Meta-learning for drug discovery

A primer

Author:
Basile Dura

Supervisor:
Prudencio Tossou

Mila supervisor:
Cem Subakan

June 1, 2020

A report for a Mitacs Accelerate internship at
InVivo AI

Mila - Quebec AI Institute
Université de Montréal
Canada

Contents

Introduction

Drug discovery today is a long, costly and arduous process. On average, it costs around one-and-a-half billion dollars to run a drug discovery campaign, which takes over ten years to complete. After impressive results in a range of applications and ever more powerful algorithms being developed, artificial intelligence might be the key to accelerate the entire process, finding treatment to orphan disease along the way.

Traditionally, drug discovery has relied on the mere magnitude of the scope, testing thousands of compounds at the same time and with very little prior idea of the results, in order to stumble upon the handful of molecules that can move on clinical trials –with no guarantee of success.

How do you evaluate the potential of a novel molecule regarding a particular task? To put it simply, until very recently, you did not. Drug generation was and still is a long and arduous process, wherein you need to actually test each of your candidate molecules *in vitro* to have any idea whether or not they can have a positive impact on human health.

This is not a fatality. Fast-forward to today, and computational drug discovery is blossoming, with the promise of substantially expanding the scope of problems that can be answered and reducing the time and costs associated with solving them. Machine-learning has the potential to give us the prospect of a new molecule without any actual experiment, and one can even dream of an algorithm able to generate novel compounds perfectly tailored to attack a given disease.

To make such promise a reality, artificial intelligence still has a long way to go. Data efficiency, a long goal for machine-learning scientists, is still out of reach and yet paramount in a fundamentally low-data problem such as drug discovery.

At InVivo AI, I worked alongside great talents to find answers to these formidable challenges, by applying myself to fundamental science in artificial intelligence. The crux of my contribution in that direction involved working on a novel family of algorithms, which we called "Adaptive Deep Kernel Learning", or ADKL for short. ADKL aims to tackle some of the limitations of modern machine-learning algorithms with a principled method.

The report will be organised as follows. First, I will present a brief overview of the drug discovery process and how artificial intelligence is poised to revolutionise it, provided it can first address some of the challenges specific to drug discovery. Then, I will broadly discuss some state-of-the-art machine-learning solutions employed today in drug discovery. Third, I will focus on a new family of methods with great potential that can learn from multiple tasks directly. Finally, I will present our proposed method, ADKL, and describe what makes it a step towards computational drug discovery as we envision it.

1 A primer on computational drug discovery

Before we dive into the technical details relating to the artificial intelligence side of drug discovery (my personal strong suit, coming from a mathematics and computer science background), I wanted to help the layperson get a grasp of what drug discovery is, what computer science can do to help, and how.

At the time I started working at InVivo AI, I only had a very basic knowledge of chemistry, a discipline I had largely abandonned eight years before. As a computer scientist working with molecular data, getting up to speed quickly became a necessity. One can certainly get by for a while without any domain knowledge: after all, from a strictly machine-learning standpoint it is just "*matrix in, matrix out*"... Nonetheless, having the right intuition regarding the system you are modelling is paramount to choose the right architecture for the right problem. That intuition can only be as good as one's understanding of the processes at play.

In this particular endeavour, I was fortunate enough to be surrounded by great minds at InVivo AI. With their help, I quickly got to a point where I could make the right assumptions about what I was modelling.

Athough I am by no means an expert in drug discovery, I wanted to begin this report with a quick recapitulation on what drug discovery is and what its main challenges our, as well as the role of artificial intelligence.

1.1 What is drug discovery ?

Drug discovery is the process of finding novel compounds apt to act on a specific target in order to stop, treat and cure a disease. For example, if a team of researchers find that a particular type of cancer relies on a protein for growth, then our goal is to find a molecule that binds to this protein, in the hope of rendering it unusable to the cancer cells.

For a chemical compound to be suitable as the basis for a new drug, it needs to verify a number of desirable properties. To enumerate a few:

1. *High activity*, such that the drug performs well on the target.
2. *Low toxicity*, for obvious reasons.
3. *Specificity*, to limit adverse effects. An ideal compound is only active on the very specific objective targeted by the drug.
4. *Solubility*, so that the drug can be easily administered.

In the traditional drug discovery pipeline, depicted in figure 1.1, the search begins with a long and tedious process of *in vitro* testing on as many as 10 000 to 15 000 candidate molecular compounds. During that process, biochemists look for those that perform well in all categories.

At the end of that step, which lasts three to six years, only a handful of compounds move on to the next phase. In many ways, this step is akin to looking for a needle in a haystack:

although the process can be guided by some form of empirical knowledge, the latter is vastly insufficient to reduce the duration of the pre-clinical phase by any significant amount.

From there, another six to seven years of human clinical trials are necessary to validate the effects of the retained compounds.

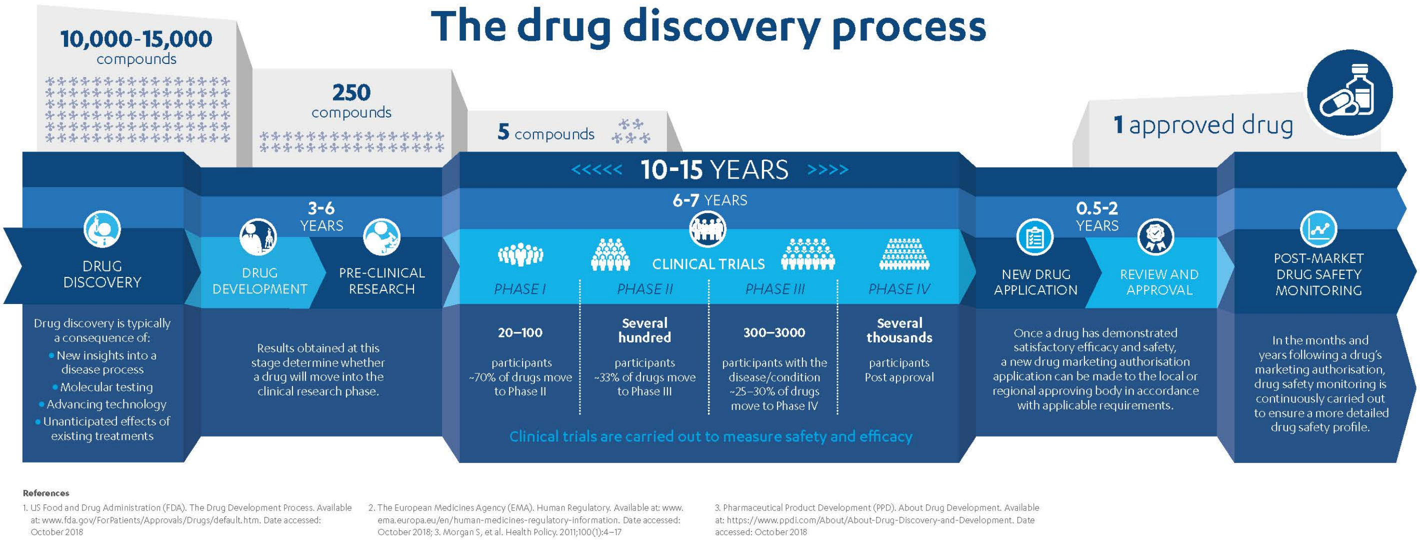

The drug discovery process
10,000-15,000
compounds
250
compounds
5 compounds
1 approved drug
<<<<< 10-15 YEARS >>>>
DRUG DISCOVERY
3-6 YEARS
DRUG DEVELOPMENT
PRE-CLINICAL RESEARCH
6-7 YEARS
CLINICAL TRIALS
PHASE I
PHASE II
PHASE III
PHASE IV
0.5-2 YEARS
NEW DRUG APPLICATION
REVIEW AND APPROVAL
POST-MARKET DRUG SAFETY MONITORING
Drug discovery is typically a consequence of:
• New insights into a disease process
• Molecular testing
• Advancing technology
• Unanticipated effects of existing treatments
Results obtained at this stage determine whether a drug will move into the clinical research phase.
20–100
participants ~70% of drugs move to Phase II
Several hundred
participants ~33% of drugs move to Phase III
300–3000
participants with the disease/condition ~25–30% of drugs move to Phase IV
Several thousands
participants Post approval
Clinical trials are carried out to measure safety and efficacy
Once a drug has demonstrated satisfactory efficacy and safety, a new drug marketing authorisation application can be made to the local or regional approving body in accordance with applicable requirements.
In the months and years following a drug's marketing authorisation, drug safety monitoring is continuously carried out to ensure a more detailed drug safety profile.
References
1. US Food and Drug Administration (FDA). The Drug Development Process. Available at: www.fda.gov/ForPatients/Approvals/Drugs/default.htm. Date accessed: October 2018
2. The European Medicines Agency (EMA). Human Regulatory. Available at: www.ema.europa.eu/en/human-medicines-regulatory-information. Date accessed: October 2018; 3. Morgan S, et al. Health Policy. 2011;100(1):4–17
3. Pharmaceutical Product Development (PPD). About Drug Development. Available at: https://www.ppdi.com/About/About-Drug-Discovery-and-Development. Date accessed: October 2018

Figure 1.1: The traditional drug discovery pipeline (Janssen, 2019)

1.2 Computational drug discovery

Imagine you could completely predict the properties of a compound as well as its effects on the human body *in silico* (that is, computationally). Such capabilities would enable us to bypass the time-consuming process of quasi-randomly testing molecules during the pre-clinical phase. Instead, we would first use computational bio-chemistry to pre-select a handful of compounds, whose *in vitro* and *in vivo* testing would become a mere formality. Drug discovery would be revolutionised by such a technical leap.

Artificial intelligence holds the promise of enabling not only fast and accurate pre-screening, but also direct compound optimisation for a set of properties. With such a technology at disposal, new insights into the operation of a disease would be translated into a set of active molecules ready for human trials in a matter of days, leading to a new treatment shortly thereafter.

Computational drug discovery can be summarised into three distinct but coupled problems:

1. *Scoring.* "Scoring" a molecule, or estimating its properties, is the first building block of an artificial intelligence that can help discover new drugs. Computational bio-chemists talk of *quantitative structure–activity relationship* (QSAR) modelling. The properties regularly tested are known as ADME/Tox for "absorption, distribution, metabolism, excretion" and toxicity.

2. *Generation.* There are an estimated 10^{60} possible small molecules (Bohacek et al., 1996b), while the largest molecular databases hold at most a hundred millions of them. Until now, the industry has been relying on candidate compounds with poor diversity relative to the unfathomable size of the unexplored chemical space.
3. *Optimisation.* Combining a good generation strategy to efficiently explore the space of small molecules and an accurate scoring function, we may one day be able to optimise for compounds directly: given a set of desirable properties, the algorithm would propose candidates that are designed to perform best.

Of course, scoring is the main piece of the puzzle. We can handle a poor generative model, by relying on empirical (and often random) methods used in the field until now. Although the candidate compounds will be the same, a good scoring model means that candidates are first tested and screened *in silico*, before a significantly lower share moves on to lengthier and costlier *in vitro* and *in vivo* testing. Conversely, we cannot optimise if the scoring model itself is unreliable.

At InVivo AI, we primarily focus on scoring molecules, by developing algorithms able to propose chemical compounds that have a high probability of scoring well on every desirable properties, thus limiting tremendously the amount of time and money needed on the first pre-clinical phase. At the same time, we actively develop generative and optimisation models, such that they may use the most recent and state-of-the-art methods for scoring as they come out.

Last, note that an algorithm that scores well necessarily uncovered patterns in the molecular structures. Said patterns can be analysed by chemists, leading to new insights and helping the scientific community in the exploration of high-potential avenues of research.

1.3 The challenges ahead of computational drug discovery

Computational drug discovery is a formidable endeavour and its success can be broken down to smaller challenges. To wit, computational biochemists need to address many sub-problems :

1. *Representation.* Learning from data in a well-structured environment, such as inferring tree height from tunk circonference (a well-known and almost toyish regression problem), is considerably easier than learning from heterogeneous data. Hence, the very first question computer scientists and biochemists need to answer deals with the representation of molecules.
2. *Extrapolation.* As we have established, the molecular space is vast. By focusing on an infinitesimal portion, we effectively limit ourselves to compounds that have a very low probability of being the optimal treatment for a given disease. However, scoring significantly different compounds implies that the algorithm is able to extrapolate beyond the molecular subspace it was trained on.
3. *Data efficiency.* One of the main obstacles to the advent of artificial intelligence in the field of drug discovery in the small amount of data. Hence, we need to devise methods and training strategies that can learn efficiently from very few datapoints.

4. *Calibration.* For a scoring algorithm to be useable in critical applications, it should be able to assess its level of confidence on its predictions.

1.3.1 Representation

The very first challenge that computational bio-chemists need to address before artificial intelligence can help drug discovery is the representation of the molecules. Indeed, there are many ways of describing a chemical compound, but the canonical representations are not readily applicable to machine-learning algorithms, which work with tensor data.

We can formulate a few desiderata regarding a suitable molecular representation, which needs to be :

- Informative. Artificial intelligence has become extremely effective at recognising patterns within the data it is presented with. However, it goes without saying that the prediction can only be as good as the information conveyed by the representation.

  For instance, molecular formulae only convey the number of atoms from each element present in the molecule, thus glossing over important configuration details that can dictate whether the molecule is a cure or a poison.

- Structured. Machine learning algorithms work primarily with tensor data. Over the years, computer scientists have develop methods to treat images, textual data, and even graphs.

  Depending on the representation we choose, some of these methods can be applied directly.

Astonishingly, a team of researchers obtained near-state-of-the-art results by using the two-dimensional representation of molecules, fed as images to an Inception-like convolutional neural network (Goh et al., 2017). However, I for one do not expect this avenue to be the future of drug discovery. Molecules are well-structured objects, and thus naturally call for inductive biases to be included in the way we represent and treat them. Using images breaks that structure, and needs the network bore the burden of understanding chemistry from bonds described by pixels, a remarkably unnatural representation indeed.

1.3.2 Compositionality and extrapolation

Given the immensity of the molecular space, it quickly becomes paramount to be able to extrapolate, at least to some extent, the insights gathered on the training data to unseen regions of the molecular space.

However, the above statement is problematic. As of today, machine-learning algorithms are essentially function approximators that are very effective at *interpolating* between datapoints, provided there come in sufficient numbers. On the other hand, extrapolation is not only elusive but also impossible without formulating some heavy assumptions on the structure of the problem.

Consider figure 1.2, that represents four univariable functions that coincide perfectly on some region of the reals. Imagine now that we train an algorithm on the shared portion of the x axis: how could it decide which function best describes the training data once we go beyond their scope ?

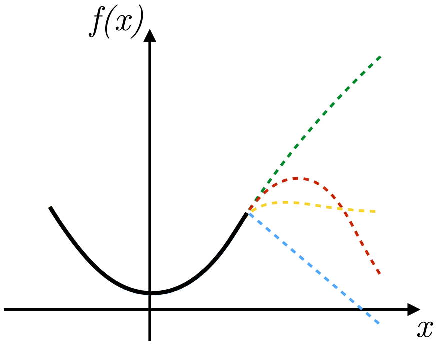


Figure 1.2: Which one do we choose ?
Four functions that coincide...

In the case of drug discovery and more specifically molecular scoring, the underlying hypothesis that lets us hope to one day extrapolate beyond the scope of the training data is the *compositionality* assumption. In a compositional problem we can derive some law that governs, at least to some extent, the observed phenomenon. Hooke was able to formulate his law linking the force needed to extend a spring to the elongation of the latter precisely because that problem is indeed compositional.

If you can retrieve the compositional structure of a problem, provided it exists, then you get two precious assets. First, the amount of data needed to train your model is substantially reduced: in the case of Hooke's law, you only need a single measurement to estimate the stiffness of the spring. Second, the compositional nature of the problem means that you may apply it far from the training examples.

Hence, we posit that much of a compound's activity is driven by the interactions between its chemical substructures, along with some fringe region-specific adjustments. That is to say, we do not discart region-specific knowledge as non-informative. Rather, we hypothesise the bulk of a compound's property can be inferred by looking at features that are compositional, and thus apply in a broader region of the molecular space.

At this point, you might get anxious that the assumption may not hold, and computational drug discovery be doomed. Let me share some reassuring thoughts.

- First, we know that compositionality exists in the realm of molecular properties. Indeed, it is the only reason chemistry itself, as a discipline, exits.
- Second, computational drug discovery itself does not rely on extrapolation. Should the compositionality assumption break, we would be forced to stick to the limited scope of the training data. But keep in mind that it still represents plenty of samples to go through, and traditional drug discovery has thriven without extrapolation since its beginnings.

1.3.3 A fundamentally low-data problem

When computer scientists design a machine-learning algorithm, they merely build an architecture. The whole idea of artificial intelligence is to *train* the machine rather than set its parameters by hand. And in order to train the algorithm, computer scientists need labelled data.

Depending on the complexity of the problem we would like to solve, our algorithm might need formidable amounts of data: in fact, there is a tradeoff between the expressivity of a network and the data efficiency of the training phase.

For instance, a simplest linear regression model will need very few points to be trained, but will not be capable of capturing complex patterns. At the far end of the spectrum, you will find mamoth-sized networks such as BERT (Devlin et al., 2018), a state-of-the-art language representation model with 304 million parameters and trained on a 3,300 million word corpus[1].

In the case of drug discovery and molecular scoring, we can safely assume that the relationship between the structure of a molecule and its properties requires a high expressivity to model the intricacies of molecular activities. Consequently, computational drug discovery is an inherently data-hungry process.

In the drug discovery context, labelled data come from past trial campaings wherein a myriad of studies are performed on the candidate compounds to have a sense of their activity, solubility, toxicity, and so on. In such studies, called biological assays or bioassays, only a handful of molecules are tested at a time. What is more, each bioassay reports the activity of these compounds relating to a very specific property. For instance, one such study might report the antibacterial effect of a few molecules on a specific strain of bacteria.

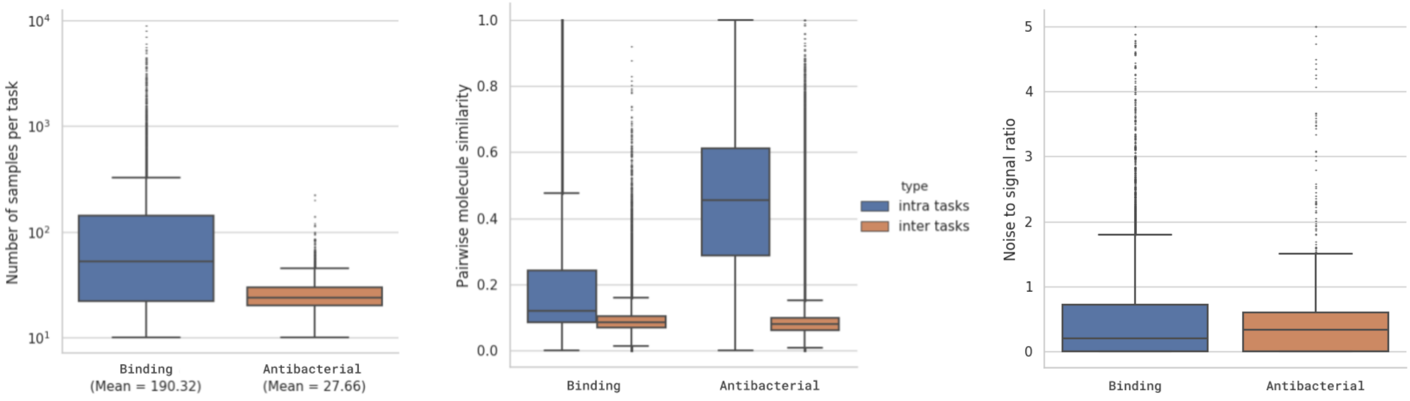
In other words, each bioassay pertains to a different machine-learning problem, with very little overlap. Moreover, for each of these problems, saying that data is scarce would be an understatement –and remember that modeling a bioassay is a most complex endeavour.

To give more substance, let us consider two bioassay collections that we at InVivo AI introduced in 2019. Compiling datasets coming from the public domain (BindingDB and PubChem), we proposed the community use the `Binding` and `Antibacterial` collections, that are available here[2]. See appendix A.2 for a thourough description of the collections.

The two collections epitomise the scarcity of data in the discipline. Figure 1.3a illustrates that issue eloquently, by reporting the number of compounds that are tested for `Binding` and `Antibacterial`: the average number of molecules tested in a single bioassay is as low as 190 for the `Binding` collection and 28 for the `Antibacterial` collection ! Remember, Inception (Szegedy et al., 2015) was trained on 1.2 million images... Although computational drug discovery necessitates huge numbers of examples, it remains a fundamentally low-data problem. We need to reconcile this apparent contradiction.

[1] Unlike molecular scoring, language representation is an inherently self-supervised task: any sentence or paragraph written in the language you are training on is a valid example, which helps gather massive amounts of data.

[2] The full URL for the paper version: `https://github.com/invivoai/molecular-datasets`



(a) Number of samples

(b) Similarity between molecules

(c) Signal to noise ratio

Figure 1.3: A few statistics for the `Binding` and `Antibacterial` collections

But scarcity is not the only issue. Looking at figure 1.3c, we quickly realise that data is not only limited but also incredibly noisy. Indeed, the noise-to-signal ratio (computed on the measurements that were done twice or more) is extremely high, an additional challenge for training machine-learning algorithms.

Moreover, a concern raised in the previous section about extrapolation is already dicernable in the two collections. To wit, figure 1.3b shows that molecules are much more similar[3] within a bioassay than accross them, which suggests that the results will be hard to generalise even to known molecules without some kind of extrapolation strategy.

The diagnosis is unequivocal: drug discovery is an inherently low-data problem. Data is not limited *per se*: rather, data related to each individual property is painfully scarce. Now, how do we go about working towards a solution? Two directions are before us:

1. We may work towards automating *in vitro* testing, to be able to collect data with a more systematic approach. The amount of data would increase dramatically, along with its quality.

2. We can also develop novel methods that extract information from seemingly unrelated studies or bioassays. The underlying assumption is that the different molecular properties are dictated by the structure of the compounds, which can be learned across bioassays.

These two approaches are not mutually exclusive. Surely, having better machine-learning methods would help the first approach, much like cleaner and more abundant data will certainly help the second avenue. However, while the former necessitates huge investments to develop and produce automated test machines, the latter can benefit from the data that is already available –be it in the public domain or not. At InVivo AI, we have chosen to focus on the second approach.

[3] Here similarity is computed for molecule pairs using Tanimoto's similarity with ECFP4 on 2048 bits (see section 2.1.2).

1.3.4 Calibration

Imagine you are a cartographer, looking to construct a map of an unknown world. You have explorers at you disposal, that you can pay to explore a specific part of the land and thus improve the precision of the map. In order to maximize the information obtained about the world, you should send your explorers to regions for which you have minimal information, rather than places for which you already have a somewhat precise idea.

However, to be able to do that, a cartographer needs to assess its confidence in the precision of the current map. Your explorers are flawed –aren't we all?–, and might give you less than reliable insights on the New World.

In the context of drug discovery, we are building a map of the molecular space. We do not send explorers to the far corners of the world: rather, we ask researchers to perform experiments in the form of bioassays. But those may be noisy, and are certainly costly.

Hence, another desirable property for our algorithms is for them to be aware of their shortcomings. More formally, an algorithm capable of correctly assessing the confidence interval around its predictions (what is called calibration) can come extremely handy in the setting of biochemistry. Indeed, the limited amount of data and the high acquisition cost mean that being able to pinpoint the compounds that will give the most information when tested has enormous advantages.

With a well-calibrated algorithm, you can guide the *in vitro* testing of new compounds in such a way that maximizes the information you get about the world when you generate new labels. This is called *active learning*: you have a set of labelled examples and a collection of samples for which you can get new labels at a cost, and the goal is to select for which of those samples you will pay to get labels.

2 Traditional machine-learning for drug discovery

In the previous section, I set out to discuss the challenges that computational drug discovery faces to reach its enormous potential as a revolutionising tool for biochemistry.

The machine-learning community has already made huge progress in tackling these challenges, and this section is dedicated to presenting some of the ideas that have been proposed. It will be the opportunity for me to give a more formal presentation of machine-learning applied to drug discovery, which will surely please the more mathematics-enthused reader.

I will focus on two aspects of drug discovery: representation and scoring. Bear in mind that this section is by no means an comprehensive enumeration of the state-of-the-art methods used today in the field. Rather, it is meant as a collection of examples whose goal is to better illustrate the stakes of using artificial intelligence in drug discovery, and how fundamental research in machine-learning has provided ways to address some of those challenges.

2.1 Representation

Before we can effectively train an algorithm to recognise the extremely compex patterns that govern molecular activity, we need a representation method that satisfies two main *desiderata*:

1. The representation method must be adapted to a machine-learning algorithm, that is to say either presented as a tensor or as an object that we know how to transform into a tensor.

2. The representation method needs to include as much information as possible. Otherwise, the algorithm will not be able to perform the downstream task.

Figure 2.1 presents eight ways to represent a given molecule that can be used in machine-learning. Let us gloss over some of them:

- Fingerprints (1) are the most readily usable representation, since they describe a molecule by a binary vector, that is, a vector of ones and zeros that encode the presence or absence of a set of substructures.

- The SMILES description (2) is a textual representation that uniquely characterises a given molecule. The community can use well-established methods borrowed from natural language processing to extract its information.

- The 3D geometry (7) is able to account for the full conformation of the molecule, and thus conveys important additional information. However, it remains a poorly structured representation not usable as is by a machine-learning algorithm.

At InVivo AI, we primarily use SMILES and fingerprint, since they are easy to produce and achieve good performance. In the following, I will focus on the two and them in more depth.

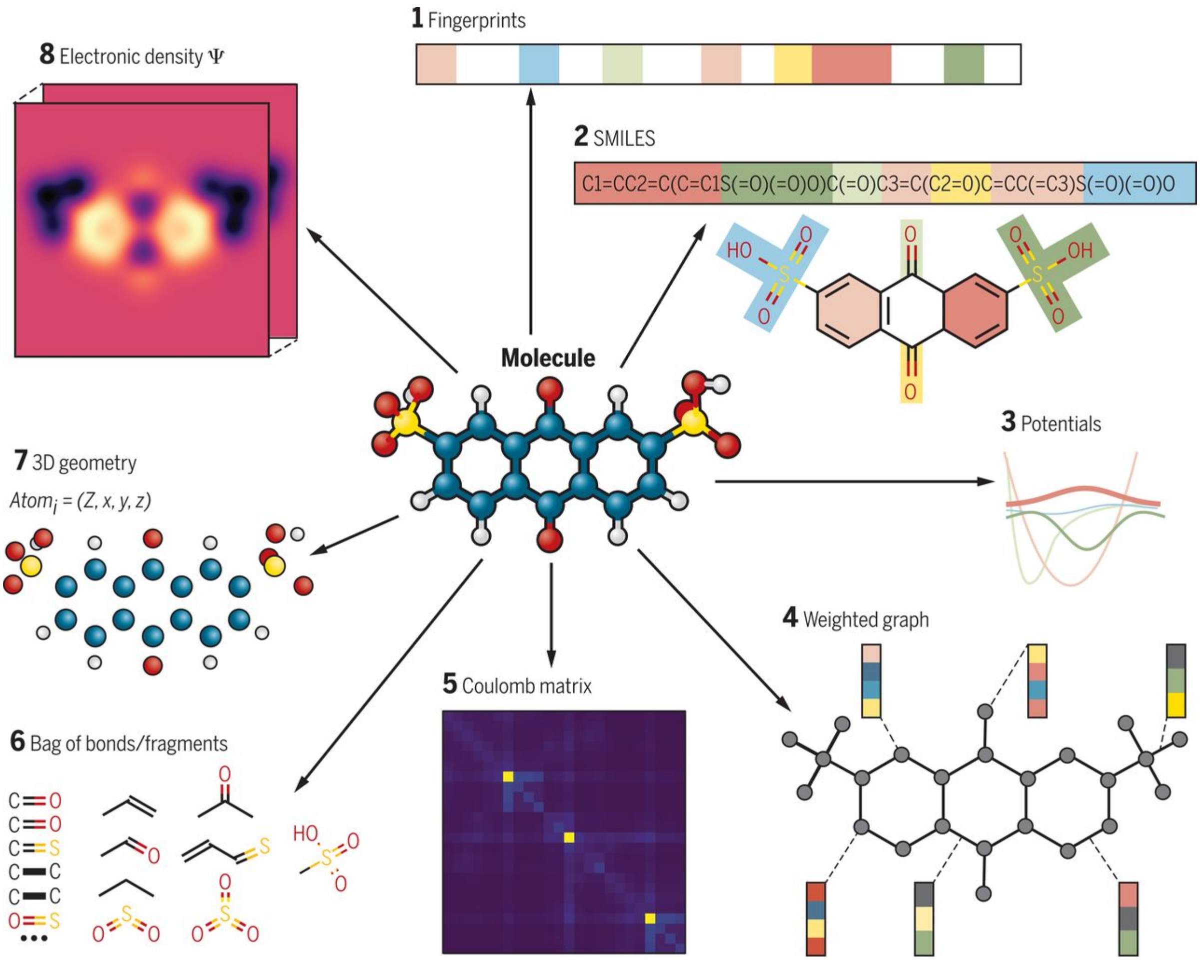


Figure 2.1: Different ways of representing a molecule (Sanchez-Lengeling and Aspuru-Guzik, 2018)

2.1.1 The SMILES representation

The *simplified molecular-input line-entry system*, or SMILES (not a plural), provides a convenient, human-readable and normalised way to represent molecules. It was proposed by Weininger (1988).

The principle is straightforward: the system provides a way to describe the molecular structure of a compound by defining the rules that govern how the molecular graph should be traversed in order to obtain unambiguous string representations. Figure 2.2 proposes an example of that procedure: the algorithm breaks the cycles, defines a main chain and describes the branches according to the syntaxic and grammatical rules of the SMILES notation system.

Hence, SMILES yields a string notation that fully describes the 2D configuration of the molecule. As such, it is a particularly rich representation, since it can uniquely represent any molecule.

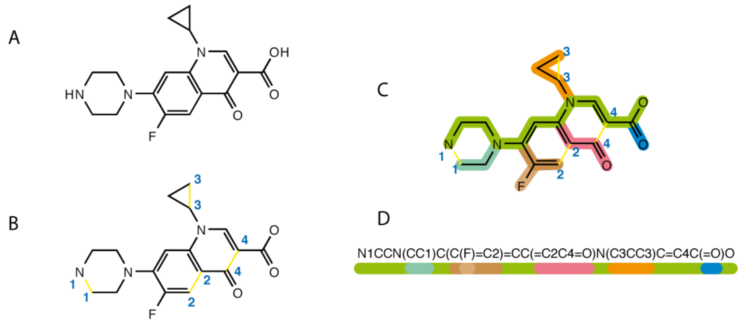


Figure 2.2: Construction of the SMILES notation for Ciprofloxacin; the algorithm breaks the cycles, defines a main chain and describes the branches (Fdardel and DMacks, 2007)

Moreover, although it cannot be used directly in a machine-learning setting, it may as well be: tremendous progress made in the field of natural language processing (NLP) have led to extremely effective algorithms (namely “recurrent neural networks”, or RNNs) to work with textual data governed by syntaxic and grammatical rules.

Hence SMILES is not a complete representation in a machine-learning sense: it is merely a convenient intermediary notation that relies on an auxiliary network whose role is to provide a meaningful tensor representation. One question remains: how do you train that auxiliary network? You have two main possibilities, which are not exclusive:

1. You can train the entire network in an end-to-end fashion, meaning that the representation is learned to minimise the error on the task you are training on.

2. Alternatively, you might want to pre-train the “auxiliary network”, using unsupervised learning.

The first solution adds a potentially large number of parameters to the trained model, and thus may necessitate vast amounts of data. As such it might not be the best choice in our context, even though it would learn features more specific to the problem at hand.

The second solution is the workhorse of natural language processing (NLP), whose latest major advances precisely come from pre-trained models such as BERT (Devlin et al., 2018). The representation network can learn to provide a meaningful representation using an encoder-decoder strategy: we add a decoder network whose role is to retrieve the original SMILES notation from the code provided by the representation network. The two network are trained jointly on a library of molecular compounds, letting the representation network learn to recognise the SMILES grammar implicitly. Once training is complete, the decoder network is discarded.

More formally, this pre-training strategy amounts to the following minimisation problem:

$$\underset{\theta,\phi}{\arg\min} \quad \sum_{s\in\mathcal{S}} \ell_{\theta,\phi}(\hat{s}, s) \qquad \text{where} \quad \hat{s} = \text{decoder}_{\phi}\left(\text{encoder}_{\theta}(s)\right) \tag{2.1.1}$$

Here θ and ϕ represent the parameters of the encoder and the decoder respectively, while ℓ denotes the chosen loss function. Moreover, $\mathcal{S}$ symbolises the set of available SMILES strings. Note that the PubChem database identifies around 100 million unique compounds: there are plenty of examples to train our representation model.

Although I only mentionned neural networks so far, they are not the panacea and many alternatives do exist. For instance, one could use a bag-of-words approach. However, the expressivity of deep learning makes it the methodology with the highest potential in our quest for computational drug discovery.

To sum up, the SMILES notation is indeed informative. However, it requires an auxiliary model to provide a useful encoding of molecules –and that model is seldom easy to produce.

2.1.2 Molecular fingerprints

Molecular fingerprints are designed to deliver a binary vector representation, giving up some information in return for a method that does not necessitate any pre-training. The term "molecular fingerprint" describes a vast family of representation methods. I will focus solely on the *extended connectivity fingerprint*, or ECFP (Rogers and Hahn, 2010), since it has emerged as the industry standard —and the only method we use at InVivo AI.

ECFPs apply an iterative hashing mechanism to represent substructures of different lengths (in terms of number of bonds). Two hyper-parameters need to be set: the maximum substructure diameter to be considered and the size of the resulting binary vector.

During the generation procedure, the method collects the hashes that represent the different substructures. The result of the iterative process is a set of 32-bit integers[4]. To represent them in the form of a fixed-size vector, the codes are *folded* to the vector length via a modulo operation. Most researchers use a binary vector as the final representation, but the alternative (counting the number of substructures that are represented by a given bit) is also used.

With this binary representation, we can define a similarity between molecules very easily, for example by counting the number of bits that are simultaneously on, divided by the number of bits that are on in at least one of the two molecules (what is known as the Tanimoto similarily). Note however that being close in fingerprint space bears no guarantee as to the likeness of behaviour.

Remember that the main assumption made by computational biochemists is that the properties of a molecular compound are governed by its substructures how they relate to one another. Fingerprints are an attempt to capture that information, albeit loosely: it is worth noting that off bits are more precise than on bits, since an on bit does not inform on which pattern is actually represented.

Daylight Chemical Information Systems, Inc. (2019) propose a good overview of the different fingerprinting methods and how they relate to one another. I suggest the interested reader explore their material.

[4] Rogers and Hahn do not recommand a particular hashing function, nor do they limit it to map to a 32-bit integer space. However, the folding operation limits the advantage of a larger code space.

2.2 Scoring

The crux of computational drug discovery hinges on our ability to predict the properties of a novel molecule.

Machine-learning, in the form of pattern recognition, can naturally play a major role in this subfield of computational drug discovery. We can assess the potential of a molecule by using its representation, be it learned or static, and plug a machine-learning algorithm to learn a representation-to-property mapping.

2.2.1 Molecular scoring as a machine-learning problem

Quantitative structure-activity relationship (QSAR) models aim to unearth the link between the structure of a molecule and its activity. It amounts to learning a function that maps from a molecular representation to a ground truth measurement, making sure that it generalises well to unseen data.

More formally, let x represent a given molecule, or rather, its representation. Let y denote its ground truth activity, and $\hat{y}$ its predicted activity. Molecular scoring aims to learn a function f_θ, parametrised by θ, such that $\hat{y} = f_\theta(x)$ is as close to y as possible, according to a chosen loss function ℓ.

Hence, molecular scoring amounts to the following optimisation problem:

$$\arg\min_{\theta} \quad \mathbb{E}_{x,y\sim p(x,y)}\left[\ell(y, f_\theta(x))\right] \tag{2.2.1}$$

Of course, we do not have access to the actual distribution of samples within the task. Thus, algorithms are trained to minimise the “empirical risk”, that is, the average loss on a set of training examples. The optimisation problem becomes:

$$\arg\min_{\theta} \quad \underbrace{\mathbb{E}_{x,y\in D_{\text{train}}}\left[\ell(y, f_\theta(x))\right]}_{\mathcal{L}(D_{\text{train}}|\theta)} \tag{2.2.2}$$

Collected samples are divided into three datasets: 1) the training set D_{train}, used to compute the error and minimise it; 2) the validation set D_{valid}, used to tune the hyperparameters; and 3) the test set D_{test}, held out and used only once to estimate the generalisation error.

2.2.2 Fingerprints and Random Forests

The random forest algorithm has shown great success in predicting molecular properties in the past (Svetnik et al., 2003; Kensert et al., 2018). Being robust to noise and little prone to overfitting, it is able to reach a good accuracy where deep learning methods fail to generalise to unseen data.

Random forest is an ensemble method where a collection of weak learners are each trained on a different subset of examples (drawn with replacement), as well as a different set of features. The most straightforward representation in this context (and the one used in practice) is the fingerprint representation.

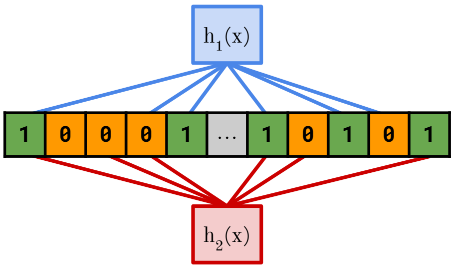


Figure 2.3: Example of a random forest with two learners using a fingerprint representation

Figure 2.3 gives an example of the algorithm. The two weak learners h_1 and h_2 (both regression trees) observe different bits in the fingerprint representation of the molecule, and offer their prediction. The (full) learner takes these weak outputs into account to produce a more robust prediction.

Using random forests in conjuction with fingerprints often provides the best predictor in the single task setting, where the scarcity of data forbids the use of more complex methods such as deep learning. However, it cannot learn accross tasks.

2.2.3 Deep learning and drug discovery

Following breakthroughs in domains including computer vision, autonomous driving, and natural language processing, deep learning methods have entered the domain of pharmaceutical research and development. Recent successes include the deconvolution of biological targets from omics data (Min et al., 2017), generation of drug-like compounds via *de novo* molecular design (Xu et al., 2019), and chemical synthesis planning (Segler and Waller, 2017; Segler et al., 2017).

A common characteristic of these applications, however, is the availability of high quality data, and in high quantities. Unfortunately, many critical prediction tasks in the drug discovery pipeline fail to satisfy these requirements, in part due to resource and cost constraints (Cherkasov et al., 2014).

The lack of abundant and clean data certainly explains why deep learning is still relatively stealth in the worl of drug discovery and more particularly scoring. To be able to use more expressive (and thus more data-hungry) models, we need to find a way to provide them with more training data.

3 Learning from multiple tasks

As we have established in section 1.3.3, the main issue computational biochemists need to tackle is the limited availability of examples. Yet, although data is indeed scarce for each biochemical property, the number of bioassays is in fact enormous. Moreover, we make the assumption that some amount of structural information can be shared between tasks.

Imagine you're first learning to recognise cats from dogs, and then birds from zebras. In the traditional machine-learning pipeline, you would simply discard everything you learned from the first task, even though it is quite easy to realize that the problems share a lot of information: for instance, the low-level features such as edge and shape detection learnt in the first task can be used in the second one with minimal adaptation.

Transpose the issue to drug discovery, where training data is painfully scarce, and you're looking at catastrophic underperformance if you naively apply the traditional machine-learning methodology. Indeed, the vast majority of biochemistry assays only come with a few tens of examples, which makes using the knowledge acquired across a number of distinct tasks all the more important.

Hence, methods have been developed to cope with the lack of data for individual assays by capturing knowledge from multiple tasks directly. I will discuss two solutions:

1. *Multi-task learning*, where a model is trained to output a vector that predicts the activity for every tasks at once.
2. *Meta-learning*, a radically different approach where the model is explicitly trained to learn new problems faster (ie with fewer datapoints).

3.1 Multi-task learning

Multi-task learning is the first step to benefit from the knowledge shared across tasks. It is very much a traditional machine-learning technique, but uses clever tricks to learn from multiple tasks at once.

The idea is simple: instead of learning to perfom each task individually, we consider only one multi-dimensional regression problem. Once again, let x be the representation of a given molecule. Let y_t denote its activity related to the t^{th} task; note that this information may not be available to us since all molecules are not tested in each bioassay. We can define $\boldsymbol{y}$ to be the concatenation of those targets, with the convention of setting to zero the missing labels:

$$\boldsymbol{y} = (y_1, y_2, \dots, y_n) \qquad \text{and} \qquad \hat{\boldsymbol{y}} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n) \tag{3.1.1}$$

The machine-learning problem now amounts to predicting the entire vector of activities. By transforming the task that way, we can exploit the hundreds of thousands of examples and training a deep learning algorithm using back-propagation becomes possible.

But there is one piece missing: we do not want to propagate the error on a target that we abitrarily set to zero. The trick is to keep track of the task index and only consider the corresponding component for each training example.

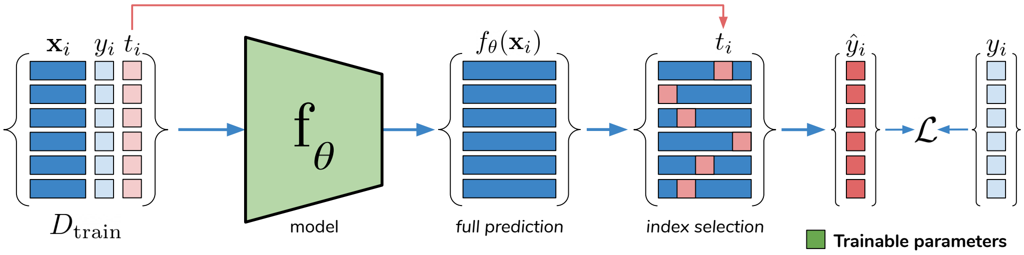


Figure 3.1: Diagram of a multi-task model

Figure 3.1 shows how the multi-task model is adapted to work in a setting where some targets are missing. The network is trained to output the full prediction vector, but only the component that corresponds to the considered bioassay is compared to the target. That way, we propagate an error signal using every single training example.

More formally, the optimisation problem associated with this method is:

$$\arg\min_\theta \quad \mathbb{E}_{x,t,y \in D_{\text{train}}} \left[\ell(y, f_\theta(x)_t) \right] \tag{3.1.2}$$

Where t is the index of the task. Note that a single molecule may appear multiple times in the training dataset should it have been tested in different bioassays.

Multi-task learning is a first step towards learning across task, using a natural idea that adapts traditional methods. However, this family of algorithms is still not particularly well-suited for few-shot learning. That is especially true in a drug discovery setting, where we would perform preliminary *in vitro* testing for a new property and expect the algorithm to generalise well such that it may guide the molecular search efficiently.

In other words, multi-task learning focuses on learning from multiple tasks in parallel. It can still construct a better representation of molecules in the intermediate layers, since that latent encoding is specifically designed to perform well on a variety of tasks. It has however a limited effectiveness when a new task comes along, a common case in the world of drug discovery.

3.2 Meta-learning

A new range of algorithms has been developed in recent years in order to tackle the issue of generalising from very few examples (what is known as few-shot learning). The field, called meta-learning, aims at "learning to learn" from a collection of machine-learning problems that share a substantial amount of structure. The crux of meta-learning is to design the objective function in such a way that the algorithm is explicitly trained to learn faster.

In what follows, I will first discuss the vocabulary and notations that surround this relatively new field. Then, I will insist on the objective function and show how it leads the algorithm to *learn how to learn*. Finally, I will present some methods from the discipline and focus on what is arguably the most popular meta-learning algorithm, namely model-agnostic meta-learning or MAML.

MAML serves as a great pedagogical tool when it comes to presenting meta-learning, since it examplifies particularly well what is expected to be learnt in that paradigm.

3.2.1 Vocabulary and notations

If you come from a traditional machine-learning background, meta-learning can be mildly confusing at first, and the vocabulary does not help.

Unlike the traditional machine-learning setting where you want to train a model on a distribution of individual examples $p(x)$, in meta-learning we train an algorithm to learn as fast as possible when presented a new task $t \sim \tau(t)$. However, we do not have access to either $p(x)$ or $\tau(t)$: as an alternative, we use a dataset assumed to have been sampled from that distribution.

	Traditional machine-learning	Meta-learning
Input	Training set D_{train}: $D_{\text{train}} = \{(\boldsymbol{x}_i, y_i)\}$	Meta-training set $\mathcal{D}^{\text{meta}}_{\text{train}}$: $\mathcal{D}^{\text{meta}}_{\text{train}} = \left\{(D^t_{\text{support}}, D^t_{\text{query}})\right\}_{t \in T}$
Output	Model parameters	Meta-model parameters
Objective	Minimise generalisation error on the test set	Minimise generalisation error on D^t_{query}s from meta-test set

Table 1: Comparison of traditional machine-learning and meta-learning

In meta-learning, we work with a *collection of datasets*, each representing one given task –and containing individual examples from it. The collection is divided into a meta-training set $\mathcal{D}^{\text{meta}}_{\text{train}}$ and a meta-test set $\mathcal{D}^{\text{meta}}_{\text{test}}$, and its datasets are each separated between training and test sets.

To limit confusion in the meta-learning setting, each task within the meta-collection will be divided between *support* and *query* sets to avoid over-using the *training* and *test* vocabulary, which will be restricted to the separation of the tasks. Hence:

$$\mathcal{D}^{\text{meta}}_{\text{train}} = \left\{\left\{D^t_{\text{support}},\ D^t_{\text{query}}\right\}\right\}_{t \in T_{\text{train}}} \quad \text{and} \quad \mathcal{D}^{\text{meta}}_{\text{test}} = \left\{\left\{D^t_{\text{support}},\ D^t_{\text{query}}\right\}\right\}_{t \in T_{\text{test}}}$$

3.2.2 The meta-objective

In the meta-learning setting, we train algorithms to *learn how to learn* from a variety of tasks that share some amount of structural information. The purpose of the training procedure is to teach the model to adapt as fast as possible to each task using the support set, in order to optimise the generalisation performance on the query set. More formally, a meta-learning algorithm aims to optimise the following minimisation problem:

$$\arg\min_{\theta} \mathbb{E}_{t \in \mathcal{D}^{\text{meta}}_{\text{train}}} \left[\mathcal{L}(D^t_{\text{query}} \mid \theta, D^t_{\text{support}})\right] \tag{3.2.1}$$

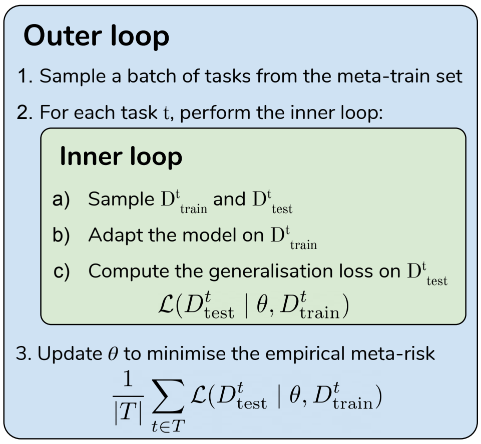
where $\mathcal{L}$ denotes the full loss and $\mathcal{L}(D^t_{\text{query}} \mid \theta, D^t_{\text{support}})$ the meta-loss on the support set after adaptation.

More specifically, letting θ' be the parameter adapted on D^t_{support}:

$$\mathcal{L}(D^t_{\text{query}} \mid \theta, D^t_{\text{support}}) = \sum_{\boldsymbol{x}, y \in D^t_{\text{query}}} \ell(f_{\theta'}(\boldsymbol{x}), y) \tag{3.2.2}$$

At first glance, a few details may worry the layman in meta-learning. To wit, the approach is all about back-propagating (or more generally, minimising) the error on the *test set*, which raises a major red flag in the traditional machine-learning setting.

But there is no contradiction, nor is there a leak of information. Indeed, the goal of meta-learning is to learn as fast as possible on *new tasks*. Instead of learning on a distribution of examples, represented by the datasets D_{train} and D_{test}, a meta-learning algorithm learns to adapt quickly within a *distribution of tasks*.



Outer loop

1. Sample a batch of tasks from the meta-train set
2. For each task t, perform the inner loop:

Inner loop

a) Sample D^t_{train} and D^t_{test}

b) Adapt the model on D^t_{train}

c) Compute the generalisation loss on D^t_{test}

$$\mathcal{L}(D^t_{\text{test}} \mid \theta, D^t_{\text{train}})$$

3. Update θ to minimise the empirical meta-risk

$$\frac{1}{|T|} \sum_{t \in T} \mathcal{L}(D^t_{\text{test}} \mid \theta, D^t_{\text{train}})$$

Figure 3.2: The general meta-learning algorithm

Perhaps a more intuitive way to look at meta-learning is to understand that the training procedure (depicted in figure 3.2) is divided between an inner-loop, where the algorithm *adapts* to a new task, and the outer-loop, where the algorithm *learns* how this adaptation could be better-performed. At test time, only the inner loop is performed on a previously unseen task.

In figure 3.2, I present the general training procedure behind meta-learning. It is worth noting that most meta-learning algorithms are trained in the so-called "episodic" setting, where the support and query sets are subsampled during training to remain in the few-shot learning regime, hence the sampling operation in the inner-loop. That operation is often left out at test time, since limiting the number of support examples would likely hurt the generalisation performance[5].

Both the adaptation and the meta-loss minimisation can be performed in a number of ways, and numerous methodology have been proposed. In what follows I will focus on one algorithm in particular, model-agnostic meta-learning (MAML), since it has become a ubiquitous baseline achieving state-of-the-art results in a variety of tasks.

3.2.3 Meta-learning methods

Most meta-learning algorithms differ in two aspects:

- The nature of the meta-knowledge, or information shared across tasks, captured;
- The amount of adaptation performed at test-time for new tasks or datasets.

In a drug discovery setting, due in part to the unfathomable size of the chemical space (Bohacek et al., 1996a), we need methods that are expressive enough to allow extrapolation

[5] Performance decrease is likely but not automatic: the meta-learning algorithm is trained in a few-shot regime and as we deviate from it, performance may decrease before ultimately going up again.

and uncertainty estimation in unseen regions of chemical space at test-time. What is more, a single molecule can have results be reported for multiple bioassays, which means test-time adaptation is particularly important.

Authors have proposed to use metric learning methods (Koch et al., 2015; Vinyals et al., 2016; Snell et al., 2017; Garcia and Bruna, 2017; Bertinetto et al., 2018) for meta-learning. Such techniques accumulate meta-knowledge in high capacity distance functions and use simple base-learners such as k-nearest neighbors (Snell et al., 2017; Vinyals et al., 2016) or low capacity neural networks (Garcia and Bruna, 2017) to produce adequate models for new tasks. The idea is simple: we can afford to share a complex and expressive network, since it will be trained on all task and therefore on numerous examples, but the task-specific part needs to be as light as possible. However, the covariance functions are not adapted at test-time.

Another avenue explored in recent years relates to initialisation- and optimisation-based methods (Finn et al., 2017; Kim et al., 2018; Ravi and Larochelle, 2016). These algorithms respectively learn the initialisation point and update rule for gradient descent-based algorithms, allowing them to adapt faster on new tasks –at the cost of necessitating slow and memory inefficient training.

3.2.4 Model-agnostic meta-learning

Model-agnostic meta-learning (Finn et al., 2017), or MAML, is one of the most commonly used meta-learning method. Its popularity stems in part from the fact that it can adapt any differentiable machine-learning algorithm to meta-learning.

The approach, presented in figure 3.3, is extremely straightforward: it looks for a parameter initialisation that enables the model to learn faster on new tasks, using gradient descent. Since gradient descent itself is differentable, MAML simply back-propagates the error on the query set through the gradient descent steps all the way to the "initial parameters".

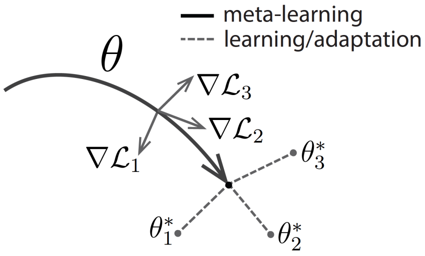


Figure 3.3: Diagram of MAML (Finn et al., 2017)

More formally, let f_θ be a machine-learning model parametrised by θ. MAML minimises the meta-objective described in equation (3.2.1) by optimising θ to reach better performance on average after adaptation, using the procedure presented in algorithm 1.

Note that the procedure described by algorithm 1 requires a distribution over tasks. In practice, we do not have access to such a distribution; we can however make the assumption that the meta-datasets $\mathcal{D}_{\text{train}}^{\text{meta}}$ and $\mathcal{D}_{\text{test}}^{\text{meta}}$ are sampled from $\tau(t)$.

As algorithm 1 illustrates, MAML is strikingly simple. It hinges on a simplistic idea: since we want our algorithm to learn more efficiently, let us just adjust the objective function to train for that explicitly. And seeing that gradient descent –the workhorse of vast majority of machine-learning algorithms today– is differentiable, Finn et al. proposed a method that adapted virtually the entire field to meta-learning.

Algorithm 1 Model-Agnostic Meta-Learning (Finn et al., 2017)

```
Require: $\tau(t)$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyper-parameters
Randomly initialize $\theta$
while not done do
    Sample batch $T$ of tasks $t \sim \tau(t)$
    for all $t$ do
        Sample support and query sets $D^t_{\text{support}}$ and $D^t_{\text{query}}$
        Evaluate $\nabla_\theta \mathcal{L}(D^t_{\text{support}} \mid \theta)$ with respect to the support set
        Compute adapted parameters with gradient descent: $\theta_t = \theta - \alpha \nabla_\theta \mathcal{L}(D^t_{\text{support}} \mid \theta)$
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{t \in T} \mathcal{L}(D^t_{\text{query}} \mid \theta_t)$
end while
```

Being gradient-based, MAML presents the advantage of truly adapting to the new task at test time and in a principled manner, where some other methods merely learn huge representation networks in the hope of mitigating the effects of domain adaptation. Of course, it does have some drawbacks. For one thing, negative adaptation[6] remains possible and not yet well-understood (Deleu and Bengio, 2018). It is also painfully slow and resource-intensive to train, owing to the need to compute the Hessian of a gradient.

[6] Negative adaptation refers to the cases where the generalisation performance is worse off after adaptation. Being gradient-based, MAML should eventually reach a good local minimum after enough adaptation steps and examples.

4 Adaptive deep kernel learning

In the previous section, we saw two general paradigms to learn from more than one task. First, I presented multi-task learning, a natural step to adapt traditional machine-learning to a setting where multiple tasks are available for training. Then, I focused on a radically different strategy whose goal is to *learn to learn efficiently*, hence its name: meta-learning.

As we have established in section 1, building methods apt to learn efficiently from very few datapoints is vital in a drug-discovery context. The field is particularly well-suited for a meta-learning approach: it advertises ample amounts of data, scattered into a myriad of different tasks. Making the reasonable assumption that molecular structure is at the root of the activity, and that we can use common patterns from one task to the other as a low-level representation, meta-learning becomes a natural research avenue to tackle computational drug discovery.

Striving to unleash the potential of artificial intelligence in the field of drug discovery, InVivo AI aims to develop novel methods that can learn from very few datapoints, and we actively explore the route of meta-learning, leading to novel methods being developed in-house as well as submissions to higher-tier publications and conferences.

In this section, I will focus on one such project on which I spent most of my time at InVivo AI, namely a novel framework we called Adaptive Deep Kernel Learning, or ADKL for short. The ADKL family is a set of algorithms that can efficiently learn from a large collection of problems and generalise the gathered knowledge to new tasks.

That work was led with Prudencio Tossou, co-founder and head of research at InVivo AI (and my supervisor), along with Mario Marchand and François Laviolette from Université Laval as well as Alexandre Lacoste from Element AI, and with the help of Daniel Cohen, CEO and co-founder of InVivo AI.

4.1 Desiderata and contributions

Being a drug discovery company, InVivo AI focuses on modelling bioassays, specifically those relevant in the early stages of drug discovery: primarily binding and cellular read-outs. Data from these assays, consisting of libraries of molecules and their associated real-valued activity scores, is often extremely small and noisy (see appendix A.2 for an eloquent demonstration of that issue).

As we have seen in section 1, data collected in bioassays are not only scarce but also incredibly specific[7], owing to the combinatorial nature of possible test configurations. That specificity makes it impossible to compare data collected across different assays directly.

Furthermore, bioassay modelling is intended to be the first step in a full drug discovery pipeline, that will eventually include direct molecular optimisation (e.g. with Bayesian optimization) and efficient exploration of the chemical space (e.g. active learning). Hence, accurate prediction and uncertainty estimation using few datapoints is critical to successful downstream applications in drug discovery.

[7] Examples of specific assays include testing the antibacterial effect on a given strain of bacterium, in presence of a particular adjuvant...

In the paper we submitted, our contributions were three-fold. First, we framed few-shot regression as a deep kernel learning (DKL) problem and demonstrated its advantages relative to classical metric learning methods. Then, we derived the ADKL framework by learning a task-dependent kernel, allowing for more test-time adaptation than the DKL framework. Finally, we introduced two real-world datasets for modelling biological assays.

4.2 Deep kernel learning

As the name suggests, ADKL is part of the family of kernel methods. In simple words, that means that in order to make a prediction about a new molecule, the algorithm will compare that compound to the labelled examples seen in the support set. We call it "deep" kernel learning because ADKL uses a deep network to create a representation of the compounds which makes this comparison easier.

Recall that meta-learning algorithms can be roughly divided into two families: those that store meta-knowledge in an extremely expressive network that is common to all tasks but limit the amount of task-adaptation at test time, and those that do adapt to the new task at the cost of slower and more resource-intensive training.

Working on ADKL, we were looking to make a step towards combining the strengths of both types of methods, a crucial feature to be able to score molecular compounds efficiently. For that reason, we chose to frame the bioassay modelling task as a deep kernel learning problem.

4.2.1 Overview of the method

Before I present the adaptive deep kernel learning method itself, let us first dive into the deep kernel learning (DKL) framework.

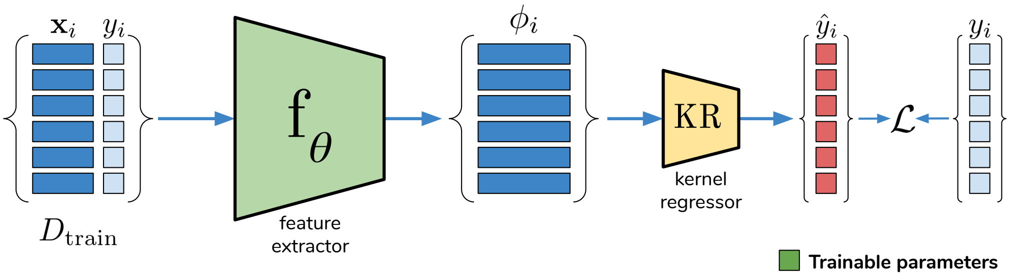


Figure 4.1: Diagram of the deep kernel learning framework

Figure 4.1 shows a diagram of the deep kernel learning method. DKL is not specifically designed for meta-learning, and was developed long before few-shot learning became an issue. Once again, the intuition behind the framework is simple: the goal is to train a large feature extractor network, whose role is to transform inputs into a meaningful representation that can then be used by a kernel method such as kernel Ridge regression (KRR).

The deep kernel learning algorithm consists of two parts:

1. The feature extractor f_θ, the only trainable section of the algorithm. It performs a non-linear mapping of the inputs into an embedding where the representation is trained to be more meaningful for the machine-learning problem;
2. The kernel regressor KR. Not trainable (it merely records the training examples), it outputs a prediction based on the distance between each examples.

As long as the kernel method itself is differentiable, we can back-propagate the empirical risk on the training set through the kernel to the parameters of the feature extractor.

Candidate kernels include the radial basis, polynomial, and linear kernels. Kernel Ridge regression (KRR) and gaussian processes (GP) are two popular and differentiable kernel methods that can be plugged onto the deep kernel learning framework.

Let $\mathbf{K}$ represent the kernel map: $\mathbf{K}_{\boldsymbol{u},\boldsymbol{v}} = \mathrm{k}(f_\theta(\boldsymbol{u}), f_\theta(\boldsymbol{v}))$. Let $\boldsymbol{y}$ represent the stacked labels for the training set.

Kernel Ridge regression KRR minimises the mean squared error of the training prediction, penalising the L^2 norm of the regressor: $\arg\min_{\boldsymbol{a}} \lambda \left\|\boldsymbol{a}\right\|^2 + \left\|\mathbf{K}_{\text{train}\times\text{train}} \cdot \boldsymbol{a} - \boldsymbol{y}\right\|^2$. Its prediction is in the form:

$$\mathrm{KR}^*(\boldsymbol{x'}) = \mathbf{K}_{\boldsymbol{x'}\times\text{train}} \cdot \boldsymbol{a}^* \quad \text{where} \quad \boldsymbol{a}^* = (\mathbf{K}_{\text{train}\times\text{train}} + \lambda\mathrm{Id})^{-1}\boldsymbol{y}$$

Gaussian process When using the negative log likelihood loss function, the GP algorithm gives a probabilistic regressor in the form $\mathrm{GP}(\boldsymbol{x'}) = \mathcal{N}(\mu, \mathbb{C}\mathrm{ov})$. Assuming a normal observation noise of standard deviation σ, the predictive mean μ and the covariance $\mathbb{C}\mathrm{ov}$ are given by:

$$\begin{aligned}\mu &= \mathbf{K}_{\text{test}\times\text{train}} \cdot (\mathbf{K}_{\text{train}\times\text{train}} + \sigma^2\mathrm{Id})^{-1}\boldsymbol{y}\\ \mathbb{C}\mathrm{ov} &= \mathbf{K}_{\text{test}\times\text{test}} - \mathbf{K}_{\text{test}\times\text{train}}(\mathbf{K}_{\text{train}\times\text{train}} + \sigma^2\mathrm{Id})^{-1}\mathbf{K}_{\text{train}\times\text{test}}\end{aligned}$$

Note that in both cases, the predictor is obtained through a differentiable procedure, which means the feature extractor can indeed be optimised during training. However, it does necessitate the full kernel map $\mathbf{K}$, which can become cumbersome to compute should we have a large training dataset.

All in all, the deep kernel learning framework offers a way to drastically increase the expressivity of principled methods that otherwise present nice generalisation properties.

4.2.2 Deep kernel learning for meta-learning

It turns out that deep kernel learning can be adapted to meta-learning very easily. In the few-shot learning setting, we can share the feature extractor across task, while differentiating the last layer. Since DKL uses a regressor that is already task-specific, there is not much work to do!

Figure 4.2 illustrates the minor changes done to the algorithm to be "meta-learning-ready". Notice that we went from a single to a double track: one for training, and the other to test the generalisation performance on examples from the same task. Remember, in meta-learning we are interested in the minimisation of the generalisation error. There is still a meta-test set, there to make sure that the algorithm performs well on completely new tasks.

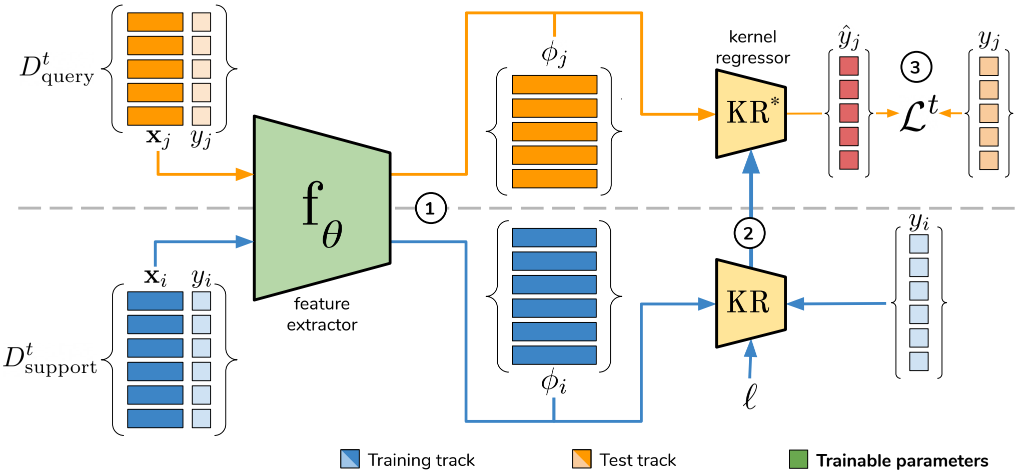


Figure 4.2: Diagram of the adaptive deep kernel learning framework

The procedure depicted in figure 4.2 is also described more formally in algorithm 2.

Algorithm 2 Deep kernel learning for meta-learning

```
Require: $\tau(t)$: distribution over tasks
Require: Model hyper-parameters
Randomly initialize $\theta$
while not done do
    Sample batch $T$ of tasks $t \sim \tau(t)$
    for all $t$ do
        Sample support and query sets $D^t_{\text{support}}$ and $D^t_{\text{query}}$
        ① Map support and query sets to the latent space $f_\theta(D^t_{\text{support}})$ and $f_\theta(D^t_{\text{query}})$
        ② Get the kernel regressor KR$^*$ from $f_\theta(D^t_{\text{support}})$
        ③ Compute the generalisation error $\mathcal{L}(D^t_{\text{query}} \mid \text{KR}^* \circ f_\theta)$
    end for
    Update $\theta$ to minimise the objective $\frac{1}{|T|} \sum_{t \in T} \mathcal{L}(D^t_{\text{query}} \mid \text{KR}^* \circ f_\theta)$
end while
```

To summarise, few-shot DKL aims to find a representation common to all tasks such that the kernel method (in our case, GP and KRR) will generalise well from a small amount of samples. It is worth noting that in doing so, the method alleviates two of the main limitations of single task DKL:

1. The scalability of the kernel method is no longer an issue since we are in the few-

shot learning regime. Even with several hundred samples, the computational cost of embedding each example is usually higher than inverting the Gram matrix.

2. The parameters are learnt across a potentially large amount of tasks and samples, providing the opportunity to learn a rich representation without overfitting.

Despite shared characteristics with the metric learning framework, the few-shot deep kernel learning framework is more powerful and flexible. It provides better task-specific adaptation due to the inference of the appropriate model using the kernel methods compared to shared model parameters in metric learning.

After meta-training, any task-specific model also inherits the generalisation guarantees of kernel-based models, and consequently increasing the number of shots for new tasks can only improve generalisation performance. Moreover, using kernel-based methods enables the incorporation of expert knowledge within the network by choosing user-specific kernel functions, which is also a major advantage of DKL over metric learning. For example, one could use periodic kernels in a regression task that relates to recurrent functions.

In the framework, the feature extractor does not get adapted at test time. Rather, DKL finds a mapping that work as well as possible for every task in the meta-training set, relying on the kernel regressor to obtain good downstream performance.

4.3 Adaptive deep kernel learning

The lack of adaptation at test time is a huge drawback for the deep kernel learning framework. Indeed, relying solely on an adequate feature mapping puts a huge burden on the embedding network, as it is tasked with finding a representation that works well in every case.

Consider the following example. Suppose that you are training a deep kernel learning algorithm to recognise (and complete) sine functions[8], when provided only a handful of examples. This task would be relatively simple for a human to perform: knowing that you are looking at a sine wave, a handful of datapoints is enough to have a good idea of the phase and pulsation of the function. On the other hand, a meta-learning algorithm might struggle:

- For a plain DKL algorithm with no adaptation capabilities, the task will be hard. It needs to figure out a way to spread inputs in a way that lets the kernel regressor make the correct prediction at test time, and thus on an unknown function.
- If your algorithm can adapt to the task, however, it is bound to perform better, having the tools to possibly recognise the characteristics of the sine wave, much like a human would do.

This is the intuition behind adaptive kernel learning framework: we aim to extend the desirable properties, the simplicity and the potential of deep kernel learning, by adding an adaptation component, such that our algorithm can explicitly adapt at test time.

[8] It turns out that very task has become one of the most commonplace testbeds for few-shot regression (see appendix A.1).

The crux of the task adaptation component resides in an additional network that learns a representation of the task itself, using the entire support set as input rather than individual examples. ADKL consists of three independent neural networks that I will indicate with f_θ, g_θ and h_θ in what follows. The networks are different but I decided to use only θ to denote the parameters to limit overloading notations.

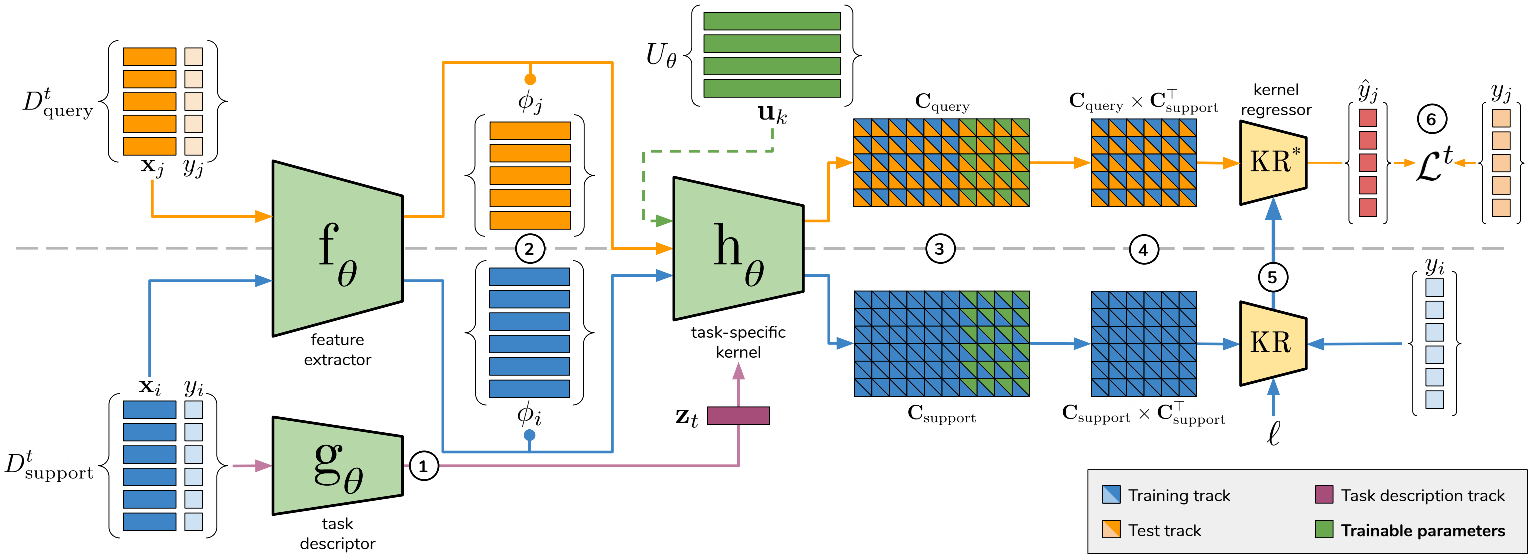


Figure 4.3: Diagram of the adaptive deep kernel learning framework

Figure 4.3 presents the architecture of the adaptive deep kernel learning framework. Let us go through the steps it depicts:

1. Given a task t, ADKL first computes a task embedding $\boldsymbol{z}_t = g_\theta(D^t_{\text{support}})$.
2. The method computes the feature embedding for both D^t_{support} and D^t_{query}, using the feature extractor f_θ.
3. The algorithm produces a task-specific kernel map of the datasets using h_θ.
4. We use empirical kernel mapping to obtain positive semi-definite kernel, useable by the downstream kernel regressor.
5. Using the support set and the choses kernel method, ADKL computes a regressor.
6. The regressor is applied to the query set to get a prediction.

In what follows, I will give more detail on the key steps of the adaptive deep kernel learning procedure.

4.3.1 The task descriptor

The challenge of the task descriptor network g_θ is to capture complex dependencies in the training set to provide a useful task encoding $\boldsymbol{z}_t$. That network should be invariant to permutations of the training set and be able to encode a variable amount of samples. After exploring a variety of architectures, we found that more complex solutions, such as Transformers (Vaswani et al., 2017), tend to underperform –possibly due to overfitting or the sensitivity of training such architectures.

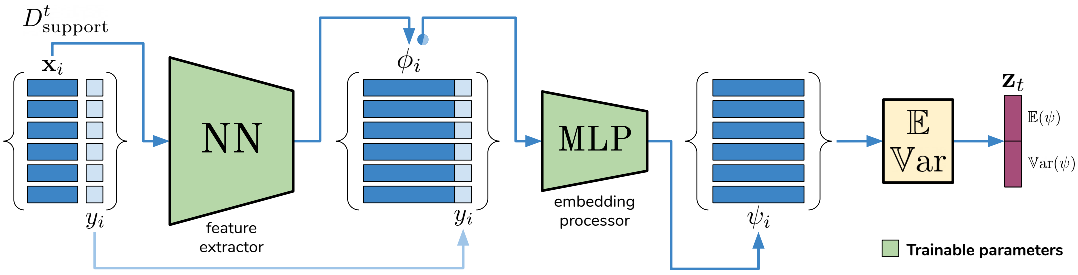


Figure 4.4: Diagram of proposed task descriptor

Consequently, inspired by the work on "DeepSets" by Zaheer et al. (2017), we propose a simple order invariant network that captures the first and second order statistics of regression datasets. The procedure of the proposed method, presented in figure 4.4, is as follows. Given a dataset D^t_{support}, the network:

a) processes the samples individually via a neural network module (NN);
b) concatenates input features with the target and embed the obtained vector using a simple multi-layer perceptron (MLP) model;
c) computes the element-wise first and the second order statistics of the results, and concatenates them to produce the representation.

More formally,

$$\boldsymbol{z}_t = \text{Concat}\left[\mu_t, \sigma_t^2\right] \quad \text{with} \quad \mu_t = \mathbb{E}_{D^t_{\text{support}}}\left[\psi\right] \qquad \sigma_t^2 = \mathbb{V}\text{ar}_{D^t_{\text{support}}}\left[\psi\right] \tag{4.3.1}$$

$$\text{where} \quad \psi_i = \text{MLP}_\theta(\text{Concat}\left[\phi_i, y_i\right]) \quad \phi_i = \text{NN}(\boldsymbol{x}_i) \tag{4.3.2}$$

using the same notation as figure 4.4. The variance operator is applied element-wise.

During training, the task descriptor sees only one example per task, compared to the other parts of ADKL which are trained on batches of samples for each task. Hence, training is significantly slower and less stable. To help train the task descriptor, we add a regularisation term that maximises the mutual information between $g_\theta(D^t_{\text{support}})$ and $g_\theta(D^t_{\text{query}})$. This penalisation encourages the network to produce similar task encodings when presented different data partitions for a given task. In practice, we maximize a lower bound on the mutual information between the task representations given by the support and the query sets (Belghazi et al., 2018).

Using a batch T of tasks and the cosine similarity c as the similarity measure the between two task encodings, the lower bound $\widehat{\text{MI}}$ is defined as:

$$\widehat{\text{MI}} \overset{\text{def}}{=} \frac{1}{|T|}\sum_{t\in T} c(\boldsymbol{z}^t_{\text{s}}, \boldsymbol{z}^t_{\text{q}}) - \log\left[\frac{1}{|T|(|T|-1)}\sum_{t\in T}\sum_{\substack{t'\in T\\ t'\neq t}} \exp\left[c(\boldsymbol{z}^t_{\text{s}}, \boldsymbol{z}^{t'}_{\text{q}})\right]\right], \tag{4.3.3}$$

where $\boldsymbol{z}^t_{\text{s}}$ and $\boldsymbol{z}^t_{\text{q}}$ are the task description for task t computed on the support and query set respectively.

4.3.2 Task-specific kernel

I just presented how the task descriptor that we propose is able to provide an encoding of the current task, as well as the solution we advocate to train it better, namely a regularisation that maximises the mutual information between the description in different partitions of the same dataset. One question remains: how does ADKL make use of that task description to adapt itself? The answer resides in the task-specific kernel.

In deep kernel learning, the kernel is chosen in advance, and the only way to help the network adapt to novel tasks is to increase the capacity of the feature extractor. With ADKL, to the contrary, the kernel is computed using a trainable network that takes the task descrition as input to explicitly adapt to the current task.

The approach we propose uses a multi-modal neural network h_θ. Given a pair of input representations $\phi \overset{\text{def}}{=} f_\theta(\boldsymbol{x})$ and $\phi' \overset{\text{def}}{=} f_\theta(\boldsymbol{x}')$ and a task encoding $\boldsymbol{z}_t$, h_θ computes the input pair similarity parametrised by the task encoding as follows:

$$h_\theta(\phi, \phi' \mid \boldsymbol{z}_t) = \mathrm{MLP}_\theta\left(\mathrm{Concat}\left[\left\|\phi - \phi'\right\|^2_{\text{element}}, \boldsymbol{z}_t\right]\right), \tag{4.3.4}$$

where $\|\cdot\|^2_{\text{element}}$ is the element-wise L^2 distance and MLP_θ is a multilayer perceptron whose last layer is a single neuron, such that it returns a scalar value.

Notice that with this definition, h_θ is symmetric and stationary, meaning that the output only depends on the element-wise distance between ϕ and ϕ' (and the task description). However, the upstream processing makes the full ADKL pipeline non-stationary with respect to the actual inputs. The task-specific flavour of the module comes in the most simplistic way, by concatenating the task representation to the input of the multi-layer perceptron network, insuring that the kernel will indeed adapt to the current task.

However, the output $\mathbf{C}$ of the network is not positive semi-definite (PSD) and cannot be directly used for downstream kernel regressors such as KRR and GP. Therefore, we use empirical kernel mapping (Schölkopf et al., 1999) to compute the task-specific PSD kernel $\mathbf{K}$ associated with a given task representation $\boldsymbol{z}_t$ obtained from D^t_{support}.

Moreover, using the empirical kernel map of $\mathbf{C}$ to compute $\mathbf{K}$ offers the opportunity to introduce *pseudo-representations* that can improve the kernel evaluations, especially in low data settings. To wit, instead of computing the empirical kernel map with regard to $f_\theta(D^t_{\text{support}})$ alone, we use $S \overset{\text{def}}{=} f_\theta(D^t_{\text{support}}) \cup U_\theta$ where U_θ is a trainable set of pseudo-representations.

The final kernel map becomes:

$$\mathbf{K}_{\mathrm{q}\times\mathrm{s}} = \mathbf{C}_\mathrm{q} \times \mathbf{C}_\mathrm{s}^\top \qquad \text{and} \qquad \mathbf{K}_{\mathrm{s}\times\mathrm{s}} = \mathbf{C}_\mathrm{s} \times \mathbf{C}_\mathrm{s}^\top \tag{4.3.5}$$

where

$$\mathbf{C}_\mathrm{s} = \left\{h_\theta(\phi, \phi' \mid \boldsymbol{z}_t)\right\}_{\phi,\phi' \in f_\theta(D^t_{\text{support}})\times S} \quad \text{and} \quad \mathbf{C}_\mathrm{q} = \left\{h_\theta(\phi, \phi' \mid \boldsymbol{z}_t)\right\}_{\phi,\phi' \in f_\theta(D^t_{\text{query}})\times S}$$

The subscripts s and q stand for “support” and “query” respectively.

The number of pseudo-representations is a hyper-parameter of ADKL (in our experiments we chose $|U_\theta| \in [0, 50]$) and the set of pseudo-representations U_θ is a learnable parameters that is shared across tasks and learned during meta-training. To prevent them from collapsing into a single point during training and to ensure that they are well distributed in the feature space, we add a regularisation term (D_U) to the training loss function.

To introduce this regularisation term, let us consider p and q to be the distributions that generate the true input representations and the pseudo-representations, respectively. We make the assumption that p and q are both multivariate Gaussian distributions with diagonal covariance matrices and have respective parameters $(\mu_\phi,\, \sigma^2_\phi)$ and $(\mu_U,\, \sigma^2_U)$.

The parameters of p can be estimated using the running mean and variance of all input representations computed over batches of tasks. Those of q are estimated using U directly. Hence, we regularise U to constrain q to be as close to p as possible, such that the pseudo-representation effectively map the representation space. To that end, we add a regularisation term that minimises the Kullback-Leibler (KL) divergence between p and q:

$$D_U = \mathrm{KL}(\mathcal{N}(\mu_U, \sigma^2_U) \parallel \mathcal{N}(\mu_\phi, \sigma^2_\phi)) \tag{4.3.6}$$

which has a closed-form solution since both distributions are Gaussian.

Hence, the full training objective for ADKL is:

$$\arg\min_\theta \mathbb{E}_{t \in \mathcal{D}^{\text{meta}}_{\text{train}}} \left[\mathcal{L}^t - \gamma_{\text{task}} \widehat{\text{MI}} + \gamma_{\text{pseudo}} D_U\right] \tag{4.3.7}$$

The complete ADKL algorithm is described in algorithm 3, referencing the steps featuring in figure 4.3.

Algorithm 3 Adaptive deep kernel learning

```
Require: $\tau(t)$: distribution over tasks
Require: $\gamma_{\text{task}}$, $\gamma_{\text{pseudo}}$, $|U|$: model hyper-parameters
Randomly initialize $\theta$
while not done do
    Sample batch $T$ of tasks $t \sim \tau(t)$
    for all $t$ do
        Sample support and query sets $D^t_{\text{support}}$ and $D^t_{\text{query}}$
        ① Compute the task representation $\boldsymbol{z}_t \overset{\text{def}}{=} g_\theta(D^t_{\text{support}})$
        ② Map support and query sets to the latent space $f_\theta(D^t_{\text{support}})$ and $f_\theta(D^t_{\text{query}})$
        ③ Compute the task-specific kernel using $h_\theta$
        ④ Compute the empirical kernel maps
        ⑤ Get the kernel regressor KR$^*$ from $\mathbf{K}_{\text{s}\times\text{s}}$
        ⑥ Compute the generalisation error $\mathcal{L}(D^t_{\text{query}} \mid \theta, D^t_{\text{support}})$
    end for
    Update $\theta$ to minimise the objective $\frac{1}{|T|}\sum_{t\in T} \mathcal{L}^t - \gamma_{\text{task}}\widehat{\text{MI}} + \gamma_{\text{pseudo}} D_U$
end while
```

4.4 Experiments

Now that we have looked at the theory behind ADKL, it is time to focuse on the actual performance of the framework.

To do that, we worked with three dataset collections. The first, `Sinusoids`, is synthetic. Initially proposed by Kim et al. (2018), it consists of a set of sine wave regression tasks: the model is provided five, ten or twenty samples from the $[-5, 5]$ interval, and is tasked with interpolating the function. The two other collections, `Binding` and `Antibacterial`, report real-world bioassay data coming from the public domain, and pre-processed by InVivo AI.

All three collections are presented in much more detail in appendix A, along with additional resource to access the datasets.

In the following paragraphs, I will first present the results of an ablation study that helped us understand the influence of the different hyper-parameters. Then, we will look at a benchmarking analysis to show how our framework compares to other meta-learning methods that I will introduce. Finally, we will focus on the active-learning capabilities of ADKL.

I will differentiate the kernel regressor used by the framework, using the denominations "ADKL-KRR" and "ADKL-GP". The former uses a kernel Ridge regressor while the latter implements a gaussian process estimator.

4.4.1 Ablation study

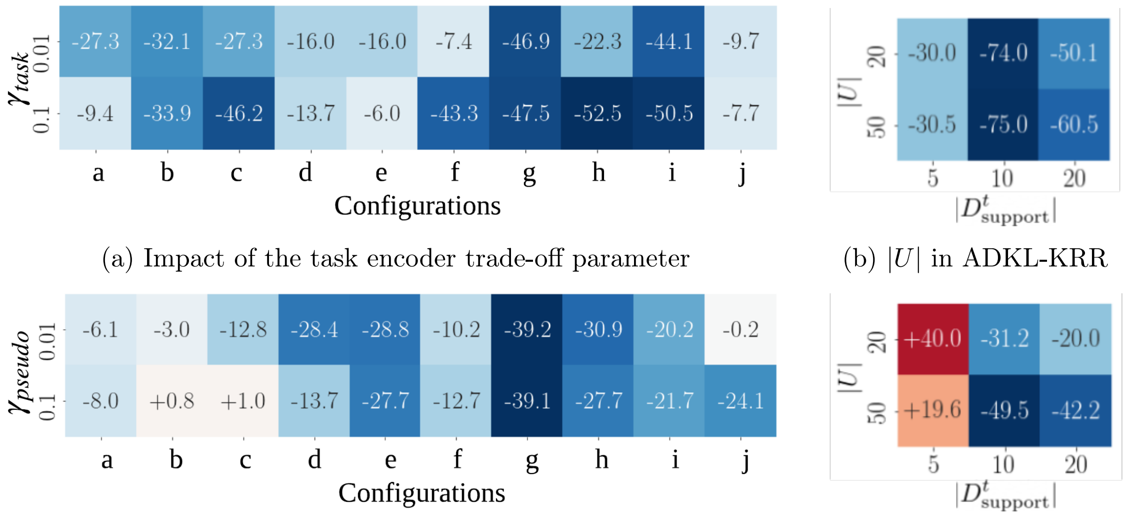
The role of the ablation study is paramount whenever a team proposes a new framework. Indeed, it is of the utmost importance that the community can assess the effect of the different hyper-parameters, particularly when the method introduces new ideas such as a task descriptor trained with mutual information.

Hence, we set out to closely evaluate the impact of the task encoder and the pseudo-inputs on the meta-generalisation error. We focused on the `Sinusoids` collection, since unlike the other two meta-datasets at our disposal, it provides a synthetic environment where the assumptions of meta-learning are entirely satisfied: we want to evaluate the impact of the parameters, not the flawed environment.

Thus, we trained and evaluated ADKL on `Sinusoids` with different hyper-parameter combinations. Figure 4.5 shows the relative improvements in the meta-test MSE compared to different baselines. The specific configurations can be found in appendix B.

First, figure 4.5a, by comparing $\gamma_{\text{task}} \in \{0.01, 0.1\}$ relative to $\gamma_{\text{task}} = 0$, figure 4.5a demonstrates that regularizing the task encoder significantly improves the generalisation performance. Other results presented in appendix B.1 show that the conclusion holds for all support set sizes tested. It confirms the importance of using a task representation for generalisation in few-shot learning and the relevance of the regularisation term we introduced.

Then, figure 4.5c shows that improving the kernel map evaluations using *pseudo-input representations* can significantly help the meta-generalisation performance of ADKL. The observation holds no matter the number of support examples (see appendix B.2), although



(a) Impact of the task encoder trade-off parameter

(b) $|U|$ in ADKL-KRR

(c) Impact of the pseudo inputs trade-off parameter

(d) $|U|$ in ADKL-GP

Figure 4.5: Relative decrease/increase in the meta-test MSE compared to no regularisation (lower is better)

the improvements were more consistent for smaller support sets: improving the kernel map estimations in these cases is more critical.

Finally, figures 4.5b and 4.5d illustrate how the number of pseudo-representations affects performance of ADKL-KRR and ADKL-GP differently. In general, we can confirm that adding more pseudo-representations improves generalisation. However, the improvements are more prominent with KRR in comparison to GP. It may be due to the fact that GP attributes a part of the modelling noise to the kernel evaluations, leading to more constraints on the optimisation of the pseudo-representation parameters.

We just went through the individual effect of the two hyper-parameters we introduced as part of the framework, see appendix B for an analysis of their joint impact.

4.4.2 Benchmarking analysis

We compared ADKL to a number of other methods that are related to our work. We first compared the performance on the synthetic collection `Sinusoids`, and susbsequently on two real-world molecular collections, `Binding` and `Antibacterial`

Conditional Neural Processes (CNPs), proposed by Garnelo et al. (2018), learn conditional stochastic processes parameterised by task-specific conditions derived from the support sets. By comparison, ADKL-GP also learns conditional stochastic processes but has mathematical guarantees thanks to gaussian processes and PSD kernels.

Few-shot deep kernel learning (R2D2) Proposed by Bertinetto et al. (2018), R2D2 is the meta-learning version of deep kernel learning seen in section 4.2.2. They tackle the lack of adaptation for new tasks by using Ridge regression, but do not adapt the kernel itself at test time.

Learned Basis Function, developed by Loo et al. (2019), is linked to deep kernel learning since its output is a linear combination of the processed support examples. However, the architecture is trained to produce a task descriptor that is used to generate the weights for the final regressor directly. Hence, it adapts to the task, but does not explicitly train the task descriptor as ADKL does, and loses the principled approach of kernel regression by outputing weights directly.

MAML (see section 3.2.4)

Bayesian MAML (BMAML): Kim et al. (2018) proposed a Bayesian variant of MAML where a feature extractor is shared across tasks, while multiple MAML particles are used for the task-level adaptation.

These algorithms have all proven to have efficient and effective test-time adaptation routines and therefore constitute strong baselines for benchmarking.

For bioassay modelling benchmarks, we looked at the performance using solely ECFP4 (Extended Connectivity FingerPrints of diameter 4) as input representation. We made that decision to put every method on an equal footing and avoid letting the representation impact the performance. Moreover, we also compared ADKL to two single-task methods considered to be state-of-the-art in chemoinformatics (Olier et al., 2018):

a) ECFP4 with random forest (ECFP4+RF)
b) ECFP4 with kernel Ridge regression using tanimoto similarity as a kernel function (ECFP4+KRR)

At test time, we computed the mean squared error (MSE) of every method on each task thirty times, using different support/query partitions (the partitions were the same for every method). This scheme mitigates the pobability that the result of a particular method on a task is obtained through blind luck, since the returned performance is averaged over thirty independent runs. In the following, I report the MSE averaged over every tasks.

Synthetic data Table 2 shows that ADKL achieves the best performance in every few-shot learning regime. I also provide a visualisation of the predictions from each model in appendix C.

$\lvert D^t_{\text{support}} \rvert$	5	10	20
BMAML	2.042	1.371	0.844
CNP	1.616	0.392	0.117
Learned Basis	3.587	0.800	0.127
MAML	2.896	1.634	0.901
ADKL-GP	1.178	0.084	0.007
ADKL-KRR	**0.867**	**0.061**	**0.005**
R2D2	1.002	0.073	0.009

Table 2: Average MSE on `Sinusoids`

We note that the kernel methods perform much better in this scenario. One interpretation is that using kernels, ADKL and R2D2 are able to construct an embedding where sine functions are favoured. By contrast, initialisation-based methods such as MAML need an extremely expressive model to produce an initialisation that can properly adapt to sine waves of different characteristics.

Real-world bioassays Tables 4 and 3 present the results on the `Antibacterial` and `Binding` collections, respectively.

$\lvert D^t_{\text{support}} \rvert$	5	10	20
ADKL-KRR	0.0380 ± 0.0020	0.0348 ± 0.0009	0.0322 ± 0.0020
BMAML	0.0813 ± 0.0571	0.0486 ± 0.0071	0.0487 ± 0.0012
CNP	0.0416 ± 0.0019	0.0393 ± 0.0030	0.0397 ± 0.0027
ECFP4+KRR	0.0376 ± 0.0012	0.0352 ± 0.0014	0.0317 ± 0.0016
ECFP4+RF	**0.0373 ± 0.0012**	**0.0339 ± 0.0013**	**0.0311 ± 0.0012**
LearnedBasis	0.0761 ± 0.0040	0.0754 ± 0.0042	0.0616 ± 0.0215
R2D2	0.0492 ± 0.0015	0.0460 ± 0.0110	0.0342 ± 0.0012

Table 3: Average MSE on `Binding`

Table 3 shows a sad truth about computational drug discovery: datasets are so noisy and and ill-prepared that in some cases, learning across task can have no effect on the results. In this case, the best estimator of molecular binding is the single-task algorithm random forest using ECFP4 fingerprints. However, we do note that ADKL is a close second, performing significantly better than other methods.

$\lvert D^t_{\text{support}} \rvert$	5	10	20
ADKL-GP	0.1017 ± 0.0013	0.0895 ± 0.0015	**0.0860 ± 0.0016**
ADKL-KRR	**0.1000 ± 0.0012**	**0.0893 ± 0.0015**	0.0862 ± 0.0009
BMAML	0.1059 ± 0.0021	0.1020 ± 0.0029	0.4616 ± 0.4210
CNP	0.1063 ± 0.0023	0.1239 ± 0.0219	0.1382 ± 0.0049
ECFP4+KRR	0.1166 ± 0.0020	0.1003 ± 0.0009	0.0956 ± 0.0009
ECFP4+RF	0.1129 ± 0.0002	0.1016 ± 0.0008	0.0970 ± 0.0003
LearnedBasis	0.1274 ± 0.0037	0.1308 ± 0.0032	0.1329 ± 0.0043
R2D2	0.1104 ± 0.0023	0.0962 ± 0.0021	0.0921 ± 0.0010

Table 4: Average MSE on `Antibacterial`

On the other hand, table 4 confirms that in some cases, using meta-learning can indeed help. When and why meta-learning can perform better is an open question and very much an active field of research. In the meantime, computer biochemists need to test both approach before they can settle for one, provided they have enough data to spare on a validation study...

4.4.3 Active learning

Active learning is a paradigm where the trained algorithm is provided both labelled and unlabelled examples. It can interact with the teacher to query new labels as learning progresses, such that queried samples can maximise the information gathered about the task. This setting is particularly relevant in a biochemistry context, where labelling is slow and costly: using active learning, experimenters can choose to prioritise molecules that will incur more information for the trained algorithm. With a well-tuned active-learning algorithm, biochemists could make *in vitro* testing on a handful of molecules, and then explore the rest of the molecular space using the queries from the model.

For an algorithm to be useable in an active learning setting, it needs to be well-calibrated. To wit, it should be able to assess its level of confidence on the predictions is produces. ADKL, when used with a gaussian process estimator, provides a predictive variance and can therefore be used for active learning. Hence, we tested ADKL-GP in this setting on our three collections. We compared the results with those of CNP, since it also output a predictive distribution.

During the experiment, training was performed using support sets of five examples. At test-time, we randomly partition each task from the meta-test set into three subsets:

a) A support set D^t_{support}, containing only five examples;

b) A query set D^t_{query}, to compute the generalisation error;

c) A pool of unlabelled data D_{u}.

First, the algorithm adapts to the task using D^t_{support} to constitute the initial hypothesis. Then, it updates iteratively by querying the most informative sample from D_{u}, in our case using a maximum entropy criterion. The iterative adaptation continues until the allocated budget runs out or the unlabelled pool is entirely consumed. These steps are presented more formally in algorithm 4.

Algorithm 4 Active learning on a new task t

```
Require: learning algorithm with a probabilistic output
Require: $l$: labelling function
Require: $D^t_{\text{support}}$, $D^t_{\text{query}}$ and $D_{\text{u}}$: support, query and unlabelled sets
Require: $b$: the query budget
Adapt algorithm to the task using $D^t_{\text{support}}$
while $b > 0$ or $D_{\text{u}}$ not consumed do
    Select maximum entropy sample $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in U} \mathbb{E}\left[\log p(y \mid \boldsymbol{x}, D^t_{\text{support}})\right]$
    Update $D^t_{\text{support}} \leftarrow D^t_{\text{support}} \cup \{\boldsymbol{x}^*, l(\boldsymbol{x}^*)\}$
    Update $D_{\text{u}} \leftarrow D_{\text{u}}\{\boldsymbol{x}^*\}$ and $b \leftarrow b - 1$
    Re-adapt algorithm using the updated support set.
end while
```

In the following, we report the results obtained for query budgets up to $b = 20$. In the case of the `Sinusoidal` collection, we could spare repeating the experiments a number of times in order to obtain a standard deviation. However, it was too time-consuming and costly to do so on `Binding` or `Antibacterial`, given the complexity of the networks for this task.

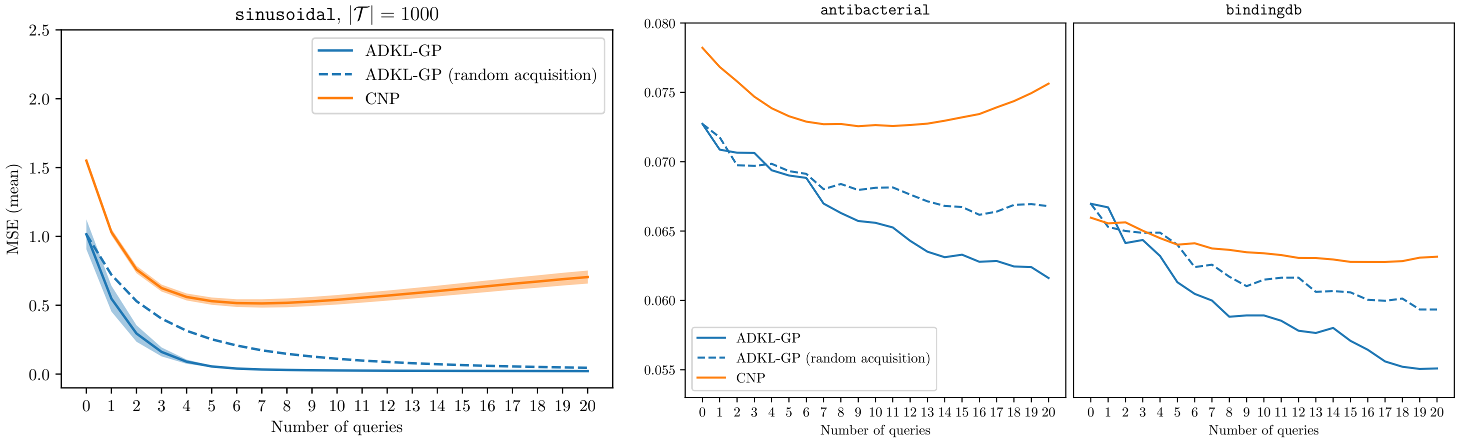


Figure 4.6: Average MSE performance with active learning for query budgets up to $b = 20$

Figure 4.6 illustrates that under the active learning strategy, ADKL-GP consistently outperforms CNP. In particular, we observe that very few samples are queried by ADKL-GP to capture the data distribution whereas CNP performance quickly plateaus.

Moreover, since using the maximum predictive entropy strategy is better than querying samples at random (solid vs. dashed line), these results suggest that the predictive uncertainty provided by ADKL-GP is informative and the algorithm is relatively well calibrated.

When the number of queries is greater than 10, we observe a performance degradation for CNP while ADKL-GP remains consistent. This observation highlights the generalisation capacity of deep kernel learning methods, even outside the few-shot regime where they have been trained —a property that does not hold true for CNP.

In conclusion, the adaptive deep kernel learning framework is a simple yet powerful novel method that is able to accumulate knowledge across a number of related tasks, and leverage this knowledge at test-time to outperform other methods on synthetic testbeds but also –and more importantly for InVivo AI– when it comes to evaluating biochemical properties in a few-shot learning setting. What is more, ADKL is well-calibrated, making it a well-suited framework for computational biochemistry since it cannot only predict the correct output from a very narrow set of examples but also guide the costly exploration of the chemical space to optimize the information obtained from *in vitro* testing.

Conclusion

Computational drug discovey is challenging. The subject matter itself is extremely complex: molecules form elaborate structures whose basic properties, let alone their effect on human health, are inherently tricky to predict and would necessitate extremely expressive machine-learning algorithms, complex enough to extract deeply nested information. Such algorithm would need clean and abundant data so they can be trained, but the sad truth about drug discovery is that in this field, data is neither plentiful nor orderly.

However, datasets do come in numbers: we may not have large sets of examples related to one given task, but we do have a few datapoints for a huge number of bioassays. Hence, we can hope to tackle computational drug discovery through two ways: we can either start producing clean and abundant data, requiring massive investments, or we may work to capture information from every source at our disposal.

At InVivo AI, we have chosen the second route and strive to push the boudaries of computational drug discovery ever further, through intense efforts in both applied and fundamental research. As an intern there, I worked on the bleeding edge of research in meta-learning, a paradigm that seeks to learn how to learn from multiple tasks at once. I took part in developing ADKL (adaptive deep kernel learning), a novel framework that explicitly adapts to new tasks at test time. The method is a step in the right direction and a good candidate to lead the advent of computational drug discovery, but it is certainly not the final answer.

There is still much to understand about meta-learning. From the influence of the task distribution to the causes of negative adaptation, the inner workings of learning to learn remain largely a mystery. During my internship, I tried to bring some answers to these fundamental questions.

Through this report, I wanted to introduce the reader to the world of computational biochemistry, since I expect they will mostly be coming from the same background as I did, namely mathematics and computer science. I pointed at on-going research in the field of meta-learning that tackle the same kind of issues that drug discovery faces, and I presented the research I did working on ADKL.

Acknowledgement

I would like to express my deepest gratitude to my supervisor, Prudencio Tossou, for his patient guidance, enthusiastic encouragements and useful critiques. He provided me with precious advice, corrected my code and nudged me in the right direction every time I needed it.

In fact, I wish to thank the entire team at InVivo AI, who made this internship a formidable experience personally as much as professionally. Their continued technical and emotional support got me through the hard times rushing for a deadline or waiting for a review. I am particularly thankful for the many ping-pong and coffee breaks that kept my sanity in checks in these times of turmoil. My experience of this internship would have been very different had it not been for them.

My internship was also made a wonderful experience by my friends and collegues, from Mila and elsewhere. Julien, Paul, Mandana, Clément and Tristan, in particular, helped me a great deal over the last eight months, giving me perspective, letting me think out loud and providing me with numerous insights that unlocked me more than once.

I would also like to address special thanks to Cem Subakan and Aurélie Hélouis from Mila. Cem gave me guidance during the internship, offering his research acumen and standing for the numerous changes in the project we were working on together. Aurélie helped me a great deal throughout my Master's at Mila, and especially over the course of my internship.

Last but not least, I want extend my thanks to the people of the Mitacs Acceleration Program who deserve praise for making the internship a possibility. They have worked hard to provide diligent assistance and funding, and I am extremely grateful to them.

References

M. I. Belghazi, A. Baratin, S. Rajeswar, S. Ozair, Y. Bengio, A. Courville, and R. D. Hjelm. Mine: mutual information neural estimation. *arXiv preprint arXiv:1801.04062*, 2018.

L. Bertinetto, J. F. Henriques, P. H. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

R. S. Bohacek, C. McMartin, and W. C. Guida. The art and practice of structure-based drug design: a molecular modeling perspective. *Medicinal research reviews*, 16(1):3–50, 1996a.

R. S. Bohacek, C. McMartin, and W. C. Guida. The art and practice of structure-based drug design: A molecular modeling perspective. *Medicinal Research Reviews*, 16(1):3–50, 1996b.

A. Cherkasov, E. N. Muratov, D. Fourches, A. Varnek, I. I. Baskin, M. Cronin, J. Dearden, P. Gramatica, Y. C. Martin, R. Todeschini, et al. Qsar modeling: where have you been? where are you going to? *Journal of medicinal chemistry*, 57(12):4977–5010, 2014.

Daylight Chemical Information Systems, Inc. Fingerprints, 2019. URL https://www.daylight.com/dayhtml/doc/theory/theory.finger.html.

T. Deleu and Y. Bengio. The effects of negative adaptation in model-agnostic meta-learning. *arXiv preprint arXiv:1812.02159*, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

Fdardel and DMacks. Smiles, 2007. URL https://commons.wikimedia.org/w/index.php?curid=2556784. [Original by Fdardel, slight edit by DMacks, re-edited by the author].

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*, pages 1126–1135. JMLR. org, 2017.

V. Garcia and J. Bruna. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*, 2017.

M. Garnelo, D. Rosenbaum, C. J. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. Eslami. Conditional neural processes. *arXiv preprint arXiv:1807.01613*, 2018.

G. B. Goh, C. Siegel, A. Vishnu, N. O. Hodas, and N. Baker. Chemception: a deep neural network with minimal chemistry knowledge matches the performance of expert-developed qsar/qspr models. *arXiv preprint arXiv:1706.06689*, 2017.

Janssen. Addressing the challenges of drug discovery. https://www.janssen.com/emea/drug-discovery, 2019.

A. Kensert, J. Alvarsson, U. Norinder, and O. Spjuth. Evaluating parameters for ligand-based modeling with random forest on sparse data sets. *Journal of Cheminformatics*, 10 (1):49, 2018.

S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, et al. Pubchem 2019 update: improved access to chemical data. *Nucleic acids research*, 47(D1):D1102–D1109, 2019.

T. Kim, J. Yoon, O. Dia, S. Kim, Y. Bengio, and S. Ahn. Bayesian model-agnostic meta-learning. *arXiv preprint arXiv:1806.03836*, 2018.

G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, 2015.

T. Liu, Y. Lin, X. Wen, R. N. Jorissen, and M. K. Gilson. Bindingdb: a web-accessible database of experimentally determined protein–ligand binding affinities. *Nucleic acids research*, 35(suppl_1):D198–D201, 2007.

Y. Loo, S. K. Lim, G. Roig, and N.-M. Cheung. Few-shot regression via learned basis functions. *open review preprint:r1ldYi9rOV*, 2019.

S. Min, B. Lee, and S. Yoon. Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869, 2017.

I. Olier, N. Sadawi, G. R. Bickerton, J. Vanschoren, C. Grosan, L. Soldatova, and R. D. King. Meta-qsar: a large-scale application of meta-learning to drug design and discovery. *Machine Learning*, 107(1):285–311, 2018.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations*, 2016.

D. Rogers and M. Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010. doi: 10.1021/ci100050t. URL https://doi.org/10.1021/ci100050t. PMID: 20426451.

B. Sanchez-Lengeling and A. Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018. ISSN 0036-8075. doi: 10.1126/science.aat2663. URL https://science.sciencemag.org/content/361/6400/360.

B. Schölkopf, S. Mika, C. J. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. J. Smola. Input space versus feature space in kernel-based methods. *IEEE transactions on neural networks*, 10(5):1000–1017, 1999.

M. H. Segler and M. P. Waller. Neural-symbolic machine learning for retrosynthesis and reaction prediction. *Chemistry–A European Journal*, 23(25):5966–5971, 2017.

M. H. Segler, M. Preuss, and M. P. Waller. Learning to plan chemical syntheses. *arXiv preprint arXiv:1708.04202*, 2017.

J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston. Random forest: a classification and regression tool for compound classification and qsar modeling. *Journal of chemical information and computer sciences*, 43(6):1947–1958, 2003.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL http://arxiv.org/abs/1409.4842.

E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, and H. Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples, 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

D. Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988. doi: 10.1021/ci00057a005. URL https://pubs.acs.org/doi/abs/10.1021/ci00057a005.

Y. Xu, K. Lin, S. Wang, L. Wang, C. Cai, C. Song, L. Lai, and J. Pei. Deep learning for molecular generation. *Future medicinal chemistry*, 11(6):567–597, 2019.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

A Few-shot regression collections

There is a dire need in the meta-learning community for standardised benchmarks, especially in the case of few-shot regression. To wit, meta-learning research is dominated by image classification, owing in part to the fact that such problems validate the assumptions of the field quite well – learning to recognise shapes can indeed be learned accross a variety of classification tasks.

Thus, in the case of image classification, benchmarks exist. For instance, Triantafillou et al. from Google Research have proposed in 2019 "Meta-Dataset", a collection of image classification datasets to be used as a testbed by the community.

On the contrary, the few-shot regression community mostly relies on *ad hoc* meta-datasets, with little systematic benchmarking other than on synthetic data. I for one believe that synthetic testing is most important in fundamental research, since it helps pinpoint the capabilities of an algorithm by controlling explicitly the assumptions one can make on the data. However, we do need to challenge ourselves to real-world applications, lest meta-learning remain a niche and unapplicable discipline.

In what follows, I describe four meta-datasets we work with at InVivo AI.

A.1 A synthetic meta-dataset : the `Sinusoids` collection

This synthetic few-shot regression benchmark, introduced by Kim et al. (2018), consists of 5,000 tasks defined by functions of the form :

$$y = A\sin(\omega x + b) + \varepsilon \qquad \text{with} \qquad \begin{gathered} A \in [0.1, 5.0] \\ b \in [0.0, 2\pi] \\ w \in [0.5, 2.0] \\ \varepsilon \sim \mathcal{N}\left(0, (0.01A)^2\right) \end{gathered} \tag{A.1.1}$$

The examples are generated by sampling inputs x from the $[-5.0, 5.0]$ interval, as well as the observation noise ε. The tasks themselves are divided into a meta-train, a meta-validation and a meta-test sets, which contain respectively 56.25%, 18.75% and 25% of the functions.

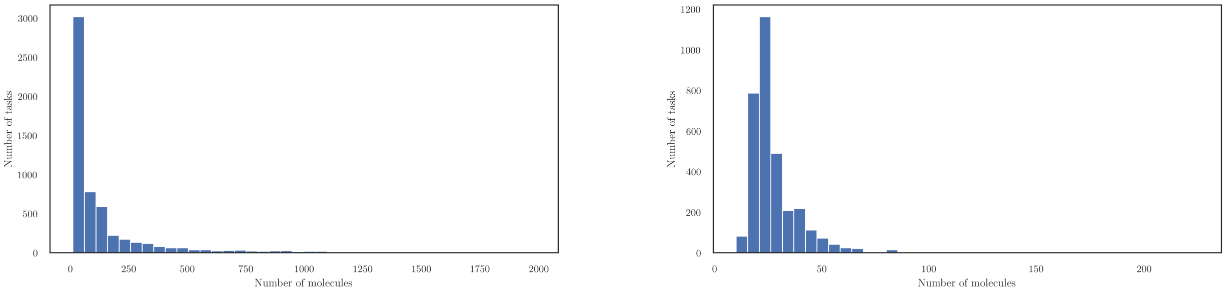
By design, this synthetic collection validates the assumptions of meta-learning : the functions are indeed sampled from a task distribution, and the goal of the algorithm is to learn to recognise the parameters of the sine function from very few points.

The `Sinusoids` collection, although it may be given other names, has become a standard benchmarking tool in the community.

A.2 `Binding` and `Antibacterial`, two real-world collections

One important flaw in the `Sinusoids` collection is its lack of realism. Moreover as a drug discrovery company, InVivo AI strive to develop algorithms that work on real-world, bio-chemical data.

To that end we propose two collections, whose content is available here, compiled from data in the public domain. Both contain data from bioassays that are representative of real-world few-shot regression tasks that arise in drug discovery.



(a) `Binding`

(b) `Antibacterial`

Figure A.1: Distribution of the number of examples in each task for the `Binding` and `Antibacterial` collections

Figure A.1a shows just how scarce data is for both collections, reflecting the time and money needed to perform an exhaustive mapping of the molecular space using *in vitro* and *in vivo* testing.

A.2.1 `Binding`

This collection is extracted from the public database BindingDB (Liu et al., 2007). We altered it by removing bio-assays with targets correlated above 0.8 or those with less than 10 experimental measurements, leaving 5,717 tasks.

Each task from the collection reports the binding affinity of small molecules to a target protein. Hence, the task distribution is defined by the characteristics of the proteins.

A.2.2 `Antibacterial`

This task collection is extracted from PubChem (Kim et al., 2019), a public registery of molecules and bioassays. After also removing bio-assays with correlations above 0.8 and those with less than 10 samples, we obtain $3,255$ tasks.

Each task in `Antibacterial` reports the antibacterial effect of a set of small molecules on a given strain of bacterium. More specifically, it describes the minimum concentration needed to wipe out at least half the population of bacteria.

Note that this collection is particularly noisy. Indeed, the results are obtained by testing different concentrations of the same molecules, leading to discrete approximation riddled with thresholding effects.

A.3 A synthetic collection of molecular data : `Properties`

The noise issue in the two bioassay collections led us at InVivo AI to create another meta-dataset, which we called `Properties`.

Although `Properties` is a collection of synthetic tasks, it still relates to drug discovery. Hence, is reports a set of 767 computable physico-chemical properties for more than 600,000 molecules, thus completely removing the noise issue while providing a rich benchmarking tool to computational bio-chemists.

With properties, the community can test out algorithms specifically designed for molecular scoring with the challenges it entails (such as the representation issue), while still benefiting from a very large dataset.

B Ablation study

This section presents more results from the ablation study. See the ICLR submission for a more comprehensive presentation.

B.1 Task regularisation

Table 5 presents the hyper-parameter combinations used in the experiments to assess the impact of the trade-off parameter γ_{task}. We report the MSE performance obtained on the meta-test for each configuration.

				γ_{task}	0.00	0.01	0.10
Conf.	algorithm	K	architecture	γ_{pseudo}			
a	ADKL-KRR	20	attention	0.01	0.0585	0.0327	**0.0289**
b		10	deepset	0.00	0.4051	**0.2944**	0.3671
c				0.10	0.4363	0.2964	**0.2882**
d	ADKL-GP	5	attention	0.10	2.4920	**2.2511**	2.2994
e	ADKL-KRR	20	attention	0.00	0.0574	0.0305	**0.0302**
f	ADKL-GP	5	attention	0.01	2.5611	**2.1511**	2.2112
g			deepset	0.01	3.2933	**2.7663**	3.0971
h		10	deepset	0.01	0.7675	0.7105	**0.4352**
i		20	deepset	0.00	0.1201	0.0873	**0.0646**
j	ADKL-KRR	20	attention	0.10	0.0575	0.0447	**0.0273**

Table 5: Effect of using task regularisation (parameter γ_{task}) on the MSE performance

To make reading this table easier, we also repeat figure 4.5a showing the improvement of the MSE relative to $\gamma_{\text{task}} = 0$ (no regularisation).

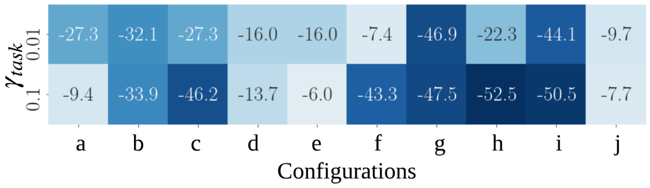


Figure B.1: Relative improvement of the MSE depending on the γ_{task} parameter

B.2 Pseudo representation

Table 6 presents the hyper-parameter combinations used in the experiments to assess the impact of the trade-off parameter γ_{pseudo}, which governs the penalty applied to the divergence between the distribution of learned pseudo-representations and the distribution of actual representations. We also repeat figure 4.5c for better readability.

Table 6: Effect of the pseudo-examples regularisation (parameter γ_{pseudo}) on the MSE performance

				γ_{pseudo}	0.00	0.01	0.10
algorithm	K	architecture	γ_{task}	Conf.			
ADKL-GP	10	deepset	0.10	a	0.6079	**0.4352**	0.5244
	20	deepset	0.01	b	0.0873	**0.0761**	0.0882
ADKL-KRR	20	deepset	0.00	c	0.0526	**0.0375**	0.0380
ADKL-GP	5	attention	0.10	d	2.2801	**2.2112**	2.2994
ADKL-KRR	20	deepset	0.01	e	0.0535	**0.0325**	**0.0325**
ADKL-GP	5	deepset	0.01	f	2.9466	2.7663	**2.7121**
	20	attention	0.10	g	0.1147	0.1144	**0.0870**
		deepset	0.00	h	0.1201	0.0958	**0.0940**
	5	attention	0.01	i	3.1136	**2.1511**	2.2511
			0.00	j	2.8528	2.5611	**2.4920**

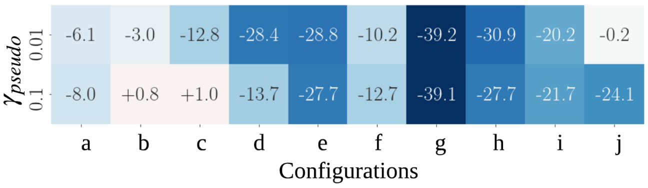


Figure B.2: Relative improvement of the MSE depending on the γ_{task} parameter

B.3 Joint impact of the task and pseudo-representation regularisations

Since both γ_{task} and γ_{pseudo} have a high impact on the training and the generalization performance, we need to assess the relationship between the two. Figure B.3 shows for different values of $|D^t_{\text{support}}|$ the relative improvement of the test MSE compared to the case where no regularisation is done. Overall, higher is better in both dimensions, although there seems to be a sweet spot on the grid for each value of $|D^t_{\text{support}}|$ and therefore we can only advise the user to cross-validate on those hyper-parameters.

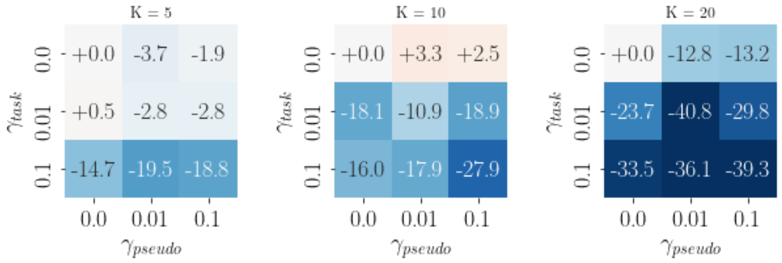


Figure B.3: Average relative improvement of the MSE and joint impact of γ_{task} and γ_{pseudo}.

C Prediction curves on the Sinusoids collection

Figure C.1 presents a visualization of the results obtained by each model on three tasks taken randomly from the meta-test set in the `Sinusoids` collection.

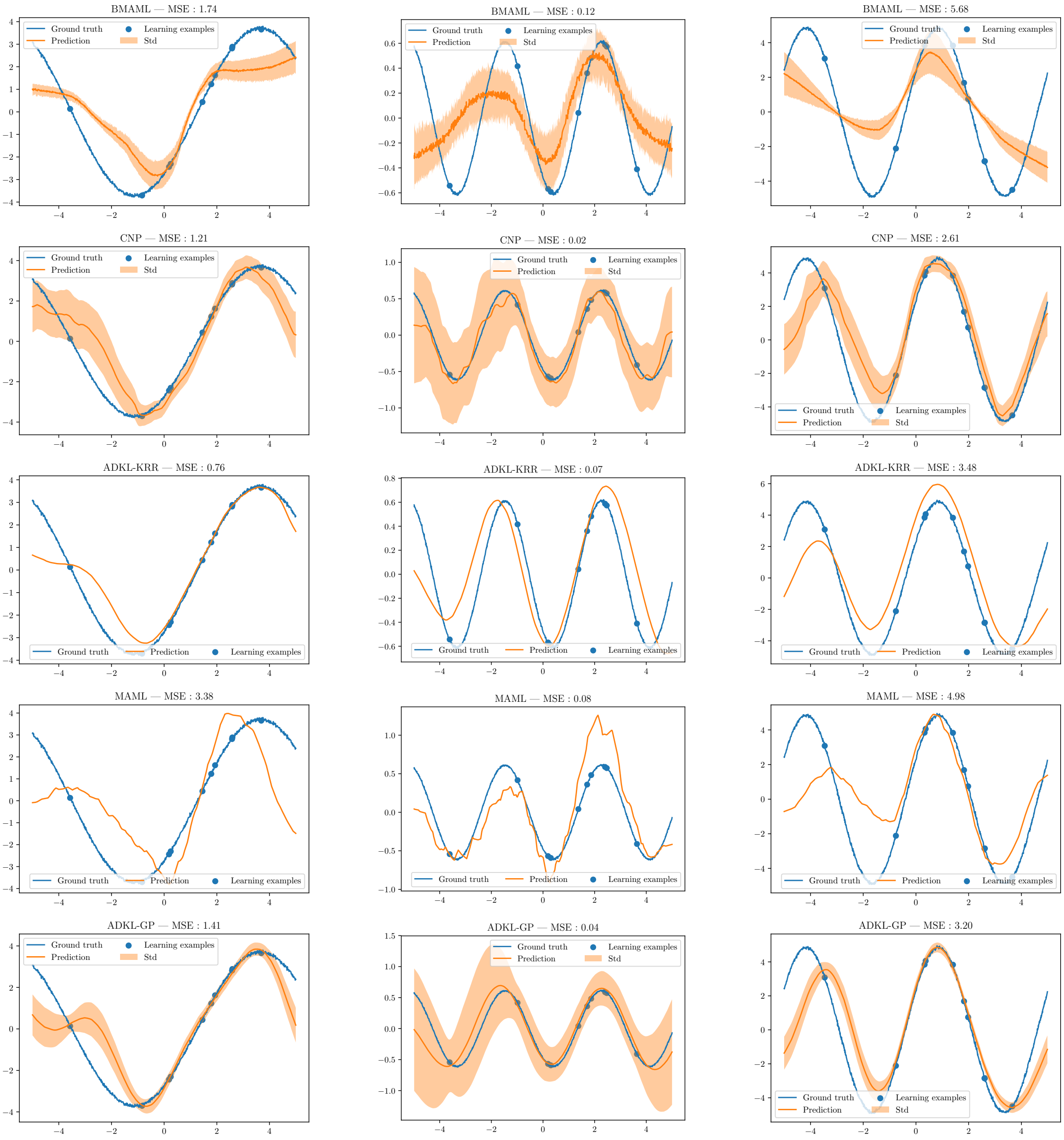


Figure C.1: Meta-test time predictions on the `Sinusoids` collection