Differentiable RNA Folding with Applications

Benasque 2024

Dr Max Ward
Department of Computer Science and Software Engineering
University of Western Australia

July 26, 2024

The Team

- Talk co-author (couldn't be here today)
  - Ryan Krueger (Harvard University)
- Collaborators
  - Marco Matthies (University of Hamburg)
  - Dave Matthews (University of Rochester)
  - Sharon Aviran (University of California, Davis)
  - Elena Rivas (Harvard University)
  - Andrew Torda (University of Hamburg)
  - Michael Brenner (Harvard University)

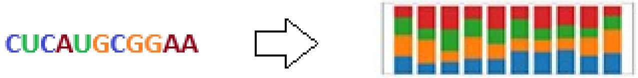
Figure: Ryan Krueger. 3rd year PhD in Applied Mathematics at Harvard University

Overview

- Algorithms can be differentiable and the gradient can be used for optimization
- Gradient-based optimization is very powerful and flexible!
- We have a differentiable RNA folding algorithm
- Some proof-of-concept applications
    - RNA structure design
    - mRNA design

Continuous Inputs

- The derivative shows how the output of the algorithm changes if we adjust the input
- The input and output must be continuous
- Typical RNA folding (Zuker-Stiegler, McCaskill) deals with a discrete sequence
- We generalise McCaskill's algorithm so its input is a continuous distribution of sequences rather than a single sequence



Continuous Inputs (ft. math!)

- One way to do this is to construct independent nucleotide distributions for each position
- Call the distribution of sequences Ψ
- $\Psi \in [0,1]^{4\times n}, \sum_{i=1}^{4} \Psi_{i,j} = 1$
- The probability of sampling a sequence $p(\pi|\Psi) = \prod_{j=1}^{|\pi|} \Psi_{\pi_j,j}$
- The partition function generalizes: $Z_\Psi = \sum_\pi \sum_{s\in S_\pi} p(\pi|\Psi) e^{-\beta E(s|\pi)}$
- You can think of this either as an expected partition function or the partition function for a probabilistic/continuous sequence

Generalized McCaskill's Algorithm (more math)

- The partition function can be calculated using a generalized McCaskill's algorithm

$$\mathcal{P}(b_i, b_j, i, j) = \sum \begin{cases} \mathcal{B}(\textsc{one-loop}(b_i, b_j, i, j)) \\ \mathcal{P}(b_k, b_l, k, l) \cdot \Psi_{b_k,k} \cdot \Psi_{b_l,l} \\ \quad \cdot \mathcal{B}(\textsc{two-loop}(b_i, b_j, b_k, b_l, i, j, k, l)) \\ \quad \forall b_k, b_l \in A, i < k < l < j \\ \mathcal{M}(2, i+1, j-1) \cdot \mathcal{B}(M_i) \cdot \mathcal{B}(M_p) \end{cases}$$

$$\mathcal{M}(p, i, j) = \sum \begin{cases} \mathcal{M}(p, i+1, j) \cdot \mathcal{B}(M_u) \\ \mathcal{P}(b_i, b_k, i, k) \cdot \mathcal{M}(\max(0, p-1), k+1, j) \\ \quad \cdot \mathcal{B}(M_p) \cdot \Psi_{b_i,i} \cdot \Psi_{b_k,k} \\ \quad \forall b_i, b_k \in A, i < k \leq j \end{cases}$$

Implementation

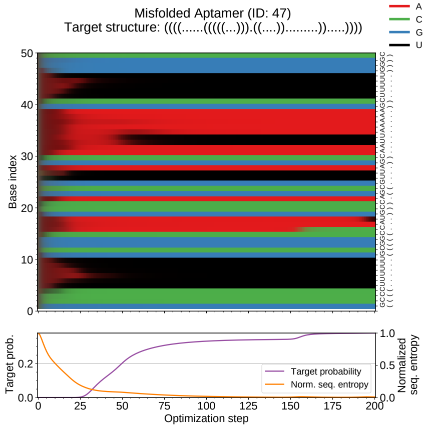
- Key observation. All the operations in this algorithm are differentiable
- We implemented this algorithm using an optimizing GPU autodifferentiation compiler (JAX)
- In practice, somewhat complicated:
  - No branches (if/else) allowed
  - No dynamic memory
  - In short, static computation only!

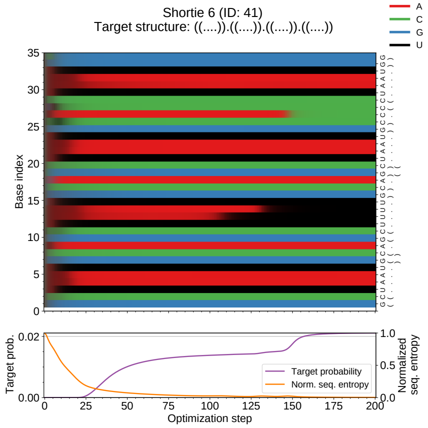
Gotchas and Caveats

- The nearest neighbour model is tricky without if statements
- There were memory issues with our first version
- Coaxial stacks and dangling ends
    - We initially targeted parity with ViennaRNA
    - Their default treatment of dangling ends (`-d2`) is bad for the generalized algorithm
- The time complexity is $O(n^3)$ but the memory complexity is $O(n^3)$. We need to store all linearization points for back propagation
- Our GPU had 80GB, so memory is the limit
- Our first experiments were limited to 50nts

Eterna100 Results

- We optimized the probability of the target structure via gradient decent





Eterna100 Results

- To optimize the probability of a target structure we compute the partition function for Ψ considering only a single structure s. Call this Z_Ψ^s
- Probability $\approx \frac{Z_\Psi^s}{Z_\Psi}$

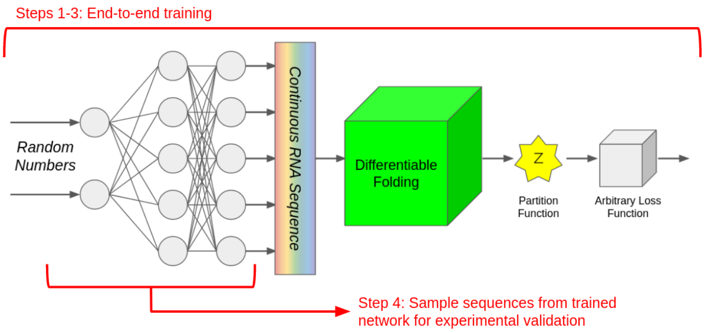
Puzzle ID	Initial	Optimized	Answer 1	Answer 2
1	0.017	$\underline{0.976}$	0.402	0.909
3	$\approx 10^{-13}$	$\underline{0.988}$	0.420	0.798
8	0.206	$\underline{0.984}$	0.545	0.596
10	$\approx 10^{-21}$	$\underline{0.962}$	0.530	0.716
11	$\approx 10^{-11}$	$\underline{0.941}$	0.449	0.562
15	$\approx 10^{-20}$	0.002	0.403	$\underline{0.540}$
20	$\approx 10^{-14}$	0.209	0.244	$\underline{0.588}$
23	$\approx 10^{-8}$	$\underline{0.563}$	0.005	0.021
26	$\approx 10^{-7}$	$\underline{0.987}$	0.235	0.241
30	$\approx 10^{-9}$	$\underline{0.980}$	0.094	0.162
33	$\approx 10^{-27}$	$\underline{0.726}$	0.676	0.594
40	$\approx 10^{-16}$	$\underline{0.990}$	0.835	0.794
41	$\approx 10^{-23}$	$\underline{0.021}$	0.001	$\approx 10^{-6}$
47	$\approx 10^{-31}$	$\underline{0.381}$	0.002	0.007
57	$\approx 10^{-35}$	$\underline{\approx 10^{-8}}$	$\approx 10^{-12}$	$\approx 10^{-14}$
65	$\approx 10^{-21}$	0.101	0.133	$\underline{0.136}$
66	$\approx 10^{-25}$	$\underline{0.003}$	0.001	$\approx 10^{-4}$

Citations

- The work presented thus far is from *Matthies, M.C., Krueger, R., Torda, A.E. and Ward, M., 2024. Differentiable partition function calculation for RNA. Nucleic Acids Research, 52(3), pp.e14-e14.*

Neural Network Projection

- We can add a neural network before the differentiable folding algorithm. The network's output is Ψ
- Gradients from differentiable folding can be used to update the network weights instead of updating Ψ directly
- In short, this is a higher dimensional projection


Steps 1-3: End-to-end training
Random Numbers
Continuous RNA Sequence
Differentiable Folding
Z
Partition Function
Arbitrary Loss Function
Step 4: Sample sequences from trained network for experimental validation

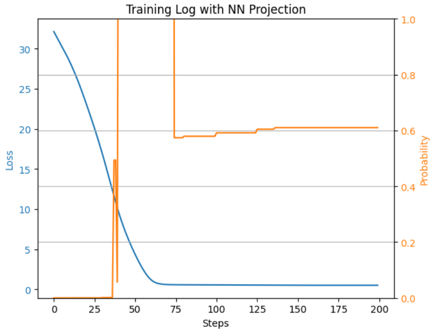
Eterna100 Results (with neural net)

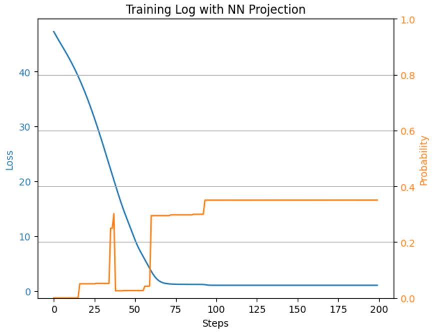
- Experiments with poor performance were re-run with a basic fully-connected network
- No hyperparameter optimisation or restarts were done

Puzzle ID	**Neural Net**	**Original**	**Answer 1**	**Answer 2**
15	0.416	0.002	0.403	$\underline{0.540}$
20	$\underline{0.610}$	0.209	0.244	0.588
41	$\underline{0.407}$	0.021	0.001	$\approx 10^{-6}$
57	$\underline{\approx 5.5^{-7}}$	$\approx 10^{-8}$	$\approx 10^{-12}$	$\approx 10^{-14}$
65	$\underline{0.351}$	0.101	0.133	0.136
66	$\underline{0.006}$	0.003	0.001	$\approx 10^{-4}$

Training Plots

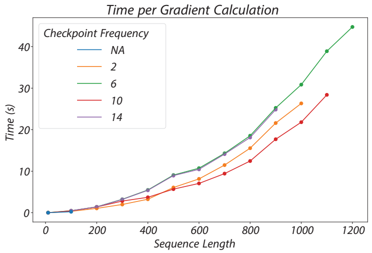
- Sometimes interesting things happened

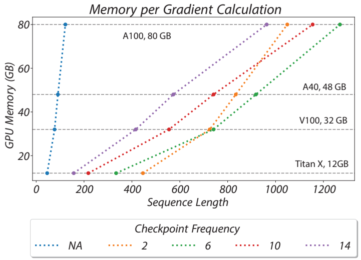




Algorithmic Improvements

- 50nt is too small
- We developed a checkpointing strategy for backprop to reduce the memory to $O(n^{2.5})$ at the cost of a 2x increase in compute time
- Don't use `-d2` (64x improvement)
- Better recursions to exploit symmetries (e.g., in internal loops 96x improvement)
- With all these optimizations we can get to 1650nt on an 80GB GPU





mRNA Design

- We wanted to test our improved method on mRNA design
- Objective: ensure CAI is above a threshold, maximize the partition function
- We use a neural network projection and train by gradient descent as before

$$\Omega(\pi|\alpha) = \begin{cases} Z_\pi & \text{if } \mathrm{CAI}(\pi|\alpha) \geq \tau \\ -\infty & \text{otherwise} \end{cases}$$

Loss Function

- Problem. $\Omega(\pi|\alpha)$ is not differentiable and is not a function of Ψ

Loss Function

$$\mathcal{L}(\Psi, \alpha) = -\log(Z_\Psi) \cdot f(\mathrm{ECAI}(\Psi)) \cdot g(P(\alpha|\Psi))$$

Definitions

- $\mathrm{ECAI}(\Psi)$ is the expected CAI sampled from Ψ
- $P(\alpha|\Psi)$ is the probability of sampling a valid coding sequence for the protein α
- f and g are hinge functions (e.g., ReLu) that punish going under a threshold

mRNA Results with a Seed

- We can consistently improve a good seed for EFE (e.g., LinearDesign)

	Unconstrained		**CAI $\geq$ 0.8**	
	LinearDesign	**Our Method**	**LinearDesign**	**Our Method**
MEV	-114.84	-114.92	-112.96	-113.04
Mini-GFP	-207.65	-208.59	-205.15	-205.15
Nanoluciferase	-452.34	-452.38	-451.29	-452.01
spike RBD	-411.55	-412.59	-407.50	-408.61
eGFP + degron	-546.92	-547.71	-546.56	-547.17

Krueger, R. and Ward, M., 2024. Scalable Differentiable for mRNA Design. bioRxiv, pp.2024-05.

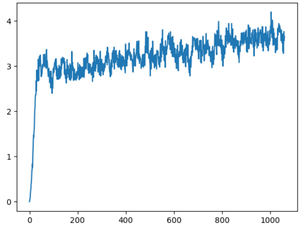
mRNA Results with Refinement

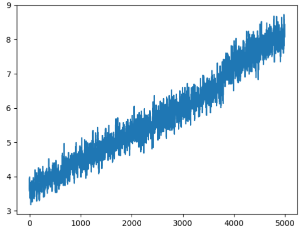
- We ran some experients to optimize AUP
- Gradient optimisation gets us to a good location in sequence space
- We sample from the optimized distribution and refine with an adaptive walk

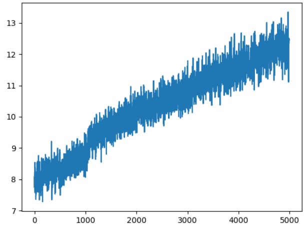
	Linear Design		**Our Method**	
	CAI	**AUP**	**CAI**	**AUP**
MEV (target CAI=0.8)	0.825	0.171	0.805	0.147
Mini-GFP (target CAI=0.9)	0.901	0.263	0.900	0.192
nLuc (target CAI=0.9)	0.885	0.203	0.888	0.184

General Network Pretraining

- We're trying to pretrain a general network
- No data needed–the model learns directly from the nearest neighbor model
- Proof of concept: train a neural network for sequences at most 50aa
- We train in batches of 256 randomly generated sequences

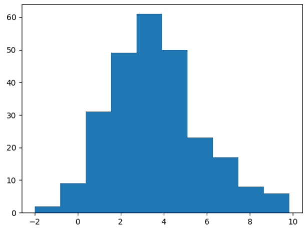


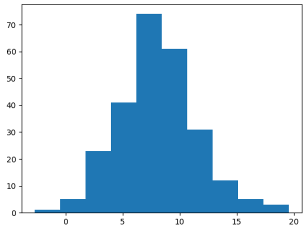


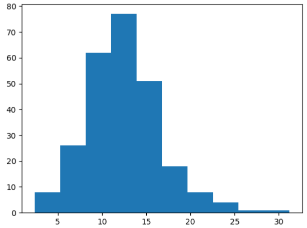


General Network Pretraining

- Distributions of $\log(Z_\Psi)$ differences to baseline random valid sequences







Future Plans & More

- Differentiable folding is a powerful and flexible tool with numerous applications
- Future plans
  - Difficult objective functions (e.g., forbidden motifs)
  - Foundation model for mRNA design
  - Scale existing structural design method
  - Foundation model for structural design
- Things I didn't have time to talk about
  - Parameter optimization
  - Module in structure prediction pipelines
  - Reparameterization trick for ideal training data