Generating and designing DNA with deep generative models

Nathan Killoran, Leo J. Lee, Andrew Delong, David Duvenaud, Brendan J. Frey

arxiv 2017

Reviewed by : Jack Lanchantin

[1]Department of Computer Science, University of Virginia

Outline

Generative Models

Generative models are good for many uses, including:

- Simulating data
- Exploring the space of possible data configurations
- Tuning generated data to have specific properties
- Inventing novel, unseen configurations

This Paper

- **Goal**: create synthetic DNA sequences and tune these sequences to have certain desired properties.
- **Methods**:
  1. GAN-based deep generative network for the creation of new DNA sequences
  2. Activation maximization method for designing sequences with desired properties
  3. Joint method of 1 & 2

Outline

Outline

GAN Generator

- Generator G transforms continuous variable z into synthetic data, $G(z)$, where z is a high-level latent encoding for the data
- Discriminator D produces a continuous valued number output $D(x)$ to score between real and generated output.
- Discriminator's training objective:

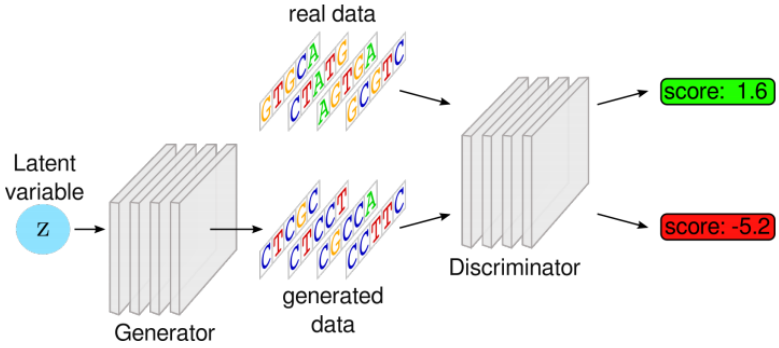
$$max_{\theta_D}\mathcal{L}_{disc} = max_{\theta_D}[\mathbb{E}_{x\sim P_{real}}D(x) - \mathbb{E}_{z\sim P_z}D(G(z))] \qquad (1)$$

- Generator's training objective:

$$max_{\theta_G}\mathcal{L}_{gen} = max_{\theta_G}[\mathbb{E}_{z\sim P_z}D(G(z))] \qquad (2)$$

GAN Generator for DNA

Wasserstein GAN (Arjovsky et al.): discriminator's output is adapted to an arbitrary score $D(x) \in \mathbb{R}$, and an optimization penalty is introduced to bound the discriminators gradients, making the model more stable and easier to train



Outline

Generative Optimization

- Instead of generating realistic-looking data, the focus in this alternative approach is to synthesize data which strongly manifests certain desired properties

Activation Maximization for DNA

- Let P be a function which predicts a target property $t = P(x)$ (e.g, x is a dog)
- P can be generalized to some combination of explicit functions $\{f_i\}$ and learned functions $\{f_{\theta_j}\}$:

$$P(x) = \sum_i \alpha_i f_i(x) + \sum_j \beta_j f_{\theta_j}(x) \tag{3}$$

- **Activation Maximization**: starting with an arbitrary x, change x to maximize t by following the gradient w.r.t x:

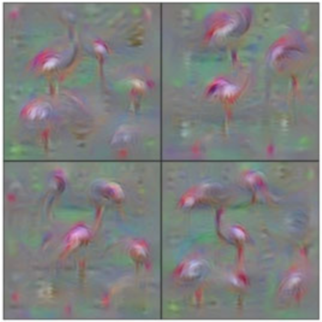
$$x \to x + \epsilon \nabla_x t \tag{4}$$

- Final sequence can be found by taking a softmax over the 4 characters at each position, and taking an argmax.

Outline

Joint method

- One drawback with activation maximization is that it ignores realism of data in its pursuit of optimal attributes
- E.g., such images are often exaggerated or nightmarish, with the target property manifesting in unrealistic ways



Flamingo

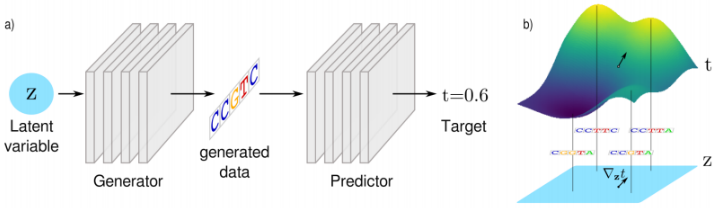
Joint method: Plug & Play Generative Network

- **“plug & play generative networks”** (Nguyen et. al.): combine activation maximization with a generative model
- Idea: Let a generator capture the generic high-level structure of data, while using predictors to fine-tune specific properties

Joint method: Plug & Play Generative Network

This joint architecture requires two components:

- Generator G transforms latent codes z into synthetic data x (e.g. a trained GAN generator), and a predictor P, mapping data x to the corresponding attributes $t = P(x)$.
- The two modules are plugged back-to-back, so that they form a concatenated transformation $z \to x \to t$



Joint method: Plug & Play Generative Network

- Goal is still the same as activation maximization: tune data to have desired properties
- To do this, we calculate the gradient of the prediction t with respect to the generators latent codes z:

$$\nabla_z t = \sum_i \frac{\partial t}{\partial x_i}\frac{\partial x_i}{\partial z} = \sum_i \frac{\partial P(x)}{\partial x_i}\frac{\partial G_i(z)}{\partial z} \tag{5}$$

Outline

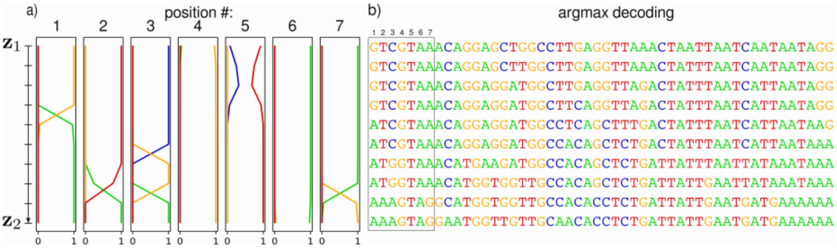
Outline

Experiment 1: Generative DNA Model

- Perform several experiments intended to more fully understand the capabilities of the DNA generator architecture

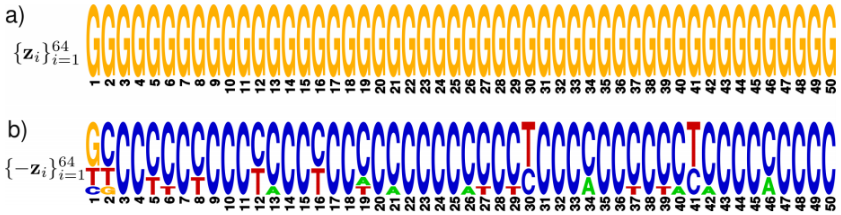
Exploring the Latent Encoding

- Trained a WGAN model on a dataset of 4.6M 50-nucleotide-long sequences encompassing chr 1 of hg38
- Consider interpolation between points in the latent space. Show how the generated data varies as we traverse a straight line between two arbitrary latent coordinates z_1 and z_2.

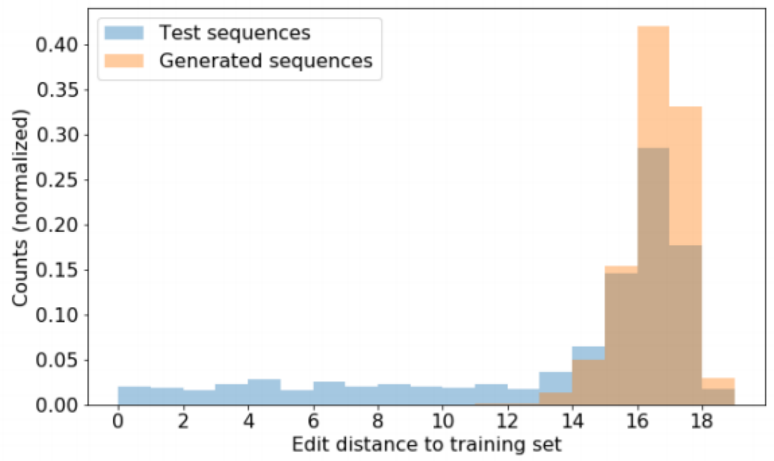


1.1: Exploring the Latent Encoding

- Reflection in the latent space: $z \to -z$
- Fix a sequence x^* (e.g. all "G") and find, via gradient-based search, 64 different latent points z_i which each generate x^* , i.e., $G(z_i) = x^*$ for all z_i
- Reflect each of these latent points and decode the corresponding generated sequences



1.1 Verification of GAN: Distance to training sequences

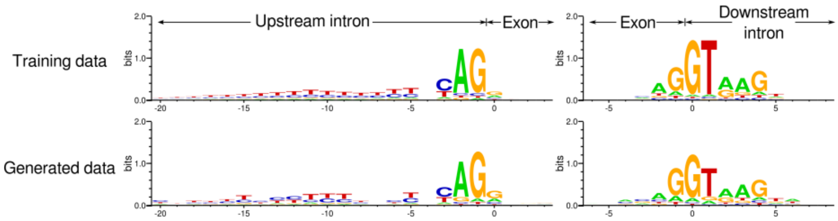


1.2: Capturing Exon Splice Site Signals

- Trained GAN on 116k 500-nt-long human genomic sequences, each containing exactly one exon (varying between 50-400 nt).
- Included an additional flag such that nucleotides within an exon $= 1$, and non-exon positions $= 0$
- Model must simultaneously learn to separate exons while also capturing the statistical information of nucleotides relative to these exon borders (splice sites)

1.2: Capturing Exon Splice Site Signals

- Used the generated flag positions to align the corresponding generated sequences (taking the first/last value above 0.5 as the start/end of the exon)
- Model has picked up on various splice site signals



Outline

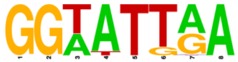
Experiment 2: Designing DNA

- Run several experiments for designing DNA sequences
- The running theme will be DNA/protein binding

2.1: Explicit Predictor (PWM): Motif Matching

- **Goal**: design DNA sequences using an explicit biologically motivated predictor function
- Predictor Function:
  1. 1-D convolution scans across the data, computing the inner product of a fixed PWM with every length-K subsequence
  2. Select the convolutional output with the highest value to get the final score for the sequence
- Used the joint method, employing a generator trained on sequences from human chr 1

a)



b)

```
TGAGAGTGATGTATTGGAATTGATGCCTCACCTCTGCTTGCAGACTGTCA
GGAATGAACTGGGGAGACAGGCCCAGAGGAATTGAGAAAGTAATGAGCAC
GCCCTGGGTTTTAAGAAATACTGTTGCATCAGGGCAAATGTAAGATTTTG
TTTTGTTTGAGATCTGTGGGGTATGCTGGAATTAAAGTCTGGACTACCAC
CTGATACTGAATGCAGATTTGAAGAACAAAGGGTATTAAAACACATGCTT
GATCCCCAAGTGTGGAATTGAGAAGGAAGCTGGAGAATCCCCAAACTCTG
CAGCCACATCAGCTTACCTAAGGAAGTGATGTGTTTTAAAACCAGCTTTG
TAGAATTTTTCTTGGTATTAATGATGATCTAGGCTTACACAGGACATCA
GACATTGCTTAGTCTGAGGGATACAGTGGGGAGTGGGTATTAAAATCTCC
```

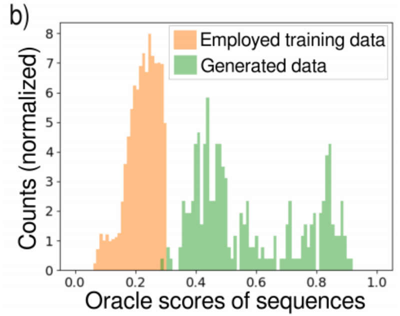
2.2: Learned Predictor (DeepBind): Protein Binding

- **Goal**: Explore the use of a predictor model which has been learned from data $\rightarrow$ Design new sequences which have high binding scores
- **Oracle model** To simulate the process of evaluating candidate sequences, use a proxy model which is trained on Chip-Seq data. This model can be queried with new designed sequences to gauge their expected binding score

2.2: Learned Predictor: Protein Binding

GAN for generating new sequences

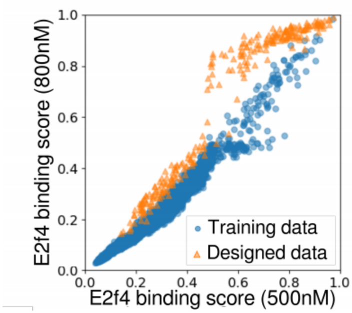
- Using only samples with oracle scores less than 40% binding likelihood, train a gan to generate new sequences, and then test the generated sequences on the oracle.

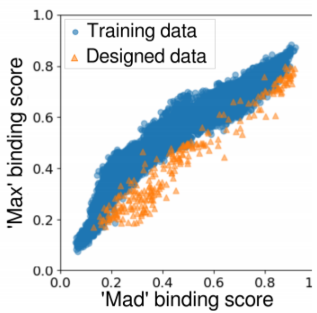


2.2: Learned Predictor: Protein Binding

Optimizing Multiple Properties

1. Design DNA sequences which preferentially bind to one protein in a family but not the other
2. Similarly, design sequences where two predictors model binding of the same protein, but under two different molecular concentrations





Future Directions

- Train an encoder E which maps data back to latent codes: $E(x) = z$, making it easier to find latent encodings for specific sequences
- Build a conditional GAN model and combine it with the joint architecture - allowing some properties to remain fixed while others were tuned
- Domain adaptation. E.g. provide a map of where we want certain components (introns, exons, promoters, enhancers) to be, and a generative model would dream up plausible sequences with the desired properties

Outline

RNNs

Training: maximize the likelihood of predicting the next char
Generating: Sample the model's prediction at each time t and feed back as the input to the next step $t+1$ (arbitrarily long seqs)

- Can be trained to generate sequences in conditional manner, producing outputs which have some desired property. Do this by appending extra labelled data y (e.g. sentiment) to the inputs x_t.

RNNs

Suitability for DNA

- No successful variant of activation maximization or plug & play that operates on RNNs.
- Also, without a learned latent encoding, we are limited to tune a conditional RNN for which we explicitly train the model for (e.g. no flipping the sequence).

Deep Autoregressive Models

Training

- Instead of feeding inputs only one at a time and relying on the network to memorize past inputs, we can alternatively show it the entire past history up to that point

Generating

- Similar to RNNs, feed the history of previous predictions as input for each time step.
- Can also be built as conditional models, enabling the generation of sequences with tailored properties.

Deep Autoregressive Models

Suitability for DNA: Similar to RNNs, they require supervised training with a labelled dataset and that these properties must be chosen beforehand and built in during training.

Variational Autoencoders

- In contrast to the 2 previous models, VAEs have the ability to learn a controllable latent representation of data in an unsupervised manner
- By changing the latent variable z, we can modify the synthetic data that the model generates.

Variational Autoencoders

- Encoder E: transforms data to latent variables, $x \to z$
- Decoder (or generator) G: transforms latent variables to generated data, $z \to x'$
- VAEs use probability distributions rather than deterministic functions to model these transformations
  - To encode, we sample z from a distribution $q(z|x)$
  - To decode, we do likewise for x from a distribution $p(x|z)$.
  - q and p are modelled via DNNs.

Variational Autoencoders

Training:

- Goal: make the error from $x \to z \to x'$ as small as possible
- For VAEs, this reconstruction error is given by

$$\mathcal{L}_{recon} := \mathbb{E}_{z \sim q(z|x)}[-logp(x|z)] \tag{6}$$

- In order to reconstruct successfully, the model must learn how to capture the essential properties of the data within the latent variable z
- Regularization encourages the latent codes to vary smoothly. This is captured by a KL divergence term between $q(z|x)$ and a fixed prior on the latent space $p(z)$ (e.g. normal)
- The full VAE objective which is minimized during training is

$$\mathcal{L}_{recon} + D_{KL}(q(z|x)||p(z)) \tag{7}$$

Variational Autoencoders

Suitability for DNA:

- It has been observed that if we use a strong decoder network, such as an RNN, VAEs will exhibit a preference to push the KL divergence term to zero
- This causes the latent code to be ignored and the generative process is handled completely by the decoder
- Without learning a meaningful latent code, such models are no better than a standard RNN

All Methods

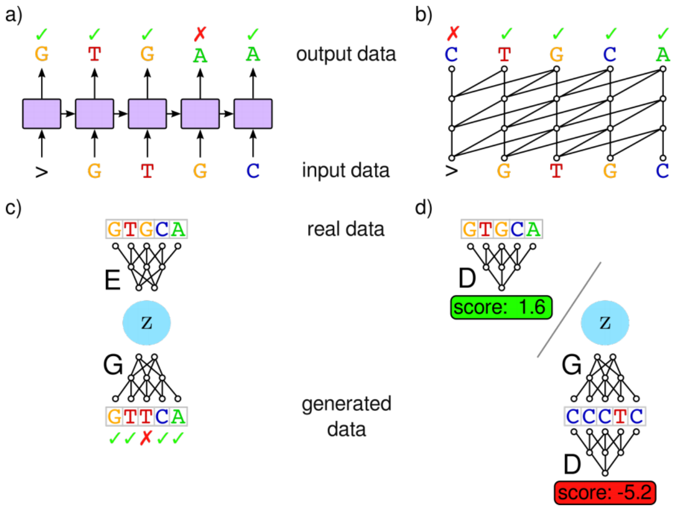


Figure 12: Generative neural network models shown with short example sequences: a) Recurrent neural network; b) PixelCNN; c) Variational Autoencoder; d) Generative Adversarial Network. A generic starting character (e.g., '>') is used to prompt the RNN and PixelCNN at the first time step.