Application of Curriculum Learning for de novo design of small molecules

Master’s thesis in Physics

Juan Diego Arango

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2021
www.chalmers.se

Abstract.

In recent years, Deep Learning has given new energy to the field of de novo design. This field is the generation of novel chemical compound ideas, which can be used for new applications. AstraZeneca has developed software for this task called REINVENT. The software uses Reinforcement Learning and a user-defined scoring function to create new compound ideas. The objective of this work is to implement, within REINVENT, the technique of Curriculum Learning. Here, the scoring function during the training phase is actively modified. Besides the implementation, this work explores how this approach improves performance compared to the classical Reinforcement Learning approach.

Table of Contents

Introduction

The challenge of designing a new drug is, at its most fundamental, to find a molecule that is *active* towards a particular biological target[1]. For example, if one wants to create a new drug analogous to caffeine, then a new compound that binds to the adenosine receptor on the neurons must be found. The problem is exploring the chemical space and finding a new candidate among all possible compounds. Seen this way, designing new drugs is a problem of exploration of the chemical space and exploitation of a particular region of space that fulfills criteria, which may include the desired binding criteria and molecular weight, among other properties. This situation is challenging as the size of the chemical space is enormous. Some estimates indicate that at least the chemical space has around 10^{60} constituents (Reymond et al., 2012).

Currently, Machine Learning has been gathering interest as a possible tool to deal with the problem. Examples include:

- The work done by (Segler et al., 2018) shows that the machinery of natural language processing can be used for compound generation. This work is done by training Recurrent Neural Networks (RNN) on string representations of molecules. The trained model is used as a generator, and the compounds it generates are found to correlate to ones used in training.
- The work done by (Arús-Pous et al., 2019), a model of RNN, using LSTM, is trained on a million molecules represented as strings. After the training, the model is used as a generator. The results suggest that the model can generalize, as it outputs molecules that belong to a bigger chemical space than the one of the training samples.
- The work done by (Awale et al., 2019) is another example. Here, the RNN model is used again, and it is trained to produce new compounds that are analogs to know drugs.

In this thesis, the work is centered around a solution offered by the group of Molecular AI at AstraZeneca. The group has implemented a software called REINVENT, which is a Machine Learning (ML) software that can perform automatic exploration or exploitation of the chemical space and returns compounds that fulfill a set of user-defined characteristics (Blaschke et al., 2020).

[1] The term active encompasses multiple possible interaction but at it simples means binding to certain biomolecule of an organism. For example, caffeine is active, as it binds to the adenosine receptor on the neurons.

REINVENT has the difficulty that under certain situations, it takes a long time to output compounds. There are two situations where this is encountered: **1)** the set of properties desired for the compounds generated are orthogonal, and **2)** the conditions given allow for a very narrow solution space. The objective of this thesis project is to help to optimize the performance of REINVENT under these conditions. The approach is to create Curriculums for the learning. This approach can be understood as sequentially modifying the criterion desired for the compounds, making it more restrictive until the desired criterion is reached.

The objective is to add a Curriculum Learning functionality to REINVENT. This new functionality is applied over a set of cases to observe how this modification changes the performance.

Theoretical background of REINVENT

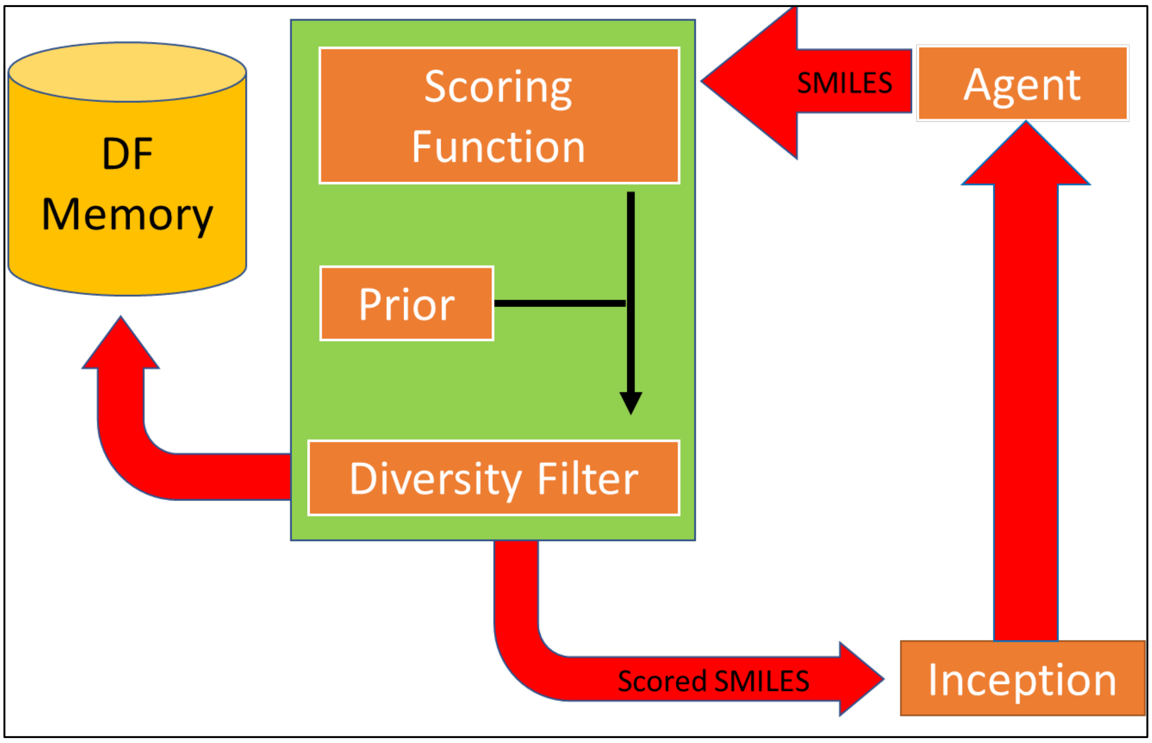

Figure 1. **(1)** The RNN generates SMILES, **(2)** The SMILES are evaluated and the Diversity Filter memory updated, **(3)** The resulting Scored SMILES are passed through the inception modulus, and it is updated. **(4)** The resulting batch after the inception modulus is used to update using backpropagation to update the agent.

REINVENT has multiples running modes. The one relevant for this work is the one that involves Reinforcement Learning. In this section, this running mode of REINVENT is introduced to the reader.

The operation of Reinforcement Learning mode can be split into three phases:

1. Generate batches of molecules
2. Score and Filter generated the batches of molecules.
3. Improve generation based on the results of the previously generated batches.

Generate molecules

In this first step, a question arises: *How to teach a computer to produce molecules?*

The solution to this problem is the implementation of generative models. A generative model, through samples, approximates a distribution. This approximate distribution can, in turn, be sampled, allowing for the generation of new samples that can mimic the ones coming from the original distribution.

In this particular case, REINVENT relies on a Recurrent Neural Network (RNN) (Abiodun et al., 2018) model trained on the dataset ChEMBL[2] (Gaulton et al., 2017) as a generative model. The selection of RNN comes from their success as generative models in various fields. Examples of this include language generation (Jozefowicz et al., n.d.), speech generation (Graves et al., 2004), image generation (Kalchbrenner et al., 2016), generation of sequences video (Srivastava et al., n.d.), and generating music of a particular style (Eck et al., n.d.).

An RNN works in the following way. Given a sequence of elements $(w_1, w_2, \cdots, w_n)$, an RNN model outputs a distribution probability for the next element: w_{n+1}. For example, the language model receives the sentence “Mondays are,” and will return a probability for a word in the vocabulary to be the next in the sentence. In this case, the word “boring” will get the probability of 0.8, “red” will get the probability of 0.2, and “apple” will get 0.

An RNN has the following construction. It consists of a function R. This function can either be a standard sequence of perceptron layers or more complicated structures, such as LTSM memory cells. This function receives a hidden state vector h_i and an input w_i and returns a hidden state vector h_{i+1}.

[2] It is a manually curated chemical database of bioactive molecules with drug-like properties represented as strings

$$R(h_i, w_i) = h_{i+1}$$

The resulting h_{i+1} vector is used to obtain the probability distribution of the next element in the sequence w_{i+1}: There are more advanced architectures, but this is the essential idea of the construction.

When a generative model is used, a start token is given as an initial input. This start token is feed into the R function with the initial hidden state, and this provides an updated hidden state and a prediction of the first element in the sequence. This prediction is feedback into the R function with the updated hidden state. This process updates the internal vector state again and gives a forecast for the second element. This set of steps is repeated until a termination token is the predicted element. (Olivecrona et al., 2017)

For the training, the approach relevant to this work is Teacher Forcing. For a sequence $(w_1, w_2, \cdots, w_n)$, the element w_i is feed into the function R with the hidden state vector h_i.

The hidden state vector is updated:

$$R(h_i, w_i) = h_{i+1}$$

The probability distribution of the next value in the sequence is generated. The distribution is compared with the next element in the sequence with the loss function. The process is repeated for the element w_{i+1} and hidden state vector h_{i+1} until the sequence is concluded. The critical part is that the prediction of the next element is the sequence is never used to update the hidden state vector (Olivecrona et al., 2017). The losses of each element are used to backpropagate through time (Mehlig, 2021). This procedure is done to tune the parameters of the neural network model. Graphic representation is shown in *Figure 2*.

Regarding the use of these models to generate compounds, there has been work done, such as the one of (Segler et al., 2018). A successful generative model for molecules is implemented using molecules in a string representation to train the RNN. The representation is the SMILES notation, which stands for *Simplified Molecular-Input Line-Entry System* (Weininger, 1988). Segler gives a good description of the SMILE notation:

> SMILES is a formal grammar which describes molecules with an alphabet of characters, for example **c** and **C** for aromatic and aliphatic carbon atoms, **O** for oxygen, and **−**, **=**, and **#** for single, double, and triple bonds. To indicate rings, a number is introduced at the two atoms where the ring is closed. For example, benzene in aromatic SMILES notation would be **c1ccccc1**. Side chains are denoted by round brackets. To generate valid SMILES, the generative model

would have to learn the SMILES grammar, which includes keeping track of rings and brackets to eventually close them.

To give the reader and idea of the appearance of the SMILE notation, the representation of glucose (sugar) is given:

C(C1C(C(C(C(O1)O)O)O)O)O

The work by Segler shows that you can successfully train an RNN to generate compounds that respect the syntax of the SMILE.

In summary, REINVENT is initially trained using the Teacher Forcing approach and a data set of SMILES. The resulting trained network knows the syntactic structure of the SMILE notation. The fully trained RNN obtained at the end of this process is denominated as **Prior**.

Additionally, when generating compounds, the RNN model is run multiple times such that batches of compounds are outputted.

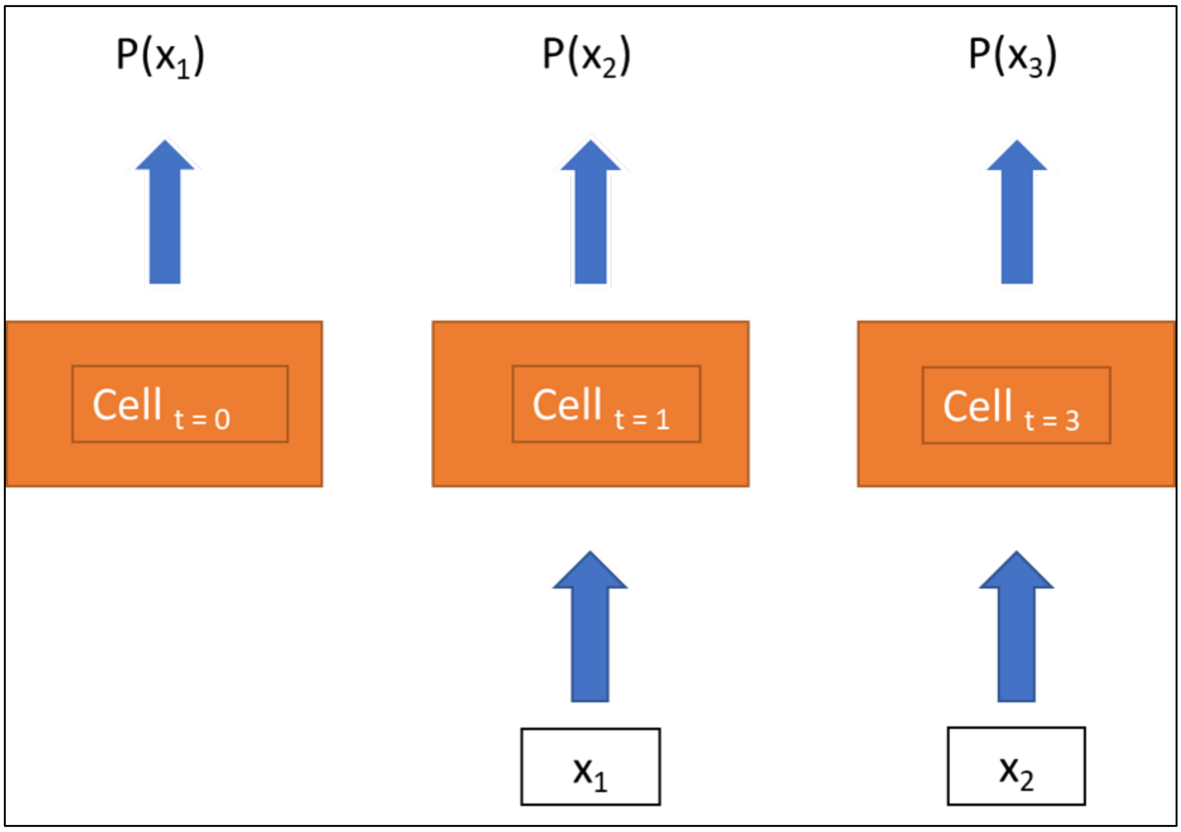

Figure 2. For the training, the RNN outputs probability distributions for the first **$P(x_1)$**, second **$P(x_2)$**, and third **$P(x_3)$** tokens in the sequence. The teach forcing approach does not sample those distributions to get an input for a cell. Instead, the tokens **x_1** and **x_2** are taken directly from the sequence used for the training.

Scoring and Filtering

REINVENT performs **three** types of evaluations on the molecules produces by the generative model. Each of these evaluations addresses a different aspect of the batches of compounds outputted by the generative model. The three evaluations are the **Scoring Function**, **Diversity Filter**, and the **Prior Likelihood.**

The first one is the **Scoring Function**. It is the normalized sum or product of individual functions that access how well the generated compounds follow the properties selected by the user. The formulas for the normalized sum and product of the functions are presented below. w_i refers to the weight assigned to a function i and $p(x_i)$ to the score given by a function i (Blaschke et al., 2020) :

- Normalized product:

$$\left(\prod_{i=0} p(x_i)^{w_i}\right)^{1/\sum_i w_i}$$

- Normalized sum:

$$\frac{\sum_i w_i \cdot p(x_i)}{\sum_i w_i}$$

These individual functions that score certain characteristics in the generated compounds will be referred to as **Components from this point forward**. These components have parameters and inputs that the user defines. For example, the molecular weight can be scored. In this case, the user controls the range of molecular weight that receives good scores.

A list of relevant components to be used in this work are introduced with the formal description given in REINVENT's documentation:

- **Matching Substructure**: Evaluate whether the generated compound contains a particular structure defined by the user. "Requires a user-defined set of (SMILE). This is a penalty component. Returns one if there is a substructure match and 0.5 otherwise." (Blaschke et al., 2020)
- **Tanimoto Similarity:** Measures how similar are generated compounds to a predefined one. "Requires a user-defined set of SMILE and returns the highest similarity score to the provided set." (Blaschke et al., 2020)

- **Jaccard Similarity:** Measures how dissimilar are generated compound to a predefined one. "Requires a user-defined set of smiles and returns the lowest distance score to the provided set." (Blaschke et al., 2020)
- **Activity:** Promotes the activity of the produced compounds towards a specific target. "Uses scikit-learn models. Works with both classification and regression models." (Blaschke et al., 2020)
- **Selectivity:** Promotes that the produced compounds are not active towards a specific target. "Uses two scikit-learn models. Works with both classification and regression models. One model is predicting the target activity and the other is providing an off-target prediction. The score is reflecting a user-defined activity gap between the target and the off-target predictions." (Blaschke et al., 2020)

The **Diversity Filter** (DF) controls that the batches produced have variety. Given that the RNN generates batches of compounds, the DF controls that the production does not suffer a mode collapse. A mode collapse refers to the situation where the production given by an RNN is composed of a single compound (Blaschke et al., 2020). For the operation of the DF, the user defines a score threshold and a limit for similar compounds. Every time the RNN outputs a batch of compounds, their scores, given by the scoring function, are calculated. If the score of a compound is equal to or higher than the threshold, then the scaffold[3] is extracted. There is a memory space, called a *bucket*, for every scaffold. Then, the generated compounds are stored in a specific bucket depending on their scaffold. If the amount of molecules in a bucket exceeds the limit, then the score for a newly generated compound (which belongs to a full bucket) becomes 0.

Prior likelihood controls that the produced molecules have correct SMILES syntax. For every compound A generated by the RNN, there are N tokens $(x_N, x_{N-1} \dots x_1)$ in the string. The likelihood, in this case, is $p(A = (x_N, x_{N-1} \dots x_1)\,)$, which is the probability of generation of the entire sequence.

This value corresponds to:

$$p(A = (x_N, x_{N-1} \dots x_1)\,) = \prod_{i=1}^{N} p(T_i = x_i | T_{N-1} = x_{N-1} \dots T_1 = x_1)$$

The probabilities $p(T_i = x_i | T_{N-1} = x_{N-1} \dots T_1 = x_1)$ are provided by the RNN model after the training with the ChEMBL dataset, denominated **prior**. These probabilities are dependent on the previous tokens in the sequence. (Olivecrona et al., 2017)

[3] Basic structure of the compound which refers to the way the bonds are arranged.

In this case, Log-Likelihood (LL) is:

$$LL(A)_{prior} = -\ln(p(A)) = -\sum log\big(p(X_i = T_i | X_{i-1} = T_{i-1} \ldots X_1 = T_1)\big)$$

The value of the LL assesses how unexpected is the production of a compound. The result can be interpreted as a measurement of whether the generated compounds are within the sampling space of a RNN trained to know the basic SMILE syntax. For example, a low value can be interpreted as a sign that a generated string deviates from what a SMILE should look.

All the evaluations are aggregated as:

$$LL(A)_{augmented} = LL(A)_{prior} + \sigma \cdot SF(A)$$

Here $SF(A)$ is the **Scoring Function**, taking into account the effect of the **Diversity Filter**. $LL(A)_{prior}$ is the **Prior Log Likelihood**. The final score that a molecule *A* gets is interpreted as the interplay between two factors: preservation of the syntax and exploration of the chemical space. Given that $LL(A)_{prior}$ measures the basic syntax of the SMILES, it works as a baseline value measuring how much *A* deviated from what a SMILE must look. The second term, $SF(A)$, gives a value for how well the generated compound follows the desired properties. The sigma factor (σ) is a parameter defined by the user, and it controls the level at which the scoring function can overrule the LL. A compound might yield a high value on the scoring function and a low value LL. The value of σ will control which of the two characteristics has predominance.

Reinforcement Learning Basics

In the situation present at hand, there is not a target to which to compare the result. Consequently, this situation is not classification tasks where the result is compared with a target using a loss function. Then, a resulting loss is used as a minimization target for backpropagation to modify the model's parameters.

The only information about the performance is the score. The objective is to modify the parameter to maximize the score by exclusively using the scores as input. The problem is then one of exploration through the parameter space of the RNN, looking for some set of parameters that will maximize a score. Reinforcement Learning (RL) is the natural fit for this problem.

Reinforcement learning works with situations where the aim is to maximize a numerical reward in a closed-loop process. The agent is not told what actions to take but must discover which actions are needed to get the highest rewards (Sutton & Barto, n.d.).

A brief description of the formulation of Reinforcement Learning is given. In RL, there is a set of states **S**. The learner (or as it also called an agent) can exist at any state s, which is part of the set **S**. Additionally for each state, there is a set of possible actions available for the agent to do. By taking an action, the agent suffers a change in its state. The next state, s_{new} , reached by the agent, after taking an action $\boldsymbol{A}$ from a state s_{new} , is given by a probability distribution:

$$P(s_{new} \mid s_{old}, A)$$

Given the transition from state s_{old} to state s_{new} through an action A, a reward is defined as:

$$r_A(s_{new} , s_{old})$$

The rewards are a value returned to the agent that assesses the quality of the action taken at a particular state. A policy gives the action done by the agent throughout the evolution of states:

$$\pi(A|s)$$

The policy is a probability distribution over all available actions. The objective of an RL problem is to find the policy that optimizes the value function:

$$V^{\pi}(s) = E\left[\sum_{i=0}^{\infty} \gamma^{i} \cdot \mathrm{r}_{i} \mid s_0 = s\right]_{\pi}$$

This value function is the cumulative value of all rewards obtained by the agent through a given policy π and an initial state s. The factor γ is a discount factor with a value between 0 and 1. It indicates how important are the rewards received at states reached earlier in comparison to more recent ones.

Reinforcement Learning in REINVENT

In the context of REINVENT, the RNN is the agent, the set of parameters used for RNN is the state, and the actions would be SMILE ***A*** outputted by the RNN. The policy (the probabilities for which action to take) are the probabilities of producing any SMILE ***A*** given by the RNN:

$$\pi(A|s) = \prod_{i=1}^{N} p(T_i = x_i | T_{N-1} = x_{N-1} \ldots T_1 = x_1)$$

The policy is a parametric model. Under these conditions, the way to proceed is through a policy gradient method. This family of techniques optimize a parametric policy through gradient descent (Zeugmann et al., 2011). In particular, the method selected by REINVENT for the optimization is the following:

1. The agent produced a batch of SMILEs.
2. The loss is calculated: $-\left(LL(A)_{augmented} - LL(A)_{agent}\right)^2$.
   - $LL(A)_{agent}$ is the likelihood of SMILE under the current parameters of the agent.
3. The parameters of the RNN model are modified through backpropagation using the loss defined above.
4. The steps are sequentially repeated.

The justification for this update rule is given by the work of (Olivecrona et al., 2017). The proposed algorithm was shown to be equivalent to the REINFORCE algorithm, which is one of the classic policy gradient algorithms.

The last element left to introduce into the context of REINVENT is the inception modulus. It works in the following way. A space in memory for a fixed amount of compounds is created. In every iteration, the memory is updated. During the update, the highest-scoring molecules found so far during the run are kept. These molecules are re-used in every update step of backpropagation. The objective of the inception modulus is to fulfill the function of an experience replay and prevent the agent from losing track of which are the best results found so far.

Curriculum Learning

The modification proposed in this work is the inclusion of Curriculum Learning (CL) within the context of REINVENT. The first important step is defining what CL is and some context of successful implementations in the past.

The term CL is borrowed from the field of education. In any educational institution, topics are arranged, so that previously learned subjects are leveraged to understand more advanced topics. This construction helps because things learn at specific points will give tools for what is to be learned in the future. The most classic example of this construction is a math class. A student first needs to know how to add, then the basic training in adding makes multiplying easier. Afterward, the knowledge of adding and multiplying makes the job of integrating easier. This kind of intuition is the motivation behind CL.

In formal terms, it can be described as narrowing solutions space by starting from a less constrained problem with broader solutions space and guiding the learning process by gradually complicating the problem. In practical terms, CL is performed by designing a sequence of tasks M_1, M_2, . . . M_t for an agent's training. This process is done so that learning speed or performance on a target task M_t is improved. (Narvekar & Stone, 2018).

Implementations done in the past give credence to its capacity to improve the performance of RL learning tasks. Some examples include the following works. A set of face pictures is divided between "hard" and "easy." A face recognition software is trained in the collection of "easy" samples and then on "hard" samples. The result achieves better performance than training the face recognition software on the entire set from the start (Huang et al., 2020).

In a study (Bengio et al., 2009), a convolutional neural network architecture is designed to recognize whenever a given picture contains rectangles, triangles, or ellipses. Two data sets are created; one has special cases like squares, equilateral triangles, and circles; and the other has more general shapes, proper rectangles, non-equilateral triangles, and ellipses. The CL setup is done by two stages of learning. First, the network is trained on a data set with only the special cases. After a defined number of iterations, the training data set is switched, and the training proceeds with the dataset of more general shapes. For comparison, a copy of the same network architecture is trained on a union of both data set. According to the author of this study, the performance is much worse than the case without CL.

According to the literature, for Curriculum Learning, there are three essential parts. First, one defines which are the different stages of the Curriculum (the tasks the agent is trained on). Second, one determines the order in which each stage of the Curriculum takes place. Third, one trains the agent in the Curriculum. It is important during the training to prevent that by learning something new, there is a catastrophic forgetting of earlier stages of the Curriculum (Narvekar et al., 2020).

In this case, the Curriculum is a series of components used to modify the Scoring Function used on the REINVENT reinforcement learning loop. Additionally, as it is usually understood in CL, the order of the stages goes from the easiest to the hardest. Then, the components would be ranked based on which can train an agent to generate optimum compounds more efficiently. Then, this ranking is used for the training of the agent.

Methods

In this section, the work done for the implementation of Curriculum Learning in REINVENT is introduced. First, it is explained how the different components used in the Curriculum are obtained from the user-defined scoring function. Second, it is explained how the components are ranked. Third, it is explained how the agent is trained on the Curriculum. All this process is done through the three newly implemented methods: Ranking, Merging, and Production. These methods are also explained.

As a technical note, the construction of the methods (Ranking, Merging and, Production) allows for multiple running modes. As a way to implement these different running modes, a structure of abstract classes is used. In this construction, a base class with global functions is defined for each method. The different running modes of each method use the abstract class (they share common internal functions), but at the same time, each one can have unique internal functions on its own.

Components of CL process

Two scenarios are considered for obtaining the components used in the Curriculum: **1)** the scoring function is composed of a single unique component, and **2)** the scoring function is composed of a set of different components.

In the case of a scoring function that rewards a single property (it is made up of a single component), then Curriculum is formed of multiples components of the type as the one used on the scoring function. The conditions defined in those components are less restrictive so that they are “easier” to learn. For example, suppose the scoring function gives rewards to compounds that are within a specific range of molecular weights. In that case, the Curriculum will be a set of components that also provide rewards to compounds that fall within a range of molecular weight, but each will have a broader range. Here, it is a user-defined process. It is also important to note that those components need to have increasing difficulty. In the example, this will correspond to a progressively narrower interval of molecular weights. The user defines the number of functions used and how to relax the conditions.

In the case of a scoring function with multiple components, the generation of the functions used by the Curriculum is automatic. The components used are the individual components of the scoring function. It is expected that optimizing the individual components is easier than optimizing the whole scoring function, as there are fewer restrictions imposed on the output of SMILES. *Table 1* summarizes the way Curriculums are built.

Table 1. This table is a summary of which functions are used in the Curriculum, depending on the situation.

Cases	**Curriculum Functions**
A scoring function made of a single component	Main scoring function + functions that are the same component but easier to learn.
A scoring function made of different components	All the individual components that are used in the scoring function.

Ranking Stage

Before the initialization of the Reinforcement Learning Loop, the first task is to rank the complexity of the different parts of the Curriculum to define the order in which they will be used. A base method denominated Ranking performs the ranking. The method has three running modes, implemented during the work of this thesis. They order the parts of the Curriculum in different ways. The aim of all three is to create an order in which the components are sequentially more difficult. The three methods are:

- User-defined ranking
- Score-defined ranking
- Iteration-defined ranking

The user-defined ranking, as its name indicates, allows the user to order. This method is helpful in the context where the order of the Curriculum is self-evident. An example of this is when the scoring function has single component, and a set of “easier” components are used as Curriculum. In this case, the user knows which ones are the easiest, as he/she defined them.

The Score-defined ranking and Iteration-defined ranking are used in the context of a scoring function with multiple components. The idea of these two methods is to assess how difficult each component is. Then, the components are order in the Curriculum from the easiest to the most challenging. These running modes are used when there is uncertainty about the optimum way of performing the Curriculum.

In particular, in the Iteration-defined setup, each component of the scoring function is used to train an identically initialized agent on an independent RL loop with. The ordering used is done according to which component took less time to reach a user-defined score threshold. The Score-defined setup has a maximum number of iterations, and the ranking is done according to which component gets the highest score during a user-defined time limit. An illustration of both running modes is presented in plot *Figure 3*.

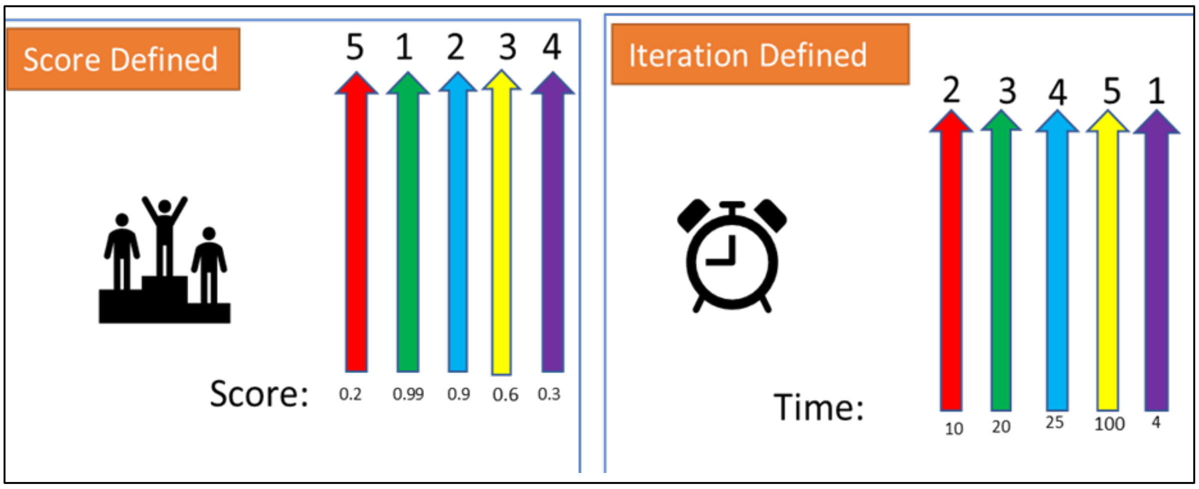

Figure 3: In both modes, each component is used as the scoring function in independent Reinforcement Learning Loops with an identical copy of the initial state of the agent. In the case when **Time Defined** mode is used, a score threshold is introduced by the user. The components of the scoring function are ranked according to the number of iterations it takes to reach that threshold. In the example give, each arrow represents a component; the number above indicates the final ranking used in the sequencing and the number below the number of iterations. When **Score Defined** mode is used, the user-defined a maximum amount of iterations, and the ranking is done in function of which component got the highest score during the period. In the picture, the number below is the total score obtained by each agent, and the number above is the sequencing.

Merging Stage

The final aspect to consider is how to teach the Curriculum. In practice, the following is done by the method Merging. Merging takes the ranking given by the Ranking method. Using the ranking of the components, an agent is trained by the Reinforcement Learning Loop, and the scoring functions are altered as the loop progresses. By alteration, it is meant replacing or including one or more new components to the scoring function used in the RL loop. The way in which the scoring function is updated is dependent on the running mode selected by the user. During the loop, the same agent remains, and only the feedback of the loop modifies the agent's parameters.

There are three Merging running modes, and they are differentiated by the criteria used to define when to update the scoring function and the way to perform those updates. The running modes are:

- Time-Based criterium
- Score-Based criterium with inclusion of components
- Score-Based criterium with replacement of components

The "Time-Based criterium" modifies the scoring function according to time. The user defines a fixed length of time. The component that comes first in the ranking is used to train the agent, and after the threshold time passes, a new component is included in the scoring function. The training loop continues with the updated scoring function, and after the time interval passes again, the change of scoring function is performed once again. These steps are repeated until the ranking is depleted.

The "Score-Based criterium with inclusion of components" uses a threshold score defined by the user. Every time the score of the output of SMILES gets to the threshold, the modification of the scoring function takes place. The changes on the scoring function consist of including a new component or multiple new components into the scoring function. This is done iteratively until the ranking is finished. "Score-Based criterium with replacement of components" uses the threshold score the same way as the "Score-Based criterium with inclusion of components" to define when to do the update. The difference is that all existing components in the scoring function are replaced by a single or multiple new components during the update. This is done iteratively until the ranking is finished. In *Figure 4*, a diagram is presented which gives a visual description of the two score base running modes.

"Score-Based criterium with inclusion of component" and "Iteration-Based criterium" methods try to preserve the knowledge gained by not discarding previous components present in the scoring function, but instead keeping previous components and simply update the scoring function by incorporating new factors. The reasoning behind this is that by keeping previous factors, the agent will be penalized if it begins to perform poorly in previously used components. On the other hand, "Score-Based criterium with replacement of component" does not preserve old components. The motivation for its inclusion is to give the user more flexibility. Some learning tasks might not require the continuous reinforcement; therefore, this running mode allows the user to dump previous components. The utility is that by drooping components, the number of operations required reduces. It is helpful if there is no improvement in the performance that can justify the computational cost of additional operations.

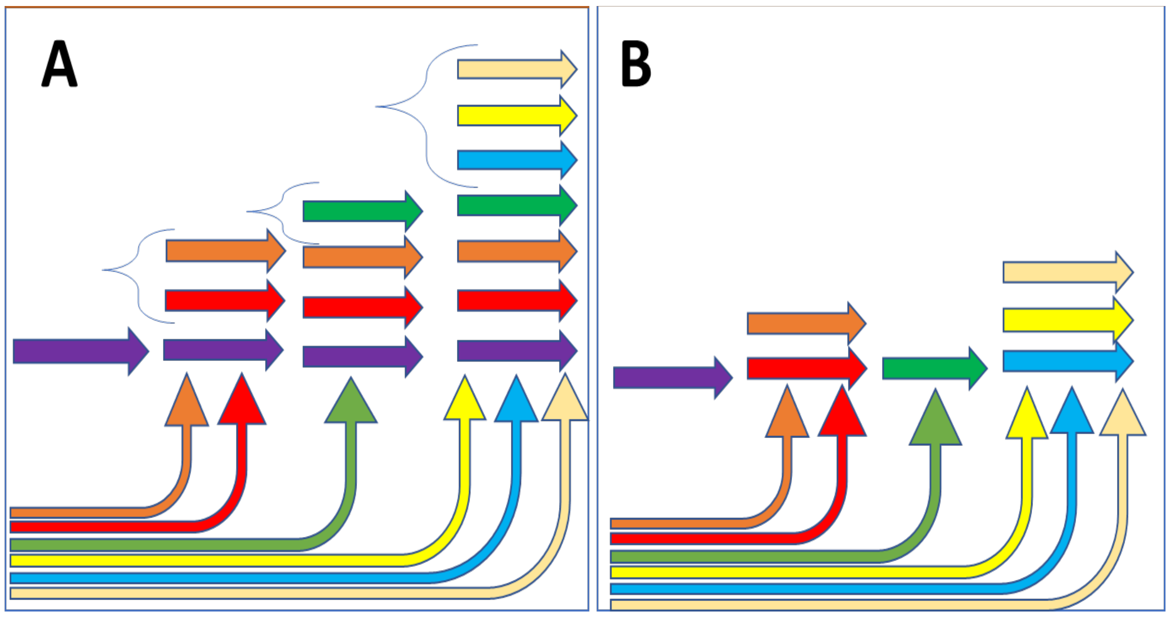

Figure 4. These diagrams give a representation of the evolution of the components considered in the scoring function used in the loop of RL. **(A)** Every time that the scoring function reaches a threshold, additional parts are included in the scoring function. It is essential to clarify that multiple components can be added in each update round and not just a single one. **(B)** Every time that the scoring function reaches a threshold, the update eliminates all components used in the scoring function up to that point and replaces them with a new set.

Production Stage

After the final configuration of the scoring function reaches the user-defined threshold score or time limit, the mergining phase ends. The resulting agent is considered to be fully trained with the Curriculum. The next part of the implementation is denominated as Production. This stage has two objectives: to use the trained agent to produce novel compound ideas and to perform further training in the agent in the case that it is deemed necessary by the user.

The way to perform this is by introducing the agents generated at the end of the merging phase into another RL loop. The learning rate in this stage is independent of the one present during the merging phase. This detail gives freedom to the user to reduce it as much as possible if it is deemed that additional learning is unnecessary. Here, the compounds generated by the agent are stored in memory.

There are two running modes for this stage:

- Production with all components.
- Production with selected components.

"Production with all components" simply uses all the components present in the Curriculum to evaluate the agent's output during the production phase. Meanwhile, in the "Production with selected components" mode, the evaluation is done with a set of user-selected components. The introduction of this can give the user the ability to know how the fully trained agent is performing in a group of metrics and further train on them if it is deemed necessary. Diagrammatic representation of the components used in this phase is presented in Figure 5.

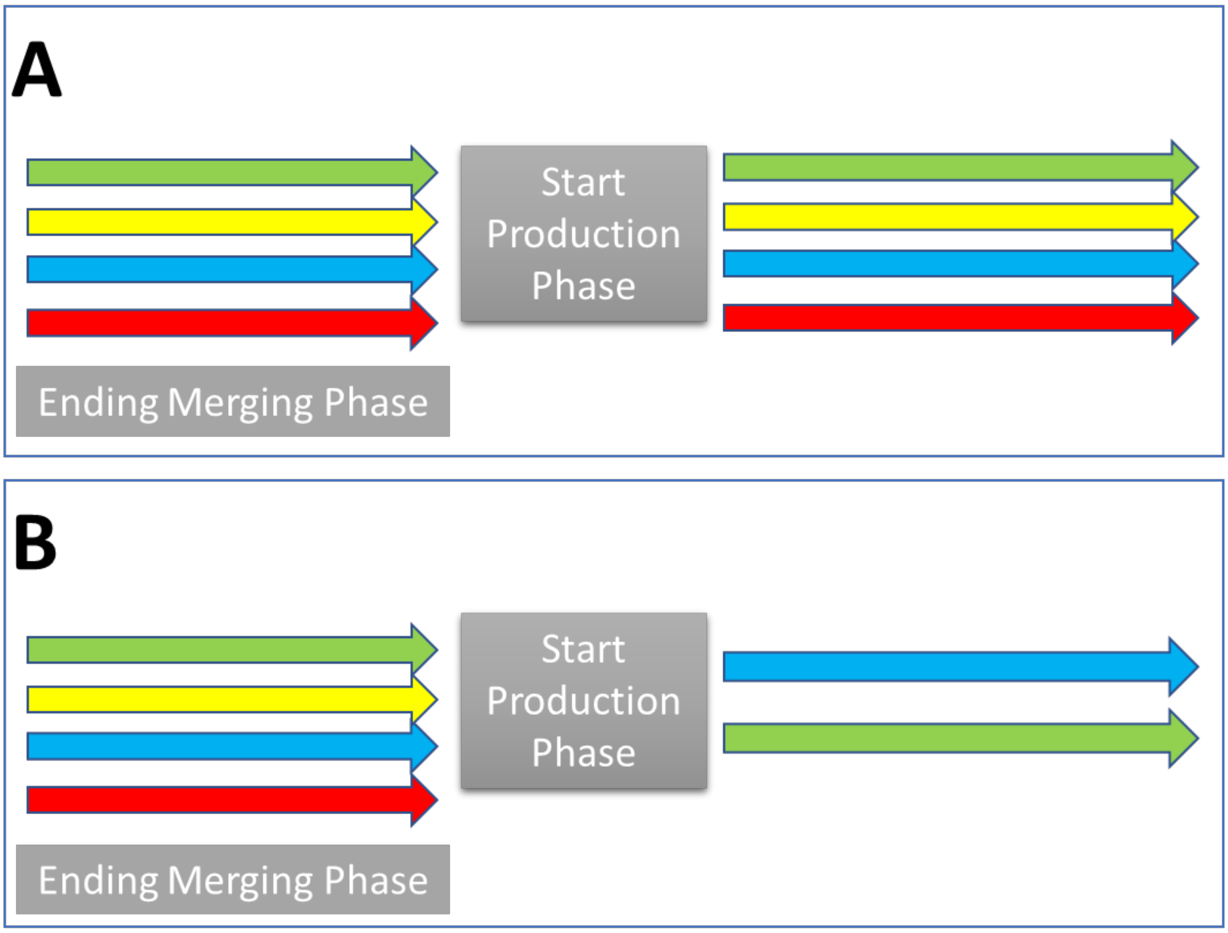

Figure 5. The production step can be defined in two ways. **(A)** Evaluate the output of the agent overall at the components used in the merging phase. **(B)** The user can select specific components for this phase and restrict the evaluation to just those metrics.

The conclusion of this section is the diagram in *Figure 6*. It shows the union of all the implementation done to give an overall understanding of all the previously mentioned steps.

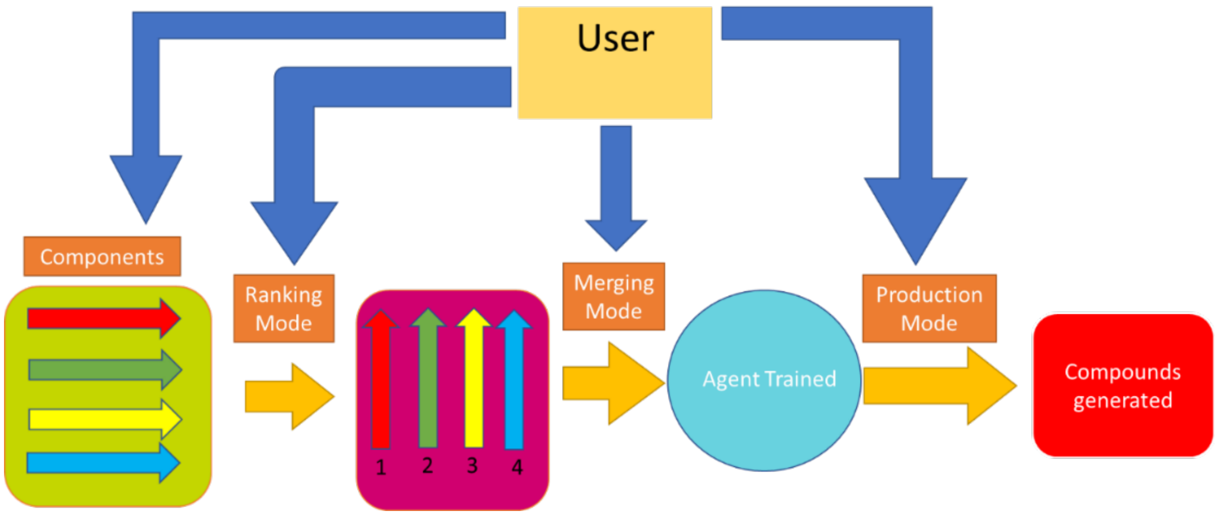

Figure 6. The overall structure has the following workflow. **First**, the user defines a set of components. This set is the components that are used to build up the Curriculum sequentially. **Second**, the user establishes a method of Ranking that builds the sequencing of the components. **Third**, the user establishes a method for training the agent using the generated sequencing. **Fourth**, the resulting fully trained agent is outputted, and the user selects a production method. **Finally**, the newly generated compounds are stored in memory.

Experiments

To examine the implementation of Curriculum Learning on *REINVENT,* four study applications are examined. The aim is to observe how and to which degree CL can lead to an improvement in an agent's performance.

Compounds with complex substructures.

The most direct application of *REINVENT* is its use to create new compounds that contain a predefined structure. The issue that can be encountered is that the problem can become intractable if the desired structure is complex.

In the standard case without Curriculum Learning, the way to approach the problem is by using a single "matching substructure" component as a scoring function. This scoring function gives a score of 1 when a generated compound has the desired target and 0,5 otherwise.

The proposed Curriculum is done in the following way. Given that the objective is to optimize compounds for a single "matching substructure, then the Curriculum is a sequence of the "matching substructure" components with sequentially more complicated structures used as targets. Additionally, the target structure in one stage will be a substructure in the subsequent stages of the Curriculum.

The motivation for this approach is that if one agent trains with an easy target, then the agent will not struggle and begin producing compounds that contain the substructure. As the substructure becomes more complex, then the compounds outputted by the agent will already be related to the new objective, as they will share a substructure. Given the similarity, the agent will generate an appropriate compound by chance and sequentially improve the generated batches. It can receive a signal indicating where the high rewards are within the chemical space.

The example selected is the generation of analogs to Camptothecin. This is a molecule that is an inhibitor of the enzyme Topoisomerase, which plays a role in the replication of DNA. As the objective is to create analogs to Camptothecin, the agent needs to output molecules with Camptothecin as a substructure. The sequence of structures used in the Curriculum is shown in *Figure 7*. The molecule used in each stage is a substructure of the molecule used in the next stage. The last compound present in the Curriculum corresponds to the target desire for all produced compounds to contain, which is Camptothecin.

For the update of the scoring function, the method selected is the “Score-Based criterium with inclusion of components.” This means that the previous components in the Curriculum are preserved, and the updates are done every time that the scoring function reaches a threshold value. The motivation for this selection is that in preliminary runs performed using the removal setup the agent stagnated in a score value every time that the scoring function was modified. This is a case when constant reinforcement of previously learned tasks is necessary.

As an additional remark, components 1, 2, and 3 are eliminated during the production phase, and the production proceeds only with the Camptothecin as the target in the matching substructure components. Additionally, training is allowed in this stage.

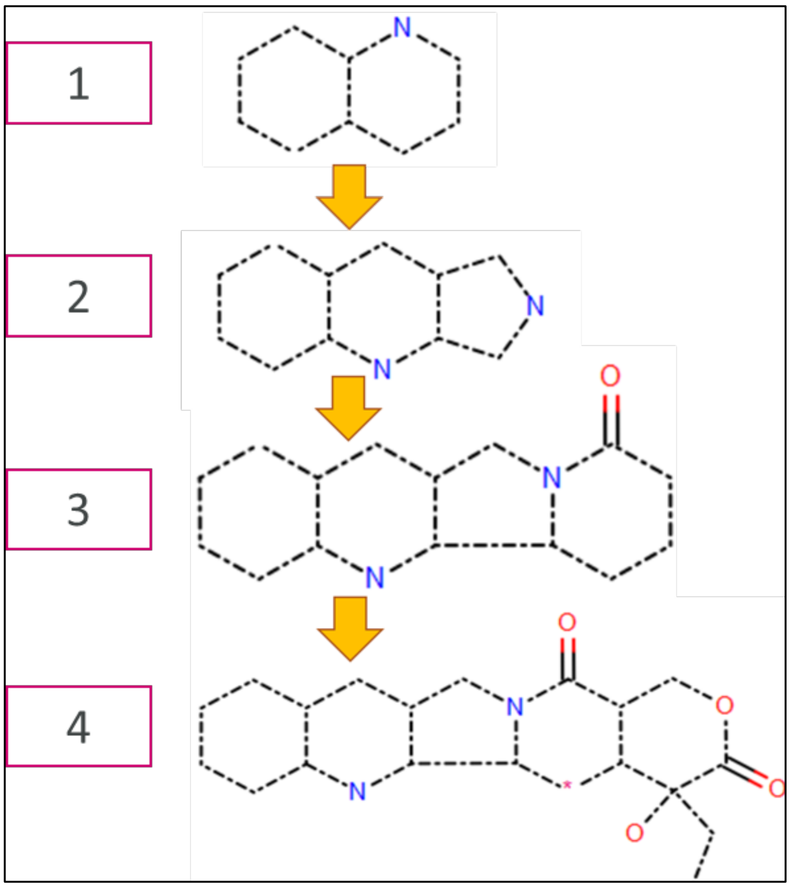

Figure 7. The Curriculum used for the training. Structure number **four** is Camptothecin.

Orthogonal Components

The next experiment considered for the application of Curriculum Learning is the training on a scoring function composed of two components that reward opposite properties. The idea is to observe whether the implementation of Curriculum learning can offer improvements in running time and the quality of compounds generated.

The reasoning for why Curriculum learning may offer a solution for this situation is that by ordering the different components, an agent can be trained to output compounds that have the characteristics rewarded by the first components. The introduction of the second component works as a restriction over the set of compounds generated. On the other side, if both components are used simultaneously, it is expected that any improvements in one will lead to a decrease in the other. This situation can lead to a stagnation situation.

For this application, two components are used: Jaccard Dissimilarity and Tanimoto Similarity. In principle, those two components reward opposite properties. Jaccard gives the highest score to compounds more different from a defined target; meanwhile, Tanimoto rewards compounds similar to a target. The scenario is to promote similarity to a defined target and encourage dissimilarity towards a set of compounds. This configuration should make the RNN output compounds that have a close appearance to the target but are not present in the group to which dissimilarity is desired. Camptothecin (see *Figure 7* compound 4) is used as a target for the Tanimoto Similarity, and a set of analogs are used for the Jaccard Dissimilarity. Some of the analogs included in the set are presented in *Figure 8*. It can be seen that the compounds in the set and the Camptothecin are similar. Thus, a significant improvement in Tanimoto Similarity will result in a decrease in the Jaccard Dissimilarly and the other way around.

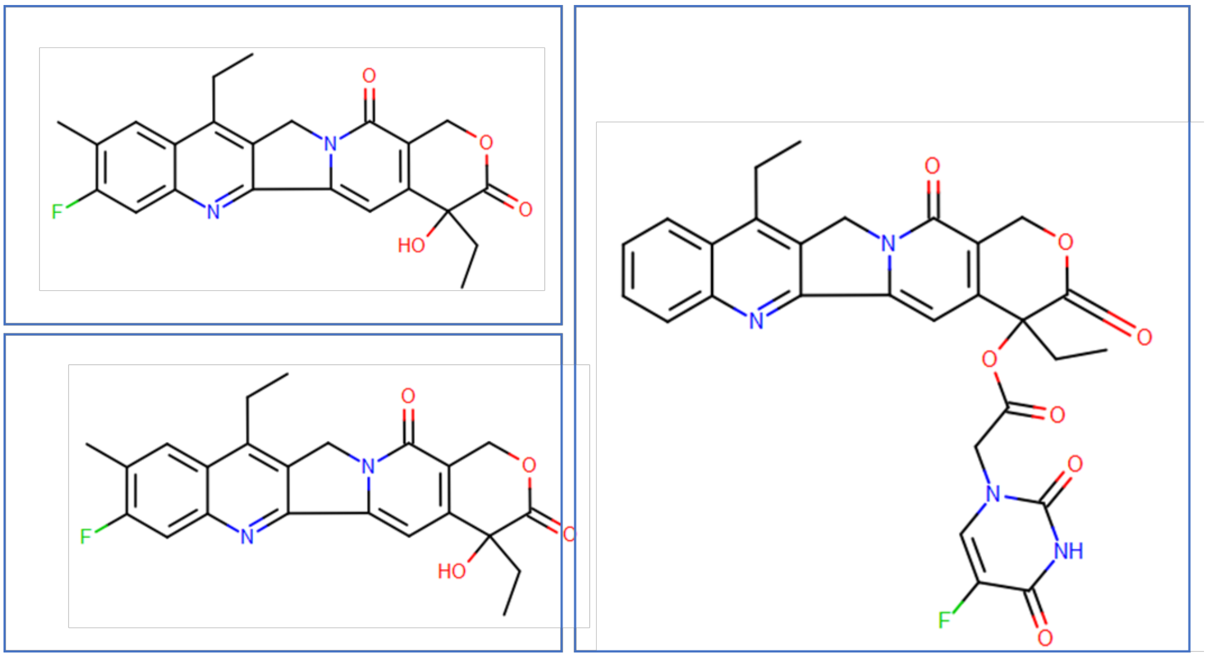

Figure 8. Set of analogs to Camptothecin used for Jaccard dissimilarity.

Additionally, for the sake of completeness, in the initial test, the agent struggled to train with only the Tanimoto similarity properly. The solution was to multiply the Tanimoto similarity with a matching substructure component using a simple molecule as target (see *Figure 7* compound 1). This change made the learning of the Tanimoto faster. Its use does not affect, as it a fairly simple structure that a solution should have. Also, it is used in multiplication, so after it becomes 1, the scoring function is only the Tanimoto component. When referring to the Tanimoto component, the actual component use is the conjunction of Tanimoto and the matching substructure components.

As the objective is to maximize both components (Jaccard dissimilarity and Tanimoto similarity) simultaneously, the scoring function uses both components. Then, the Curriculum is made up of the individual components of Jaccard dissimilarity and Tanimoto similarity. As both components can actively cancel each other, the training in the merging phase is done using the mode of “Score-Based criterium with inclusion of components.” This selection of mode in the merging phase is justified by the orthogonal nature of the components. If the agent trains on a single one, then the score for the other has the potential to decrease as a consequence.

As a final remark, the sequencing is selected as, first, Tanimoto Similarity and, second, Jaccard Dissimilarity. The dissimilarity component cannot be used to train in the initial stage, as it will always get a high score. The agent's initial production is very dissimilar to the set of forbidden compounds defined for Jaccard. This situation leads to a local minimum.

Selectivity and Activity

For the third application, the aim is to maximize Activity and Selectivity components at the same time. Those two components, also mentioned in the introductory selection, have the following purposes:

- Activity: This component rewards that the generated molecules are active towards a particular target.
- Selectivity: This component rewards that the generated molecules do not bind towards a particular target.

The idea is in principle similar to the case studied previously; the aim is to teach the agent to produce compounds that align towards a property but at the same time prevent another one. The difference here is that the situation is not orthogonal.

For the runs, the Activity target is the enzyme Aurora kinase and the Selectivity target is the protein B-RAF kinases. The aim is to maximize the score of both components. Additionally, the sequence of the Curriculum is to train first on the Activity and then on the Selectivity. This ordering is such because it is desired to have compounds that are first Active and then Selective. The training in the opposite order will produce compounds that are maximally distant to the B-RAF, but they might not have any interaction with Aurora kinase.

Also, a point to clarify is whether the update of the scoring function is done by replacing the components or adding them to the existing ones. The approach selected is to keep the trained components. The reasoning is that by eliminating the Activity, the subsequent training might improve the Selectivity to the detriment of the Activity. By keeping the Activity present, this problem can be prevented.

Selectivity and Activity using Diversity Filter

An additional study is done using the same Activity and Selectively Curriculum Learning setup with the inclusion of the Diversity Filter (DF). The DF was excluded from the first two experiments because the properties that the agent tried to optimize in those cases are related to similarity in structure. The inclusion of a DF is counterproductive, as this filter activity discourages structural similarity in the generated compounds, which is at the same time the property that is actively encouraged. In the case of Activity and Selectivity, the DF can be applied, as the two components are not as tied with the scaffold's structure.

Results

In all cases, the experiment done with Curriculum Learning are compared against a baseline case. These baseline scenarios are the agent's training on the desired objective without the use of Curriculum Learning. These cases are denominated as No Curriculum Learning or as No-CL. These additional runs are done as control cases.

Compounds with complex substructures.

In this case, the baseline is the training done only with a "matching substructure" component that will reward only compounds containing the desired Camptothecin target (see *Figure 7* compound 4). The CL and No-CL scenarios are shown in *Figure 9*. The main observation is that the score is not able to go beyond 0.5 in the No-CL case. This result means that the agent is not able to produce compounds with the Camptothecin substructure. Meanwhile, in the CL case, the agent gets a score close to 1.

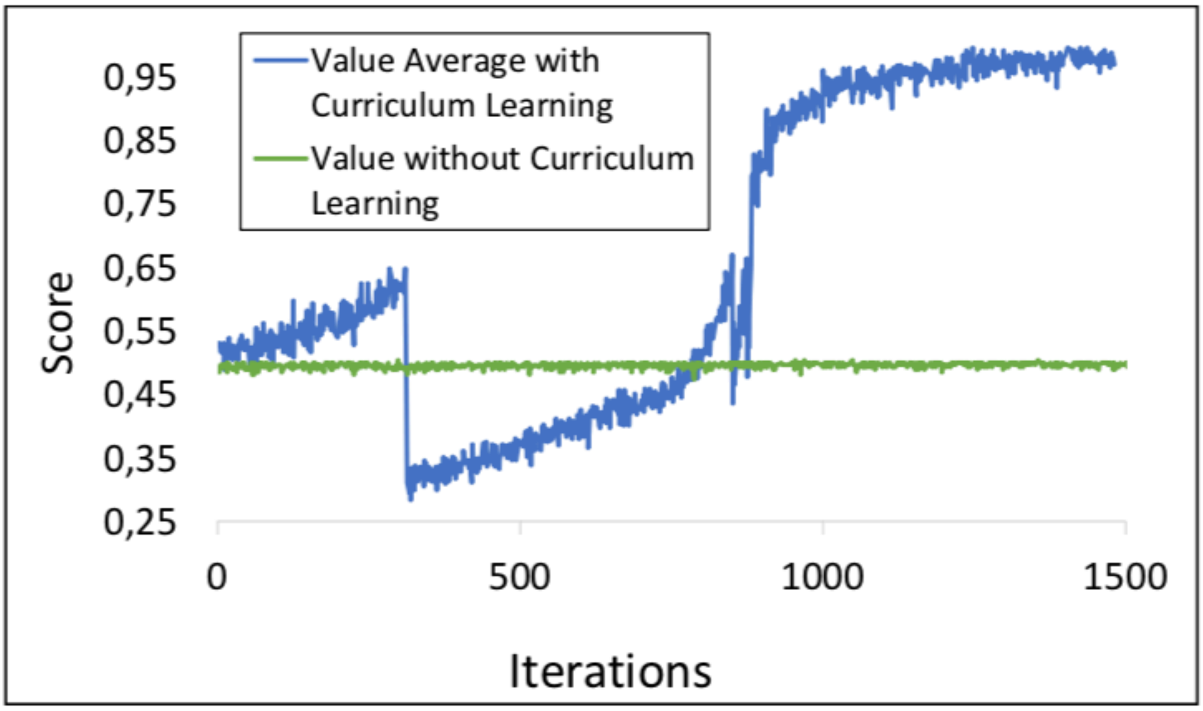

Figure 9. There are significant differences between the two training. In the No-CL case, the value of the score gets stagnated at 0.5. There are fluctuations in value in the case with CL, but the final score when the training concludes is very close to 1.

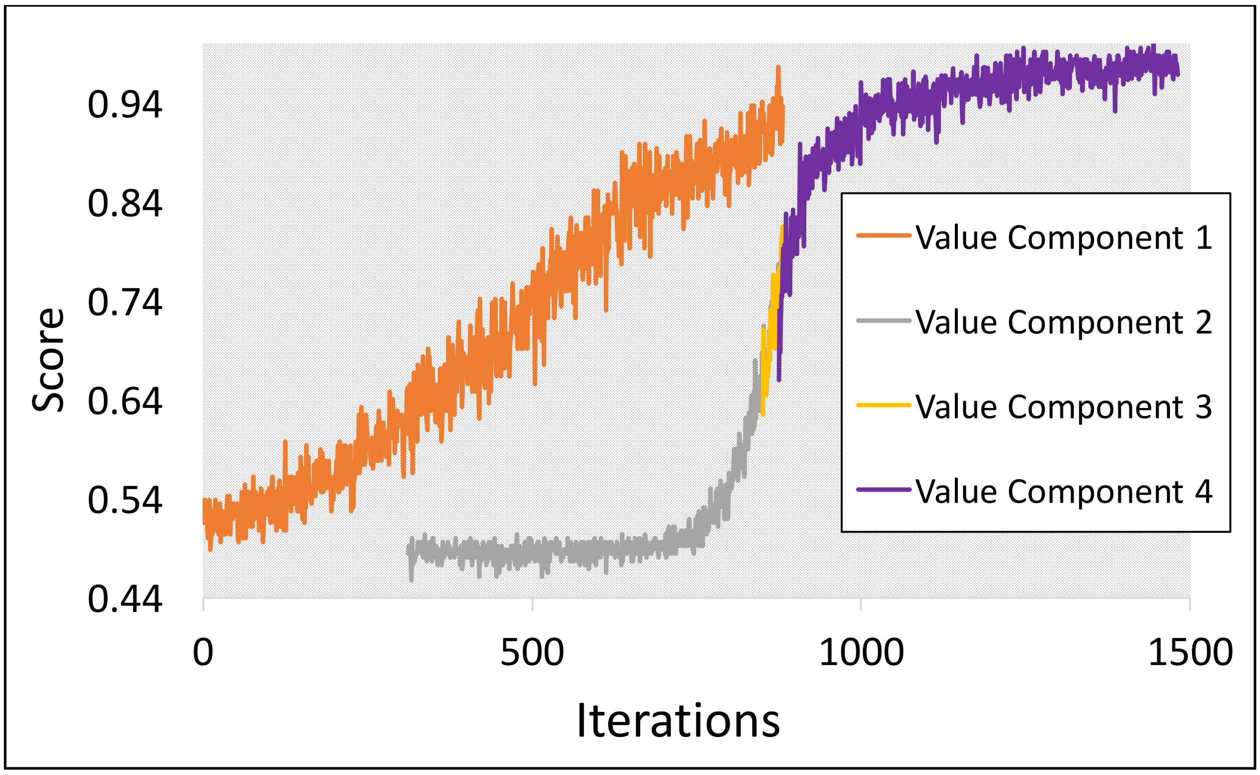

Figure 10. This picture shows the individual values of the components during the training of agents with the Curriculum Learning approach. For every iteration, the lines on it correspond to the components used for the training at the moment. The training with 1-2-3 lasts for a short period.

To assess the impact of the individual parts of the Curriculum, the score of each individual "matching substructure" component is individually plotted (see *Figure 10*). The results in *Figure 10* and *Figure 9* show that the significant fluctuations are due to the modification in the scoring function. Additionally, it shows that not all components seem to have an impact on the overall training. In particular, Component 3 does not seem to produce a considerable fluctuation, and the iterations need for the subsequent modification are few.

To observe whether the inclusion of Component 3 has any effect on the performance, the number of iterations to finish the merging phase is measured twenty times for different configuration of the Curriculum. The results are presented in *Figure 11*.

The figure shows that the distributions of iterations are very similar in the cases when the Curriculum has four stages and when three-stage Curriculums are used. This result suggests that there is an optimal number of components. In summary, a large number of components in the Curriculum does not lead to better performance.

The main observation one can draw from these experiments is that the construction of Curriculum in the context of forming compounds with defined complex substructures can, in principle, be benefited from the use of Curriculum Learning, as its use seems to make the training faster. Also, it is essential to remark that including a long Curriculum does not necessarily translate into faster performance.

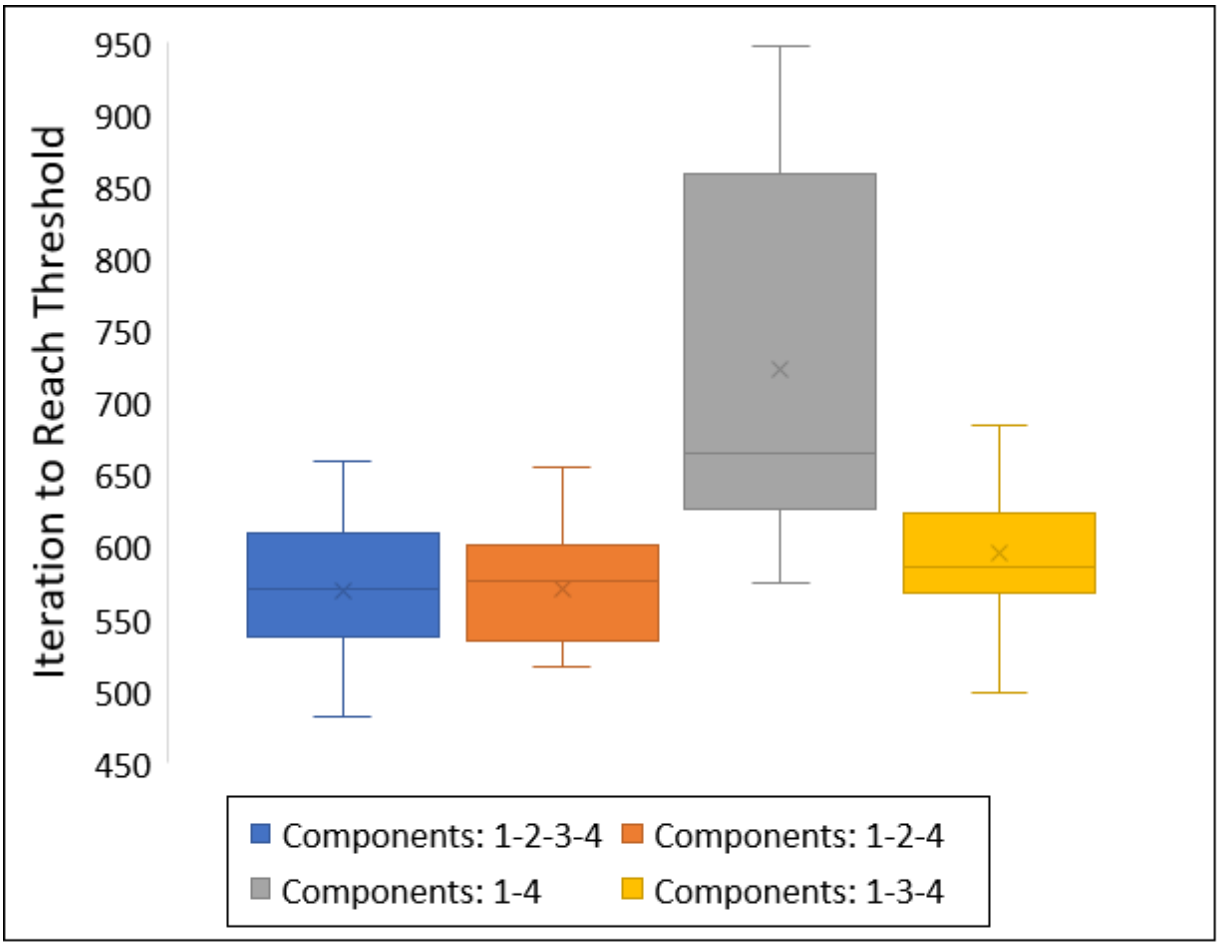

Figure 11. Observation of the distribution of iterations for four different Curriculums. That a Curriculum is more extensive does not translate into improved performance in comparison to cases with simpler Curriculums. This plot is generated using twenty runs for each box.

Orthogonal Components

As the No-CL baseline, the agent is directly trained with both components at the same time. Therefore, the first metric of analysis is the iterations required to finish the merging phase. This value corresponds to the number of iterations for the fully trained agent to overcome a threshold defined as 0.5. In the CL case, this number of iterations corresponds to the cumulative number of iterations required until the score obtained from a fully assembled scoring function gets to the threshold. On the other side, the number of iterations considered in the No CL setup is the total number of iterations that take an agent trained within the whole scoring function to reach the threshold score. Twenty runs of the CL and No-CL setups were performed to generate the data for comparison. The results are presented in Figure 12. The impact of the Curriculum is apparent in the performance. The inclusion of Curriculum Learning lowers the necessary number of iterations required, at the very least by half. Additionally, it seems to reduce the variation of the number of iterations.

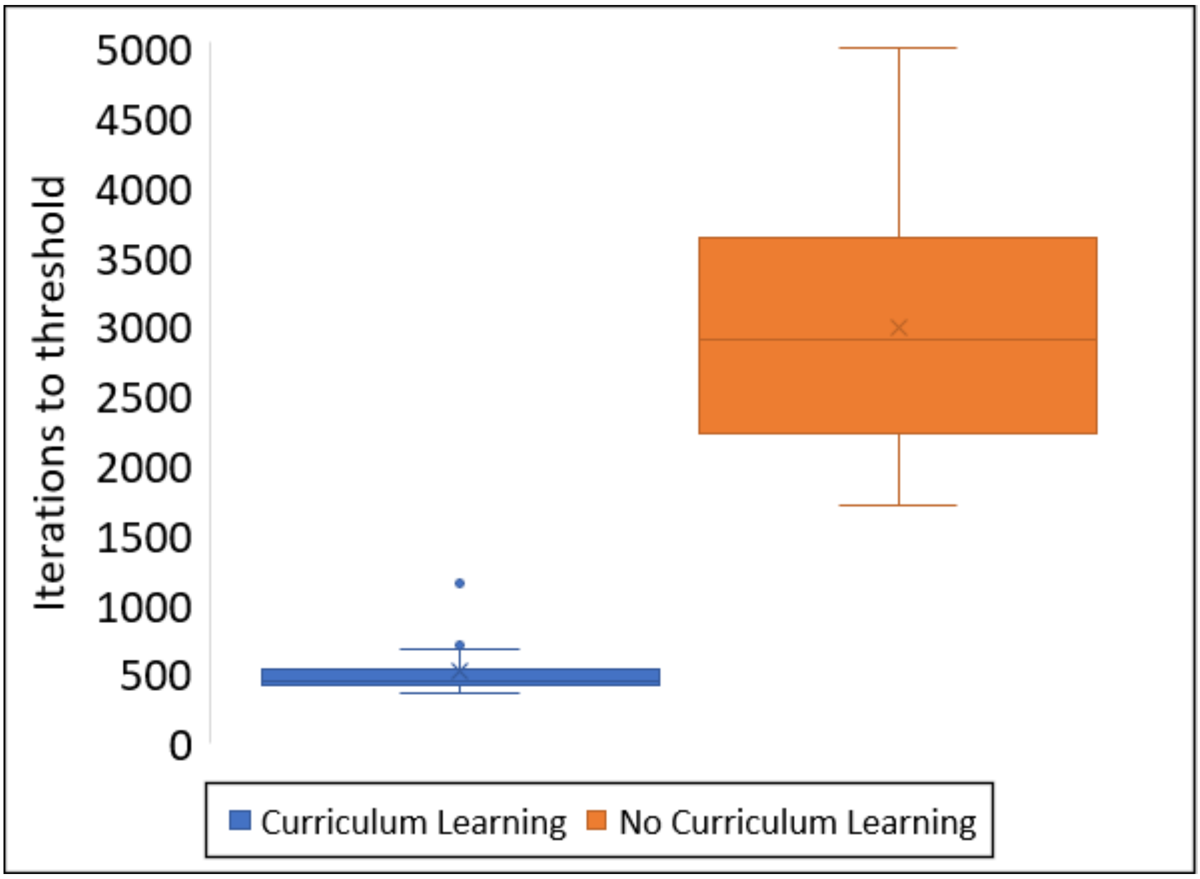

Figure 12. This figure shows the number of iterations. For each setup, 20 runs were performed. One can draw from these distributions that the Curriculum set is less expensive iteration-wise to reach a threshold training value. This plot is generated using twenty-five runs of REINVENT.

Two plots of the merging phase for both setups are provided (see Figure 13 and *Figure 14*) to study why there is an improvement in the performance. In the CL case, the training only maximizes the Tanimoto similarity. The later introduction of the Jaccard dissimilarity stops the maximization of the Tanimoto score. By introducing the Jaccard component, the agent begins producing compounds that keep having a similarity with Camptothecin but are as dissimilar as possible as the set of forbidden analogs.

The situation present with the No-CL approach is different. As the initial compounds outputted by the agent are far away from the set used in the dissimilarity, the agent gets an initial high score. This situation prevents exploration. The agent will not be able to move to a region where the Tanimoto similarity gives high rewards because the act of approaching that region would be punished by the Jaccard dissimilarity. This situation is in Figure 14. Here, the Tanimoto similarity stays low, and the Jaccard dissimilarity pushes up the value of the overall score.

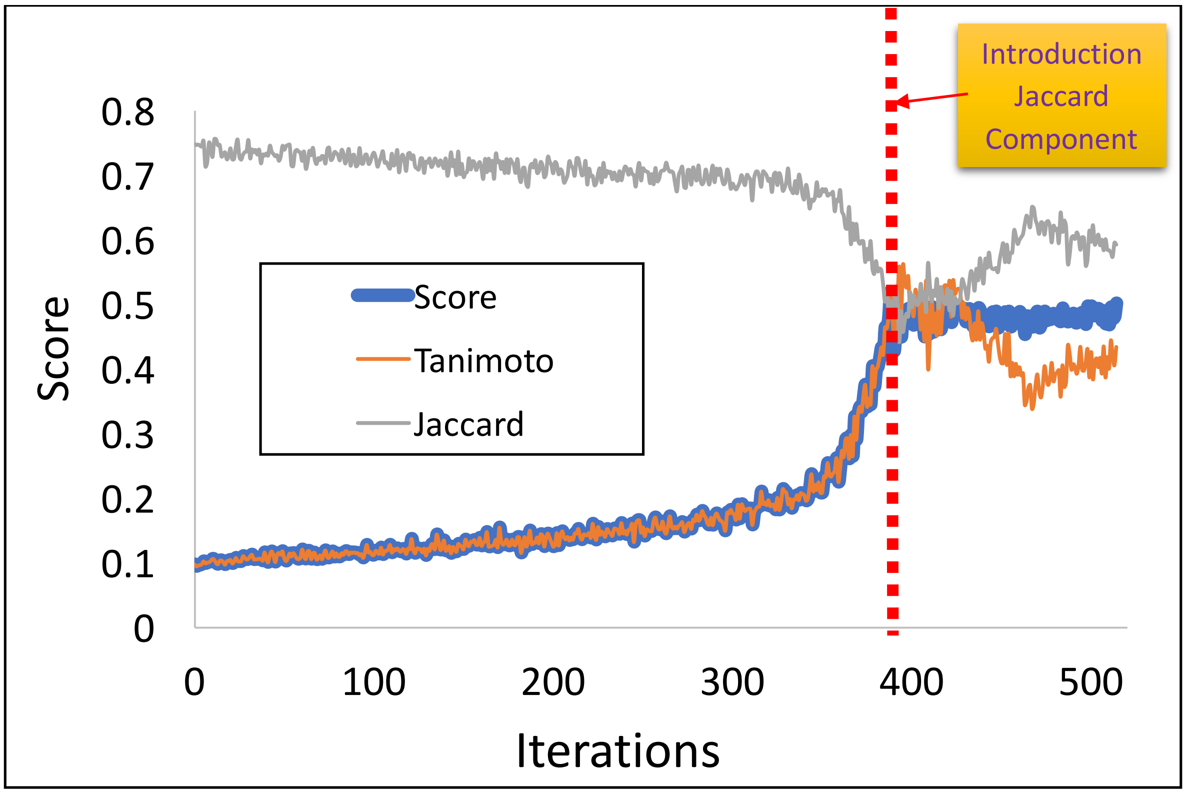

Figure 13. The plots show the behavior during the merging phase for the CL case. The Curriculum is to train first on Tanimoto and then train in the combination of Tanimoto and Jaccard. The change in Curriculum is performed at iteration number 380. Initially, the value of similarity increases until the inclusion of dissimilarity stops it.

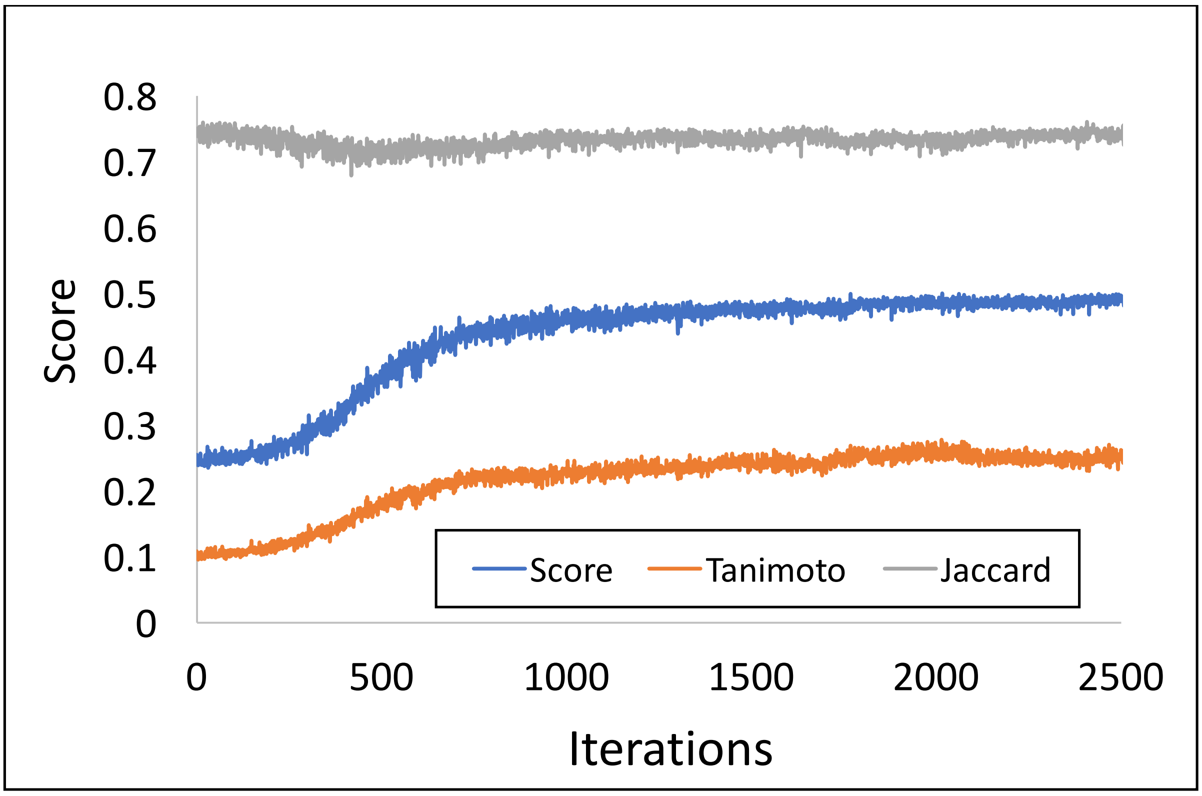

Figure 14. In the No CL case, all components are used for the training from the beginning. As a result, the Jaccard component becomes maximum for the starts. This maximization prevents exploration, and the training has a difficult time improving the Tanimoto component.

Beyond the time it takes the agent to reach a target score, it is also useful to analyze the overall quality of the compounds generated. The first factor to examine this is the amount of actual new compounds that are being generated. REINVENT can produce strings that do not correspond to any existing molecule. Then, a way to assess performance is to look at the compounds stored (SMILES that correspond to something that can exist) and compare it to the whole number of strings produced during this stage. The ratio of these two numbers is called yield. In *Table 2*, the average yield over five runs for CL and No-CL are presented with their respective standard deviation.

Table 2. Values of the Yield of the Tanimoto-Jaccard experiment with and without Curriculum learning.

Tanimoto + Jaccard	**Mean Yield**
Curriculum Learning	$39\% \pm 5\%$
No Curriculum Learning	$52\% \pm 6\%$

The yield given by CL is smaller than the one of the No CL. However, it is important to note that when the margin of error is taken into account, the gap between the yield result is not substantial.

A third tool to study the performance are the score distributions of generated compounds. Distributions of the values of the scoring function and of the individual components are presented in Figure 15, Figure 16, and Figure 17. These histograms show the fraction of the entire production of valid SMILES during the production phase within a certain range of score. To consider the stochastic nature of the production, the results used for this study are compiled over five runs. The histograms are generated for each run, and then they are averaged, and their standard deviation is obtained.

The distribution of the scoring function values indicates that the components generated in both setups are close. Most of the compounds have a score between 0,45 and 0,6. On the other side, there are differences in the distributions of the values given by the individual components. The No-Cl cases have distributions accumulate in narrow intervals. For Jaccard, most of the compounds generated are located in the interval between 0,6 and 0,8. At the same time, the distribution of Tanimoto scores congregates at the range of 0,1 and 0,3. The results reaffirm what was observed when studying the results in *Figure 14*. The No-CL approach is unable to generate good similarity results.

The results for the CL distribution are similar in the sense that the results are also very tilted towards low similarity and high dissimilarity, as is the case for No-CL. The significant difference is that both for dissimilarity and similarity, there are tails in the distribution not present in the No-CL case. Given that the overall distribution of the combined score is also around 0,5 for the CL, this implies that those small subpopulations correspond to the same generated compounds. This result is valuable as it indicates that CL can give a more well-balanced distribution than the No-CL.

As a concluding observation for this scenario, the experiments indicate that the application of CL can accelerate the training processes when orthogonal components are used. Additionally, it can output molecules more well-balanced and not excessively tilted towards a unique property.

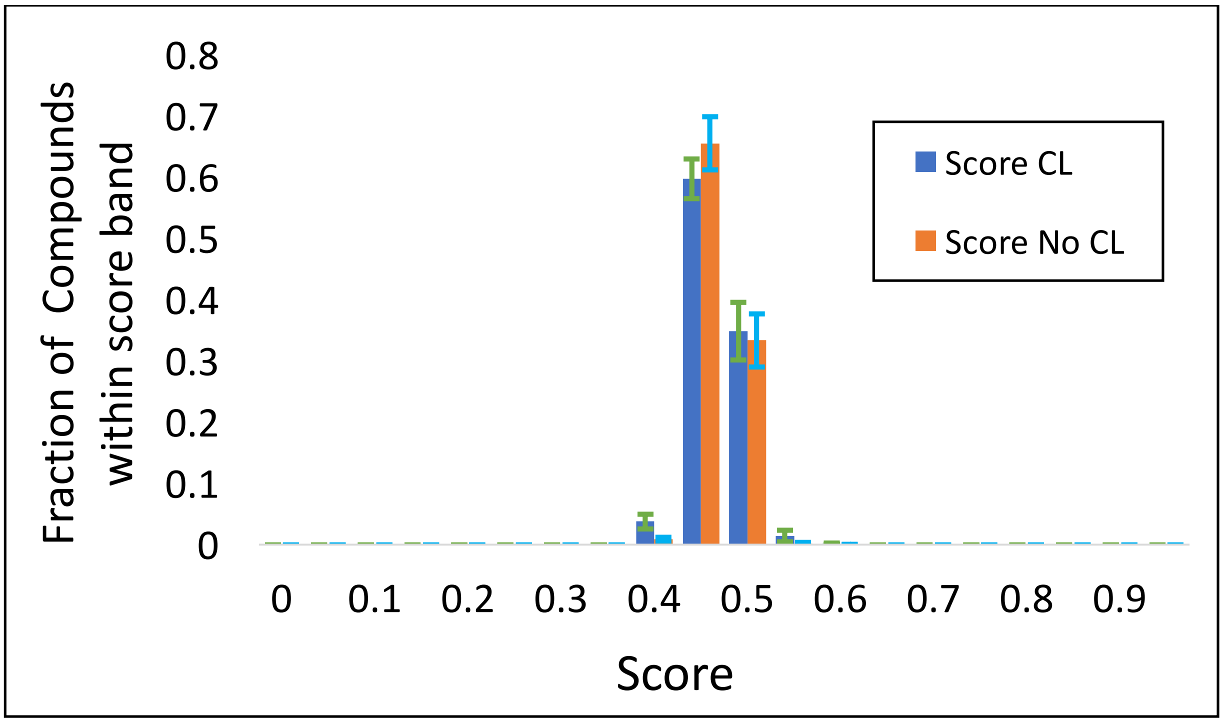

Figure 15. The concentration of the output score is around the range between 0,4 and 0,6. This plot is generated by performing five runs and computing the fraction of the whole population residing in all 0,05 intervals between 0 and 1. These fractions are averaged for the five runs, and the standard deviation is obtained. These repetitions are done to account for the noise characteristic of the stochasticity of the training.

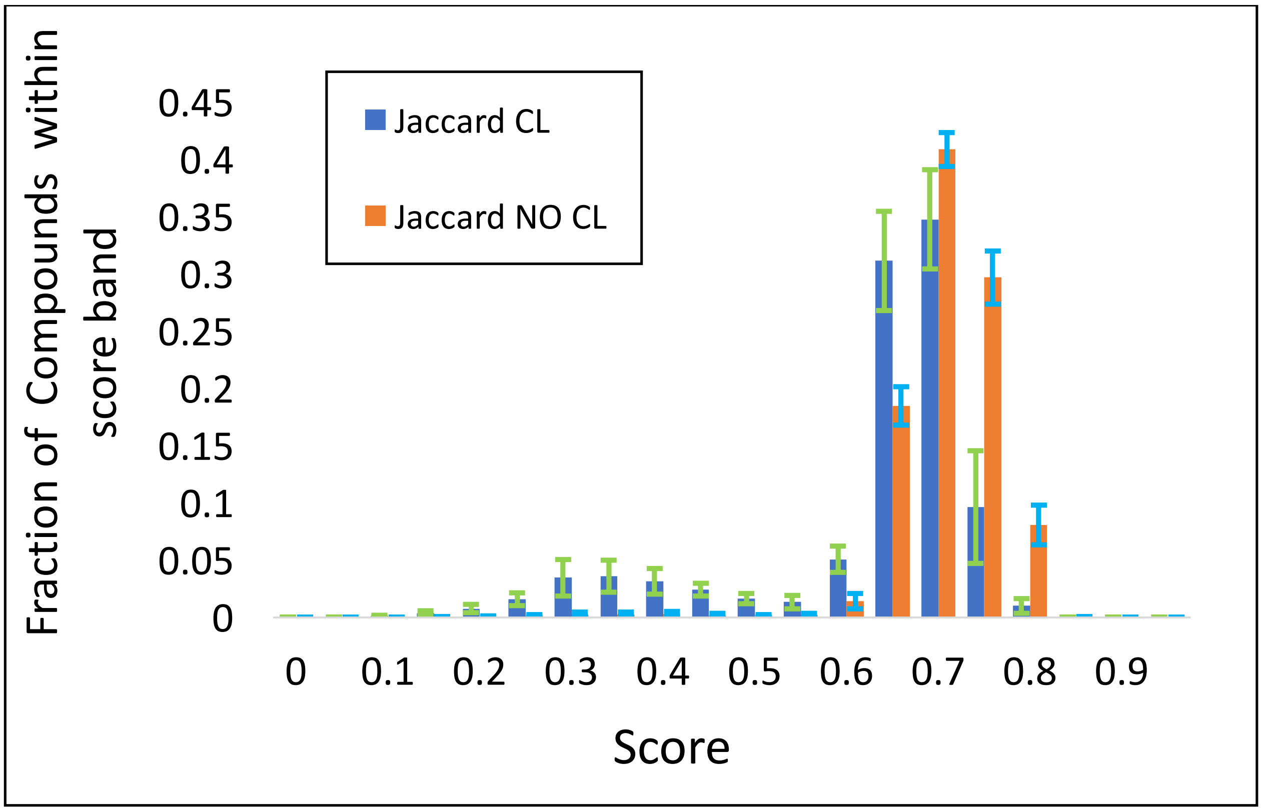

Figure 16. This plot is generated the same way as Figure 15, but the score observed is only the Jaccard component's values. The results indicate that the production in the No CL case is more heavily tilted towards high dissimilarity. Meanwhile, the output of the CL case still showcases a high concentration of high dissimilarity, but there is a fraction present at lower values.

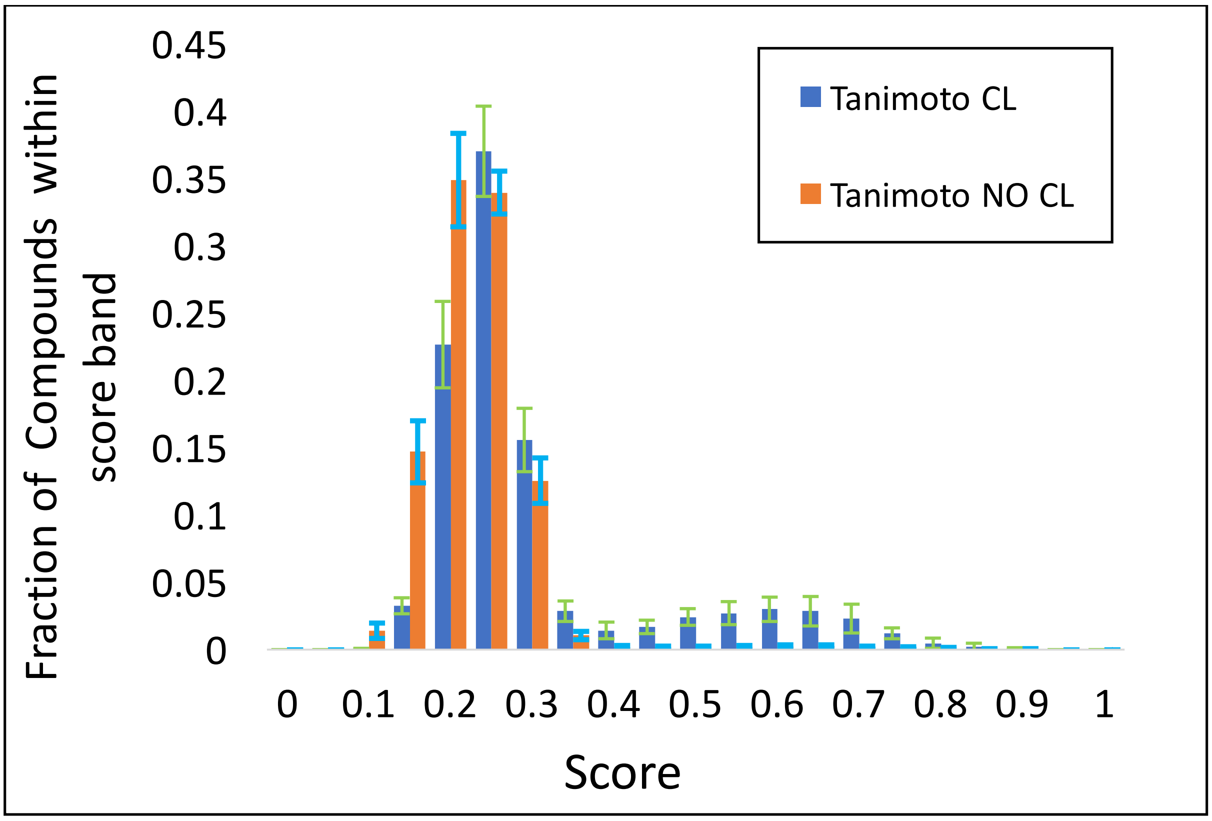

Figure 17. This plot is generated the same way as Figure 15. The case of No CL presents low values of similarity. At the same time, CL still produces compounds with low similarity, but there is a representative fraction with high similarity scores.

Selectivity and Activity

As the No-CL baseline, the agent is directly trained with both components at the same time. The iteration study is repeated twenty times for three setups: the CL approach, the No-CL approach, and the Selectivity alone. The results are presented in Figure 18.

The results indicate that there is an improvement regarding the number of iterations necessary for the merging phase. The difference is not as conclusive as the one observed for the Tanimoto and Jaccard setup. Still, the results suggest an improvement with CL. The box plot indicates that around 75% of all measurements with CL are lower than 75% of the measurements with No-CL. Additionally, training directly on Selectivity has a higher number of iteration. This observation indicates that the use of an inverted Curriculum would result in more iterations.

The yield is done to observe if the implementation CL affects the number of valid SMILES generated in the production phase. The yield is generated for five different runs, and the average and standard deviation are generated. The results indicate no significant changes in the yield. The mean values are very close and are within each other margin of error. Therefore, it can be concluded that there is no reason to assume any significant differences in the yield of the two setups.

Table 3. Values of the Yield of the Activity-Selectivity experiment with and without Curriculum learning.

Activity + Selectivity	**Mean Yield**
Curriculum Learning	$44\% \pm 11\%$
No Curriculum Learning	$47\% \pm 11\%$

The compounds generated during the production phase of five runs are stored. First, for every population generated, histograms of the distribution of values for the Activity and Selectivity are generated. Then, the histograms are averaged, and the standard deviation for each one is calculated. The results are presented in Figure 19 and Figure 20.

The results show a vast amount of variability within the distributions, and there is a significant overlap between the margin of error. Only the region beyond 0,95 shows a possible deviation in performance quality. Here, the value of the CL case is significantly higher than the No-CL case, and the margin of error indicates that it can be substantially higher. Apart from that observation, the distributions do not show significant differences.

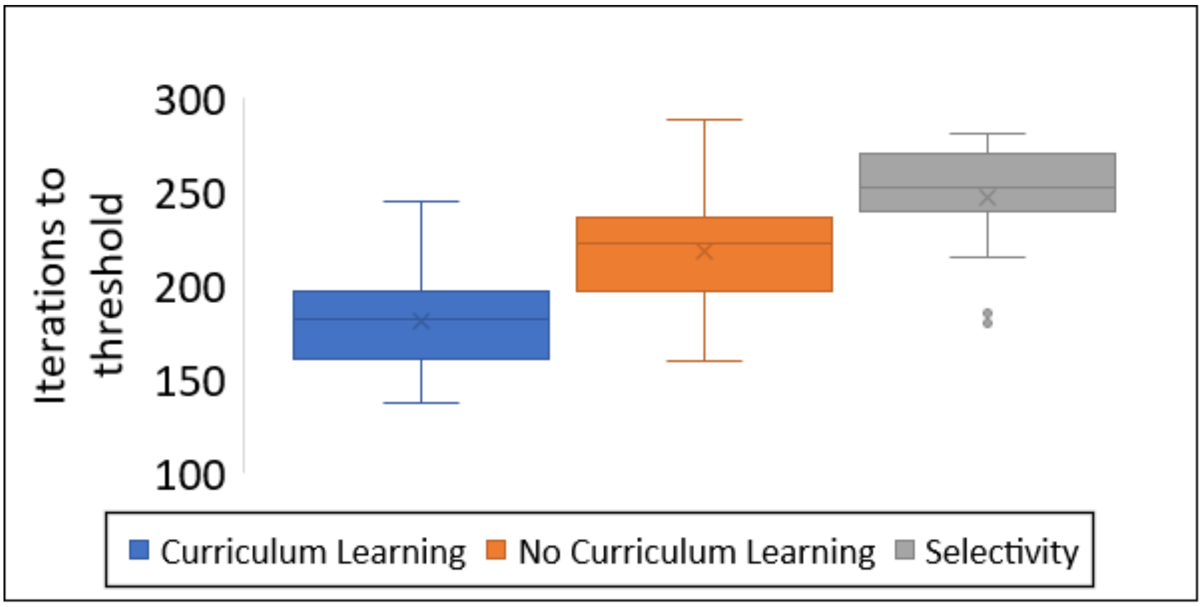

Figure 18. The performance measure in the time to reach a threshold during the merging phase is superior for the CL approach compared to No-CL. This plot is generated using twenty runs of REINVENT.

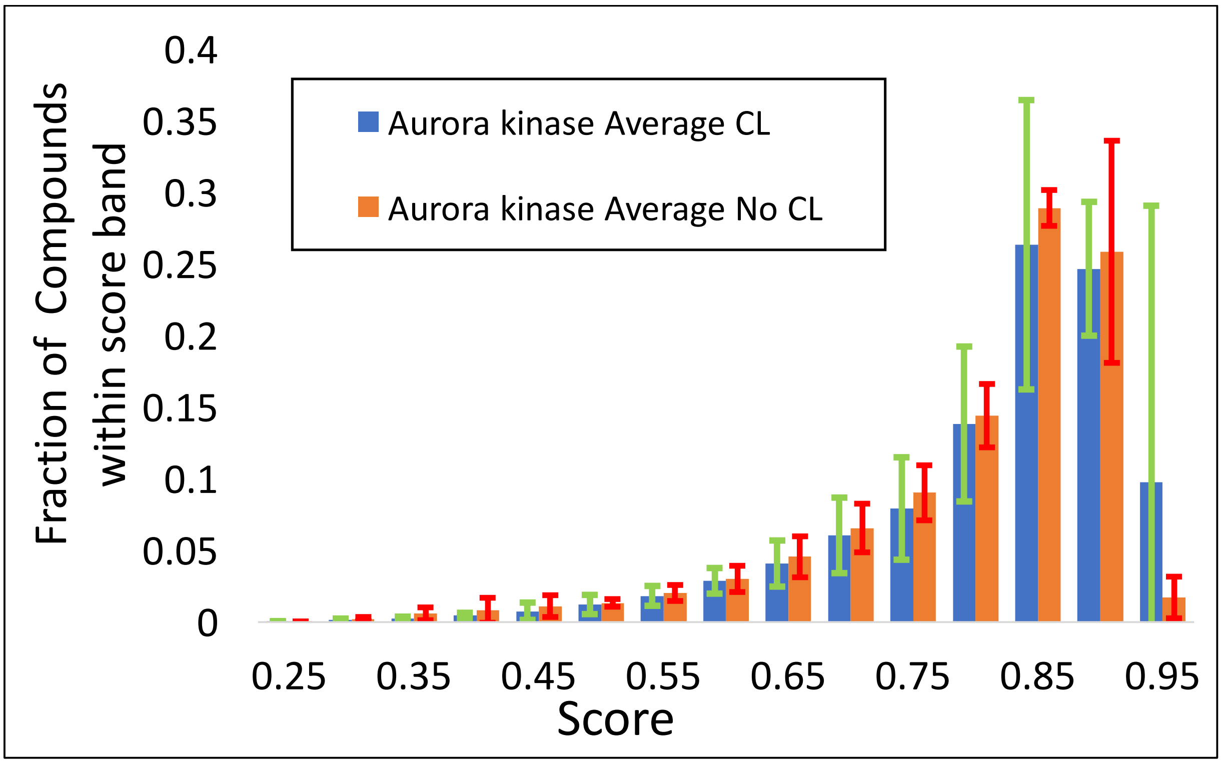

Figure 19. The plot indicates that the activity distribution of the generated compounds by the CL and No CL. The results show no significant differences between the distribution. The only difference is that the CL case tends to produce more compounds with activity bigger than 0,95, but this must be taken with skepticism due to the high variance.

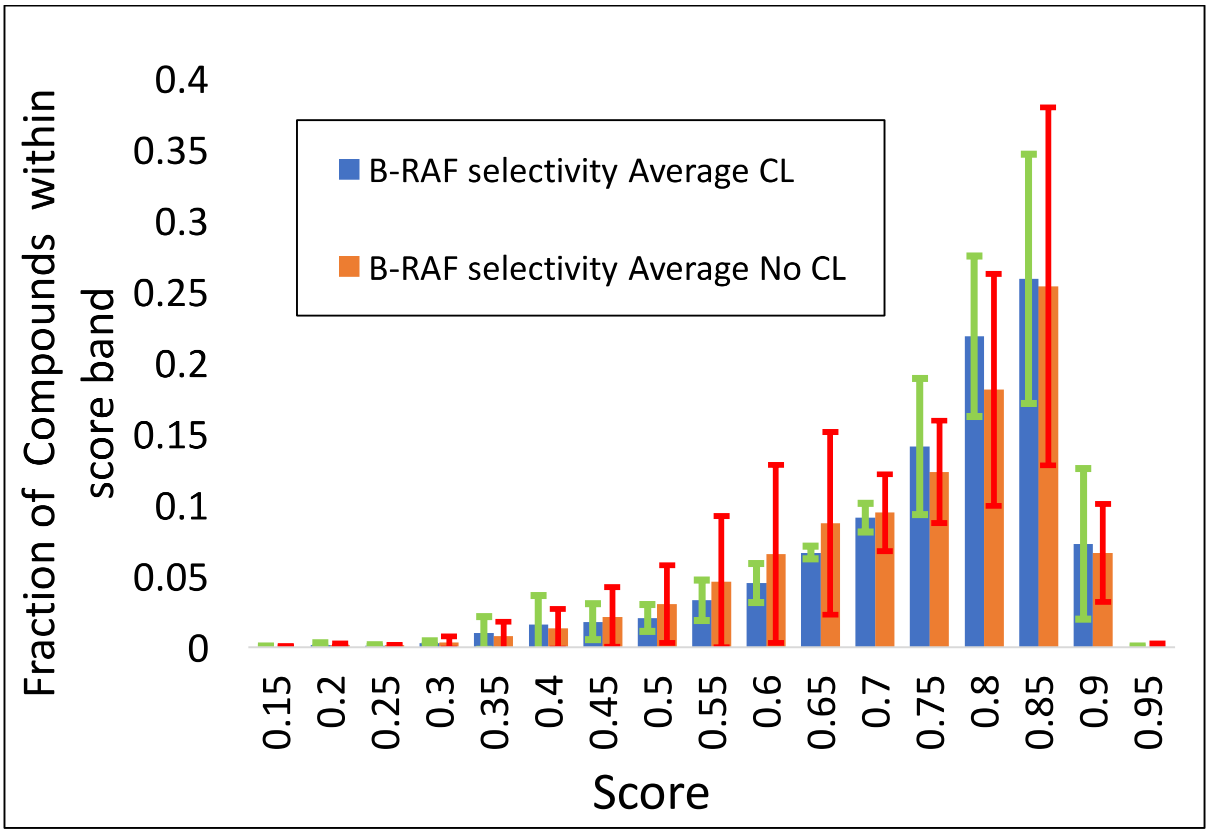

Figure 20. The selectivity distribution between the two set up shows no significant differences. However, there is considerable overlap between the margins of error, so it is inaccurate to assume any difference between the results.

Selectivity and Activity using Diversity Filter

The iteration distributions in Figure 21. The resulting distributions are closer to each other in comparison to the case when DF is not implemented. The mean of the CL approach is approximately ten iterations smaller than the No-CL. The result is too narrow to argue it is conclusive. Additionally, the number of iterations needed to stop the merging phase is more significant when the DF is used compared to the case when it is not (see *Figure 18*).

The analysis of the score distributions is done with the inclusion of the DF. The results are presented in *Figure 22* and *Figure 23*. An interesting point of contrast is how the distributions with CL and No-CL change when DF is included. In preceding section, the distribution of the population tends for higher Activity in the case with CL, but besides that, the distribution does not show significant differences. In the case with DF, the proportion of the entire population that has high Activity in the CL case increases in comparison to the No-CL case. Especially at 0.9, the activity is superior in the case of CL, and there is no overlap between the margin of error. There is still uncertainty as the margins of error are broad, but it is an improvement over the case with no DF, where the two scenarios are practically indistinguishable. In Selectivity, the fraction of the population in the range of values between 0,9 and 1 is superior for CL than for No-CL, and the overlap of margins of error is small. These results indicate that the population being generated achieves higher Activity and Selectivity.

As an observation, this extension of the Activity and Selectivity setup shows that Curriculum Learning can, in principle, work well with a Diversity Filter in the generation of an output with a better profile (higher scores). On the other side, the results suggest that the combination of Diversity Filter and Curriculum Learning slows down the merging phase.

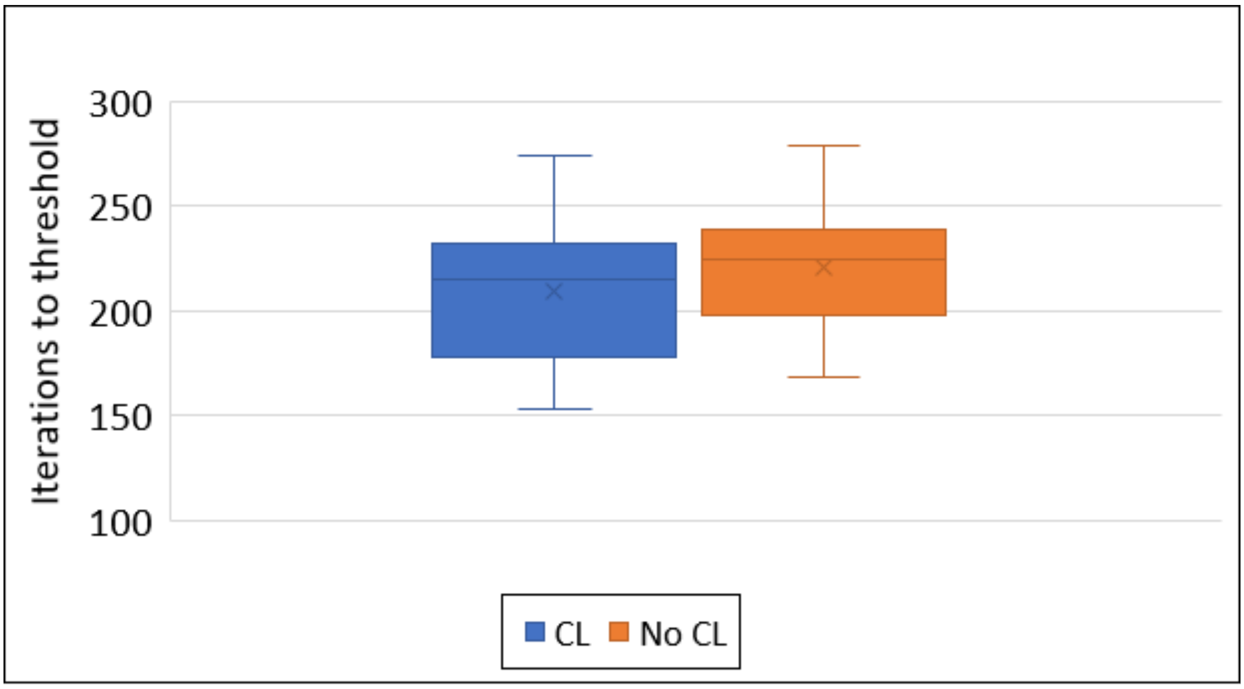

Figure 21. The inclusion of the diversity filter impacts the results by decreasing the difference in performance in what respects the number of iteration needed to reach the threshold. This plot is generated using twenty runs of REINVENT.

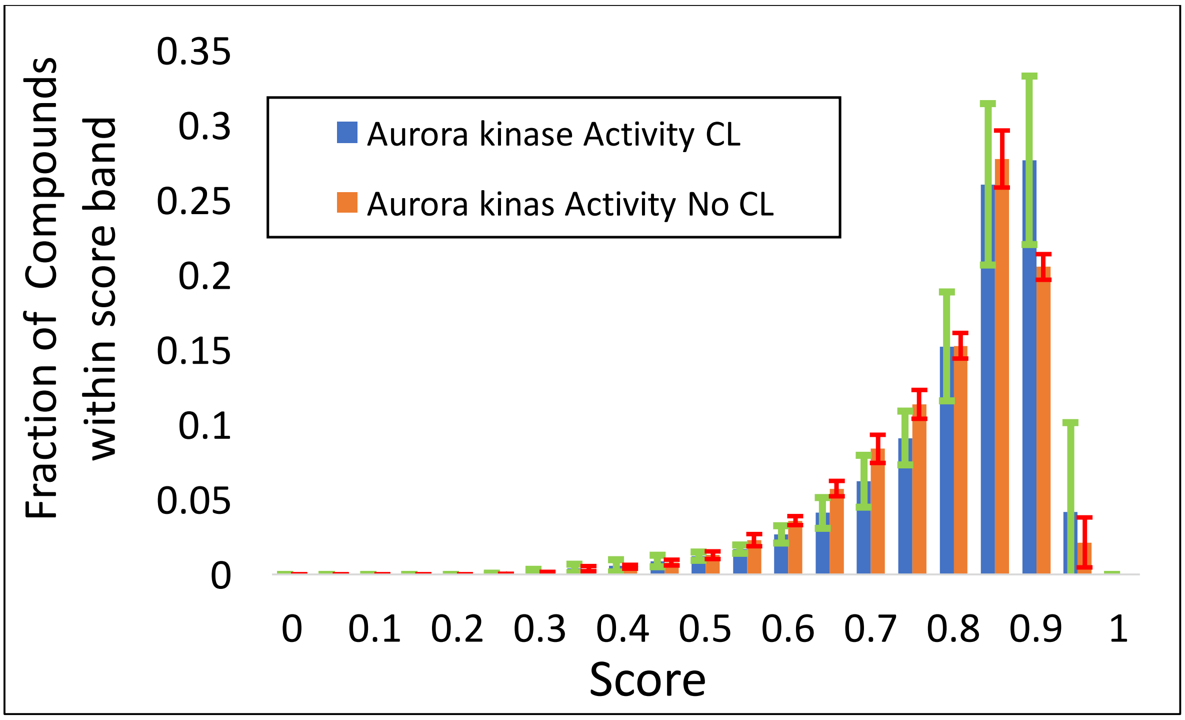

Figure 22. There is a more considerable fraction of the overall production with activity values between 0.9 and 1 in CL when compared to No-CL. It Is worth noting that the values of 0.9 do not overlap in their margins of errors.

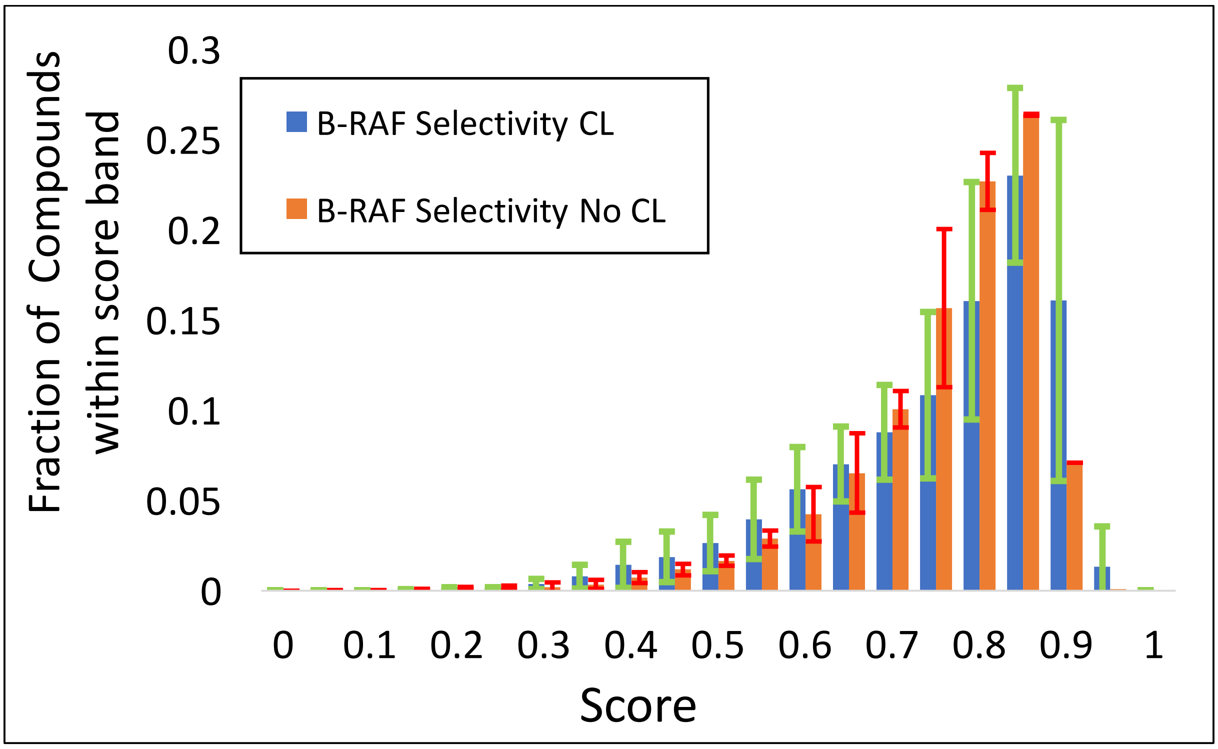

Figure 23. There is a more significant fraction of the overall production with Selectivity values between 0.8 and 1 in CL when compared to the case with No-CL.

Conclusion

For all the applications, two outcomes were observed: a reduction in the number of iterations necessary to train the agent and an improvement in the quality of the generated molecules. By improvement, it is meant an increase in the values assigned by the components to the compounds generated or more balanced distribution of values, with respect to a No-CL baseline.

Regarding the number of iterations, it must be noted that the degree of the impact is not consistent in all cases. It is present in all cases, but the size of the effect varies. In the case of Tanimoto-Jaccard, the result is evident (see *Figure 12*). Still, for its use in Activity-Selectivity with Diversity Filter, the reduction is not present (see *Figure 21*). Additionally, it has been seen on the Tanimoto-Jaccard setup (see *Figure 12*) and Matching Substructure setup (see *Figure 11*) that CL has an additional effect of reducing the variance of the number of iteration required to finish the merging phase.

In the following paragraph, an interpretation of how Curriculum Learning works in REINVENT is given, and how this effect might connect with the reduction in variance and iterations. The agent is pre-trained on a set of compounds. Therefore, the agent can sample compounds from a region of chemical space with the properties of the pre-trained set. During the Reinforcement

Learning loop of *REINVENT*, the agent's parameters are modified. Every modification can be seen as a displacement of the region of the chemical space from where the agent samples. A set of compounds that score high correspond to some particular part of the chemical space where compounds with the desired physicochemical properties are located. In principle, the agent's initial state can be a region far away in the chemical space from the section where the molecules with the desired properties are located. This situation makes it unlikely that any sample will give any significant variation in the score. This lack of "signal" makes exploration hard, as there is no clear path of how to modify the parameters. What the Curriculum does in this context is to train an agent with a scoring function that is "easy," which means that the solution space is ample and, therefore, will take few iterations to reach it. After the agent reaches that solution region, the Curriculum updates the scoring function making it "harder." This process translates into narrowing the solution space. The previous stage helps because it moves the agent to a part of the chemical space that is closer to a region of the next stage. Given the proximity, the agent can detect the trace of where to go. Then, it can find the direction faster, but it does not run the risk of going into long trajectories in the chemical space that do not lead to improvements. This idea explains why Curriculum Learning helps to reduce the agent's number of iterations and variance.

Another result worth pointing out is that the application of Curriculum Learning seems to work better in the problems connected to the structure of a compound. Both the results of using CL for generating compounds with a particular substructure and the Tanimoto-Jaccard setups give credence to it. It is theorized that this might be because building a sequence of progressively more challenging problems can quickly be done when the situation is building substructures that are gradually more complex. On the other hand, the results for the Activity-Selectivity cases are promising, but further work might be required as the high amount of variance in the results make it hard to make concrete conclusions.

Two relevant points were left unaddressed during this work's progression: the automatic construction of the stages of the Curriculum and the ranking methods. When the scoring function is made up of a single component, the users are responsible for creating the Curriculum on their own. Given the effect that the extension Curriculum can have (see *Figure 11*), it would be insightful to investigate possible paths for automatic Curriculum generation as a way to address this. Regarding the Ranking, the methods for automatically ranking components were constructed, so the users of REINVENT can take advantage of them. Still, none of the scenarios studied during this work used them. The reason for this is that there was already an understanding of the optimum ranking for the cases that were studied. In any future work that follows, it is considered relevant to take a deeper look into how efficient are the implemented methods for assessing the difficulty of a task. Also, the methods constructed define the "hardness" of a component by:

1. Generating copies of an initial agent.
2. Training each copy on a different component of the scoring function.
3. Comparing how efficient the training on each component is in terms of the number of iteration required or the score attained.

This method might have a blind spot, as it assumes that the complexity of learning a component is independent of the preexisting training of the agent. Some of the tasks might become harder depending on the training that the agent has already received. This aspect should be further studied.

Another remark for further improvement is that the applications were limited. It would be necessary to consider a different set of compounds to see if the results here are general.

The conclusion of this work is that, in principle, Curriculum Learning in REINVENT works. It is necessary to improve further and refine the job done, but the utility of its use is demonstrated here.

Bibliography

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. E., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. In *Heliyon* (Vol. 4, Issue 11, p. e00938). Elsevier Ltd. https://doi.org/10.1016/j.heliyon.2018.e00938

Arús-Pous, J., Johansson, S. V., Prykhodko, O., Bjerrum, E. J., Tyrchan, C., Reymond, J. L., Chen, H., & Engkvist, O. (2019). Randomized SMILES strings improve the quality of molecular generative models. *Journal of Cheminformatics*, *11*(1), 1–13. https://doi.org/10.1186/s13321-019-0393-0

Awale, M., Sirockin, F., Stiefl, N., & Reymond, J. L. (2019). Drug Analogs from Fragment-Based Long Short-Term Memory Generative Neural Networks. *Journal of Chemical Information and Modeling*, *59*(4), 1347–1356. https://doi.org/10.1021/acs.jcim.8b00902

Bengio, Y., Jérôme Louradour, U., Collobert, R., & Weston, J. (2009). Curriculum Learning. *Proceedings of the 26th International Conference On Machine Learning, ICML 2009*, 41–48.

Blaschke, T., Arús-Pous, J., Chen, H., Margreitter, C., Tyrchan, C., Engkvist, O., Papadopoulos, K., & Patronov, A. (2020). REINVENT 2.0 – An AI tool for de novo drug design. In *ChemRxiv*. ChemRxiv. https://doi.org/10.26434/chemrxiv.12058026.v3

Eck, D., workshop, J. S.-P. of the 12th I., & 2002, undefined. (n.d.). Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. *Ieeexplore.Ieee.Org*. Retrieved March 24, 2021, from https://ieeexplore.ieee.org/abstract/document/1030094/

Gaulton, A., Hersey, A., Nowotka, M. L., Patricia Bento, A., Chambers, J., Mendez, D., Mutowo, P., Atkinson, F., Bellis, L. J., Cibrian-Uhalte, E., Davies, M., Dedman, N., Karlsson, A., Magarinos, M. P., Overington, J. P., Papadatos, G., Smit, I., & Leach, A. R. (2017). The ChEMBL database in 2017. *Nucleic Acids Research*, *45*(D1), D945–D954. https://doi.org/10.1093/nar/gkw1074

Graves, A., Eck, D., Beringer, N., & Schmidhuber, J. (2004). Biologically plausible speech recognition with LSTM neural nets. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3141*, 127–136. https://doi.org/10.1007/978-3-540-27835-1_10

Huang, Y., Wang, Y., Tai, Y., Liu, X., Shen, P., Li, S., Li, J., & Huang, F. (2020). CurricularFace: Adaptive Curriculum Learning Loss for Deep Face Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 5900–5909. http://arxiv.org/abs/2004.00288

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., Wu, Y., & Brain, G. (n.d.). Exploring the Limits of Language Modeling. In *arxiv.org*. Retrieved March 24, 2021, from https://arxiv.org/abs/1602.02410

Kalchbrenner, N., Com, K., & Deepmind, G. (2016). *Pixel Recurrent Neural Networks Koray Kavukcuoglu*. PMLR. http://proceedings.mlr.press/v48/oord16.html

Mehlig, B. (2021). *Machine learning with neural networks An introduction for scientists and engineers*.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., & Stone, P. (2020). Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. In *Journal of Machine Learning Research* (Vol. 21).

Narvekar, S., & Stone, P. (2018). Learning Curriculum Policies for Reinforcement Learning. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, *1*, 25–33. http://arxiv.org/abs/1812.00285

Olivecrona, M., Blaschke, T., Engkvist, O., & Chen, H. (2017). Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, *9*(1), 48. https://doi.org/10.1186/s13321-017-0235-x

Reymond, J.-L., Ruddigkeit, L., Blum, L., & van Deursen, R. (2012). The enumeration of chemical space. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, *2*(5), 717–733. https://doi.org/10.1002/wcms.1104

Segler, M. H. S., Kogej, T., Tyrchan, C., & Waller, M. P. (2018). Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Central Science*, *4*(1), 120–131. https://doi.org/10.1021/acscentsci.7b00512

Srivastava, N., Mansimov, E., & Salakhutdinov, R. (n.d.). Unsupervised Learning of Video Representations using LSTMs. In *proceedings.mlr.press*. Retrieved March 24, 2021, from http://proceedings.mlr.press/v37/srivastava15.html

Sutton, R. S., & Barto, A. G. (n.d.). *Reinforcement Learning: An Introduction Second edition*.

Weininger, D. (1988). 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. https://doi.org/10.1021/ci00057a005

Zeugmann, T., Poupart, P., Kennedy, J., Jin, X., Han, J., Saitta, L., Sebag, M., Peters, J., Bagnell, J. A., Daelemans, W., Webb, G. I., Ting, K. M., Ting, K. M., Webb, G. I., Shirabad, J. S., Fürnkranz, J., Hüllermeier, E., Matwin, S., Sakakibara, Y., … Fürnkranz, J. (2011). Policy Gradient Methods. In *Encyclopedia of Machine Learning* (pp. 774–776). Springer US. https://doi.org/10.1007/978-0-387-30164-8_640