A Comparison of Generative Models for Sequence Design

Andreea Gane*, David Belanger, David Dohan, Christof Angermueller, Ramya Deshpande
Suhani Vora, Olivier Chapelle, Babak Alipanahi, Kevin Murphy, Lucy Colwell

Google Research
*Correspondence to gandreea@google.com and lcolwell@google.com

Abstract

Recent work has explored variants of the cross entropy method for black-box optimization of discrete structures such as DNA sequences. Using multiple rounds of calls to the black-box function, a generative sequence model is trained to place high probability on sequences that achieve high function values. These works employ sophisticated deep generative models, which, however, have high sample complexity and require extensive tuning. On the other hand, simple generative models, such as hidden Markov models, have achieved widespread success in computational biology applications. In response, we evaluate the performance of simple generative models when used within the cross entropy method. We find that simple generative models are competitive with more sophisticated models on two synthetic optimization tasks inspired by biological sequence design.

1 Introduction

Discrete black-box optimization problems of the form $\arg\max_{x \in \mathcal{X}} f(x)$, where $\mathcal{X}$ is a discrete high-dimensional set and f is an unknown scoring function, arise in many practical applications. Notable recent examples include optimization tasks in computer systems and infrastructure [23, 12, 22], chemistry [8], and life sciences [2, 1].

The focus of this paper is on biological sequence design, where the goal is to identify DNA or protein sequences x with desired properties $f(x)$. Here, $f(x)$ indicates gene expression [17], the fluorescence of a protein [18], or the binding affinity of a transcription factor [3]. Although methods to more efficiently optimize $f(x)$ can have a huge impact in medicine, agriculture, and manufacturing, the problem is far from being solved. The challenges arise from the combinatorial search space, the inability to compute derivatives of the scoring function to guide the search, and potentially restricted access to the oracle. For instance, when performing optimization over DNA sequences to enforce properties such as high binding affinity to a specific target [10], obtaining oracle outputs requires laboratory experiments. As a result, the optimization process must limit the number of queries, where each query scores a batch of sequences. Existing approaches for biological sequence design include directed evolution and evolutionary algorithms, which guide the search using prior knowledge about relevant mutations [2, 1]. Alternatively, optimization methods based on generative models guide the search through a parametrized distribution over data points: the optimization problem becomes $\arg\max_\theta E_{p(x;\theta)}[f(x)]$ [4]. This formulation enables using state-of-the-art reinforcement learning algorithms [23, 12], as well as estimation of distribution algorithms, such as the Cross Entropy Method (CEM) [15].

Here we focus on CEM, which has been recently successfully applied to biological sequence design using different generative models, including *variational autoencoders* [5, 6], *autoregressive generative models* [13, 19], or *generative adversarial networks* [9]. Nevertheless, the influence of the

generative model on the optimization performance of CEM is not well understood. In particular, it is unclear how simpler models such as *fully-factorized models* or *hidden Markov models*, traditionally used in bioinformatics applications [7], perform relative to complex deep generative models.

We compare generative models of varying complexity for designing DNA and protein sequences using CEM. On two synthetic optimization tasks, we find that the simple methods perform competitively with deep generative models. Surprisingly, we also find that simple evolutionary algorithms, which are generally considered to be sample-inefficient optimizers [16], can outperform CEM, irrespective of the choice of generative model, when performing many rounds of optimization.

2 Cross Entropy Method

Our goal is to solve $\hat{x} = \arg\max_{x \in \mathcal{X}} f(x)$, where $\mathcal{X} = V^L$, for a discrete vocabulary V, and an integer length L. The Cross Entropy Method produces a sequence of generative models $p(x; \theta_t)$, $t = 1, 2, \ldots$, that generate data points with increasingly high values of $f(x)$. Specifically, at time step t, it solves the following optimization problem [5]: $\theta_t = \arg\max_{\theta'_t} \sum_{x \in A_{t-1}} w(x) \log p(x; \theta_{t'})$ where $A_{t-1} = \{x | x \sim p(x; \theta_{t-1})\}$ is a set of samples from the previous distribution, and $w(x)$ is a sample weighting function, monotonic in $f(x)$. A typical choice is $w(x) = 1[x \in S_t]$, where $S_t \subseteq A_{t-1}$ is a set of elite samples x selected based on their score $f(x)$, for instance, by using the top-k samples with the highest score [13]. In this paper, we follow a similar approach, used by Brookes et al. [5] (DbAs) in using a quantile cutoff for selecting the elite set and the corresponding sample weights, and we investigate alternative generative model architectures for modeling $p(x; \theta)$.

2.1 Generative Models for the Cross Entropy Method

A generative model can be used in the context of CEM as long as it possesses an efficient (weighted) maximum likelihood estimation (MLE) procedure $\arg\max_\theta \sum_{k=1}^{|\mathcal{D}|} w^{(k)} \log p(x^{(k)}; \theta)$, given M sequence-weight pairs $\mathcal{D} = \{(x^{(k)}, w^{(k)}) | k = 1 \ldots M\}$. We consider the following model classes:

Fully-factorized (FF). The simplest approach one can design for modeling sequences is a fully-factorized model, where each sequence position is modeled independently. Fully-factorized models are appealing due to their simplicity and ability to scale to high-dimensional data. The modeling distribution is given by $p(x; \theta) = \prod_{i=1}^{L} p(x_i; \theta_i)$, where $p(x_i; \theta_i)$ is parameterized by the probabilities per position and token: θ_{i,x_i}. For each position, we use a uniform Dirichlet prior over the parameters and perform maximum a-posteriori estimation.

K-th order Markov model (KMM). More expressive generative models capture the conditional dependency between adjacent positions. Markov models assume that the assignment per sequence position depends only on the previous k positions. In particular, the modeling distribution is given by $p(x; \theta) = p(x_1; \theta_1) \prod_{i=2}^{L} p(x_i | x_{i-k:i-1}; \theta_i)$. Similar to the fully-factorized case, the start and each conditional distribution are not tied along the length of the sequence, and we employ uniform Dirichlet priors with tuned concentration parameters. Note that the FF model is a KMM with $k = 0$.

Hidden Markov model (HMM). One can extend a KMM with a sequence of discrete latent variables such that observations are conditionally independent given the latent variables. The modeling distribution is given by $p(x; \theta) = \sum_{h \in \mathcal{H}} p(h_1; \theta_S) \prod_{i=2}^{L} p(h_i | h_{i-1}; \theta_T) \prod_{i=1}^{L} p(x_i | h_i; \theta_E)$, where θ_S, θ_T and θ_E are the start, transition and emission probabilities, respectively. We are not learning sequence position specific parameters, but the presence of hidden variables enables the model to learn specialized hidden states that incorporate information about the location within the sequence and can model higher sequence diversity. Training is performed with expectation-maximization, which is considerably more complex than training FF and KMM models, which rely on simple counting.

Recurrent neural network (RNN). Recurrent neural networks are the standard neural network approaches for sequential data, modeling the next token probability as a complex (nonlinear) function of the previous token and a continuous hidden state vector. Similar to the hidden Markov models, the sequence of input tokens is encoded as a sequence of states. While recurrent neural networks are generally more powerful than hidden Markov models, which make strong assumptions about the state-transition dynamics, they are more challenging to train, especially on long sequences.

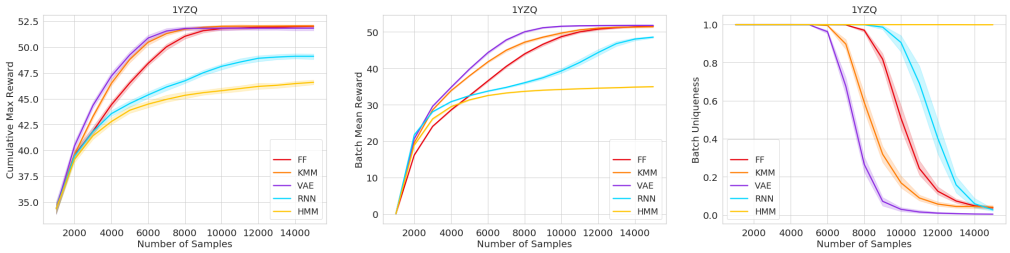

Figure 1: Optimization results for one PDB ising problem (target protein structure: 1YZQ). Shown are the cumulative maximum reward (left), batch mean reward (middle), and batch uniqueness (right), as a function of the number of proposed samples. We find that DbAs-KMM performs competitively with DbAs-VAE.

Variational autoencoder (VAE). Generating sequences in one-shot, via a fully-connected neural network decoder, enables encoding high-order dependencies. The distribution modeled by a variational autoencoder is defined by passing samples from a generic distribution ("noise") through a neural network decoder, and the resulting latent variable model can be trained via stochastic gradient descent by using variational inference and the reparametrization trick [11]. The work by Brookes et al. [5], which is our focus in this paper, used a variational autoencoder to model the elite set distribution, and we compare this approach with the simpler models outlined above.

3 Experiments

The goal of our experiments is to understand how the performance of DbAs is influenced by the choice of the generative model, and how DbAs performs relative to evolution-based methods. We use the following baselines: (1) *Random solver*, where each proposed batch contains random sequences from the search space, (2) *Regularized evolution*, an improved evolution algorithm [14], and (3) *Single mutant walker*, a simple solver inspired by directed evolution [2], described in the appendix.

We evaluate methods via the following metrics, evaluated per batch as optimization progresses: (1) *Cumulative max reward*: the maximum reward found in the current or any previous batch. This is the primary measure of performance, and it indicates a solver's ability to discover good sequences. (2) *Batch mean reward*: the average reward in the current batch. We use the resulting curve to indicate whether the model succeeds in producing samples with increasingly high rewards. (3) *Batch uniqueness*: the percentage of unique samples within the batch, used to indicate whether the model converges towards a (potentially sub-optimal) subset of configurations.

3.1 Datasets

We perform the evaluation on two synthetic optimization tasks designed to reflect the characteristics of a real wet-lab experiment:

PDB Ising Model. Our first task is based on the structures of naturally occurring proteins that have been experimentally determined with coordinates deposited in the Protein Data Bank (PDB). Each protein structure gives rise to an Ising model in which the sequence positions are nodes, and two nodes are connected if the 3D distance between their α-carbon atoms is within a given threshold (here we use 8.0Å). For simplicity, we assume that each sequence position can take on one of 2 values (e.g. hydrophobic and hydrophilic) instead of the 20 possible amino acids [20]. The edge weights are chosen to produce a frustrated system, where neighboring nodes are encouraged to take on different values, and the goal is to discover binary sequences that minimize the energy of the resulting model. Our synthetic task is a simplified version of protein design: given a target structure, we first describe it as an Ising model and then seek to find the configuration of amino acids that has lowest energy under this model [21]. We allow the solver to perform a limited number of queries (set to 14 in our experiments) in batches of fixed size of 1,000 sequences. The initial round starts with 1,000 randomly chosen sequences that are fixed across all runs.

5' UTR. A common task in biological engineering is to identify regulatory elements which increase the expression level of a particular gene. A 5' untranslated region (5' UTR) is one of such regulatory elements, which is located just before the coding region of a gene. Sample et al. [17] recently showed that the expression level of a gene can be predicted from the corresponding 5' UTR sequence using a convolutional neural network. We used this network as a black-box oracle and sought to find 5'

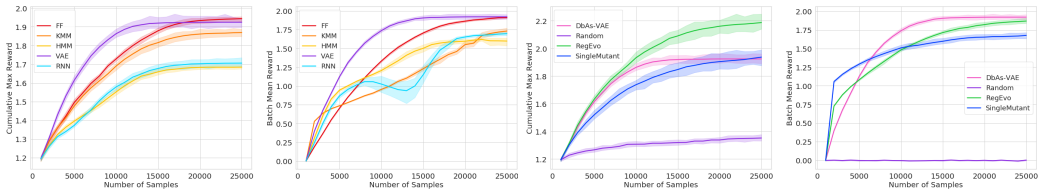

Figure 2: Optimization results on the 5' UTR problem. We compare DbAs-VAE with alternative generative models in the first two figures, and with baselines in the last two figures (in each case we show the cumulative max reward and batch mean reward). We find that simple evolutionary optimization (RegEvo) outperforms DbAs after enough optimization rounds. The fully-factorized model performs competitively with the VAE - note that it obtains a slightly higher maximum reward and a similar batch mean reward at the end of optimization.

UTR sequences x that maximize the predicted gene expression level $f(x)$. The proposed sequences are fixed-length, 50 nucleotides long, resulting in a search space of 4^{50}. We performed 24 rounds of queries to the oracle, with a batch size of 1000 sequences. Akin to the PDB Ising Model Problem, the initial round starts with 1,000 randomly chosen sequences.

3.2 Optimization Results

PDB Ising Model. Figure 1 shows results for a chosen protein structure and results for two additional proteins are presented in the appendix. The hyper-parameters were selected on a 4^{th} protein. We note that the KMM is close in performance to the VAE, both in terms of the maximum reward attained, and the increase in maximum reward as a function of the number of sequences tested. The VAE and KMM consistently performed better than the recurrent neural network in the CEM formulation, likely due to the expected difficulties of fitting a recurrent neural network to large sequences (our sequences have length 50), or to the small number of samples that the models are being trained on. The HMM is likely to underperform due to sharing the transition matrix parameters across steps (for the KMM, the use of untied parameters improved performance). Finally, the batch mean reward is monotonically increasing, suggesting that the models successfully capture samples with increasingly high scores.

The VAE attains a high batch mean reward earlier than the KMM, though the batch uniqueness results suggest that the VAE also reduces batch diversity more quickly than the KMM. In the context of real-world problems where the identification of diverse good solutions is important, the ability of a solver to retain additional sequences in the final batch while almost matching the batch mean reward of the VAE model makes it an attractive alternative.

5' UTR. As we have a single black-box oracle for the 5' UTR problem, to be used both for tuning parameters and evaluation, we report two sets of results. Figure 2 shows results where the hyper-parameters are extracted from the PDB Ising model problem, while the appendix shows results where the hyper-parameters are tuned on the 5' UTR problem. The FF and VAE perform comparably after enough rounds of optimization, with DbAs-VAE being more sample efficient. This difference is reduced when tuning the hyper-parameters of the simple models. In particular, among the generative models, the KMM improves significantly as a result of tuning: when using a smaller conditioning block size, it approaches the fully-factorized model's performance (shown in the appendix).

4 Discussion

Using the cross entropy method for biological sequence design requires choosing between simple models, which are easy to learn but capture limited interactions between the output variables, and expressive models with costly and potentially intractable learning and inference procedures [1]. Neural network-based approaches enable learning expressive models without explicitly requiring to characterize the interactions and without tractability considerations, but can be sample inefficient and require careful hyper-parameter tuning. We propose investigating whether richer models improve optimization results. On two synthetic biological sequence design task we show that simple generative models are competitive with neural network-based models. While further work is required to fully characterize when and why the simple approaches might be sufficient when compared to neural network approaches, we hope that our analysis motivates further comparisons in this respect.

References

[1] Rubén Armañanzas, Iñaki Inza, Roberto Santana, Yvan Saeys, Jose Luis Flores, Jose Antonio Lozano, Yves Van de Peer, Rosa Blanco, Víctor Robles, Concha Bielza, et al. A review of estimation of distribution algorithms in bioinformatics. *BioData mining*, 1(1):6, 2008.

[2] Frances H Arnold. Design by directed evolution. *Accounts of chemical research*, 31(3):125–131, 1998.

[3] Luis A Barrera, Anastasia Vedenko, Jesse V Kurland, Julia M Rogers, Stephen S Gisselbrecht, Elizabeth J Rossin, Jaie Woodard, Luca Mariani, Kian Hong Kock, Sachi Inukai, et al. Survey of variation in human transcription factors reveals prevalent dna binding changes. *Science*, 351(6280):1450–1454, 2016.

[4] David H Brookes, Akosua Busia, Clara Fannjiang, Kevin Murphy, and Jennifer Listgarten. A view of estimation of distribution algorithms through the lens of expectation-maximization. *arXiv preprint arXiv:1905.10474*, 2019.

[5] David H Brookes and Jennifer Listgarten. Design by adaptive sampling. *arXiv preprint arXiv:1810.03714*, 2018.

[6] David H Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. *arXiv preprint arXiv:1901.10060*, 2019.

[7] Sean R Eddy. What is a hidden markov model? *Nature biotechnology*, 22(10):1315, 2004.

[8] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[9] Anvita Gupta and James Zou. Feedback gan for dna optimizes protein functions. *Nature Machine Intelligence*, 1(2):105, 2019.

[10] Tatsunori B Hashimoto, Steve Yadlowsky, and John C Duchi. Derivative free optimization via repeated classification. *arXiv preprint arXiv:1804.03761*, 2018.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2430–2439. JMLR. org, 2017.

[13] Daniel Neil, Marwin Segler, Laura Guasch, Mohamed Ahmed, Dean Plumbley, Matthew Sellwood, and Nathan Brown. Exploring deep recurrent models with reinforcement learning for molecule design. 2018.

[14] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[15] Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

[16] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[17] Paul Sample, Ban Wang, David W. Reid, Vlad Presnyak, Iain J. McFadyen, David R. Morris, and Georg Seelig. Human 5 utr design and variant effect prediction from a massively parallel translation assay. *Nature Biotechnology*, 2019.

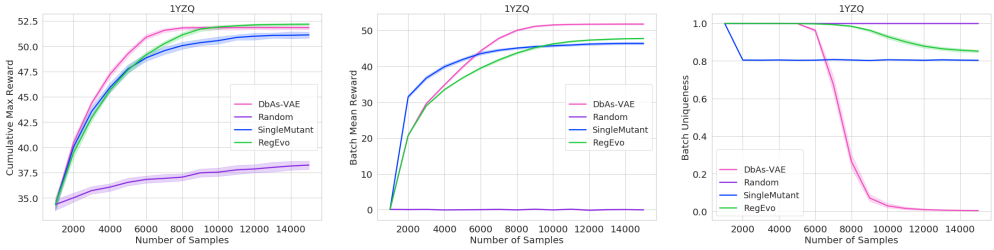

Figure 3: Optimization results for a PDB ising task (protein target structure 1YZQ). Shown are the cumulative maximum reward (left), batch mean reward (middle) and batch uniqueness (right) as a function of the number of proposed samples. We compare DbAs-VAE with simple optimization baselines: random solver, single mutant solver and regularized evolution. DbAs-VAE outperforms the baselines in terms of maximum reward and reaches a higher batch mean reward given sufficient optimization rounds.

[18] Karen S Sarkisyan, Dmitry A Bolotin, Margarita V Meer, Dinara R Usmanova, Alexander S Mishin, George V Sharonov, Dmitry N Ivankov, Nina G Bozhanova, Mikhail S Baranov, Onuralp Soylemez, et al. Local fitness landscape of the green fluorescent protein. *Nature*, 533(7603):397, 2016.

[19] Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1):120–131, 2017.

[20] Eugene I Shakhnovich and Alexander M Gutin. Engineering of stable and fast-folding sequences of model proteins. *Proceedings of the National Academy of Sciences*, 90(15):7195–7199, 1993.

[21] Eugene I Shakhnovich and AM Gutin. A new approach to the design of stable proteins. *Protein Engineering, Design and Selection*, 6(8):793–800, 1993.

[22] Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. Learning to design circuits. *arXiv preprint arXiv:1812.02734*, 2018.

[23] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

5 Appendix

5.1 Additional Results

In Figure 3, we show additional optimization results for the same PDB ising model task discussed in the main text (protein target structure 1YZQ). We compare DbAs-VAE with simple optimization baselines: random solver, single mutant solver and regularized evolution. DbAs-VAE outperforms the baselines in terms of maximum reward and reaches a higher batch mean reward given sufficient optimization rounds. That said, note that the VAE concentrates more quickly, which may result in reduced exploration in the later stages of optimization.

In Figure 4, we show results for the PDB Ising model task with three target protein structures (the first protein is also shown in the main text). Similar to the results presented in the main paper, DbAs-KMM achieves performance competitive with DbAs-VAE. In future work, we aim to understand the properties of the PDB Ising model task that enable the simple models to perform well.

Results on the 5' UTR problem with tuned hyper-parameters are presented in Figures 5 and 6. Due to the lack of a train-test split, we compare these results with the ones in the main text, where we used the hyper-parameters tuned on the Ising Model problem. With the tuned hyper-parameters, the KMM meets the performance of the fully-factorized model (see also Figure 7 for a similar plot using standard deviation based confidence intervals). We used the same parameters for the VAE and the HMM as we saw no increase in performance when changing the parameters. The new hyper-parameters for the recurrent neural network improved maximum reward performance, but decreased performance with respect to the batch mean reward. We note that achieving high cumulative maximum reward in early rounds can trade off to achieving a higher maximum reward after many steps with small hyper-parameter changes.

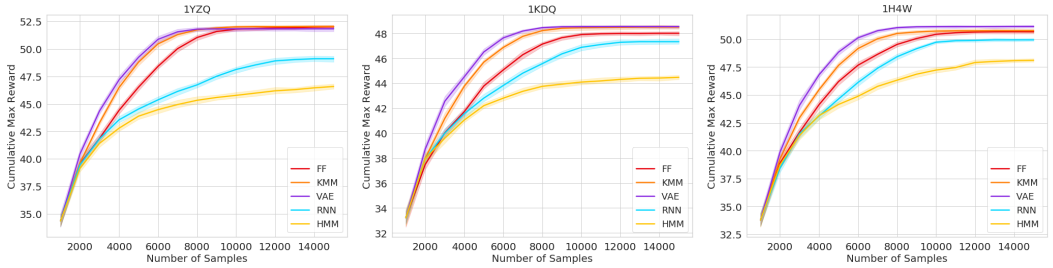

Figure 4: To test the generality of our findings across target structures, we consider alternative target protein structures from the PDB. DbAs-KMM is competitive with DbAs-VAE on each target.

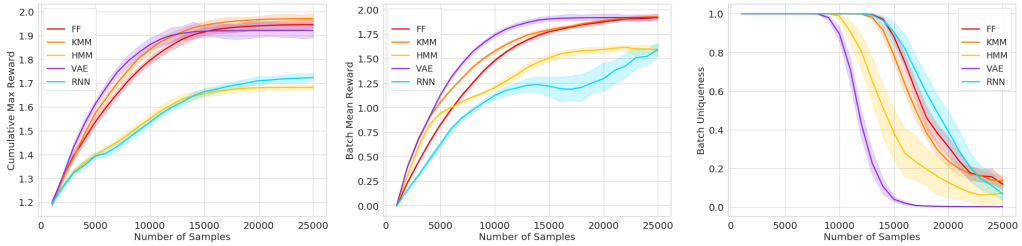

Figure 5: Results on the 5' UTR problem with tuned hyper-parameters. With the tuned parameters, the KMM meets the performance of the fully-factorized model. We used the same parameters for the VAE and the HMM as we saw no increase in performance when changing the parameters. The new hyper-parameters for the recurrent neural network improved maximum reward performance, but decreased performance with respect to the batch mean reward.

The comparison between DbAs and baselines is maintained when tuning parameters directly on the 5' UTR problem (Figure 6). In particular, the regularized evolution solver outperforms DbAs-VAE when the optimization process involves a large number of rounds. We note that, in the regularized evolution implementation, we ensure that the alive population contains unique sequences, which increases the diversity of the proposed sequences. Instead, DbAs-VAE uses the top scoring sequences from the last batch for weighted MLE. This discrepancy may account for the increased cumulative maximum reward of the regularized evolution on the UTR task. In particular, the batch uniqueness plot suggests that DbAs-VAE concentrates on a few configurations earlier in the optimization process, while the evolution solver continues to explore and therefore can continue to discover high-scoring sequences, and eventually outperform DbAs-VAE. On the other side, the batch mean reward for the evolution solver is lower than for DbAs-VAE as the former may contain potentially lower scoring unique solutions in the alive population. Adding a tunable diversity-inducing regularization term to the weighted MLE objective in DbAs-VAE might encourage higher diversity in the generative models and reduce this discrepancy.

5.2 Implementation and Hyper-parameter Tuning Details

We implemented DbAs [5] following the advice in the original paper under the assumption of noiseless oracles. In particular, the original paper uses the survival function to weigh each sample according to the probability that the stochastic oracle score passes a quantile-based threshold. A

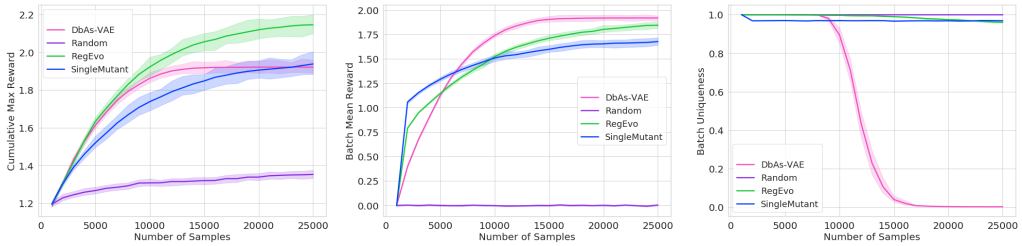

Figure 6: Results on the 5' UTR problem with tuned hyper-parameters. The comparison between DbAs and baselines is maintained when tuning parameters directly on the UTR problem. In particular, DbAs-VAE concentrates on a few configurations earlier in the optimization, while the evolution solver continues to explore and therefore can continue to discover high-scoring sequences and to eventually outperform DbAs-VAE.

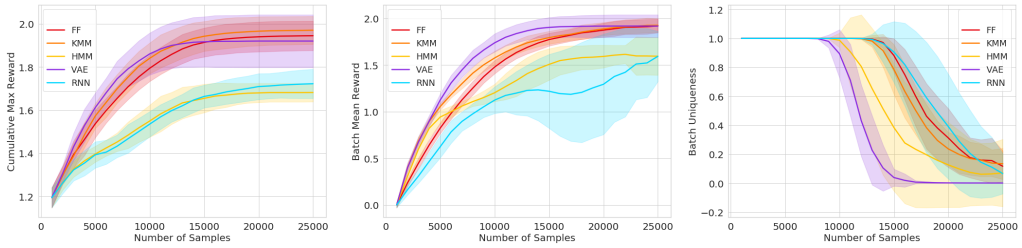

Figure 7: Results on the 5' UTR problem with tuned hyper-parameters where the curves and shaded regions represent the mean +/- the standard deviation as optimization progresses.

noiseless oracle is transformed into a stochastic one by assuming a Gaussian variable with constant variance (treated as a hyper-parameter and set to 10^{-6}). As a result, samples are dropped when their deterministic score is below the quantile threshold, and kept when they are above the threshold (with a weight of 0.5 when they match the threshold, and a weight of 1 when they are strictly above the threshold).

The elite selection process returns a dataset where a high proportion of samples are discarded. We note that alternative cross entropy method approaches use a larger proportion of samples and re-weight them based on the oracle rewards (e.g. by applying an alternative shaping function to the rewards [4]), and may lead to more stable optimization trajectories. We maintain the choice made in the original DbAs paper, and tune the quantile hyper-parameter Q for each generative model.

We run each solver 50 times, with different random seeds, and we plot the mean performance as well as confidence intervals computed using seaborn's plotting function's default arguments (bootstrap-based confidence intervals). We note that the confidence intervals are wider when using standard deviation (see Figure 7).

Fully-factorized model: For each position, we use a uniform Dirichlet prior over the parameters and perform maximum a-posterior estimation. The concentration parameter of the prior, which controls the entropy of the posterior, is tuned as a hyper-parameter. For the Ising model problem, we set the quantile Q to 0.9 and the concentration parameter to 1.25, while for the 5' UTR problem, we set the two parameters to 0.95 and 1.25, respectively. We found that larger concentration parameter values decreased performance, especially in terms of maximum reward at the end of optimization.

K-th order Markov model: For the Ising model problem, we set the quantile Q to 0.9, the block size to 3 and concentration parameters corresponding to start and transition probability priors to 1.25. We used only untied parameters, since in early experiments on the Ising model problem it lead to better performance. For the 5' UTR problem, we found that decreasing the conditioning block size to 1 improved results. The remaining parameters were set to 0.95 (quantile), 2 (start probability prior concentration parameter), 1.25 (transition probability prior concentration parameter), although they did not affect results in a significant manner.

HMM: We used the public implementation *hmmlearn*[1]. The hyper-parameters to optimize are the number of hidden states and the parameters of the Dirichlet priors on the start and transition probabilities. We set the number of components to 10, the concentration parameters corresponding to start and transition probabilities to 1.25, and we performed 100 expectation-maximization iterations for each optimization round. The quantile Q was fixed to 0.95. We note that due to increased computational burden, the grid search used for tuning the HMM was much smaller than for the alternatives. As it cannot handle weights, we extracted unweighted datasets by drawing samples with replacement proportional to the given sample weights. We drew $3 * N$ training samples (from a weighted dataset of size N) as we found no improved performance when increasing the number of training samples further.

VAE: We use fully-connected models for both the encoder and decoder models. We note for the neural network models, during each optimization round, we fine-tune the parameters from the previous iteration (rather than re-initializing the network). We tune the learning rate (0.01), the number of epochs (40), the number of hidden units (64), the latent size (50), the batch size (20), and the quantile Q (0.9). The numbers in the parentheses reflect the final hyper-parameter choice. We found that the learning rate and number of epochs are the parameters that affected performance the most. We used

[1] `https://hmmlearn.readthedocs.io/en/latest`

the same setting for both problems as tuning parameters for the 5' UTR problem did not improve results.

RNN: We use a single layer long short-term memory network (LSTM). We tune the learning rate (0.003), the number of epochs (Ising Model problem: 20, 5' UTR problem: 10), the number of hidden units (64), the embedding size (64), the batch size (20), and the quantile Q (0.95). Similar to the VAE, the learning rate and number of epochs are the parameters that affected performance the most. We note that the VAE consistently performed better than the recurrent neural network in the CEM formulation, likely due to the expected difficulties of fitting a recurrent neural network to large sequences (our sequences have length 50), or to the reduced number of samples that the models are trained on. While various tricks for tuning recurrent neural networks may improve results, most protein sequences are significantly longer than 50 amino acids, potentially limiting applicability to real-world design problems.

5.3 Baselines Details

The **single mutant walker** solver proposes each batch by starting from the sequence with the best reward in the previous batch (or the starting population, if no previous batch) and randomly sampling from the set of sequences that can be obtained via a single mutation. We note that the solver never repeats the sequence used for mutation throughout the optimization process. If the batch size is smaller than the possible number of single mutants, sequences are randomly sampled from the set of possible single mutants. However, if the batch size is larger, all of the single mutants are proposed, and the remaining sequences in the batch are sampled from the set of double mutants. If the batch size is even greater, the rest of samples are randomly selected from the set of triple mutants, and so on. The single mutant walker solver has no hyper-parameters to tune.

The **regularized evolution** solver is an implementation of the evolutionary algorithm in Real et. al [14], which proposes using *aging evolution*. In particular, parent selection is performed by maintaining a fixed-size *alive population* by removing the oldest individuals, followed by k-way *tournament selection* (randomly sampling with replacement k individuals and maintaining the highest scoring one). We ensure that the alive population contains unique sequences: we keep the newest unique samples. We allow using up to two parents in recombination, which is performed by crossover according to a crossover probability. The resulting child is mutated at each sequence position independently according to a mutation probability. If a mutation takes place, the new symbol is chosen uniformly at random, with replacement, from the vocabulary of tokens. We tune the following hyper-parameters: the *tournament size* (the number of individuals to sample for tournament selection), the *mutation probability* (the probability of mutating a token), and the *crossover probability* (the probability of crossover at each sequence position). For the Ising model task, we set the mutation probability to 0.05, the crossover probability to 0.2, and the tournament size to 20. For the UTR task, we changed the crossover probability to 0.1 which improved results only slightly over the previous configuration.