*, i.e. learn $G$ through task $B$. The easiest way to do this is through *hints* [ 10 ], i.e. directly training the model to predict the most important features.

### Representation bias

MTL biases the model to prefer representations that other tasks also prefer. This will also help the model to generalize to new tasks in the future as a hypothesis space that performs well for a sufficiently large number of training tasks will also perform well for learning novel tasks as long as they are from the same environment [ 11 ].

### Regularization

Finally, MTL acts as a regularizer by introducing an inductive bias. As such, it reduces the risk of overfitting as well as the Rademacher complexity of the model, i.e. its ability to fit random noise.

## MTL in non-neural models

In order to better understand MTL in deep neural networks, we will now look to the existing literature on MTL for linear models, kernel methods, and Bayesian algorithms. In particular, we will discuss two main ideas that have been pervasive throughout the history of multi-task learning: enforcing sparsity across tasks through norm regularization; and modelling the relationships between tasks.

Note that many approaches to MTL in the literature deal with a homogenous setting: They assume that all tasks are associated with a single output, e.g. the multi-class MNIST dataset is typically cast as 10 binary classification tasks. More recent approaches deal with a more realistic, heterogeneous setting where each task corresponds to a unique set of outputs.

## Block-sparse regularization

In order to better connect the following approaches, let us first introduce some notation. We have $T$ tasks. For each task $t$, we have a model $m_t$ with parameters $a_t$ of dimensionality $d$. We can write the parameters as a column vector $a_t = \left[a_{1,t} \cdots a_{d,t}\right]^\top$. We now stack these column vectors $a_1, \ldots, a_T$ column by column to form a matrix $A \in \mathbb{R}^{d \times T}$. The $i$-th row of $A$ then contains the parameter $a_{i,\cdot}$ corresponding to the $i$-th feature of the model for every task, while the $j$-th column of $A$ contains the parameters $a_{\cdot,j}$ corresponding to the $j$-th model.

Many existing methods make some sparsity assumption with regard to the parameters of our models. [ 12 ] assume that all models share a small set of features. In terms of our task parameter matrix $A$, this means that all but a few rows are $0$, which corresponds to only a few features being used across *all* tasks. In order to enforce this, they generalize the $\ell_1$ norm to the MTL setting. Recall that the $\ell_1$ norm is a constraint on the sum of the parameters, which forces all but a few parameters to be exactly $0$. It is also known as lasso (**l**east **a**bsolute **s**hrinkage and **s**election **o**perator).

While in the single-task setting, the $\ell_1$ norm is computed based on the parameter vector $a_t$ of the respective task $t$, for MTL we compute it over our task parameter matrix $A$. In order to do this, we first compute an $\ell_q$ norm across each row $a_i$ containing the parameter corresponding to the $i$-th feature across all tasks, which yields a vector $b = \left[\|a_1\|_q \ldots \|a_d\|_q\right] \in \mathbb{R}^d$. We then compute the $\ell_1$ norm of this vector, which forces all but a few entries of $b$, i.e. rows in $A$ to be $0$.

As we can see, depending on what constraint we would like to place on each row, we can use a different $\ell_q$. In general, we refer to these mixed-norm constraints as $\ell_1/\ell_q$ norms. They are also known as block-sparse regularization, as they lead to entire rows of $A$ being set to $0$. [ 13 ] use $\ell_1/\ell_\infty$ regularization, while Argyriou et al. (2007) use a mixed $\ell_1/\ell_2$ norm. The latter is also known as group lasso and was first proposed by [ 14 ].

Argyriou et al. (2007) also show that the problem of optimizing the non-convex group lasso can be made convex by penalizing the trace norm of $A$, which forces $A$ to be low-rank and thereby constrains the column parameter vectors $a_{\cdot,1}, \ldots, a_{\cdot,t}$ to live in a low-dimensional subspace. [ 15 ] furthermore establish upper bounds for using the group lasso in multi-task learning.

As much as this block-sparse regularization is intuitively plausible, it is very dependent on the extent to which the features are shared across tasks. [ 16 ] show that if features do not overlap by much, $\ell_1/\ell_q$ regularization might actually be worse than element-wise $\ell_1$ regularization.

For this reason, [ 17 ] improve upon block-sparse models by proposing a method that combines block-sparse and element-wise sparse regularization. They decompose the task parameter matrix $A$ into two matrices $B$ and $S$ where $A = B + S$. $B$ is then enforced to be block-sparse using $\ell_1/\ell_\infty$ regularization, while $S$ is made element-wise sparse using lasso. Recently, [ 18 ] propose a distributed version of group-sparse regularization.

## Learning task relationships

While the group-sparsity constraint forces our model to only consider a few features, these features are largely used across all tasks. All of the previous approaches thus assume that the tasks used in multi-task learning are closely related. However, each task might not be closely related to all of the available tasks. In those cases, sharing information with an unrelated task might actually hurt performance, a phenomenon known as negative transfer.

Rather than sparsity, we would thus like to leverage prior knowledge indicating that some tasks are related while others are not. In this scenario, a constraint that enforces a clustering of tasks might be more appropriate. [ 19 ] suggest to impose a clustering constraint by penalizing both the norms of our task column vectors $a_{\cdot,1}, \ldots, a_{\cdot,t}$ as well as their variance with the following constraint:

$$\Omega = \|\bar{a}\|^2 + \frac{\lambda}{T}\sum_{t=1}^{T}\|a_{\cdot,t} - \bar{a}\|^2$$

where $\bar{a} = (\sum_{t=1}^{T} a_{\cdot,t})/T$ is the mean parameter vector. This penalty enforces a clustering of the task parameter vectors $a_{\cdot,1}, \ldots, a_{\cdot,t}$ towards their mean that is controlled by $\lambda$. They apply this constraint to kernel methods, but it is equally applicable to linear models.

A similar constraint for SVMs was also proposed by [ 20 ]. Their constraint is inspired by Bayesian methods and seeks to make all models close to some mean model. In SVMs, the loss thus trades off having a large margin for each SVM with being close to the mean model.

[ 21 ] make the assumptions underlying cluster regularization more explicit by formalizing a cluster constraint on $A$ under the assumption that the number of clusters $C$ is known in advance. They then decompose the penalty into three separate norms:

- A global penalty which measures how large our column parameter vectors are on average: $\Omega_{mean}(A) = \|\bar{a}\|^2$.
- A measure of between-cluster variance that measures how close to each other the clusters are: $\Omega_{between}(A) = \sum_{c=1}^{C} T_c \|\bar{a}_c - \bar{a}\|^2$ where $T_c$ is the number of tasks in the $c$-th cluster and $\bar{a}_c$ is the mean vector of the task parameter vectors in the $c$-th cluster.
- A measure of within-cluster variance that gauges how compact each cluster is: $\Omega_{within} = \sum_{c=1}^{C} \sum_{t \in J(c)} \|a_{\cdot,t} - \bar{a}_c\|$ where $J(c)$ is the set of tasks in the $c$-th cluster.

The final constraint then is the weighted sum of the three norms:

$$\Omega(A) = \lambda_1 \Omega_{mean}(A) + \lambda_2 \Omega_{between}(A) + \lambda_3 \Omega_{within}(A).$$

As this constraint assumes clusters are known in advance, they introduce a convex relaxation of the above penalty that allows to learn the clusters at the same time.

In another scenario, tasks might not occur in clusters but have an inherent structure. [ 22 ] extend the group lasso to deal with tasks that occur in a tree structure, while [ 23 ] apply it to tasks with graph structures.

While the previous approaches to modelling the relationship between tasks employ norm regularization, other approaches do so without regularization: [ 24 ] were the first ones who presented a task clustering algorithm using k-nearest neighbour, while [ 25 ] learn a common structure from multiple related tasks with an application to semi-supervised learning.

Much other work on learning task relationships for multi-task learning uses Bayesian methods:
[ 26 ] propose a Bayesian neural network for multi-task learning by placing a prior on the model parameters to encourage similar parameters across tasks. [ 27 ] extend Gaussian processes (GP) to MTL by inferring parameters for a shared covariance matrix. As this is computationally very expensive, they adopt a sparse approximation scheme that greedily selects the most informative examples. [ 28 ] also use GP for MTL by assuming that all models are sampled from a common prior.

[ 29 ] place a Gaussian as a prior distribution on each task-specific layer. In order to encourage similarity between different tasks, they propose to make the mean task-dependent and introduce a clustering of the tasks using a mixture distribution. Importantly, they require task characteristics that define the clusters and the number of mixtures to be specified in advance.

Building on this, [ 30 ] draw the distribution from a Dirichlet process and enable the model to learn the similarity between tasks as well as the number of clusters. They then share the same model among all tasks in the same cluster. [ 31 ] propose a hierarchical Bayesian model, which learns a latent task hierarchy, while [ 32 ] use a GP-based regularization for MTL and extend a previous GP-based approach to be more computationally feasible in larger settings.

Other approaches focus on the online multi-task learning setting: [ 33 ] adapt some existing methods such as the approach by Evgeniou et al. (2005) to the online setting. They also propose a MTL extension of the regularized Perceptron, which encodes task relatedness in a matrix. They use different forms of regularization to bias this task relatedness matrix, e.g. the closeness of the task characteristic vectors or the dimension of the spanned subspace. Importantly, similar to some earlier approaches, they require the task characteristics that make up this matrix to be provided in advance. [ 34 ] then extend the previous approach by learning the task relationship matrix.

[ 35 ] assume that tasks form disjoint groups and that the tasks within each group lie in a low-dimensional subspace. Within each group, tasks share the same feature representation whose parameters are learned jointly together with the group assignment matrix using an alternating minimization scheme. However, a total disjointness between groups might not be the ideal way, as the tasks might still share some features that are helpful for prediction.

[ 36 ] in turn allow two tasks from different groups to overlap by assuming that there exist a small number of latent basis tasks. They then model the parameter vector $a_t$ of every actual task $t$ as a linear combination of these: $a_t = Ls_t$ where $L \in \mathrm{R}^{k \times d}$ is a matrix containing the parameter vectors of $k$ latent tasks, while $s_t \in \mathrm{R}^k$ is a vector containing the coefficients of the linear combination. In addition, they constrain the linear combination to be sparse in the latent tasks; the overlap in the sparsity patterns between two tasks then controls the amount of sharing between these. Finally, [ 37 ] learn a small pool of shared hypotheses and then map each task to a single hypothesis.

# Recent work on MTL for Deep Learning

While many recent Deep Learning approaches have used multi-task learning -- either explicitly or implicitly -- as part of their model (prominent examples will be featured in the next section), they all employ the two approaches we introduced earlier, hard and soft parameter sharing. In contrast, only a few papers have looked at developing better mechanisms for MTL in deep neural networks.

## Deep Relationship Networks

In MTL for computer vision, approaches often share the convolutional layers, while learning task-specific fully-connected layers. [ 38 ] improve upon these models by proposing Deep Relationship Networks. In addition to the structure of shared and task-specific layers, which can be seen in Figure 3, they place matrix priors on the fully connected layers, which allow the model to learn the relationship between tasks, similar to some of the Bayesian models we have looked at before. This approach, however, still relies on a pre-defined structure for sharing, which may be adequate for well-studied computer vision problems, but prove error-prone for novel tasks.

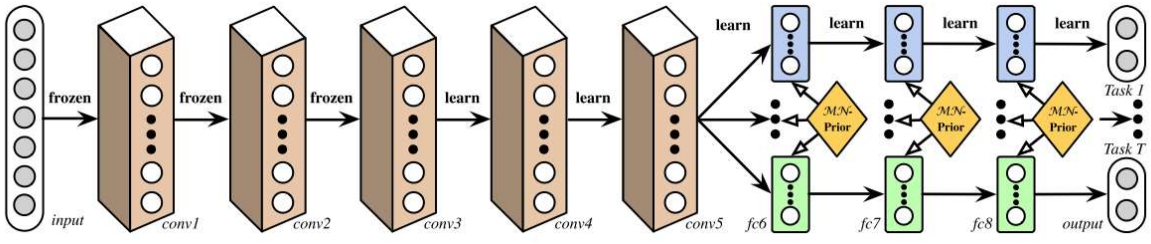


Figure 3: A Deep Relationship Network with shared convolutional and task-specific fully connected layers with matrix priors (Long and Wang, 2015).

## Fully-Adaptive Feature Sharing

Starting at the other extreme, [ 39 ] propose a bottom-up approach that starts with a thin network and dynamically widens it greedily during training using a criterion that promotes grouping of similar tasks. The widening procedure, which dynamically creates branches can be seen in Figure 4. However, the greedy method might not be able to

discover a model that is globally optimal, while assigning each branch to exactly one task does not allow the model to learn more complex interactions between tasks.

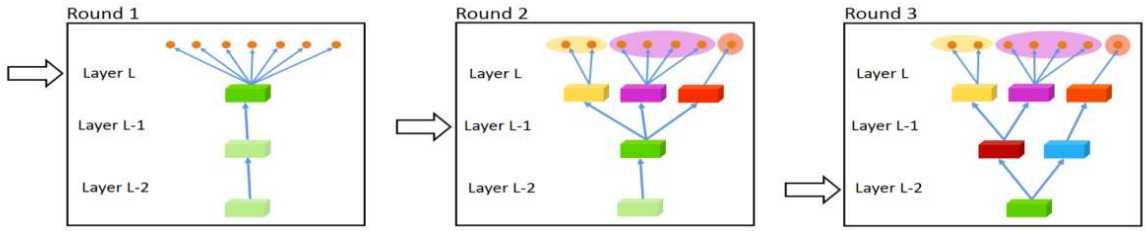


Figure 4: The widening procedure for fully-adaptive feature sharing (Lu et al., 2016).

## Cross-stitch Networks

[ 40 ] start out with two separate model architectures just as in soft parameter sharing. They then use what they refer to as cross-stitch units to allow the model to determine in what way the task-specific networks leverage the knowledge of the other task by learning a linear combination of the output of the previous layers. Their architecture can be seen in Figure 5, in which they only place cross-stitch units after pooling and fully-connected layers.

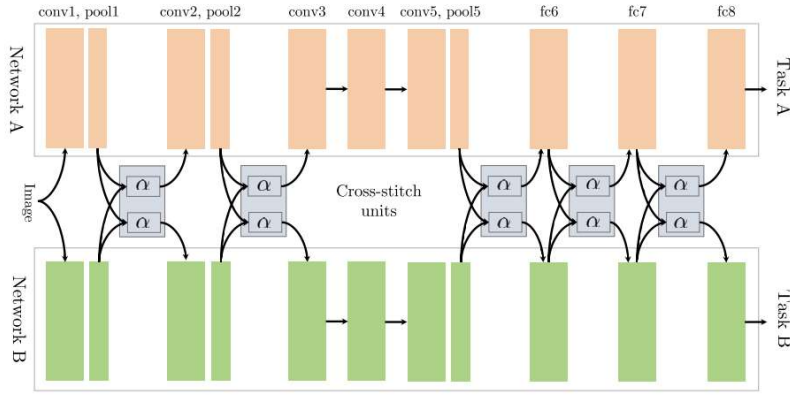


Figure 5: Cross-stitch networks for two tasks (Misra et al., 2016).

## Low supervision

In contrast, in natural language processing (NLP), recent work focused on finding better task hierarchies for multi-task learning: [ 41 ] show that low-level tasks, i.e. NLP tasks typically used for preprocessing such as part-of-speech tagging and named entity recognition, should be supervised at lower layers when used as auxiliary task.

## A Joint Many-Task Model

Building on this finding, [ 42 ] pre-define a hierarchical architecture consisting of several NLP tasks, which can be seen in Figure 6, as a joint model for multi-task learning.

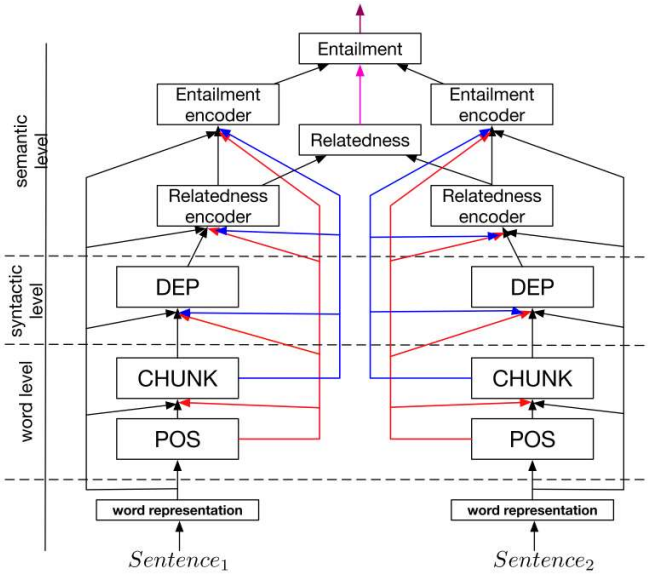


Figure 6: A Joint Many-Task Model (Hashimoto et al., 2016).

## Weighting losses with uncertainty

Instead of learning the structure of sharing, [ 43 ] take a orthogonal approach by considering the uncertainty of each task. They then adjust each task's relative weight in the cost function by deriving a multi-task loss function based on maximizing the Gaussian likelihood with task-dependant uncertainty. Their architecture for per-pixel depth regression, semantic and instance segmentation can be seen in Figure 7.

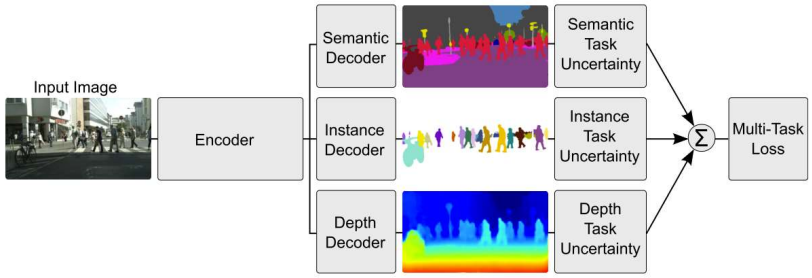


Figure 7: Uncertainty-based loss function weighting for multi-task learning (Kendall et al., 2017).

## Tensor factorisation for MTL

More recent work seeks to generalize existing approaches to MTL to Deep Learning: [ 44 ] generalize some of the previously discussed matrix factorisation approaches using tensor factorisation to split the model parameters into shared and task-specific parameters for every layer.

## Sluice Networks

Finally, we propose Sluice Networks [ 45 ], a model that generalizes Deep Learning-based MTL approaches such as hard parameter sharing and cross-stitch networks, block-sparse regularization approaches, as well as recent NLP approaches that create a task hierarchy. The model, which can be seen in Figure 8, allows to learn what layers and subspaces should be shared, as well as at what layers the network has learned the best representations of the input sequences.

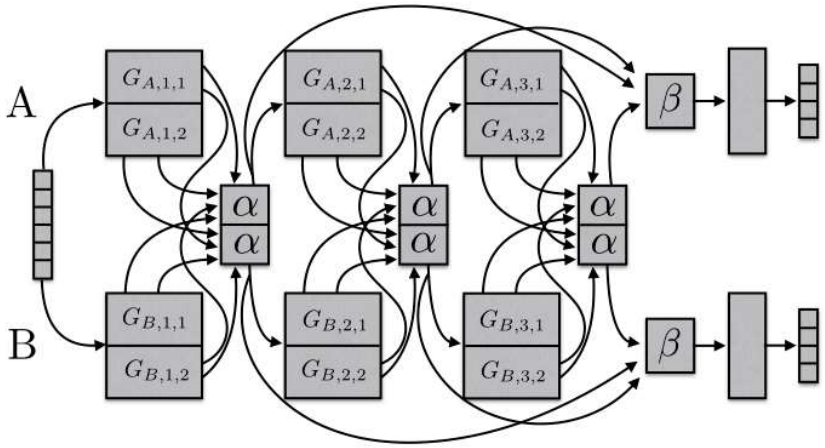


Figure 8: A sluice network for two tasks (Ruder et al., 2017).

### What should I share in my model?

Having surveyed these recent approaches, let us now briefly summarize and draw a conclusion on what to share in our deep MTL models. Most approaches in the history of MTL have focused on the scenario where tasks are drawn from the same distribution (Baxter, 1997). While this scenario is beneficial for sharing, it does not always hold. In order to develop robust models for MTL, we thus have to be able to deal with unrelated or only loosely related tasks.

While early work in MTL for Deep Learning has pre-specified which layers to share for each task pairing, this strategy does not scale and heavily biases MTL architectures. Hard parameter sharing, a technique that was originally proposed by Caruana (1996), is still the norm 20 years later. While useful in many scenarios, hard parameter sharing quickly breaks down if tasks are not closely related or require reasoning on different levels. Recent approaches have thus looked towards *learning* what to share and generally outperform hard parameter sharing. In addition, giving our models the capacity to learn a task hierarchy is helpful, particularly in cases that require different granularities.

As mentioned initially, we are doing MTL as soon as we are optimizing more than one loss function. Rather than constraining our model to compress the knowledge of all tasks into the same parameter space, it is thus helpful to draw on the advances in MTL that we have discussed and enable our model to learn how the tasks should interact with each other.

## Auxiliary tasks

MTL is a natural fit in situations where we are interested in obtaining predictions for multiple tasks at once. Such scenarios are common for instance in finance or economics forecasting, where we might want to predict the value of many possibly related indicators, or in bioinformatics where we might want to predict symptoms for multiple diseases simultaneously. In scenarios such as drug discovery, where tens or hundreds of active compounds should be predicted, MTL accuracy increases continuously with the number of tasks (Ramsundar et al., 2015).

In most situations, however, we only care about performance on one task. In this section, we will thus look at how we can find a suitable auxiliary task in order to still reap the benefits of multi-task learning.

### Related task

Using a related task as an auxiliary task for MTL is the classical choice. To get an idea what a related task can be, we will present some prominent examples. Caruana (1998) uses tasks that predict different characteristics of the road as auxiliary tasks for predicting the steering direction in a self-driving car; [ 46 ] use head pose estimation and facial attribute inference as auxiliary tasks for facial landmark detection; [ 47 ] jointly learn query classification and web search;

Girshick (2015) jointly predicts the class and the coordinates of an object in an image; finally, [ 48 ] jointly predict the phoneme duration and frequency profile for text-to-speech.

### Adversarial

Often, labeled data for a related task is unavailable. In some circumstances, however, we have access to a task that is *opposite* of what we want to achieve. This data can be leveraged using an adversarial loss, which does not seek to minimize but maximize the training error using a gradient reversal layer. This setup has found recent success in domain adaptation [ 49 ]. The adversarial task in this case is predicting the domain of the input; by reversing the gradient of the adversarial task, the adversarial task loss is maximized, which is beneficial for the main task as it forces the model to learn representations that cannot distinguish between domains.

### Hints

As mentioned before, MTL can be used to learn features that might not be easy to learn just using the original task. An effective way to achieve this is to use hints, i.e. predicting the features as an auxiliary task. Recent examples of this strategy in the context of natural language processing are [ 50 ] who predict whether an input sentence contains a positive or negative sentiment word as auxiliary tasks for sentiment analysis and [ 51 ] who predict whether a name is present in a sentence as auxiliary task for name error detection.

### Focusing attention

Similarly, the auxiliary task can be used to focus attention on parts of the image that a network might normally ignore. For instance, for learning to steer (Caruana, 1998) a single-task model might typically ignore lane markings as these make up only a small part of the image and are not always present. Predicting lane markings as auxiliary task, however, forces the model to learn to represent them; this knowledge can then also be used for the main task. Analogously, for facial recognition, one might learn to predict the location of facial landmarks as auxiliary tasks, since these are often distinctive.

### Quantization smoothing

For many tasks, the training objective is quantized, i.e. while a continuous scale might be more plausible, labels are available as a discrete set. This is the case in many scenarios that require human assessment for data gathering, such as predicting the risk of a disease (e.g. low/medium/high) or sentiment analysis (positive/neutral/negative). Using less quantized auxiliary tasks might help in these cases, as they might be learned more easily due to their objective being smoother.

### Predicting inputs

In some scenarios, it is impractical to use some features as inputs as they are unhelpful for predicting the desired objective. However, they might still be able to guide the learning of the task. In those cases, the features can be used as outputs rather than inputs. [ 52 ] present several problems where this is applicable.

### Using the future to predict the present

In many situations, some features only become available *after* the predictions are supposed to be made. For instance, for self-driving cars, more accurate measurements of obstacles and lane markings can be made once the car is passing them. Caruana (1998) also gives the example of pneumonia prediction, after which the results of additional medical trials will be available. For these examples, the additional data cannot be used as features as it will not be available as input at runtime. However, it can be used as an auxiliary task to impart additional knowledge to the model during training.

### Representation learning

The goal of an auxiliary task in MTL is to enable the model to learn representations that are shared or helpful for the main task. All auxiliary tasks discussed so far do this implicitly: They are closely related to the main task, so that learning them likely allows the model to learn beneficial representations. A more explicit modelling is possible, for instance by employing a task that is known to enable a model to learn transferable representations. The language modelling objective as employed by Cheng et al. (2015) and [ 53 ] fulfils this role. In a similar vein, an autoencoder objective can also be used as an auxiliary task.

### What auxiliary tasks are helpful?

In this section, we have discussed different auxiliary tasks that can be used to leverage MTL even if we only care about one task. We still do not know, though, what auxiliary task will be useful in practice. Finding an auxiliary task is largely based on the assumption that the auxiliary task should be related to the main task in some way and that it should be helpful for predicting the main task.

However, we still do not have a good notion of when two tasks should be considered similar or related. Caruana (1998) defines two tasks to be similar if they use the same features to make a decision. Baxter (2000) argues only theoretically that related tasks share a common optimal hypothesis class, i.e. have the same inductive bias. [ 54 ] propose that two tasks are $\mathcal{F}$-related if the data for both tasks can be generated from a fixed probability distribution using a set of transformations $\mathcal{F}$. While this allows to reason over tasks where different sensors collect data for the same classification problem, e.g. object recognition with data from cameras with different angles and lighting conditions, it is not applicable to tasks that do not deal with the same problem. Xue et al. (2007) finally argue that two tasks are similar if their classification boundaries, i.e. parameter vectors are close.

In spite of these early theoretical advances in understanding task relatedness, we have not made much recent progress towards this goal. Task similarity is not binary, but resides on a spectrum. More similar tasks should help more in MTL, while less similar tasks should help less. Allowing our models to learn what to share with each task might allow us to temporarily circumvent the lack of theory and make better use even of only loosely related tasks. However, we also need to develop a more principled notion of task similarity with regard to multi-task learning in order to know which tasks we should prefer.

Recent work [ 55 ] have found auxiliary tasks with compact and uniform label distributions to be preferable for sequence tagging problems in NLP, which we have confirmed in experiments (Ruder et al., 2017). In addition, gains have been found to be more likely for main tasks that quickly plateau with non-plateauing auxiliary tasks [ 56 ].

These experiments, however, have so far been limited in scope and recent findings only provide the first clues towards a deeper understanding of multi-task learning in neural networks.

## Conclusion

In this overview, I have reviewed both the history of literature in multi-task learning as well as more recent work on MTL for Deep Learning. While MTL is being more frequently used, the 20-year old hard parameter sharing paradigm is still pervasive for neural-network based MTL. Recent advances on learning what to share, however, are promising. At the same time, our understanding of tasks -- their similarity, relationship, hierarchy, and benefit for MTL -- is still limited and we need to learn more about them to gain a better understanding of the generalization capabilities of MTL with regard to deep neural networks.

I hope you found this overview helpful. If I made any error, missed a reference, or misrepresented some aspect, or if you would just like to share your thoughts, please leave a comment below.

## Printable version and citation

This blog post is also available as an article on arXiv, in case you want to refer to it later.

In case you found it helpful, consider citing the corresponding arXiv article as:
*Sebastian Ruder (2017). An Overview of Multi-Task Learning in Deep Neural Networks. arXiv preprint arXiv:1706.05098.*

## References


1. Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing. Proceedings of the 25th International Conference on Machine Learning - ICML '08, 20(1), 160–167. https://doi.org/10.1145/1390156.1390177
2. Deng, L., Hinton, G. E., & Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 8599–8603. https://doi.org/10.1109/ICASSP.2013.6639344
3. Girshick, R. (2015). Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1440–1448). https://doi.org/10.1109/iccv.2015.169
4. Ramsundar, B., Kearnes, S., Riley, P., Webster, D., Konerding, D., & Pande, V. (2015). Massively Multitask Networks for Drug Discovery. https://doi.org/https://arxiv.org/abs/1502.02072
5. Caruana, R. (1998). Multitask Learning. Autonomous Agents and Multi-Agent Systems, 27(1), 95–133. https://doi.org/10.1016/j.csl.2009.08.003
6. Caruana, R. "Multitask learning: A knowledge-based source of inductive bias," Proceedings of the Tenth International Conference on Machine Learning, 1993.
7. Baxter, J. (1997). A Bayesian/information theoretic model of learning to learn via multiple task sampling. Machine Learning, 28, 7–39. Retrieved from http://link.springer.com/article/10.1023/A:1007327622663
8. Duong, L., Cohn, T., Bird, S., & Cook, P. (2015). Low Resource Dependency Parsing: Cross-lingual Parameter Sharing in a Neural Network Parser. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers), 845–850.
9. Yang, Y., & Hospedales, T. M. (2017). Trace Norm Regularised Deep Multi-Task Learning. In Workshop track - ICLR 2017. Retrieved from http://arxiv.org/abs/1606.04038
10. Abu-Mostafa, Y. S. (1990). Learning from hints in neural networks. Journal of Complexity, 6(2), 192–198. https://doi.org/10.1016/0885-064X(90)90006-Y
11. Baxter, J. (2000). A Model of Inductive Bias Learning. Journal of Artificial Intelligence Research, 12, 149–198.
12. Argyriou, A., & Pontil, M. (2007). Multi-Task Feature Learning. In Advances in Neural Information Processing Systems. http://doi.org/10.1007/s10994-007-5040-8
13. C.Zhang and J.Huang. Model selection consistency of the lasso selection in high-dimensional linear regression. Annals of Statistics, 36:1567–1594, 2008

14. Yuan, Ming, and Yi Lin. "Model selection and estimation in regression with grouped variables." Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68.1 (2006): 49–67.

15. Lounici, K., Pontil, M., Tsybakov, A. B., & van de Geer, S. (2009). Taking Advantage of Sparsity in Multi-Task Learning. Stat, (1). Retrieved from http://arxiv.org/pdf/0903.1468

16. Negahban, S., & Wainwright, M. J. (2008). Joint support recovery under high-dimensional scaling : Benefits and perils of $\ell_{1,\infty}$-regularization. Advances in Neural Information Processing Systems, 1161–1168.

17. Jalali, A., Ravikumar, P., Sanghavi, S., & Ruan, C. (2010). A Dirty Model for Multi-task Learning. Advances in Neural Information Processing Systems. Retrieved from https://papers.nips.cc/paper/4125-a-dirty-model-for-multi-task-learning.pdf

18. Liu, S., Pan, S. J., & Ho, Q. (2016). Distributed Multi-task Relationship Learning. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS) (pp. 751–760). Retrieved from http://arxiv.org/abs/1612.04022

19. Evgeniou, T., Micchelli, C., & Pontil, M. (2005). Learning multiple tasks with kernel methods. Journal of Machine Learning Research, 6, 615–637. Retrieved from http://discovery.ucl.ac.uk/13423/

20. Evgeniou, T., & Pontil, M. (2004). Regularized multi-task learning. International Conference on Knowledge Discovery and Data Mining, 109. https://doi.org/10.1145/1014052.1014067

21. Jacob, L., Vert, J., Bach, F. R., & Vert, J. (2009). Clustered Multi-Task Learning: A Convex Formulation. Advances in Neural Information Processing Systems 21, 745–752. Retrieved from http://eprints.pascal-network.org/archive/00004705/%5Cnhttp://papers.nips.cc/paper/3499-clustered-multi-task-learning-a-convex-formulation.pdf

22. Kim, S., & Xing, E. P. (2010). Tree-Guided Group Lasso for Multi-Task Regression with Structured Sparsity. 27th International Conference on Machine Learning, 1–14. https://doi.org/10.1214/12-AOAS549

23. Chen, X., Kim, S., Lin, Q., Carbonell, J. G., & Xing, E. P. (2010). Graph-Structured Multi-task Regression and an Efficient Optimization Method for General Fused Lasso, 1–21. https://doi.org/10.1146/annurev.arplant.56.032604.144204

24. Thrun, S., & O'Sullivan, J. (1996). Discovering Structure in Multiple Learning Tasks: The TC Algorithm. Proceedings of the Thirteenth International Conference on Machine Learning, 28(1), 5–5. Retrieved from http://scholar.google.com/scholar?cluster=9560540185077238 32&hl=en

25. Ando, R. K., & Tong, Z. (2005). A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. Journal of Machine Learning Research, 6, 1817–1853.

26. Heskes, T. (2000). Empirical Bayes for Learning to Learn. Proceedings of the Seventeenth International Conference on Machine Learning, 367–364.

27. Lawrence, N. D., & Platt, J. C. (2004). Learning to learn with the informative vector machine. Twenty-First International Conference on Machine Learning - ICML '04, 65. https://doi.org/10.1145/1015330.1015382

28. Yu, K., Tresp, V., & Schwaighofer, A. (2005). Learning Gaussian processes from multiple tasks. Proceedings of the International Conference on Machine Learning (ICML), 22, 1012–1019. https://doi.org/10.1145/1102351.1102479

29. Bakker, B., & Heskes, T. (2003). Task Clustering and Gating for Bayesian Multitask Learning. Journal of Machine Learning Research, 1(1), 83–99. https://doi.org/10.1162/153244304322765658

30. Xue, Y., Liao, X., Carin, L., & Krishnapuram, B. (2007). Multi-Task Learning for Classification with Dirichlet Process Priors. Journal of Machine Learning Research, 8, 35–63.

31. Daumé III, H. (2009). Bayesian multitask learning with latent hierarchies, 135–142. Retrieved from http://dl.acm.org.sci-hub.io/citation.cfm?id=1795131

32. Zhang, Y., & Yeung, D. (2010). A Convex Formulation for Learning Task Relationships in Multi-Task Learning. Uai, 733–442.

33. Cavallanti, G., Cesa-Bianchi, N., & Gentile, C. (2010). Linear Algorithms for Online Multitask Classification. Journal of Machine Learning Research, 11, 2901–2934.

34. Saha, A., Rai, P., Daumé, H., & Venkatasubramanian, S. (2011). Online learning of multiple tasks and their relationships. Journal of Machine Learning Research, 15, 643–651. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84862275213&partnerID=tZOtx3y1

35. Kang, Z., Grauman, K., & Sha, F. (2011). Learning with whom to share in multi-task feature learning. Proceedings of the 28th International Conference on Machine Learning, (4), 4–5. Retrieved from http://machinelearning.wustl.edu/mlpapers/paper *files/ICML2011Kang*344.pdf

36. Kumar, A., & Daumé III, H. (2012). Learning Task Grouping and Overlap in Multi-task Learning. Proceedings of the 29th International Conference on Machine Learning, 1383–1390.

37. Crammer, K., & Mansour, Y. (2012). Learning Multiple Tasks Using Shared Hypotheses. Neural Information Processing Systems (NIPS), 1484–1492

38. Long, M., & Wang, J. (2015). Learning Multiple Tasks with Deep Relationship Networks. arXiv Preprint arXiv:1506.02117. Retrieved from http://arxiv.org/abs/1506.02117

39. Lu, Y., Kumar, A., Zhai, S., Cheng, Y., Javidi, T., & Feris, R. (2016). Fully-adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification. Retrieved from http://arxiv.org/abs/1611.05377

40. Misra, I., Shrivastava, A., Gupta, A., & Hebert, M. (2016). Cross-stitch Networks for Multi-task Learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/CVPR.2016.433

41. Søgaard, A., & Goldberg, Y. (2016). Deep multi-task learning with low level tasks supervised at lower layers. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, 231–235.

42. Hashimoto, K., Xiong, C., Tsuruoka, Y., & Socher, R. (2016). A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. arXiv Preprint arXiv:1611.01587. Retrieved from http://arxiv.org/abs/1611.01587

43. Kendall, A., Gal, Y., & Cipolla, R. (2017). Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. Retrieved from http://arxiv.org/abs/1705.07115

44. Yang, Y., & Hospedales, T. (2017). Deep Multi-task Representation Learning: A Tensor Factorisation Approach. In ICLR 2017. https://doi.org/10.1002/joe.20070

45. Ruder, S., Bingel, J., Augenstein, I., & Søgaard, A. (2017). Sluice networks: Learning what to share between loosely related tasks. Retrieved from http://arxiv.org/abs/1705.08142

46. Zhang, Z., Luo, P., Loy, C. C., & Tang, X. (2014). Facial Landmark Detection by Deep Multi-task Learning. In European Conference on Computer Vision (pp. 94–108). https://doi.org/10.1007/978-3-319-10599-4_7

47. Liu, X., Gao, J., He, X., Deng, L., Duh, K., & Wang, Y.-Y. (2015). Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval. Naacl-2015, 912–921.

48. Arık, S. Ö., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., … Shoeybi, M. (2017). Deep Voice: Real-time Neural Text-to-Speech. In ICML 2017.

49. Ganin, Y., & Lempitsky, V. (2015). Unsupervised Domain Adaptation by Backpropagation. In Proceedings of the 32nd International Conference on Machine Learning. (Vol. 37).

50. Yu, J., & Jiang, J. (2016). Learning Sentence Embeddings with Auxiliary Tasks for Cross-Domain Sentiment Classification. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP2016), 236–246. Retrieved from http://www.aclweb.org/anthology/D/D16/D16-1023.pdf

51. Cheng, H., Fang, H., & Ostendorf, M. (2015). Open-Domain Name Error Detection using a Multi-Task RNN. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (pp. 737–746).

52. Caruana, R., & Sa, V. R. de. (1997). Promoting poor features to supervisors: Some inputs work better as outputs. Advances in Neural Information Processing Systems 9: Proceedings of The 1996 Conference, 9, 389. Retrieved from http://scholar.google.com/scholar?start=20&q=author:%22Rich+Caruana%22&hl=en#6

53. Rei, M. (2017). Semi-supervised Multitask Learning for Sequence Labeling. In ACL 2017.

54. Ben-David, S., & Schuller, R. (2003). Exploiting task relatedness for multiple task learning. Learning Theory and Kernel Machines, 567–580. https://doi.org/10.1007/978-3-540-45167-9_41

55. Alonso, H. M., & Plank, B. (2017). When is multitask learning effective? Multitask learning for semantic sequence prediction under varying data conditions. In EACL. Retrieved from http://arxiv.org/abs/1612.02251

56. Bingel, J., & Søgaard, A. (2017). Identifying beneficial task relations for multi-task learning in deep neural networks. In EACL. Retrieved from http://arxiv.org/abs/1702.08303