PLANNING EXPERIMENTS WITH META-LEARNING PRIORS

Jan Olivetti, Ferran Alet
CSAIL
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{janoliv,alet}@mit.edu

1 INTRODUCTION

Suppose we have an entity from which we want to know some of its properties. Properties are binary (either true or false), and we obtain a reward for each property that we discover to be true. However, checking whether a property is true or not has a cost. For example, suppose that we have found a molecule that could cure a disease, and we want to test whether it's safe for humans. If said molecule is found to be safe for humans, the reward from distributing and selling the medicine would be large. However, testing whether it's safe for humans might be very expensive, so a cheaper alternative could be to test whether it's safe on mice first (which on its own gives no reward), and then decide based on the result we obtain whether it's worth it to perform the test on humans. Yet another option would be to test other properties, such as flammability or solubility on water, that give us less information about the property we're actually interested in but are even cheaper.

2 OVERVIEW

To solve this problem, we will create a model that can take a molecule and some known properties of that molecule and outputs a probability distribution over all the properties. Then, we'll use this model alongside the cost of each experiment and the reward for each property that we discover to be true to generate a policy of experiments we will perform.

2.1 TRANSPOSED META-LEARNING

The classic formulation of meta-learning is the following: for unseen tasks, we want to generalize to unseen inputs after seeing a few example inputs. However, in our formulation of the problem, we will not be facing new tasks (properties) but rather new inputs (molecules) for which we will use the tasks that we already learned to make predictions about them. In other words, for unseen inputs, we want to generalize to unseen tasks after seeing a few example tasks. Therefore, what we do is transpose the classic formulation of meta-learning, treating each molecule as a task and each property of the molecule as an input. Transposed meta-learning will be the basis of all the models we will train except for the multitask model.

2.2 MODELS

2.2.1 MULTITASK

The first model we will consider is the multitask model, which takes a molecule as an input and outputs a vector of probabilities of each property being true. This model consists of a Gated Graph Neural Network followed by a multi-layer perceptron, as explained in Nguyen et al. (2020). This model is unable to incorporate new information about experiments we have already performed. In theory, this should be enough since the input is Markovian and it is possible to deduce all the properties by looking at the molecule alone, but in practice we will not have perfectly accurate models so incorporating extra information will help.

2.2.2 Multitask with extended input

A simple way of incorporating known properties would be to include them as part of the input. This method is described in Pu et al. (2017). In this model, the input consists of a molecule and a vector containing the properties we already know about this molecule. This model is able to incorporate new information about experiments, but there are two problems with it. The first is that the input is flooded with many zeros since there's over a thousand properties but most of the time we will only know the results of a handful of them. The second is that there is a generalization gap: when the model receives new molecules as input, it has to retrain the relationship between the molecule and its known properties.

2.2.3 MAML

To address the aforementioned problems, we can use an optimization-based approach for the new molecule instead of an architecture-based approach. In transposed MAML (Model Agnostic Meta-Learning), we split molecules into meta-training and meta-validation, and we split properties into training and validation. Note that if we were using classic MAML, the splits would be the other way around. During meta-training, the model is trained using two loops: the inner loop uses training properties and the outer loop uses validation properties. During meta-validation, we use the meta-validation dataset to check that the model can generalize to unseen molecules by using the training tasks. That way, we hope to obtain a model that can generalize to new molecules and use known properties to predict the unknown properties. However, a disadvantage of this model is that we cannot train on several molecules in parallel.

2.2.4 CNGrad

In order to train on several molecules in parallel, we use the CNGrad method described in Alet et al. (2021). Instead of updating the entire model in both the inner and the outer loop, we insert a CN (Conditional Normalization) layer, consisting of a BN (Batch Normalization) layer followed by an affine layer, right before the last layer of the neural network. In the inner loop we will only update the CN layer and in the outer loop we update the rest of the model. Therefore, the majority of the neural network will be common across all molecules, and the differences between molecules will be entirely captured within this CN layer. In our experiments, we found that replacing the BN layer by a Layer Normalization layer improved performance, so we used that instead. By using this method, we can greatly speed up training without losing much expressiveness.

2.3 Planner

Using any of the models we trained to output probability distributions over properties, along with the cost of each experiment and the reward for each property if we discover that it is true, we can create a policy of experiments to conduct. To do so, we will perform tree search. Each node represents a set of properties we already know about the molecule (either true, false or unknown). The root of the tree will be a node that corresponds to knowing none of the properties, and from each node we can either perform one of the available experiments, or stop. The payout of stopping is equal to the rewards of the true properties that we discovered minus the cost of all the experiments we performed. Each time we perform an experiment, we generate two nodes: one where the property is true, and one where the property is false. We calculate the probability of each outcome using the trained model, and we calculate the expected value of performing that experiment by taking the expected value of the two child nodes weighted by the probability of each outcome. After expanding the tree up to a predetermined maximum depth, we generate the policy. At each node, the chosen policy will be the action with the highest expected value. An example of how this works can be found in Figure 1.

3 Experiments

We use data from the ChEMBL20 dataset to train our models and evaluate the performance of the planner. We only consider properties that are known for at least 128 molecules and we only consider molecules that have at least 4 properties (3 for training and 1 for validation). The chosen

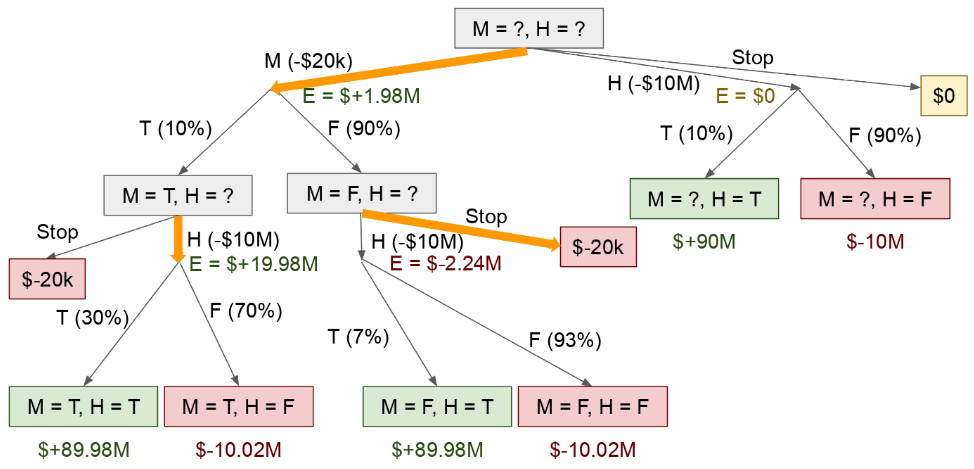

Figure 1: Example of tree search with two available experiments. M costs $20,000 but gives no reward, while H costs $10 million but gives a reward of $100 million if true. The large orange arrows represent the policy chosen by the planner, which correspond to the actions with the highest expected value at each step.

molecules are split randomly into (meta-)training and (meta-)validation, and the properties are also split randomly into training and validation. When training models that use properties as part of the input, instead of including all the known properties in the input, for each molecule we choose a number at random between 0 and 3 and we include that many properties in the input, randomly chosen among those that are available. The remaining properties are not included. The random number and chosen properties for a specific molecule will be different in each epoch. We remove inputs to simulate the situations we will face during planning, because in our experiments we will never perform more than 3 simultaneous experiments on any molecule.

To test the performance of the planner for each of the trained models, for each molecule we choose a random validation property as our ”goal” property, and discovering that it is true will give a reward R. The rest of properties give no reward. The set of available experiments for that molecule will be the properties that are present in the dataset for that molecule. We number the properties from 1 to k, assigning lower numbers to training properties, higher numbers to validation properties and the highest number to the goal property. The experiment to discover property i will cost $10^{c\frac{i-1}{k-1}}$, which means the experiment associated with the goal property will cost 10^c. During tree search, the maximum depth is 2 if there are fewer than 10 available experiments, and 1 otherwise. Using those numbers, we generate a policy of experiments for that molecule, and then we simulate that policy with the real data of that molecule.

4 Results

The (meta-)validation accuracy of each model is shown in Table 1. We observe that, for the multitask model, the model with extended input seems to perform worse than without the additional input. However, we observed experimentally that the model with extended input performs better than bare multitask when the input contains at least 1 property, so it can still be useful for planning. We also see that MAML can obtain a higher accuracy, likely because it is an optimization-based approach. Finally, we see that CNGrad outperforms MAML and that is likely because it is faster and therefore it can train for more epochs in the same window of time.

Table 1: (Meta-)validation accuracy of each model, on average and for each number of known properties. 95% confidence intervals are given for each value.

Model	**Accuracy**	**0 props.**	**1 prop.**	**2 props.**	**3 props.**
Multitask	.8679 $\pm$ .0003	N/A	N/A	N/A	N/A
Extended input	.8672 $\pm$ .0003	.8599 $\pm$ .0003	.8705 $\pm$ .0003	.8758 $\pm$ .0003	.8791 $\pm$ .0003
MAML	.8820 $\pm$ .0004	.8737 $\pm$ .0007	.8848 $\pm$ .0007	.8898 $\pm$ .0007	.8916 $\pm$ .0007
CNGrad	.9159 $\pm$ .0003	.9087 $\pm$ .0003	.9144 $\pm$ .0003	.9187 $\pm$ .0003	.9205 $\pm$ .0002

Table 2: Planner performance using each model, measured by average payout for each molecule. $R = 2000$, $c = 3$. The ”with planning” column allows the policy to perform experiments other than the goal, while ”without planning” can only use the initial prediction for the goal property. ”Always” refers to the scenario in which we always perform the goal experiment regardless of predictions. 95% confidence intervals are provided to test the hypothesis that the performance is different from multitask.

Model	**Without planning**	**With planning**
Multitask	211.75	N/A
Extended input	209.21 $\pm$ 2.55	217.28 $\pm$ 2.76
CNGrad	213.42 $\pm$ 3.07	217.81 $\pm$ 3.08
Always	-293.66 $\pm$ 6.61	N/A

The performance of the planner is shown in Tables 2 and 3. Performing tree search or not when using the multitask model does not affect the overall payout because the multitask model is unable to incorporate information from the experiments in the prediction. Multitask with extended input can, and despite having a worse performance without tree search, it outperforms the bare multitask model when we are allowed to perform other experiments. This is likely due to the fact that its overall accuracy is lower but it becomes higher when the input contains at least 1 property. CN-Grad outperforms multitask even without performing tree search, likely because the training method already makes the model learn some information about the known properties, but by letting the planner choose to perform some other experiments first we obtain even better results. We did not test the planner using the MAML model because it cannot be batched and therefore traversing all the dataset would take far too long.

5 Future work

One aspect to look into is the addition of a tailoring loss during training: if we perform an experiment on property B, for another property A it must hold that $P(A) = P(A|B)P(B) + P(A|\neg B)P(\neg B)$. We can use the Kullback-Leibler divergence between these two terms, before and after performing the inner loop update, as a tailoring loss. Furthermore, another problem with this method of obtaining a policy is that making an incorrect prediction in one node in a way that gives an overly optimistic result will cause the planner to favor that node, therefore making the entire policy incorrect. The more experiments we have available, the greater the chance we have of making one such incorrect prediction. Further investigation on this issue would be required. Another step that can be taken to improve results is trying to fine-tune hyperparameters. Lastly, it would be interesting to test this model using real world data for costs and rewards for the various experiments.

References

Ferran Alet, Maria Bauza, Kenji Kawaguchi, Nurullah Giray Kuru, Tomas Lozano-Perez, and Leslie P Kaelbling. Tailoring: encoding inductive biases by optimizing unsupervised objectives at prediction time. 2021.

Table 3: Planner performance using each model, measured by average payout for each molecule. $R = 200000$, $c = 5$.

Model	Without planning	With planning
Multitask	21175	N/A
Extended input	20921 ± 255	22120 ± 276
CNGrad	21342 ± 307	21877 ± 308
Always	-29366 ± 661	N/A

Cuong Nguyen, Constantine Kreatsoulas, and Kim Branson. Meta-learning gnn initializations for low-resource molecular property prediction. 2020.

Yewen Pu, Leslie P Kaelbling, and Armando Solar-Lezama. Learning to acquire information. 2017.