Transfer of Knowledge Across Tasks

Ricardo Vilalta and Mikhail M.Meskhi

1 Introduction

We have mentioned before that learning should not be viewed as an isolated task that starts from scratch with every new problem. Instead, a learning algorithm should exhibit the ability to adapt through a mechanism dedicated to transfer knowledge gathered from previous experience [41, 40]. The problem of how to transfer knowledge across tasks is central to the field of metalearning, and is also referred to as *learning to learn* or *transfer learning*. Here, knowledge can be understood as a collection of patterns observed across tasks. As an example, one view of the nature of patterns across tasks is that of invariant transformations. For example, image recognition of a target object is simplified if the object is invariant under rotation, translation, scaling, etc. A learning system should be able to recognize a target object on an image even if previous images show the object in different sizes or from different angles. We view transfer learning as the study of how to improve learning by detecting, extracting, and exploiting knowledge across tasks.

In this chapter, we take a look at various approaches to implement learning systems armed with the ability to transfer knowledge across tasks. We focus our description by responding to two questions: What can be transferred across tasks? and what learning architectures have been commonly used for transfer learning? We also present developments in the theoretical aspects of learning to learn. Our focus is on supervised learning; other work can be found in fields such as unsupervised learning [8] and reinforcement learning [39].

2 Background, Terminology, and Notation

We focus on the task of supervised learning or classification where we are given the task of inducing a model from a sample $\{(x, y)\}$, where vector x is an instance (feature vector) of the input space $\mathcal{X}$, and y is an instance of the output space $\mathcal{Y}$. The sample contains independently and identically

distributed (i.i.d.) examples that come from a fixed but unknown joint probability distribution, $P(X = \mathbf{x}, Y = y)$, in the input-output space $\mathcal{X} \times \mathcal{Y}$. The output of the learning algorithm is a hypothesis (i.e., model, function) $h(X)$ mapping the input space to the output space, $h : \mathcal{X} \rightarrow \mathcal{Y}$. Function h comes from a space of hypotheses $\mathcal{H}$. The idea is to search for the hypothesis that minimizes the expectation of a loss function $L(Y, h(X))$, a.k.a. the risk: $R(h) = E_{P(X,Y)}[L(Y, h(X)]$.

2.1 When is transfer learning applicable?

In transfer learning, we assume the existence of a *source domain* $\mathcal{D}_S$ from which we can leverage experience to generate an accurate model on *target domain* $\mathcal{D}_T$. Ultimately, the main goal is to induce an accurate model $h_T(X)$ on the target domain. The need to transfer knowledge across domains is prompted by the change of at least one of the following elements between domains: $\{\mathcal{X}, P(X), \mathcal{Y}, P(Y|X)\}$[1] Let us follow a concrete case study to understand these elements. If we assume the learning task of inducing a model to predict disease from laboratory tests in a medical facility, the first element refers to the case where the feature space differs, $\mathcal{X}_S \neq \mathcal{X}_T$, as it would happen if two medical centers rely on different laboratory tests. The second element refers to the marginal distribution $P(X) = \int_Y P(X, Y) d_Y$; it can be illustrated as two medical centers having populations of patients exhibiting differences in demographics, $P_S(X) \neq P_T(X)$. The third element refers to the output or class-label space; this would correspond to the case where two medical centers aim at predicting different diseases, $\mathcal{Y}_S \neq \mathcal{Y}_T$. The last element, the class posterior probability, refers to the scenario where, due to environmental, genetic, or other factors, disease is manifested differently across two medical centers, $P_S(Y|X) \neq P_T(Y|X)$. Transfer learning is justified when one or more of these elements differs across the source and target domains.

Remember that the emphasis is always placed on the target domain $\mathcal{D}_T$, corresponding to the task at hand. The main objective is to induce a model $h_T(X)$ for the target domain; when building the model, one can exploit knowledge from the source domain $\mathcal{D}_S$. A cautionary note is in place when the similarity between the source and target domains is poor; it may occur that an attempt to leverage information from the source domain leads to a loss of generalization performance on the target domain. This effect, also known as *negative transfer* [43], places a boundary on the potential benefits of adapting models to new domains.

2.2 Types of transfer learning

Different approaches are available to transfer knowledge across tasks [47]. A proposed taxonomy is shown in Figure 1. We use the term *representational*

[1] Each element will normally be referred with a subscript to differentiate between source and target domains, e.g., $\mathcal{X}_S$ and $\mathcal{X}_T$.

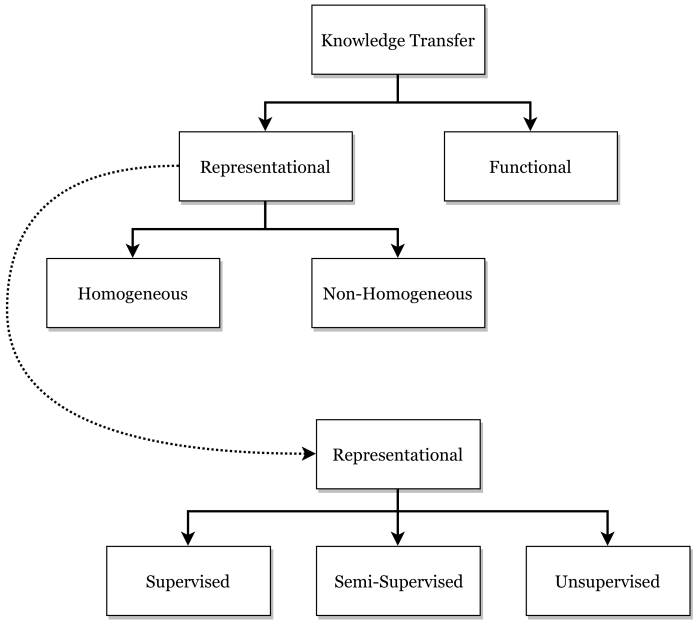

Fig. 1. A taxonomy of different approaches to knowledge transfer.

transfer to denote the case where the target and source models are trained at different times and the transfer takes place after the source model has already been trained; here, there is an explicit form of knowledge transferred into the target model. In contrast, we use the term *functional transfer* to denote the case where two or more models are trained simultaneously; in this case the models share (part of) their internal structure during learning (e.g., multi-task learning). When the transfer of knowledge is explicit, as is the case with representational transfer, further distinctions can be made. First, in terms of the the input or feature space, we can have that source and target domains share the same input space, also known as *homogeneous transfer* [47], or conversely, we can have that source and target domains do not share the same input space, also known as a *non-homogeneous transfer*. In terms of the availability of class labels, we denote as *Unsupervised transfer* the case where both source and target datasets contain no class labels. We denote as *Semi-Supervised transfer* the case where the source dataset contains labels, but the target dataset contains no class labels (or very few), and (e.g., domain adaptation). Finally, we denote as *Supervised transfer* the case where both source and target datasets contain class labels.

The need for transfer learning often points to target datasets with few or no class labels from which it is difficult to build accurate models. But it is important to note that transfer learning is also applicable under datasets with abundant class labels, where the goal is to improve over previous mistakes, further restricting the size of the hypothesis space.

2.3 What can be transferred?

While many different types of knowledge can be transferred across domains, popular techniques can be divided into three categories: *instance-based transfer learning, feature-based transfer learning, and parameter-based transfer learning*. We briefly review each technique in turn.

Instance-based transfer learning. The first type of knowledge transfer, instance-based transfer learning, aims at identifying instances on the source domain that seem to be *closer* to the distribution on the target domain. The idea in instance-based methods is to assign high weights to source examples occupying regions of high density in the target domain. A popular approach is known as the covariate shift [32, 37, 27, 38, 10]. The covariance-shift assumption is that one can build a model on the newly-weighted source sample and apply it directly to the target domain [23]. Specifically, we adopt the assumption that the difference in source $P_S(X,Y)$ and target $P_T(X,Y)$ distributions is due to a covariate shift, i.e., $P_S(X) \neq P_T(X)$, whereas, the conditional probabilities remain the same $P_S(Y|X) = P_T(Y|X)$. In this case, we can redefine the risk as $R(h) = E_{\sim P_T(X,Y)}[L(Y,h(X)]$, $R(h) = E_{\sim P_T(X,Y)}[\frac{P_T(X,Y)}{P_S(X,Y)}L(Y,h(X)]$, $R(h) = E_{\sim P_T(X,Y)}[\beta(X,Y)L(Y,h(X)]$. By obtaining the value of $\beta(X,Y)$ on every source instance X, we can minimize the risk on the target domain. A stringent requirement, however, is that source and target distributions must be close to each other.

Feature-based transfer learning. The second type of knowledge transfer, feature-based transfer learning, aims at finding a common representation where both source and target distributions overlap. Feature-based methods attempt to project source and target datasets into a latent feature space, where the covariate-shift assumption holds. A model is then built on the transformed space and used as the classifier on the target. Examples are structural corresponding learning [11], subspace alignment methods [4], among others.

Parameter-based transfer learning. The third type of knowledge transfer, parameter-based transfer learning, aims at generating a good set of initial parameters to expedite the model building phase on the target domain. As an illustration, we may perform an exhaustive search for the right model parameters on a source domain, where we can generate a set of prior distributions. Upon the arrival of a new target task, transfer learning obviates such exhaustive search; instead, we can generate a posterior distribution on

the target (using the source to obtain the priors) that would lead to finding a near-optimal set of target model parameters.

3 Learning Architectures in Transfer Learning

Many experiments in supervised learning have been reported within the neural network community, but other architectures have also played an important role. Besides neural networks, this section includes kernel methods and parametric Bayesian methods.

3.1 Transfer in neural networks

A learning paradigm amenable to testing the feasibility of knowledge transfer is that of neural networks. A neural network is capable of expressing flexible decision boundaries over the input space [21]; it is a nonlinear statistical model that applies to both regression and classification. In particular, for a neural network with one hidden layer, each output node computes the following function:

$$g_k(X = \mathbf{x}) = f(\sum_l w_{kl}\ f(\sum_i w_{li}x_i + w_{l0}) + w_{k0}) \tag{1}$$

where $\mathbf{x}$ is the input feature vector, $f(\cdot)$ is a nonlinear (e.g., sigmoid, ReLU) function, and x_i is a component of vector $\mathbf{x}$. Index i runs along the components of vector $\mathbf{x}$, index l runs along the number of intermediate functions (i.e., nonlinear transformations of the input features), and index k refers to the kth output node. The output is a nonlinear transformation of the intermediate functions. The learning process is limited to finding appropriate values for all weights $\{w\}$. The concepts described below are equally valid for *deep neural networks* [21] where there is more than just one hidden layer between the input and output nodes.

Neural networks have received much attention in the context of knowledge transfer because one can exploit the final set of weights of the *source* network (i.e., of the network obtained on a previous task) to initialize the set of weights corresponding to the *target* network (i.e., to the network corresponding to the current task). We describe different strategies to transfer knowledge between neural network models.

Functional transfer in neural networks. Most approaches to transfer learning in neural networks follow a representational approach, where some knowledge is explicitly transferred from the source network to the target network. But a functional approach is also popular, where several networks are combined into a single network architecture enabling different tasks to share the same hidden representation; this field is also known as *multi-task learning* [2]. As an illustration, Figure 2 shows two networks, one intended to classify

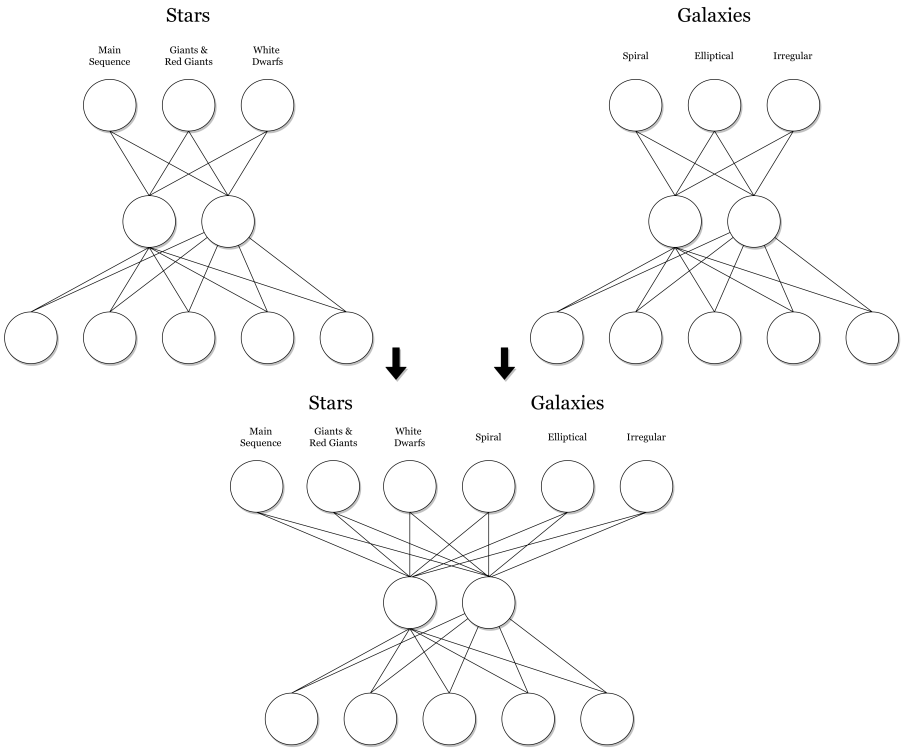

Fig. 2. One can combine tasks together into a single parallel multi-task problem; here, multiple luminous objects are identified in parallel using a common hidden layer.

stars, and the other galaxies, that can be combined into one single architecture where hidden nodes now capture patterns that are common across both domains.

Sharing part of the neural network structure. In general, many hybrid variations have been tried around the central idea of sharing a neural network structure, often by combining different forms of knowledge transfer. Examples include dividing the neural network into two parts: a common structure at the bottom of the network capturing a common task representation, and a set of upper structures each focused on learning a specific task [48]. Specifically, a source domain with an abundance of labeled examples can be exploited to generate a network model with high generalization performance. New target domains with limited training samples can re-use the bottom layers of the

source network, while simply adjusting the weights on the upper part of the target network [25, 48].

Searching for invariant transformations. An interesting example of an application of knowledge transfer in neural networks is the search for certain forms of invariant transformations. We mentioned before the importance of finding such transformations in the context of image recognition. As an illustration, suppose we have gathered images of a set of objects under different angles, brightness, location, etc. Let us assume our goal is to automatically learn to recognize an object in an image using as experience images containing the same object (albeit captured in different conditions). One way to proceed is to train a neural network to learn an invariant function σ. Function σ is trained with pairs of images generated under different conditions to identify when the images contain the same object. If the function is approximated with no error, one could perfectly predict the type of object contained in one image by simply applying σ over the current image and previous images containing several prototype objects. In practice, however, finding σ can be intractable and information about the shape of the invariant function (e.g., function slopes) can be used to improve the accuracy of the learner [41, 49].

Nested learning and k-shot learning. A general way to depict a metalearning algorithm is to divide its internal architecture into two main components: a base learner and a metalearner. The base learner works as is traditional in supervised learning, inducing a model from a set of labeled examples by searching for near-optimal model parameters on a specific task (or episode). The metalearner instead takes on the role of learning patterns (i.e., knowledge) across tasks to simplify the task of each base learner. This can be visualized as a double-loop architecture [45, 9], where the base learner iterates over a training set to learn model parameters under a fixed hypothesis space, in what is described as the *inner loop*, while concurrently, the metalearner iterates over different tasks to learn metaparameters under a family of hypothesis spaces, in what is described as the *outer loop* (see Fig. 3). This double-loop architecture has seen an explosion of different techniques and settings [18], particularly in the neural network community. A typical application is the n-way k-shot learning task, where the challenge is to train a (deep) neural network with very few examples; specifically, to induce an accurate model with only k examples for each of the n possible classes. This is only possible if the meta-learner has captured relevant patterns across multiple tasks. We briefly illustrate some instances of these ideas.

- **Learning similarity functions**. One form of metalearning uses the source task to learn a *similarity function* that can accurately predict if two objects belong to the same class [28, 15]. This is different from traditional supervised learning, where the classifier receives as input two examples (i.e., two feature vectors), and predicts if they belong to the same class or not. This verification problem can be exploited by transferring such

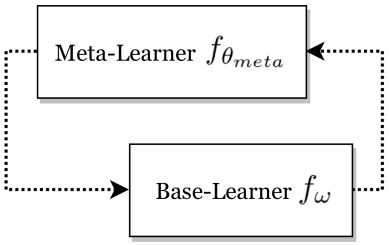

Fig. 3. The double-loop architecture in metalearning, where the base learner iterates over a training set to learn model parameters under a fixed hypothesis space, in what is described as the *inner loop*, while concurrently, the metalearner iterates over different tasks to learn metaparameters under a family of hypothesis spaces, in what is described as the *outer loop*.

similarity function to the target domain. In 1-shot learning, for example, the single labeled example on the target task can replace one matching element of the similarity function, while the other element corresponds to a target testing example. The nested learning framework can be effected by minimizing a loss over each task or episode corresponding to a specific target sample (inner loop), while improving on the similarity function across many learning tasks (outer loop) [46].

- **Learning with recurrent neural networks**. One advantage of the double-loop view of metalearning is that fixed update routines can be transformed into adaptable modules, amenable to learning. A typical framework for learning update rules is that of recurrent neural networks, particularly LSTMs (Long Short Term Memory), where the ability to remember past events provides feedback to improve the update mechanism itself [26]. As an illustration, a recurrent neural network can be designed with a double-loop architecture, where a search for model parameters on the base learner (optimizee) for a specific task, is guided by a metalearner (optimizer) in charge of learning the update rule itself after seeing several tasks [1, 33].

- **Bidirectional feedback between learner and metalearner**. One prominent line of research is to increase the interdependence between the base learner and the metalearner by adjusting the optimization process to ensure feedback is sent in both directions [29, 18, 19, 9]. Specifically, a

base learner can update its parameters θ' by relying on a global metaparameter θ (controlled by the metalearner) for parameter initialization. In the context of stochastic gradient descent (SGD), a single update step can be defined as follows:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{2}$$

where the second term on the right side of the equation is the gradient over the loss function on task $\mathcal{T}_i$. The update step above defines the inner loop (see the previous discussion), but notice the dependence on global parameter θ. The outer loop is effected when θ is updated after seeing several tasks:

$$\theta = \theta - \beta \sum_{\mathcal{T}_i} \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) \tag{3}$$

where metaparameter θ is based on the sum of local gradients. In effect, the metalearner provides an initial set of parameters on each task $\mathcal{T}_i$, to update θ_i in few steps [18, 19].

- **Memory-augmented neural networks**. Another interesting direction is to enhance neural networks to remember past events by adding memory components [22]. In transfer learning, this leads to models that remember past events, and can generalize to new tasks by leveraging past experience [35, 31], overcoming the *catastrophic forgetting* typical of deep networks. Memory becomes then an additional component to the neural network, with the capacity to store and retrieve representations relatively fast. This is critical in a k-shot learning scenario, where generalizing with few examples is difficult, requiring the storage of newly observed events. Here, the inner loop of metalearning is achieved by quickly retrieving instances for which proper generalization has not been reached, with an outer loop where the slow acquisition of patterns across tasks or episodes leads to robust and stable models.

3.2 Transfer in kernel methods

Kernel methods such as support vector machines (SVMs) have been extended to work on multi-task learning. Kernel methods look for a solution to the classification (or regression) problem using a discriminant function $g(\cdot)$ of the form:

$$g(X = \mathbf{x}) = \sum_j c_j k(\mathbf{x}_j, \mathbf{x}) \tag{4}$$

where $\{c_j\}$ is a set of real parameters, index j runs along the number of training examples, and k is a kernel function in a reproducing kernel Hilbert space [36].

Knowledge transfer can be effected using kernel methods by forcing the different hypotheses (corresponding to the different tasks) to share a common structure. As an illustration, consider the space of hypotheses made of hyperplanes, where every hypothesis is represented as $\mathbf{w} \cdot \mathbf{x}$ (i.e., as the inner product of $\mathbf{w}$ and $\mathbf{x}$). To employ the idea of having multiple tasks, we assume we have several datasets $\mathbf{T} = \{T_p\}_{p=1}^{n}$. Our goal is to produce hypotheses $\{h_p\}_{p=1}^{n}$ from $\mathbf{T}$ under the assumption that the tasks are related. The idea of task relatedness can be incorporated by modifying the space of hypotheses so that the weight vector is made of two components:

$$\mathbf{w}_p = \mathbf{w}_0 + \mathbf{v}_p, \quad 1 \leq p \leq n \tag{5}$$

where we assume all models share a common model $\mathbf{w}_0$, and the vectors $\mathbf{v}_j$ serve to model each particular task. In this case, we are in effect forcing all hypotheses to share a common component while also allowing for deviations from the common model [17][].

3.3 Transfer in parametric Bayesian models

One type of knowledge transfer uses a Bayesian model by computing the posterior probability of each class y given an input vector $\mathbf{x}$, $P(Y = y|X = \mathbf{x})$. For a fixed class y, Bayes theorem results in the following formula:

$$g(\mathbf{x}) = P(Y = y|X = \mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \tag{6}$$

where $P(y)$ is the prior probability of class y, $P(\mathbf{x}|y)$ is the likelihood of y with respect to $\mathbf{x}$ or the class-conditional probability, and $P(\mathbf{x})$ is the evidence [16]. Under this framework, a parameter-based transfer learning approach is to train a Bayesian learning algorithm on source domain D_S, resulting in a predictive model with parameter vector θ_S (parameter vector θ_S embeds the set of probabilities required to compute the posterior probabilities). For a new target domain D_T, we require that the new probability vector θ_T be similar to the previous one (i.e., $\theta_S \sim \theta_T$). To accomplish this we assume that each component parameter of θ_S and θ_T stems from a hyper-prior distribution. The degree of similarity between parameter components can be controlled by forcing the hyper-prior distribution to have small variance (corresponding to similar tasks) or large variance (corresponding to dissimilar tasks) [34, 14].

Transfer by Clustering. One approach to learning to learn consists of designing a learning algorithm that groups similar tasks into clusters. A new task is assigned to the most related cluster; knowledge transfer takes place when generalization exploits information about the cluster to which each task belongs. This idea of clustering similar tasks has also been pursued under a Bayesian approach. Essentially, each vector of hidden to output weights can be modeled as a mixture of Gaussians, where each Gaussian is in fact describing a cluster of tasks[3, 42].

4 A Theoretical Framework

Several studies have provided a theoretical analysis of the learning-to-learn paradigm. The aim is to understand the conditions under which a metalearner can provide good generalizations when embedded in an environment made of related tasks. Although the idea of knowledge transfer is normally made implicit in the analysis, it is clear that the metalearner extracts and exploits knowledge from every task to perform well on future tasks. Theoretical studies fall within a Bayesian model [5, 25] and a probably approximately correct (PAC) model [6, 30]. The idea is to find not only the right hypothesis h in a hypothesis space $\mathcal{H}$, $h \in \mathcal{H}$, but in addition to find the right hypothesis space $\mathcal{H}$ in a family of hypothesis spaces $\mathbb{H}$, $\mathcal{H} \in \mathbb{H}$.

Let us look at these studies more closely. We focus on the problem of bounding the number of examples needed to produce good generalizations when the learner faces a stream of tasks. Consider first that the goal of traditional learning is to find a hypothesis $h^* \in \mathcal{H}$ that minimizes a functional risk, $h^* = \arg\min_{h \in \mathcal{H}} R_\phi(h)$, where

$$R_\phi(h) = \int_{(\mathbf{x},y) \in \mathcal{X} \times \mathcal{Y}} L(h(\mathbf{x}), y) d\phi(\mathbf{x}, y) \tag{7}$$

The risk corresponds to the expected loss incurred by hypothesis h; $L(h(\mathbf{x}), y)$ is a particular loss function (e.g., zero-one loss) and the integral runs across the input-output space. We introduce new notation, ϕ, to denote the probability distribution over $\mathcal{X} \times \mathcal{Y}$ that indicates which examples are more likely to be seen for that particular task. Since we do not have access to all possible examples in the input-output space, we may choose to approximate the true risk with an empirical risk $\hat{R}_\phi(h)$. We do this by randomly sampling m examples according to ϕ to generate a training sample $T = \{(\mathbf{x}_j, y_j)\}_{j=1}^m$, where:

$$\hat{R}_\phi(h, T) = \frac{1}{m} \sum_{j=1}^{m} L(h(\mathbf{x}_j), y_j) \tag{8}$$

It has been formally shown that one can bound the true risk $R_\phi(h)$ as a function of the empirical risk $\hat{R}_\phi(h, T)$ if there exists a uniform bound for all $h \in \mathcal{H}$ on the probability of deviation between $R_\phi(h)$ and $\hat{R}_\phi(h, T)$ [44, 12]. Such bounds can be represented as a function of the Vapnik-Chervonenkis (VC) dimension of the hypothesis space $\mathcal{H}$, $\text{VC}(\mathcal{H})$. The VC dimension captures the degree of expressiveness or richness in delimiting flexible decision boundaries by the set of functions in $\mathcal{H}$; it provides an objective characterization of $\mathcal{H}$ [44]. Bounds for the deviation between $R_\phi(h)$ and $\hat{R}_\phi(h, T)$ take on the form

$$R_\phi(h) \leq \hat{R}_\phi(h, T) + g(m, \delta, \text{VC}(\mathcal{H})) \tag{9}$$

where function $g(\cdot)$ explicitly indicates an upper bound on the distance between the true risk and the empirical risk; the inequality is satisfied for all $h \in \mathcal{H}$ with probability $1 - \delta$.

4.1 The learning-to-learn scenario

Let us now consider the novelty brought about by the learning-to-learn scenario [6]). Here we assume the learner is embedded in a set of related tasks that share certain commonalities. In traditional learning, we assume a probability distribution ϕ that indicates which examples are more likely to be seen in such a task. Now we assume there is a metadistribution Φ over the space of all possible distributions $\{\phi\}$. In essence, Φ indicates which tasks are more likely to be found within the sequence of tasks faced by the metalearner (just as ϕ indicates which examples are more likely to be seen in such a task). As an example, if we were interested in classifying luminous objects on astronomical surveys, Φ may stand for a probability distribution that peaks over tasks that identify classes of astronomical objects. Given a family of hypothesis spaces $\mathbb{H}$, the goal of the metalearner is to find a hypothesis space $\mathcal{H}^* \in \mathbb{H}$ that minimizes a new functional risk, $\mathcal{H}^* = \arg\min_{\mathcal{H} \in \mathbb{H}} R_\Phi(\mathcal{H})$, where

$$R_\Phi(\mathcal{H}) = \int_{\phi \in \Phi} \inf_{h \in \mathcal{H}} R_\phi(h) d\Phi(\phi) \tag{10}$$

An expansion of the above formula gives:

$$R_\Phi(\mathcal{H}) = \int_{\phi \in \Phi} \inf_{h \in \mathcal{H}} \int_{(\mathbf{x},y) \in \mathcal{X} \times \mathcal{Y}} L(h(\mathbf{x}), y) d\phi(\mathbf{x}, y) d\Phi(\phi) \tag{11}$$

The new functional risk, $R_\Phi(\mathcal{H})$, represents the expected loss of the best possible hypothesis in each hypothesis space. The integral runs across all task distributions $\{\phi\}$, which are themselves distributed according to a metadistribution Φ. In practice, since we ignore the form of Φ, we need to draw samples $T_1, T_2, \ldots T_n$ to infer how tasks are distributed in our environment.

The advantage of working on a learning-to-learn scenario is that the learner accumulates experience after each new task. Such experience, here referred to as metaknowledge, is expected to result in more accurate models when the tasks share commonalities or patterns. The expectation is that as more tasks are observed, the number of examples required to attain accurate models (with high probability) decreases over time.

4.2 Bounds on generalization error for metalearners

Finding bounds on the generalization error for metalearners follows the same logic as that adopted in conventional learning theory. The idea is to formally show that it is possible to bound the new functional risk $R_\Phi(\mathcal{H})$ as a function of the empirical risk $\hat{R}_\Phi(\mathcal{H})$. Given a set of n samples $\mathbf{T} = \{T_p\}$, the empirical

risk is defined as the average of the best possible empirical error for each training sample T_p:

$$\hat{R}_{\Phi}(\mathcal{H}) = \frac{1}{n}\sum_{p=1}^{n} \inf_{h\in\mathcal{H}} \hat{R}_{\phi}(h, T_p) \tag{12}$$

The bound can be found if there exists a uniform bound for all $\mathcal{H} \in \mathbb{H}$ on the probability of deviation between $R_{\Phi}(\mathcal{H})$ and $\hat{R}_{\Phi}(\mathcal{H})$. In conventional learning theory these bounds are governed by the expressiveness of the family of hypotheses $\mathcal{H}$. Similarly, in the learning-to-learn scenario, bounds on generalization error are governed by the size of function classes associated with the family space $\mathbb{H}$. Specifically, one can guarantee that with probability $1-\delta$ (according to the choice of samples $\mathbf{T}$), all $\mathcal{H} \in \mathbb{H}$ will satisfy the following inequality:

$$R_{\Phi}(\mathcal{H}) \leq \hat{R}_{\Phi}(\mathcal{H}) + \epsilon \tag{13}$$

This holds if the number of tasks n is such that

$$n \geq \max\left\{\frac{256}{\epsilon^2}\log\frac{8\mathcal{C}(\frac{\epsilon}{32}, \Lambda_{\mathbb{H}})}{\delta}, \frac{64}{\epsilon^2}\right\} \tag{14}$$

and the number of examples m for each task is such that

$$m \geq \max\left\{\frac{256}{n\epsilon^2}\log\frac{8\mathcal{C}(\frac{\epsilon}{32}, \mathbf{\Lambda}^{\mathbf{n}}_{\mathbb{H}})}{\delta}, \frac{64}{\epsilon^2}\right\} \tag{15}$$

The theorem [6] introduces two new properties characterizing the family of hypothesis spaces $\mathbb{H}$, $\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}})$ and $\mathcal{C}(\epsilon, \mathbf{\Lambda}^{\mathbf{n}}_{\mathbb{H}})$. These functions measure the capacity of $\mathbb{H}$ in a way similar to how the VC dimension measures the capacity of $\mathcal{H}$. To provide continuity to our chapter we defer explanation of these properties to Appendix A. The bounds stated above simply show that to learn both a good hypothesis space $\mathcal{H} \in \mathbb{H}$ and a good hypothesis $h \in \mathcal{H}$, one needs a minimum number of both the number of tasks and the number of examples on each task. It is known that if ϵ and δ are fixed [6], the number of examples m needed on each task to attain an accurate model is such that

$$m = O\left(\frac{1}{n}\log\mathcal{C}(\epsilon, \mathbf{\Lambda}^{\mathbf{n}}_{\mathbb{H}})\right) \tag{16}$$

This indicates that the required number of examples on each task decreases as the number of tasks increases, in accordance with our expectations of the benefits gained when the learning algorithm has the capability of exploiting previous experience.

4.3 Other theoretical work

Bounds using algorithmic stability

The results described above can be improved if one makes certain assumptions [30]. To understand this we need to review the concept of algorithmic stability [13]. A learning algorithm is said to be uniformly β-stable if taking away one example from the training set does not modify the loss of the output hypothesis by more than β (for a fixed loss function). We update our definition of a metalearning algorithm as a function $\mathcal{A}(\mathbf{T})$ that outputs a hypothesis after looking at a sequence of samples $\mathbf{T} = \{T_p\}_{p=1}^{n}$. That is, we no longer talk about a hypothesis space, but of a single hypothesis that does well on all previous tasks. In that case, one can also think of a metalearning algorithm as being β'-stable if removing one sample from the set of samples $\mathbf{T}$ does not modify the loss of the output hypothesis by more than β'. Notice that parameter β' corresponds to the concept of stability across tasks, whereas parameter β is used to refer to stability across examples drawn from one task.

Given that $\mathcal{A}(\mathbf{T}) = h$ for a given set of samples $\mathbf{T}$, the new results show that for every environment Φ, with probability greater than $1 - \delta$ according to the selection of $\mathbf{T}$, the following inequality holds:

$$\forall \phi \; R_{\Phi}(h) \leq \frac{1}{n} \sum_{p=1}^{n} \hat{R}_{\phi_p}(h, T_p) + 2\beta' + (4n\beta' + m)\sqrt{\frac{\ln(1/\delta)}{2n}} + 2\beta \qquad (17)$$

where $\phi_p \in \Phi$ and $\hat{R}_{\phi_p}(h, T_p)$ is an estimation of the empirical loss of hypothesis h when the examples are drawn from sample T_p. The first term on the right-hand side of the inequality is then the average empirical loss of h on the set of tasks $\mathbf{T}$. It can be shown that the new bound is tighter than that of Section 4.2 (of course under the assumption of stability parameterized by β and β' on $\mathcal{A}(\mathbf{T}) = h$).

Bounds for Domain Adaptation

The context of domain adaptation leads to another set of interesting learning bounds [7]. Assume a source domain $\mathcal{D}_S$ where class labels abound, and a target domain $\mathcal{D}_T$, with few or no class labels. It is implicitly assumed that source and target domains must be *related*, with no mechanism to quantify the degree of *relatedness*. This can be helpful to understand how to bound the error of a model trained on the source domain, but applied to the target domain, where we assume the distribution over $\mathcal{X}$ has changed, i.e., $P_S(X) \neq P_T(X)$.

We begin by defining the error of a hypothesis h under a zero-one loss function as $R_{\phi}(h) = E_{(\mathbf{x},y)\sim\phi}[|h(\mathbf{x}) - y|]$, where we assume $\mathcal{Y} = \{-1, 1\}$. We refer to the source and target distributions as ϕ_S and ϕ_T, with the understanding that the only difference is in the marginal distributions $P_S(X) \neq P_T(X)$. It has been formally shown that the generalization error on the target domain can be bound as a function of three terms:

$$R_{\phi_T}(h) \leq R_{\phi_S}(h) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\phi_S, \phi_T) + \lambda \tag{18}$$

where the first term on the right side of the inequality simply refers to the generalization error on the source domain. The second term is a measure of the *relatedness* of the two distributions. Formally,

$$d_{\mathcal{H}\Delta\mathcal{H}}(\phi_S, \phi_T) = 2 \sup_{h,h' \in \mathcal{H}} |P_{\mathbf{x}\sim\phi_S}[h(\mathbf{x}) \neq h'(\mathbf{x})] - P_{\mathbf{x}\sim\phi_T}[h(\mathbf{x} \neq h'(\mathbf{x})]| \tag{19}$$

The goal is simply to capture the difference in the probability of disagreement between two hypotheses in the space of hypothesis $\mathcal{H}$. The last term λ refers to the combined error of an *ideal* hypothesis:

$$\lambda = R_{\phi_S}(h^*) + R_{\phi_T}(h^*) \tag{20}$$

where $h^* = \text{argmin}_{h \in \mathcal{H}} R_{\phi_S}(h) + R_{\phi_T}(h)$. The bound depends then on the *distance* between the source and target distributions, and on the existence of a hypothesis that can attain low generalization error on both source and target domains.

4.4 Bias vs variance in metalearning

As part of our theoretical study, we end by looking into the nature of the bias-variance dilemma in classification when immersed in a learning-to-learn framework. Let us first recall what the bias-variance dilemma states in traditional learning [24, 20]. The dilemma is based on the fact that the expected prediction error, or risk, can be decomposed into a bias and variance components.[2] Ideally we would like to have classifiers with both low bias and low variance, but these components are inversely related. On the one hand, simple classifiers encompass a small hypothesis space $\mathcal{H}$. Their small repertoire of functions produces high bias (since the hypothesis with lowest prediction error may lie far from the true target function) but low variance (since there are few hypotheses to choose from). On the other hand, increasing the size of $\mathcal{H}$ reduces the bias, but increases the variance. The large size of $\mathcal{H}$ normally allows for flexible decision boundaries (low bias), but the learning algorithm inevitably becomes sensitive to small variations in the data (high variance).

In the learning-to-learn framework, there is an equal need to find a balance in the size of the family of hypothesis spaces $\mathbb{H}$. A small $\mathbb{H}$ will exhibit low variance and high bias; here, unless we can find a good hypothesis space $\mathcal{H} \in \mathbb{H}$ with a small risk $R_{\Phi}(\mathcal{H})$, the best $\mathcal{H}$ may be far from the true hypothesis space. And just as in traditional learning, a large $\mathbb{H}$ will exhibit low bias but high variance, since a large number of available hypothesis spaces increases the

[2] A third component, the *irreducible error* or *Bayes error*, cannot be eliminated or traded [24].

chances of selecting one that simply accommodates to the idiosyncrasies of the training data. One main goal is to understand if learning the right family of hypothesis spaces $\mathbb{H}$ is inherently easier (or not) than learning the right hypothesis space $\mathcal{H}$.

Appendix A

Section 4.2 makes use of two properties characterizing the space of a family of hypothesis spaces $\mathbb{H}$, $\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}})$ and $\mathcal{C}(\epsilon, \mathbf{\Lambda^n_{\mathbb{H}}})$. These functions quantify the capacity of the space of a family of hypothesis spaces $\mathbb{H}$. We now explain the nature of these properties in more detail:[3]

Definition 1. For each $\mathcal{H} \in \mathbb{H}$, define a new function $\lambda_{\mathcal{H}}(\phi_i)$ by

$$\lambda_{\mathcal{H}}(\phi) = \inf_{h \in \mathcal{H}} R_{\phi}(h) \tag{21}$$

where $\lambda : \Phi \rightarrow [0, 1]$. In other words, function λ specifies the minimum error loss achieved after looking at every $h \in \mathcal{H}$ under distribution ϕ.

Definition 2. For the family of hypothesis spaces $\mathbb{H}$, define a new set $\Lambda_{\mathbb{H}}$ by

$$\Lambda_{\mathbb{H}} = \{\lambda_{\mathcal{H}} : \mathcal{H} \in \mathbb{H}\} \tag{22}$$

The set $\Lambda_{\mathbb{H}}$ contains all *different* functions according to Def. 1 within the space of a family of hypotheses $\mathbb{H}$. We can compute the expected difference in the minimum error loss for any two functions $\lambda_1, \lambda_2 \in \Lambda_{\mathbb{H}}$ as follows.

Definition 3. For any two functions $\lambda_1, \lambda_2 \in \Lambda_{\mathbb{H}}$, and a distribution Φ on the space of possible input-output distributions, define

$$D_{\Phi}(\lambda_1, \lambda_2) = \int_{\phi} |\lambda_1(\phi) - \lambda_2(\phi)| d\Phi(\phi) \tag{23}$$

Function D can be seen as the expected distance between two functions λ_1, λ_2. We now define the concept of an ϵ-cover as follows.

Definition 4. An ϵ-cover of $(\Lambda_{\mathbb{H}}, D_{\Phi})$ is a set $\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ such that for all $\lambda \in \Lambda_{\mathbb{H}}$, $D_{\Phi}(\lambda, \lambda_p) \leq \epsilon$ $(1 \leq p \leq n)$. Let $\mathcal{N}(\epsilon, \Lambda_{\mathbb{H}}, D_{\Phi})$ represent the size of the smallest ϵ-cover. We now define the capacity of $\Lambda_{\mathbb{H}}$ by

$$\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}}) = \sup_{\Phi} \mathcal{N}(\epsilon, \Lambda_{\mathbb{H}}, D_{\Phi}) \tag{24}$$

where the supremum runs over all probability distributions over $\mathcal{X} \times \mathcal{Y}$.

We can similarly define the second capacity $\mathcal{C}(\epsilon, \mathbf{\Lambda^n_{\mathbb{H}}})$. To begin, consider a sequence of n tasks that has been modeled with n hypotheses

[3] We follow Baxter's work [6] in different order and notation to simplify the explanation of the two properties characterizing $\mathbb{H}$.

$\mathbf{h} = (h_1, h_2, \cdots, h_n)$. We can compute the expected error loss across n tasks as follows:

$$\lambda^n_{\mathbf{h}}(\{\mathbf{x}, y\}) = \frac{1}{n}\sum_{p=1}^{n} L(h_p(\mathbf{x}), y) \tag{25}$$

Definition 5. For the space of a family of hypotheses $\mathbb{H}$, define a new set $\Lambda^n_{\mathbf{h}}$ by

$$\Lambda^n_{\mathbf{h}} = \{\lambda^n_{\mathbf{h}} : h_1, h_2, \cdots, h_n \in \mathcal{H}\} \tag{26}$$

The set $\Lambda^n_{\mathbf{h}}$ is a loss function class and as before it indicates how many *different* classes of functions (capturing the average error loss for a sequence of n hypotheses) are contained within the hypothesis space $\mathcal{H}$; the difference is that now we are comparing sets of n loss functions.

Definition 6. For the space of a family of hypotheses $\mathbb{H}$, define

$$\mathbf{\Lambda^n_{\mathbb{H}}} = \bigcup_{\mathcal{H}\in\mathbb{H}} \Lambda^n_{\mathbf{h}} \tag{27}$$

where $\mathbf{h} \subseteq \mathcal{H}$. The second capacity $\mathcal{C}(\epsilon, \mathbf{\Lambda^n_{\mathbb{H}}})$ is defined similarly to the first one but using a new distance function:

$$D^n_{\Phi}(\mathbf{h}, \mathbf{h}') = \int_{(\mathcal{X}\times\mathcal{Y})^n} |\lambda^n_{\mathbf{h}}(\{\mathbf{x}_i, y_i\}) - \lambda^n_{\mathbf{h}'}(\{\mathbf{x}_i, y_i\})| d\phi_1, d\phi_2, \cdots, d\phi_n \tag{28}$$

This brings us to the second capacity function:

$$\mathcal{C}(\epsilon, \mathbf{\Lambda^n_{\mathbb{H}}}) = \sup_{\phi} \mathcal{N}(\epsilon, \mathbf{\Lambda^n_{\mathbb{H}}}, D^n_{\Phi}) \tag{29}$$

where the supremum runs over all sequences of n probability distributions over $\mathcal{X} \times \mathcal{Y}$.

References

[1] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3988–3996, USA, 2016. Curran Associates Inc.

[2] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *Advances in neural information processing systems*, pages 41–48, 2007.

[3] B. Bakker and T. Heskes. Task Clustering and Gating for Bayesian Multitask Learning. *Journal of Machine Learning Research*, 4:83–999, 2003.

[4] F. Basura, A. Habrard, M. Sebban, and T. Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE International Conference on Computer Vision, ICCV*, pages 2960–2967, 2013.

[5] J. Baxter. Theoretical models of learning to learn. In S. Thrun and L. Pratt, editors, *Learning to Learn*, chapter 4, pages 71–94. Springer-Verlag, 1998.

[6] J. Baxter. A Model of Inductive Learning Bias. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[7] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Mach. Learn.*, 79(1-2):151–175, May 2010.

[8] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36, 2012.

[9] L. Bertinetto, J. F. Henriques, P. H. S. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[10] S. Bickel, M. Brückner, and T. Scheffer. Discriminative learning under covariate shift. *J. Mach. Learn. Res.*, 10:2137–2155, Dec. 2009.

[11] J. Blitzer, R. McDonald, and F. Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing, ACL*, pages 120–128, 2006.

[12] A. Blumer, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik Chervonenkis Dimension. *Journal of the ACM*, 36(1):929–965, 1989.

[13] O. Bousquet and A. Elisseeff. Stability and Generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[14] X. Cao, D. Wipf, F. Wen, and G. Duan. A practical transfer learning algorithm for face verification. In *International Conference on Computer Vision (ICCV)*, January 2013.

[15] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 539–546, Washington, DC, USA, 2005. IEEE Computer Society.

[16] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[17] T. Evgeniou and M. Pontil. Regularized multi-task learning. In *Tenth Conference on Knowledge Discovery and Data Mining*, 2004.

[18] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1126–1135. JMLR.org, 2017.

[19] C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 9537–9548, USA, 2018. Curran Associates Inc.

[20] S. Geman, E. Bienenstock, and R. Doursat. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, pages 1–58, 1992.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[22] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[23] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf. Covariate shift by kernel mean matching. *Dataset shift in machine learning*, 3(4):5, 2009.

[24] T. Hastie, R. Tbshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer 2nd. Edition, 2009.

[25] T. Heskes. Empirical Bayes for Learning to Learn. In *17th International Conference on Machine Learning*, pages 367–374. Morgan Kaufmann, San Francisco, CA, 2000.

[26] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *IN LECTURE NOTES ON COMP. SCI. 2130, PROC. INTL. CONF. ON ARTI NEURAL NETWORKS (ICANN-2001*, pages 87–94. Springer, 2001.

[27] T. Kanamori, S. Hido, and M. Sugiyama. A least-squares approach to direct importance estimation. *J. Mach. Learn. Res.*, 10:1391–1445, Dec. 2009.

[28] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *Who knows?*, 2015.

[29] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2113–2122, 2015.

[30] A. Maurer. Algorithmic Stability and Meta-Learning. *Journal of Machine Learning Research*, 6:967–994, 2005.

[31] T. Munkhdalai and H. Yu. Meta networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2554–2563, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[32] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

[33] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[34] M. T. Rosenstein, Z. Marx, and L. P. Kaelbling. To Transfer or Not To Transfer. In *Workshop at NIPS (Neural Information Processing Systems)*, 2005.

[35] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1842–1850. JMLR.org, 2016.

[36] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[37] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, Oct. 2000.

[38] M. Sugiyama, S. Nakajima, H. Kashima, P. V. Buenau, and M. Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in neural information processing systems*, pages 1433–1440, 2008.

[39] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[40] S. Thrun. Lifelong Learning Algorithms. In S. Thrun and L. Pratt, editors, *Learning to Learn*, pages 181–209. Kluwer Academic Publishers, MA., 1998.

[41] S. Thrun and T. Mitchell. Learning One More Thing. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pages 1217–1223, 1995.

[42] S. Thrun and J. OSullivan. Clustering learning tasks and the selective cross-task transfer of knowledge. In *Learning to learn*, pages 235–257. Springer, 1998.

[43] L. Torrey and J. Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.

[44] V. Vapnik. *The Nature of Statistical Leanring Theory*. Springer Verlag, New York, 1995.

[45] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2):77–95, Oct. 2002.

[46] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3637–3645, USA, 2016. Curran Associates Inc.

[47] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A Survey of Transfer Learning. *Journal of Big Data*, 3(1), 2016.

[48] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *arXiv e-prints*, page arXiv:1411.1792, Nov 2014.

[49] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep sets, 2017.