Machine Learning for Biological Design

Tom Blau and Iadine Chades and Cheng Soon Ong

Abstract

We briefly present machine learning approaches for designing better biological experiments. These approaches build on machine learning predictors, and provide additional tools to guide scientific discovery. There are two different kinds of objectives when designing better experiments: to improve the predictive model or to improve the experimental outcome. We survey five different approaches for adaptive experimental design that iteratively search the space of possible experiments while adapting to measured data. The approaches are: Bayesian optimisation, bandits, reinforcement learning, optimal experimental design, and active learning. These machine learning approaches have shown promise in various areas of biology, and we provide broad guidelines to the practitioner and links to further resources.

Key words: machine learning, adaptive experimental design, Bayesian optimisation, bandits, reinforcement learning, optimal design, active learning

1 Introduction

Machine learning is a paradigm for deriving computational and statistical models from observations. In the context of biological design, having accurate and reliable predictive models of phenomena is of great importance for a variety of purposes. For example, to predict the properties of proteins, what molecules they will interact with, what main effects and side effect they will have in a biological system. Machine learning has the potential to improve our understanding of biological systems, for example to understand how the DNA sequence of different regions impacts gene transcription and expression. This chapter describes approaches that build upon predictive models, and enable an adaptive approach to designing biological experiments and systems. This includes applications such as designing metabolic networks that will have desired properties, efficiently yield desired products, and retain the health of the host organism.

There are two main reasons why machine learning is worth exploring in synthetic biology. First, while computer modelling and in silico experimentation are hardly new in the biological sciences, the models derived from machine learning algorithms are often faster by orders of magnitude than traditional methods such as mechanistic simulations [1]. Second, a growing number of machine learning techniques can exploit knowledge extracted from data to search the space of possible solutions in an intelligent way, rather than searching it exhaustively [2]. Put together, these two advantages give machine learning the potential to massively accelerate progress in synthetic biology [3].

In this chapter, we will discuss *adaptive experimental design* (AED) algorithms. As the name alludes, these algorithms adapt the designs of future experiments to the observed outcomes of past experiments. AED algorithms all consist of the same components: a predictive model that maps observed quantities to predicted quantities, a learning rule that uses data to update the predictor, and a design rule that uses the predictor to design the next experiment. These components can be chained together in a cycle and repeated to iteratively improve our desired objectives. A key choice in AED algorithms is the objective we are trying to maximise: whether we want to improve the actions or decisions; or whether we want to improve the predictive model. After providing a brief introduction to predictive models, the following section discusses two different kinds of objective functions that correspond to action improvement and predictor improvement. The rest of the chapter describes the main algorithmic approaches for each kind of objective.

1.1 Predictive Models and Supervised Learning

Machine learning is a powerful paradigm for model fitting and optimisation that allows a practitioner to automatically discover a predictive model that can explain the available data. There is a wide variety of machine learning predictors for dealing with different types of data, and a corresponding variety of algorithms used to fit the

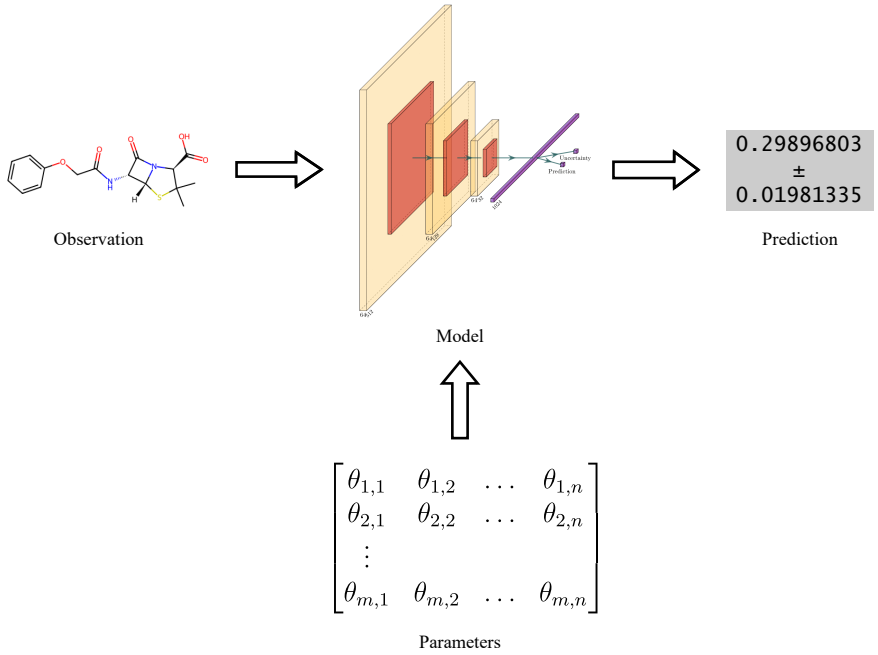

Fig. 1 Anatomy of a machine learning prediction process. An observation is passed to the predictive model. The model processes the observation based on its parameters, and outputs a prediction. The prediction includes some estimated uncertainty. Parameters of a predictive model are estimated by a process called ”training” or ”learning”.

models to the data. Broadly speaking, however, every machine learning predictive model is a function that takes as input some observation (e.g. the DNA sequence of a promoter) and outputs some prediction (e.g. gene expression strength) or decision (e.g. how to design a promoter). This generic prediction process is illustrated in Figure 1. Note that the predictor has some *parameters*, i.e. numerical values that determine the mapping from observation to prediction. For example, if the model is a weighted sum of the inputs, its parameters would simply be the weights. Parameters may map to meaningful notions, such as chirality or a molecular fingerprint, but may also be numerical values with no clear meaning. Finally, predictors often output two values: the prediction itself, and an estimate of the uncertainty about the prediction.

In the context of machine learning, the word “algorithm” is used interchangeably both for the process of fitting a predictive model to data, as well as for the process of using a predictive model for some purpose. In the former sense, often referred to as “training”, an algorithm is a procedure that takes a predictor and a data set, and iteratively updates the values of the model parameters to maximise some *objective*. For a more concrete example, the data set might consist of gene promoter sequences and their respective expression strengths. The predictor takes in a sequence and

predicts the expression strength, with the objective being the average accuracy of predictions.

In each iteration, the algorithm evaluates the prediction accuracy of the model for observations (e.g. promoters) in the data set. Using a target value associated with the input (e.g. expression strength of the promoter), the value of the objective is computed (e.g. squared difference between the predicted and true expression strength). The algorithm then applies a *learning rule* to update the model parameters, with the aim of improving the objective. Thus the algorithm *learns* the values of the parameters from data. This loop of prediction, evaluation, and parameter updates repeats until some stopping criterion is satisfied, such as reaching a pre-specified number of loops.

After training is finished, the predictive model is ready to be used for downstream tasks, such as predicting the yields of a metabolic network, or designing a network that has specific properties. If the parameters of the model have some meaning that is interpretable to an expert, conclusions may be drawn based on the learned values. For example, if a parameter is associated with production of a specific metabolite, the learned value can suggest a specific functional relationship between that metabolite and the yield of desired products.

The success of the machine learning paradigm rests on the ability to learn highly performant predictors, i.e. predictors that achieve a high value of the objective. A predictors that provides low accuracy predictions of gene expression strength would not be very useful in designing a novel gene promoter. A predictor that significantly overestimates the production of a critical metabolite may result in a metabolic network that fails to sustain the organism. It is important to remember that the mere act of using a machine learning algorithm to derive a predictor does not guarantee the predictor will be correct. Care must be taken to evaluate machine learning predictors before using them, as a practitioner can make any number of mistakes that will result in a poor predictor. Nonetheless, with proper attention to details and careful evaluation of the results, machine learning is capable of producing highly accurate computer models in less time and with less effort than many traditional approaches.

1.2 Nomenclature

Throughout this chapter, we use a number of terms that have different meanings in different scientific fields. Here we clarify the meaning we intend when using each term:

- **Predictor** - a machine learning tool that predicts attributes based on observations.
- **Model** - in the machine learning context, a mathematical representation of some physical phenomenon, in which case we will specify it is a **predictive model**. In the biology context, a model organism.
- **Training** - the process of fitting a predictive model to data.

- **System** - a physical entity or collection of physical entities. Typically a biological system such as a gene regulatory network, a bacterial population, a bioreactor etc.
- A system has **attributes**, e.g. gene expression or transcription initiation rate. We commonly wish to optimise one or more attributes.
- The attributes of the system are assumed to be influenced by its **configuration**, e.g. the DNA sequence of a gene promoter. We seek to optimise attributes by manipulating the configuration.
- **Action** - the choice of configuration of a system. E.g. choosing the DNA bases of the promoter.
- **Agent** - a computer-based actor that makes decisions and carries them out. E.g. an agent decides what the configuration of a promoter will be.
- **Objective (function)** - a mathematical measure for the performance of a predictor or agent, e.g. the average error in prediction of some quantity.

1.3 Anatomy of an Adaptive Experimental Design Workflow

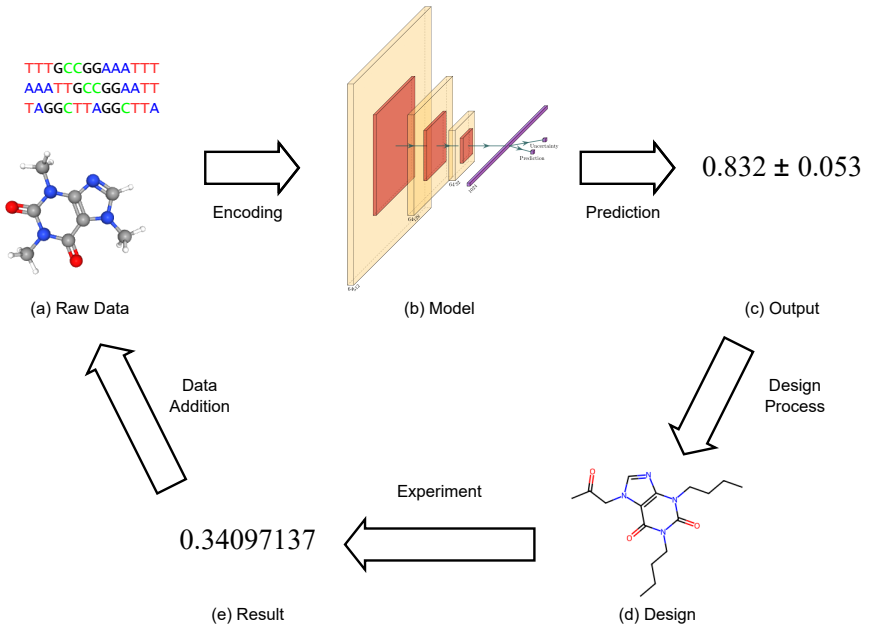

Fig. 2 The machine learning workflow for adaptive experiment design. Raw biological data is used by a model to make predictions, which then guide the design of the next experiment. The result of this experiment is added to the data, which is then fed back into the pipeline to design the next experiment.

We illustrate a generic machine learning workflow for biological design in Figure 2. Such workflows are often referred to as Biological Design Automation (BDA) or Design-Build-Test-Learn (DBTL) loops in the synthetic biology literature [4, 5], although those terms also include workflows that do not incorporate machine learning. The machine learning workflow always begins with some raw data, as seen in panel (a). This data could include molecular structures, genes, proteins, etc. which could be stored in a variety of formats. Since machine learning methods work on numerical inputs, we need to convert the raw data into a suitable numerical representation. This conversion step is often called encoding or preprocessing. The result is that each datum becomes a vector of real numbers, also known as features.

The next step, as shown in panel (b), is to use the predictive model and the encoded observation in order to predict the quantity of interest associated with the original input. Using the numerical vector, there are many different possible prediction methods, including: tree based methods such as random forest [6], kernel methods such as support vector machines and Gaussian processes [7], statistical methods such as generalized linear models [8], and deep learning methods [9]. One key requirement that is useful is that the prediction methods do not only predict the target value, but also additionally provide some level of uncertainty estimation, such as a confidence interval or even a full posterior predictive distribution. Panel (c) shows an example of such an output, showing the prediction plus/minus some error bar.

Using the predicted target value, an adaptive experimental design method proposes new measurements to take. This proposal is the *design*, shown in panel(d). Depending on the setting, a design could be a DNA sequence, a list of genes to knock out, the nominal conditions of a chemostat, etc. The space of possible designs to choose from will depend on the specifics of the problem at hand. The easiest case is that there is a finite set of designs from which to choose, e.g. a small library of known genes. In other cases, the set is finite but extremely large, e.g. the set of all genes that consist of 80 base pairs. And in some cases the space will be infinite, such as the set of all possible production rates of a protein.

The goal of the design is to produce some data that would be useful to measure. This data would be of the same class as the raw data initially passed to the predictive model. Having selected the design, the experiment can then be carried out, which results in a measurement of the quantity of interest. As per panel (e), this is the same quantity as the one predicted by the predictive model. The design and result of the experiment is then added to the set of raw data, and the process can be repeated until some stopping criterion is satisfied, e.g. we have exhausted our budget. Depending on the exact method being used, the predictive model may be re-trained when we acquire new data.

2 Different Design Objectives

Machine learning algorithms are trained to optimise a particular objective function, and adaptive experimental design is no exception. There are two major kinds of objective functions for adaptive experimental design, corresponding to two different goals of running an experiment.

- **Improve output** - to increase the value of the measured outcome, for example to maximise the yield of a particular system.
- **Improve predictor** - to improve the accuracy of the predictor, or more generally to improve our understanding of the prediction problem

These two different goals are described in more detail in Section 3 and 4 respectively. Additionally, the biological design problem itself is often a multi-objective problem: when designing a compound or a gene, or modifying a metabolic network, we often wish to maximise some outcome while minimising others, or at least keeping them within certain acceptable bounds. Alternatively, there may be multiple outcomes or properties that we want to maximise. These objectives may occur in a number of different domains, for example:

- **Metabolic changes** Increasing the production of one metabolite, without impairing the production of other critical metabolites too greatly.
- **Experimental feasibility** The experimental design must be possible to implement. A DNA sequence must be stable enough to assemble and insert into an organism or the experiment cannot be carried out.
- **Toxicity** A compound must not be too toxic for the biological system into which it will be introduced. Modifications to the system should not produce excessive toxic metabolites.
- **Cost** Different designs may incur different capital, labour and material costs. A design that exceeds budgetary constraints cannot be be implemented, and the benefits of each experiment may need to be weighed against its cost.

Any adaptive experimental design procedure in the context of synthetic biology may need to satisfy some combination of these objectives, and possibly further objectives. When assembling an experimental design workflow, the practitioner should carefully consider all the relevant objectives, and ensure they are accounted for.

Note that incorporation into the objective function is not the only way to handle multiple conflicting objectives. Suppose we have a machine learning workflow for designing gene promoters. Our main objective is to express a gene more strongly, but we must also ensure that the sequence can be synthesised by our intended synthesis method. After proposing a promoter but before manufacturing it, we could apply a test to filter out sequences that are unlikely to survive the synthesis process. If a proposed design fails this test, we will generate a new one rather than proceed to the experiment stage. Alternatively, it would be even more efficient if we could avoid proposing infeasible designs altogether. For example, if we know some sub-sequence of the promoter must have certain values (e.g. it must the GCR1 transcriptional

activator), we can restrict the search space and refrain from proposing designs that violate this constraint.

2.1 A Working Example

Given a particular gene, we may wish to choose a DNA sequence corresponding to its promoter sequence. Since promoters regulate the rate of protein production (in a complex fashion), choosing a particular DNA sequence of the promoter will essentially regulate the gene.

We assume that a suitable numerical representation of the DNA sequence is available, and a predictor such as the Gaussian process regression is trained to predict the amount of protein being produced for a given promoter sequence. The two different kinds of objective correspond to:

- **Improve output** Choose promoter sequences that increase the amount of protein produced. For example, the gene may produce a key component of some important enzyme and the researcher may wish to increase the production of this enzyme in the organism under study.
- **Improve predictor** Choose promoter sequences that improve our understanding of how genes are regulated, for example by choosing to measure sequences that reduce the prediction error (usually measured as root mean squared error for regression) of the trained Gaussian process regressor.

The space of possible promoter sequences is extraordinarily big, so that we cannot expect to search it exhaustively or even to cover a significant portion of it. We must have some kind of way to quickly rule out large segments of the space (that are unlikely to contain good promoters) without evaluating every sequence in those segments. Moreover, the functional relationship between the DNA and the objective is quite complex: the effects of individual base pairs are not independent of each other, and even a change in a single nucleotide can lead to a significant difference in outcome. Therefore, the predictor must capture the functional relationship, or at least do so well enough to guide our search.

3 Action Improvement

Action improvement concerns optimising the output or decision made by an agent. For example, if we have an agent that designs ribosome binding sites, this means maximising the volume of protein that will be produced by the resulting system [10]. It is assumed that executing the chosen action is costly and time consuming, e.g. a physical experiment that may take weeks to complete, and hence we are justified in spending a considerable amount of effort optimising the action.

One of the main challenges in action improvement is dealing with the uncertainty of the predictor. If we had perfect predictions, we could simply search the space of possible actions for the highest predicted value. This kind of action selection is referred to as "exploitation" of the predictor, since we exploit what we know. However, since there is uncertainty associated with each prediction, and that uncertainty is different for different actions, we also need to take actions that will reduce uncertainty, or we may never find the optimum. This kind of action selection is called "exploration". All action improvement algorithms have to somehow manage the trade-off between exploration and exploitation. In this section we will present the most popular approaches to maximising the action improvement objective.

3.1 Bayesian Optimization

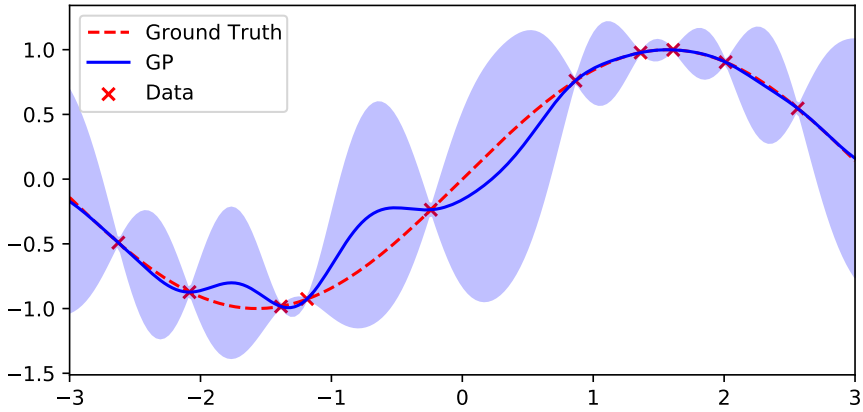

Fig. 3 A Gaussian Process trained on data from a trigonometric function. The true function is shown as a dashed red line. Red crosses indicate data points that have been observed and incorporated into the GP. The solid blue line is the mean function of the GP and the shaded blue region denotes one standard deviation from the mean.

The first approach to action improvement which we will consider is Bayesian optimisation (BO), and the workhorse of BO is the Gaussian process (GP). A GP can be thought of as a function that will map each point in the action space to a Gaussian distribution. Mathematically, this can be written as:

$$p(y|x) = \mathcal{N}(\mu(x), \sigma(x)), \tag{1}$$

where $p(y|x)$ is the predicted probability of seeing outcome y when taking action x, and $\mu(x)$ and $\sigma(x)$ are functions that map action x to the mean and standard deviation

of the Gaussian distribution $\mathcal{N}$. Figure 3 shows an example of a 1-dimensional GP trained on data from the `sinus` function. Crosses mark the data points used for training, and the underlying function is shown as a dashed red line. The solid blue line denotes the *mean function* of the GP, i.e. the mean of the predicted Gaussian distribution for each input x. The shaded region corresponds to one standard deviation of the Gaussian distribution. Note that the predictor is very accurate near the training data, and has high confidence (i.e. low standard deviation) about those predictions, whereas the predictions become less accurate and less confidence as we move farther away from the training data.

Keep in mind that not every function which outputs Gaussian distributions is necessarily a GP. Indeed, GPs have many other interesting properties, which are beyond the scope of this text. For a more thorough treatment, we recommend [11].

Because our prediction come in the form of a Gaussian distribution, we can incorporate uncertainty (as quantified by its standard deviation $\sigma(x)$) into the action improvement procedure. For example, we can choose the action that has the highest probability of improvement, i.e. the x that maximises $p(y > y^*|x)$, where y^* is the highest outcome seen in the training data. Alternatively, we can choose the action with the highest upper confidence bound for some confidence level. For example, the shaded region in Figure 3 denotes one standard deviation from the mean, therefore the top boundary of the shaded region is the upper confidence bound corresponding to a confidence level of ˜84% (i.e. there's an 84% chance that the outcome of an action will be below this boundary).

Both the probability of improvement and the upper confidence bound are examples of *acquisition functions*. These are functions that map each possible action x to some numerical value that determines how desirable it is. We always seek to take the action that maximises our chosen acquisition function. There exist many acquisition functions in the literature, but all of them have the important attribute that they manage the exploration-exploitation trade-off. Consider the example of the probability-of-improvement acquisition function. For values of x where the uncertainty of the prediction is large, the acquisition function will assign a high value even if the predicted outcome y is low. Thus the agent is encouraged to explore in regions of the action space where it has little information, but also balances this against actions with high-confidence, high-value actions.

Choosing the design that maximises the acquisition function implies yet another optimisation problem. The optimisation of acquisitions functions can be solved in a number of ways. If it is possible to compute the gradient of the acquisition function, then we can use local optimisation algorithms such as gradient descent [12]. In this case it is recommended to repeat the local optimisation multiple times from different start locations. If the gradient is not available, then a global optimisation algorithm such as the method of dividing rectangles [13] can be used.

Recall our working example of designing promoter sequences for a gene. How would we use Bayesian optimisation to solve this problem? In order to apply GPs to strings of DNA, we need to use a function such as the spectrum kernel [14, 15] which computes similarity between two strings. If any prior data is available on promoters and expression strength for the target gene, we can use them to train the GP. We

then use gradient descent to find a promoter which maximises the upper confidence bound acquisition function, and choose this as the design for our next experiment. After running the experiment and getting a measurement of promoter strength, we update the GP and repeat the process.

3.2 Bandits

The bandit approach most naturally applies to situations where we have a discrete set of choices, and we want to find the choice that maximises the output. The multi-armed bandit problem is a classic optimisation problem wherein an agent must choose between a number of options for investing its finite resources, with the goal of maximising a return on this investment. The options are analogised as choosing between different slot machines, colloquially known as "one-armed bandits", and hence options are also referred to as "arms". While the cost of each option is known in advance, the potential return is only partially understood, and more knowledge can be gained by allocating resources to this option [16]. In the context of adaptive experiment design, this corresponds to choosing from a finite set of designs, for example the set of all possible ribosome binding sites for an mRNA molecule.

The bandit framework proceeds iteratively: in each round, the agent chooses one of the options and receives some reward. For example, the reward might be the improvement in transcription initiation rate over the best known RBS. This continues for a predetermined number of rounds, which reflects the available budget for conducting experiments. The reward that would be received for each option is not known in advance, and indeed is not deterministic. The same RBS will produce different transcription initiation rates in repeated due to varying conditions that can't be fully controlled. Thus each option is associated with a true reward distribution, and when the agent chooses that option the reward it receives is a random sample from that distribution. The agent must maintain a *predictive* distribution over the rewards for each option, which reflects the belief about possible rewards that the option would yield. These distributions can be represented in any number of ways, for example using simple distributions, mixture models or flow-based models. The context around the experiment may also provide additional information that can improve the representation of the distributions [17]. The important thing is that the agent must be able to incorporate the observed rewards into its predictive reward distribution, so that this information can be exploited in the next round.

The goal of the agent is to maximise the sum of received rewards, which in our example is equivalent to maximising the transcription initiation rate of the best RBS the agent found. This reward maximisation objective can be formulated in one of two ways:

The first way is **regret minimisation**. Imagine an agent that had perfect knowledge of the true reward distributions, and could thus pick the best arm (the one with maximum expected reward) in every round. The difference between the expected sum of rewards achieved by this ideal agent, and the expected sum of rewards

achieved by a given non-ideal agent, is called the *regret* of the non-ideal agent. In this formulation, an agent seeks to minimise the total regret that accumulates over a fixed number of rounds.

Alternatively, the objective can be formulated as **best-arm identification**. Imagine that after each round, we try to predict which option is best in the sense that it has the highest expected reward. As we observe more rewards, over time the probability that our prediction is correct increases. At the end of the last round we must make a final prediction. In the best-arm identification setting, the goal of the agent is to maximise the probability that the final prediction is correct [18]. Naturally, some agent behaviours will be worse than others. For example, we intuitively expect that repeatedly choosing the same option in every round would be sub-optimal, since we gain no knowledge of the remaining options.

It is important to note that the two objectives, regret minimisation and best-arm identification, are mutually exclusive, in the sense that an algorithm that optimises one will be suboptimal for the other [19]. The intuition behind this is that the best-arm identification objective can tolerate exploratory action that yield extremely bad rewards. Such actions may allow to identify the best arm more quickly, but also accumulate regret quickly. Conversely, regret minimisation requires being more conservative in exploration, avoiding actions that may incur large regret, which slows down identification of the best arm.

Consequently, the two formulations fit different adaptive experiment design settings. In the event that we are choosing between designs that have inherently different costs or benefits (e.g. choosing between different types of bioreactors that have different operating costs), regret minimisation is appropriate as it allows minimizing the costs incurred by experimentation. On the other hand, if we are only considering designs that have identical or similar costs (e.g. choosing between bacteria strains that are equally expensive), then best-arm identification is appropriate.

There are many different subtypes of bandits in the literature, and many algorithms have been proposed to solve them. One popular class of algorithms is known as Thompson sampling. The core idea is to select options in proportion to their probability of being optimal [20]. To achieve this, in each round the agent samples a hypothetical reward for each option based on the predictive reward distributions that it maintains for that option. The agent then chooses the option that corresponds to the highest sampled reward. The probability that an option was selected is thus equal to the probability that it has the highest expected reward (according to the predictive reward distribution).

Consider the example of designing a gene promoter to maximise the expression of a gene. This could be formulated as a bandit problem by using gene expression strength as the rewards, and a library of candidate promoters as the set of options. Historical data could be used to fit a simple distribution (e.g. a log-normal distribution) for use as a prior. We could then proceed to use Thompson sampling to select a candidate promoter, and carry out the experiment to obtain a measurement of the resulting expression strength. After using this result to update the prior, we would then repeat the process until our budget is exhausted.

3.3 Reinforcement Learning

Reinforcement learning is a general framework for problems that involve making a sequence of decisions [21]. At the heart of reinforcement learning lies the Markov decision process (MDP), depicted in Figure 4. The MDP serves as a mathematical representation of the problem we wish to solve. It consists of two main components: a policy and an environment.

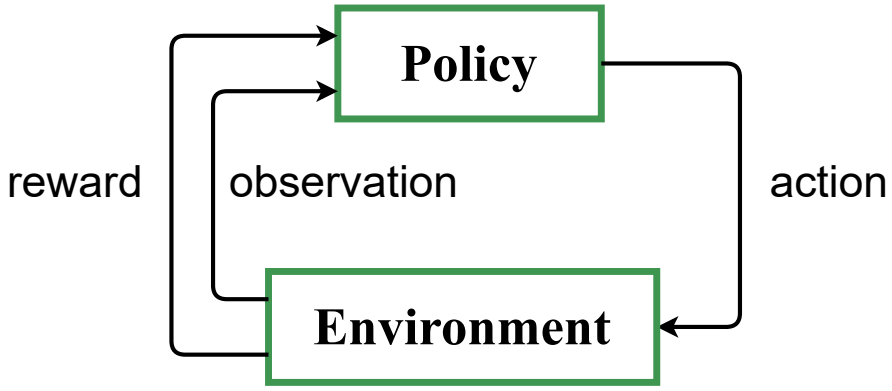

Fig. 4 The reinforcement learning framework. A policy observes the state of the environment and takes an action accordingly. The action causes the state of the environment to evolve. The environment than emits a new observation and a reward. The observation feeds back into the policy to repeat the cycle, while rewards are used to update the policy.

The environment represents a system whose state changes over time. For example, suppose we are trying to control a bioreactor to maximise the output of its product. The bioreactor is the environment, and its state can be summarised by the population sizes of the different microbial species in the reactor and the quantity of the desired product they have yielded. A policy is an object that maps observations of the environment to actions. In our example, the policy receives as input the population sizes, and chooses the rate at which to introduce different nutrients to the reactor.

When the action chosen by a policy is executed in the environment, this causes the state of the environment to evolve. The change in state is governed by the transition dynamics of the MDP, also known as a transition function, since it maps the current state and action to the subsequent state. In a bioreactor, these dynamics would be determined by the rate of nutrient consumption, the bacterial yield rate, and growth rate of each population. This entire step of changing from one state to the next by executing an action is called a "transition". That is, a transition is the collection of a state, the pursuant action determined by the policy, and the subsequent state.

After completing a state transition, the environment emits an observation of the new state, as well as a reward. This reward is a numerically value that expresses how good or desirable the transition was. In other words, it is a function not only

of the current state, but also of the action and possibly the previous state. Keeping to the bioreactor example, the reward function could include a positive term that corresponds to the bacterial yield, as well as a negative term that penalises the excessive use of nutrients. This cycle then repeats until some limiting number of transitions is reached, thus completing an *episode*.

The goal of a reinforcement learning algorithm is to learn a policy that maximises the sum of rewards accumulated over the course of an episode. Therefore the design of the reward function determines the behaviour of an optimal policy. Once a policy is trained, it can be executed in the system to achieve the desired behaviour (e.g. controlling a bioreactor) as defined by the reward. Great care must be taken in designing the reward function, as it may produce incorrect behaviours that are not immediately obvious. For example, the algorithm may discover a novel way to yield large volumes of the desired product, but also contaminates the product with some other undesirable metabolite that makes it unusable. If we had never encountered such a contamination issue before, and didn't incorporate it into the reward function, then this behaviour is in fact "optimal" from the perspective of the algorithm.

It may be the case that it is difficult to measure our desired objective during an episode, e.g. we can only measure the quantity of product after stopping and opening the bioreactor. In such an event, it is possible to train a policy using sparse rewards: assigning a reward of 0 to most transitions and awarding all the product yield to the final transition. However, practitioners should be mindful that many reinforcement learning algorithms perform poorly in the face of sparse rewards. In particular, if the reward function includes penalty terms (in our example, a penalty for using nutrients), sparse rewards may result in a policy that takes no action. This is because any action is penalised and the reward at the final transition may not be enough to counterbalance the penalty.

Another issue to be mindful of is that most implementations of reinforcement learning algorithms include a discount factor. This is a constant real number between 0 and 1 which discounts the value of rewards more heavily the later they arrive in an episode. In essence, the discount factor encodes our preference between achieving rewards immediately and achieving rewards in the long term. The closer it is to 0, the less we care about taking actions that will lead to high rewards in the future, and the more we prefer actions that will yield a high reward immediately. The closer it is to 1, the more we are willing to sacrifice short term gain in order to prioritise the total reward at the end of an episode. Practitioners should be mindful to choose a discount factor that suits their intended aim, and not simply use default value. Using the bioreactor example, we may prefer a policy that yields more product sooner and thus releases the reactor or other resources for another task, in which case a lower discount factor is appropriate. On the other hand, our resources may already be committed for a fixed time window and difficult to reallocate. In that case, we care about the total product yield over a fixed time period, and a high discount factor is a good choice.

It is important to note that reinforcement learning algorithms typically take a large number of episodes before they learn a good policy that achieves high rewards. As such, it is usually impractical to learn policies directly on physical systems. Rather,

reinforcement learning is most effective when a computer simulation of the system is available. Once an effective policy has been learned in silico, it can be fine-tuned on a physical system. In other words, the policy is transported to the physical system, and the algorithm continues to learn from interactions with this system, thus compensating for the differences between the real world and the simulation. If a good simulator is not available, it may be possible instead to use offline reinforcement learning. This subclass of reinforcement learning uses historical data on actions and observations (e.g. from a bioreactor with a hard-coded controller) in order to learn an initial policy. This policy can then be deployed in a physical system, and may continue to learn and improve over time.

As with all methods in this section, reinforcement learning algorithms have to manage the trade-off between exploration and exploitation. This is usually done by injecting some randomness into the policy. That is, instead of mapping observations to a deterministic action, the policy maps observations to a distribution over actions. The degree of randomness in this distribution can be decreased over time, either according to a fixed schedule, or in proportion to the performance of the policy (as measure by rewards). In any case, practitioners should take care to ensure that the randomness approaches zero by the end of policy learning, and to avoid deploying trained policies that still have a large amount of randomness.

3.4 Recommendations and Further Reading

This section introduced a number of approaches to the task of action improvement, and each approach can be implemented in many different ways. When faced with a specific adaptive experiment design problem, it can be difficult to choose an appropriate algorithm. Here we set up a few rules of thumb that can help guide you in making that choice.

Bayesian optimisation is highly effective at optimising actions, but its effectiveness declines rapidly as the dimensionality of the problem grows. In other words, if the number of design or observation parameters becomes large BO ceases to perform well. What constitutes as large varies depending on the exact field, but a general rule of thumb is that $10 - 20$ parameters is the maximum that BO will handle reliably. Beyond that point, it is recommended to use other methods.

Reinforcement learning, and its attendant Markov decision process framework, are built around the assumption that the system under consideration is stateful, i.e. it has some state that changes as a result of the decisions we make. While we can always apply RL to stateless systems, it will generally be inefficient to do so, and bandits are a better tool for such cases. Additionally, RL algorithms usually require a large number of system interactions to learn good policies, and as such are mainly useful when we have an efficient simulation of the system in question.

Another point to consider is whether the space of design is continuous (e.g. choosing the population ratios of a multi-species bacterial culture) or discrete (e.g. choosing between a handful of species of bacteria). Bayesian optimisation generally

works well with continuous spaces, and specialised algorithms are needed to handle discrete ones. On the other hand, bandits are mainly designed for discrete spaces, and perform poorly in the continuous case. Reinforcement learning can handle both discrete and continuous designs, although care should be taken about the specific algorithm being used.

The methods introduced thus far have many additional details that are beyond the scope of this chapter. For those wishing to understand them in greater depth, we recommend the following resources: To learn more about Bayesian Optimisation, we recommend the book *Bayesian Optimization* [22]. The canonical text on Gaussian processes is *Gaussian Processes for Machine Learning* [11]. For a broader overview of the main types of predictive models used in machine learning, see *Pattern Recognition and Machine Learning* [23] and *Mathematics for Machine Learning* [24]. *Introduction to Multi-armed Bandits* [25] is a good primer for bandit problems and algorithms. For more on reinforcement learning, a good resource is *Reinforcement Learning: an Introduction* [21].

4 Predictor Improvement

In the model improvement setting our objective is to acquire new data that can be used to improve some performance metric of the model itself (e.g. predictive accuracy, confidence bounds) rather than a metric of the model's output (e.g. binding affinity of a ligand). This is distinct from the learning that occurs in a typical machine learning algorithm, where the data is predetermined and available at negligible cost. Here the acquisition of each new data point or batch of points comes at a significant cost, and the aim is to maximise the improvement in model performance subject to a limited budget of data. In other words, the independent variable being optimised is the data itself.

More precisely, the data to be acquired consists of an independent component, called a *query* or *design*, which we completely control, and a dependent component which depends on the design in a potentially complicated way. For example, suppose we are trying to improve a model of a drug's pharmacokinetics. The design could consist of the dose to be administered, or the times at which to take blood samples. The serum concentrations we will measure depend on the design but also on the biology of the body, and this dependence is what we're trying to model. Thus the question is how to optimise the design so that, after conducting the experiment and collecting data, we will see the maximum improvement in model predictions of serum concentration.

4.1 Bayes Optimal Experiment Design

Bayesian statistics is founded on Bayes' theorem, which describes how the degree of belief about some numerical value is updated when we see new data. Suppose are working with a genetic toggle switch and a chemical inducer [26]. We need to develop a predictive model that will predict the inducer concentration inside the cell at a given time t, based on the concentration of the inducer outside the cell and the permeability of the cell wall. Thus the permeability and intracellular concentration are variables, which we will denote A and B. In a Bayesian framework, we have some prior beliefs about the values of A and B, which are probability distributions denoted as $p(A)$ and $p(B)$, respectively. For any given value of the permeability (A), our predictor gives us a belief (i.e. a probability distribution) about the value of the intracellular concentration(B). We denote this as $p(B|A)$, which is read as "probability of B given A". Bayes' theorem allows us to compute the *posterior* belief about permeability after observing the concentration:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}. \tag{2}$$

In other words, we can use Bayes' theorem to learn the permeability parameter of our predictor from data. Figure 5 visualises what a prior and posterior might look like. In this case, the prior $P(A)$ (in blue) is uniform over the range of $0 - 10$ units, meaning that before observing any data, we believe the permeability of the cell membrane is equally likely to be any value in that range. The posterior $P(A|B; t = 1)$ (in orange) is calculated following one observation of the intracellular concentration, made one minute after starting the system. It has a humped shape with a peak at ˜6.7 hours. Once we have observed the intracellular concentration at time t, we are much more confident that the permeability is in the range of $4 - 9$ units and much less confident that it is outside that range. Another posterior, $P(A|B; t = 2)$ (in green) is calculated using a single observation, but this time made at the two minute mark. It has a much narrower shape than the posterior at $t = 1$ and the peak sits at 6 units.

Two important insights follows from this example: first, the posterior can be more or less peaked depending on the time of observation. Second, we would like to have a posterior that is as narrow and peaked as possible, concentrating the majority of the probability in a small range. The degree to which a distribution is peaked is also known as its *self-information* or *entropy*, so that highly peaked distributions are said to have low entropy. The increase in peaked-ness of the distribution from the prior to the posterior is known as *information gain*.

Therefore, we want our experiments to have high information gain, i.e. to peak the posterior as much as possible. Bayes optimal experiment design (BOED) is the discipline that focuses on maximising the information gain of experiments by manipulating the independent variables [27]. In the above example, the independent variable is the time of measurement (we can also imagine manipulating the initial extracellular concentration). The choice of values of the independent variables is the *design* of the experiment [28]. It is important to note that different outcomes are pos-

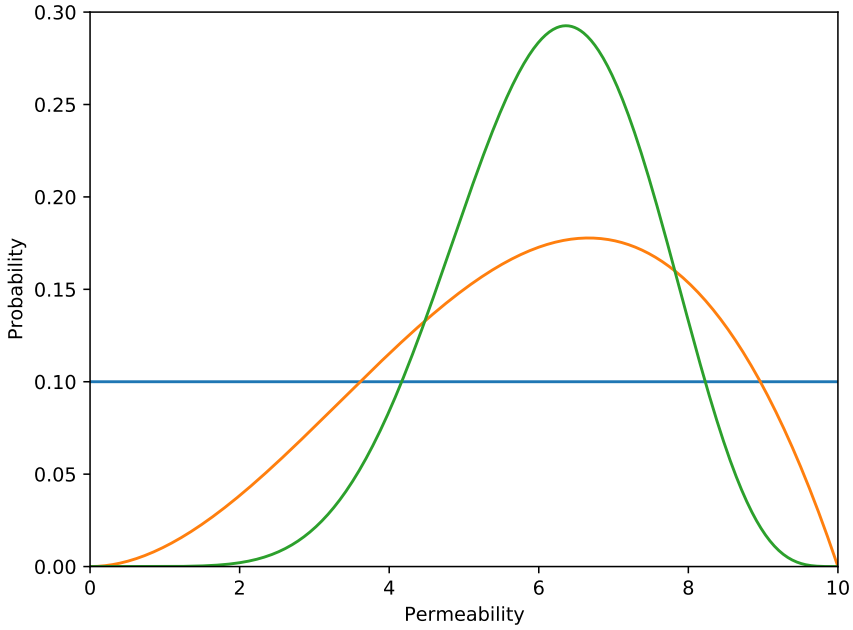

Fig. 5 Prior (blue) and two different posterior (orange and green) probability distributions for permeability of a cell membrane. Each posterior corresponds to a different experiment. The green posterior is much more peaked and concentrated than the orange one, indicating that more information was gained about permeability from the corresponding experiment.

sible when repeating the same experiment, e.g. different intracellular concentrations at the same measurement time. This will lead to different posteriors and different information gains. Therefore, it is impossible to know what the exact information gain will be for any given design before running the experiment. We only know the distribution of possible information gains, conditional on the design, and can only maximise the *expected value* of information gain.

Calculating this expected information gain, and optimising it with respect to the design variables, is greatly dependent on the methods being used to model the problem. If the distributions $p(A), P(B)$ and $p(B|A)$ are represented by simple probability distributions (e.g. Gaussian and uniform distributions), linear models, or Gausian processes, there is often an analytical solution, i.e. a relatively simple mathematical expression that exactly computes the expected information gain for any given design. However, it is very common that an analytical solution isn't available, and the expected information gain must be approximated. This approximation typically involves sampling inputs from a probability distribution and performing some mathematical operations on those inputs. As the number of sampled inputs increases, the estimation becomes more exact, but the computational cost also increases.

Whether the expected information gain can be computed exactly or only approximately for a given design, we are then left with the task of finding the design that maximises gain. In the event that the set of possible designs is small, this can be done by evaluating the gain for all possible designs and choosing the best one. In the more common case, however, the space of designs is too large (or possibly infinite) and more sophisticated methods must be used. If our model allows us to compute the gradient of expected information gain with respect to the design variables, we can use a gradient descent algorithm to optimise the design [12]. Otherwise, we can use gradient-free methods, such as coordinate descent, or simulated annealing. Note that many of the methods introduced in Section 3 can also be used to maximise information gain, for example reinforcement learning [29].

4.2 Model Discrimination

In addition to inferring the parameters of a predictor, we can also use BOED in the case where we have multiple competing predictor architectures, and have to determine which one best fits our data. For the example of predicting the permeability of a cell membrane, we could have different models reflecting different assumptions: passive diffusion, active transport, asymmetric influx and efflux, degradation of the inducer, etc. The Bayesian framework can then be applied to choose between these predictive models, in a process called *model discrimination* or *model selection*. Each predictor is assigned a prior probability that it is the correct one, thus inducing a probability distribution over predictors [30, 31]. The left panel of Figure 6 show a uniform prior over four predictors $P_1 - P_4$, meaning that before making any observation, we have no preference between the predictors. After observing a measurement of the intracellular concentration of the inducer, Bayes' theorem is applied to compute the posterior distribution, shown in the right panel.

The posterior indicates a preference for predictor P_4, but this preference is relatively mild: only a 4-to-3 ratio. A different design would lead to a stronger preference. Fortunately, information gain is related to the strength of preference for a predictor in this setting, just as it was related to the peaked-ness of the posterior in the single model setting. Thus, we can use BOED to choose an experiment design that maximises the expected information gain of the posterior over predictors.

Finally, it is possible to consider the joint distribution that combines both the distribution over predictors and the individual distribution of each of the predictors. By selecting a design that maximises information gain with respect to this joint distribution, we can simultaneously maximise the posterior preference for a predictor as well as the posterior confidence about the parameters of that predictor.

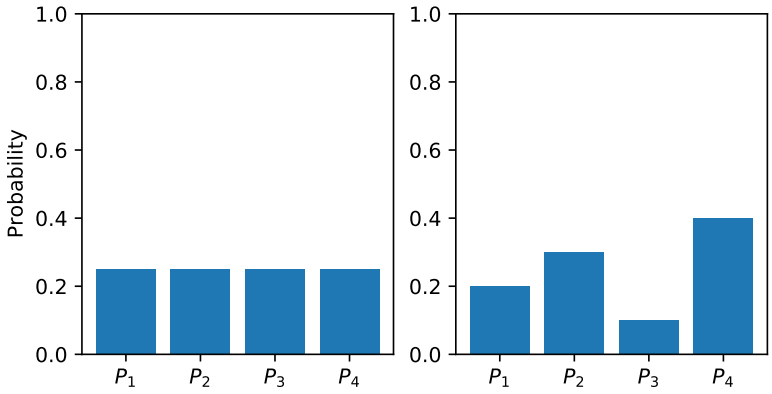

Fig. 6 Prior (left) and posterior (right) probability distributions over four different predictors. The prior accords equal probability to every predictor, since we have no reason to prefer any particular one. After observing an experiment, the posterior accords more probability to P_4, meaning the observed outcome is most consistent with this predictor.

4.3 Active Learning

Whereas BOED seeks to improve the model by obtaining an accurate estimate of the model parameters, active learning (AL) is focused on improving the predictive performance of the model directly [32, 33]. Recall our previous example of a model that predicts serum concentration of a drug based on elimination half life. Rather than trying to identify the half-life with a high degree of certainty and using this estimation to predict serum concentration, we could learn some model that predicts serum concentration directly. Our goal would then be to minimise some notion of the predictive error, for example the average squared value of the error between predicted concentration and measured concentration.

Active learning is an extension of the supervised learning setting, where we have a set of data used to train the model as well as a set that will be reserved to test the model. In other words, the predictive error will be evaluated using the test data. Additionally, in AL we assume a much larger set of unlabeled data, i.e. inputs for which we don't know the true target value. The task is to choose some subset of these inputs for which we will acquire target values, i.e. to design experiments, such that after re-training the model with the experimental results, the average predictive error on the test data will be minimised. As before, we don't know what the true target value (e.g. the measured serum concentration) will be before we measure it, so we can only minimise the expected value of the predictive error, where the expectation is with respect to the current belief (as represented by our model) about what the target value will turn out to be. Mathematically, let $f()$ be the current model, $g()$ be the model after re-training, and ℓ be the predictive error. Then we wish to minimise:

$$\mathbb{E}_{p(y|f(x))}\left[\ell(g(x^*), y^*)\right], \tag{3}$$

where x are the design variables we control, y is the true measurement associated with x, and (x^*, y^*) are the test data. $p(y|f(x))$ is the belief about y according to the current model $f()$, therefore the notation $\mathbb{E}_{p(y|f(x))}$ means we are taking the expected value with respect to the uncertainty over the measurement y. The model must provide this uncertainty in addition to the prediction $f(x)$.

In general, it is impossible to know what the model $g()$ will look like before carrying out the experiment, and therefore the above objective cannot be computed exactly. Thus we will, in most cases, need to optimise some proxy objective. A popular family of methods is to choose the design x that maximises the predictive uncertainty of $f(x)$ [34]. An alternative approach is to estimate a Bayesian posterior regarding the distribution of the model parameters after re-training, and choose the design that minimises uncertainty about the parameters (rather than about the prediction) [35].

Returning to the example of designing gene promoters, we could use active learning to improve prediction. Our predictive model could be a Bayesian neural network, a deep learning architecture that supports efficient computation of distributions over the model parameters and predictions [36]. If we have a large pool of candidate promoters (e.g. 100, 000 promoters), we can compute the predictive uncertainty for all of them, and choose a small subset (e.g. 100) of the promoters that have the highest uncertainty. After experimentally evaluating the expression strengths of these promoters, we add the results to our data and re-train the Bayesian neural network.

4.4 Recommendations and Further Reading

In the prediction improvement setting, there are two primary types of approach: Bayesian optimal experiment design and active learning. Although there is some overlap and strong mathematical connections between the two, they are generally suited to distinct use cases. BOED is primarily an approach for system identification. When studying a system, such as a genetic toggle switch, and there are parameters whose values we want to identify, e.g. the flux of a chemical inducer, then the BOED framework and algorithms are appropriate. In contrast, active learning is better suited for the setting where we don't care about the value of any specific parameter, but rather are concerned with the performance of our predictor, and wish to improve it.

The above is a general guideline to help decide which class of tools you should consider when faced with a specific research question. Because the field evolves rapidly, any specific algorithm recommendation may soon become obsolete. However, we can recommend some texts that are good entry points to the literature.

For the mathematical foundations of Bayes optimal experiment design, see *On a Measure of the Information Provided by an Experiment* [27]. An overview of deep learning methods for solving the BOED problem can be found in *Modern Bayesian Experimental Design* [37]. For an in-depth introduction to active learning,

confer *Active Learning* [32]. Finally, *A survey of deep active learning* [38] provides a rundown of contemporary active learning algorithms.

5 Discussion and Conclusion

Machine learning based approaches for adaptive experimental design have the potential to accelerate scientific discovery. There are two different kinds of goals of designing an experiment: to improve the output of the experimental system, or to improve the predictive model. Improving the output of the experimental system is useful when the scientist is interested in maximising the yield (for example the amount of protein) of the system under study. In contrast, the scientist may be interested in collecting data that improves the predictive model itself, that is to find data that provides information with the goal of improving the accuracy of the predictive model. For both scenarios, an estimate of the predictive uncertainty is highly useful to take into account the fact that predictive models are not perfect.

When improving actions, a key challenge is to manage the trade-off between exploration and exploitation. We considered three increasingly general classes of action improvement approaches. Bayesian optimisation uses a Gaussian process as a predictive model, and optimises the corresponding acquisition function. Acquisition functions provide a numerical value representing how desirable a particular action is; which could be the probability that the action improves the output, or an upper confidence bound that directly trades off exploration and exploitation. When we have a discrete set of choices, a bandit approach can be used to maximise the output of the experiment. There are two major formulations of bandit objective functions: whether to minimise the regret, or to identify the best choice (called arm). The most general approach for improving actions is reinforcement learning, which also additionally models the fact that the experimental environment may change over time.

When improving the predictor, the main goal is to acquire new data that improves the performance of the predictor itself. The classical approach based on Bayes' theorem is to find a posterior distribution that is more peaked, which indicates that the data used to update Bayes' theorem is very informative. However, since we have not yet measured the data when designing the experiment, and hence the amount of information gained by the experiment is yet unknown, optimal experimental design approaches need to optimise the expected value of information gain. The Bayesian framework can also be used for model selection. We could directly focus on the performance of the predictive model itself, instead of the distribution over the models. The active learning approach considers which of the set of currently unlabelled data that should be labelled, to potentially maximise the prediction performance.

5.1 Greedy, non-Greedy and Batch Algorithms

Thus far we have discussed only the case where a single experiment is optimised at a time, before it is carried out and its results are used to update the model. Algorithms that work in this way are known as *greedy* algorithms, because they greedily optimise for immediate gain without considering the effects that the design will have farther into the future. However, it is a common occurrence that we have the budget to carry out more than one experiment and know this in advance. In this case, we could apply a *non-greedy* algorithm that will consider the effect of each experiment on all future experiments within our budget. While such algorithms still design experiments one at a time, they sacrifice immediate gains (i.e. accepting less information gain or less reduction in predictive error from the current experiment) in exchange for higher gains later on, leading to an overall better series of experiments.

The drawback of non-greedy optimisation is that it is more difficult to carry out. In both the BOED and AL settings we saw that the objective, whether information gain or predictive error, can only be optimised in expectation, because we don't know what the outcome of the experiment will be at the time we design it. When seeking to optimise the long-term benefits of an experiment, we have to take an expectation over the possible outcomes of all future experiments. Additionally, since we are still optimising experiments one at a time, the designs of future experiments are also not known in advance, and we have to take an expectation over the possible values of future designs, as well.

Yet another setting to consider is that of *batch* algorithms, so called because they optimise a batch of experiments simultaneously. While this approach is not adaptive, since all experiments must be designed before observing any outcomes, it still has some benefits. First, there are many cases where we can conduct experiments in parallel, leading to considerable savings in time and cost compared with running them sequentially. Examples range from high-throughput biology, where thousands of experiments can be carried out simultaneously, to experiments in animal models which may take years to complete. Second, while batch algorithms can't incorporate information about the outcomes of the experiments, they can still take into consideration how the designs affect one another, which greedy algorithms cannot do. For example, when designing a batch of experiments we can consider how well the batch covers different regions of the space of possible experiments [39].

In summary: greedy algorithms optimise each design by taking an expectation over possible outcomes of the current experiment, batch algorithms optimise all designs simultaneously by taking an expectation over possible outcomes of all experiments, and non-greedy algorithms optimise designs sequentially by taking an expectation over possible outcomes and designs of all future experiments.

5.2 Summary

The approaches presented in this chapter provide additional tools for a scientist to use machine learning predictors to design better experiments. The choice of how to define "better" results in two different kinds of objectives: improving the predictive model or improving the outcome of the experiment. Depending on the experimental search space and the objective to be optimised, different approaches are more suitable than others. We have therefore surveyed five different classes of approaches that adapt to new data as it is acquired and have shown promise for biological design. Broad guidelines are presented to help practitioners in choosing the right tool for their particular problem. However, there are many nuances to every scientific question, and equally as many nuances to each machine learning approach. As such, we include a number of recommended texts that can help acquire a deeper understanding of relevant machine learning algorithms. These texts will empower the reader to get more out of the tools presented here. We hope that our exposition provides a useful entry point for experimental scientists interested in accelerating the search for better experimental designs.

References

[1] Ioana M Gherman et al. "Bridging the gap between mechanistic biological models and machine learning surrogates". In: *PLoS Computational Biology* 19.4 (2023), e1010988.

[2] Jean-Loup Faulon and Léon Faure. "In silico, in vitro, and in vivo machine learning in synthetic biology and metabolic engineering". In: *Current Opinion in Chemical Biology* 65 (2021), pp. 85–92.

[3] Frank Noé et al. "Machine learning for molecular simulation". In: *Annual review of physical chemistry* 71 (2020), pp. 361–390.

[4] Evan Appleton et al. "Design automation in synthetic biology". In: *Cold Spring Harbor perspectives in biology* 9.4 (2017), a023978.

[5] Bryan A Bartley et al. "pySBOL: a python package for genetic design automation and standardization". In: *ACS synthetic biology* 8.7 (2018), pp. 1515–1518.

[6] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[7] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[8] P McCullagh and J A Nelder. *Generalized linear models*. Chapman and Hall/CRC, 1989.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] Mengyan Zhang et al. “Machine learning guided batched design of a bacterial Ribosome Binding Site”. In: *ACS Synthetic Biology* 11.7 (2022), pp. 2314–2326.

[11] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.

[12] Léon Bottou. “Stochastic Gradient Descent Tricks”. In: *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 421–436.

[13] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. “Lipschitzian optimization without the Lipschitz constant”. In: *Journal of optimization Theory and Applications* 79 (1993), pp. 157–181.

[14] Christina Leslie, Eleazar Eskin, and William Stafford Noble. “The spectrum kernel: A string kernel for SVM protein classification”. In: *Biocomputing 2002*. World Scientific, 2001, pp. 564–575.

[15] Cheng Soon Ong et al. “Support Vector Machines and Kernels for Computational Biology”. In: *PLoS Computational Biology* 4.10 (2008), e1000173.

[16] Peter Whittle. “Multi-armed bandits and the Gittins index”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 42.2 (1980), pp. 143–149.

[17] Andreas Krause and Cheng Soon Ong. “Contextual Gaussian Process Bandit Optimization”. In: *Advances in Neural Information Processing*. 2011.

[18] Mengyan Zhang and Cheng Soon Ong. “Quantile Bandits for Best Arms Identification”. In: *International Conference on Machine Learning*. 2021.

[19] Rémy Degenne et al. “Bridging the gap between regret minimization and best arm identification, with application to a/b tests”. In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 1988–1996.

[20] Daniel J Russo et al. “A tutorial on thompson sampling”. In: *Foundations and Trends® in Machine Learning* 11.1 (2018), pp. 1–96.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[22] Roman Garnett. *Bayesian optimization*. Cambridge University Press, 2023.

[23] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Springer, 2006.

[24] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[25] Aleksandrs Slivkins et al. “Introduction to multi-armed bandits”. In: *Foundations and Trends® in Machine Learning* 12.1-2 (2019), pp. 1–286.

[26] Jean-Baptiste Lugagne et al. “Balancing a genetic toggle switch by real-time feedback control and periodic forcing”. In: *Nature communications* 8.1 (2017), p. 1671.

[27] Dennis V Lindley. “On a measure of the information provided by an experiment”. In: *The Annals of Mathematical Statistics* 27.4 (1956), pp. 986–1005.

[28] Jiali Wang et al. “Optimal design for adaptive smoothing splines”. In: *Journal of Statistical Planning and Inference* 206 (2020), pp. 263–277.

[29] Tom Blau et al. “Optimizing Sequential Experimental Design with Deep Reinforcement Learning”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 2107–2128.

[30] Alberto Giovanni Busetto, Cheng Soon Ong, and Joachim M. Buhmann. “Optimized Expected Information Gain for Nonlinear Dynamical Systems”. In: *Proceedings of the International Conference on Machine Learning*. 2009, pp. 97–104.

[31] Alberto Giovanni Busetto et al. “Near-optimal Experimental Design for Model Selection in Systems Biology”. In: *Bioinformatics* 29.20 (2013). doi:10.1093/bioinformatics/btt436, pp. 2625–2632.

[32] Burr Settles. *Active Learning*. Springer, 2012.

[33] Alasdair Tran, Cheng Soon Ong, and Christian Wolf. “Combining active learning suggestions”. In: *PeerJ Computer Science* 4 (July 2018), e157. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.157. URL: https://doi.org/10.7717/peerj-cs.157.

[34] Freddie Bickford Smith et al. “Prediction-Oriented Bayesian Active Learning”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 7331–7348.

[35] Neil Houlsby et al. “Bayesian active learning for classification and preference learning”. In: *arXiv preprint arXiv:1112.5745* (2011).

[36] Laurent Valentin Jospin et al. “Hands-on Bayesian neural networks—A tutorial for deep learning users”. In: *IEEE Computational Intelligence Magazine* 17.2 (2022), pp. 29–48.

[37] Tom Rainforth et al. “Modern bayesian experimental design”. In: *arXiv preprint arXiv:2302.14545* (2023).

[38] Pengzhen Ren et al. “A survey of deep active learning”. In: *ACM computing surveys (CSUR)* 54.9 (2021).

[39] Mengyan Zhang, Russell Tsuchida, and Cheng Soon Ong. “Gaussian Process Bandits with Aggregated Feedback”. In: *AAAI Conference on Artificial Intelligence*. 2022.