DRUG DISCOVERY HACKATHON 2020

PROBLEM STATEMENT DDT2-10

TEAM 'IISC UG'

REGISTRATION ID: 8414

Technical Document

Title	
Document Name	Technical Document for Antiviral_SeqGAN – DDT2-10, Team 'IISc UG'
Version No.	1
Date	25/10/2020
Document Author(s)	Preetham Venkatesh

Document Version History

Version	Date	Author	Change	Sections Affected
1.0	25 Oct 2020	Preetham Venkatesh	-	-

1. Introduction

This Document describes the development and usage of the Antiviral_SeqGAN program, designed by Team 'IISc UG' (Registration ID: 8414) for Problem Statement DDT2-10 of the Drug Discovery Hackathon 2020.

The requirement of the Problem Statement is stated as follows: "ML model for antiviral peptide predictions using Generative Adversarial Networks".

To complete this task, we developed a SeqGAN model to design novel antiviral peptides. SeqGAN (Yu *et. al*, 2017) is a state-of-the art method for text generation in Natural Language Processing tasks. We thus restate the next-token prediction task in NLP to next-amino acid prediction for our antiviral peptides. This is achieved by tokenization of amino acid sequences.

SeqGAN allows us to overcome the commonly faced issue in NLP of evaluating partially generated sequences by modeling the generator as a stochastic policy in reinforcement learning, and updating the performance via gradient policy update. Details are available in the referenced manuscript.

Our contribution lies in implementing this algorithm for peptide data, tuning hyperparameters for optimal performance, and evaluating the output for novelty as well as functionality.

2. Technical Design

Technical Approach

Development of a Sequence-Generative Adversarial Network (SeqGAN) to output bioactive antiviral peptides (AVPs)

Scope

2.1.1. In Scope

The model is limited in its application to generating AVPs that it learns from its training dataset

2.1.2. Out of Scope

The model is unlikely to output AVPs that do not share a biophysical basis with the examples it learns in the training dataset.

Other Affected Applications

The principles of Antiviral_SeqGAN could be extended to other functional bioactive peptide families, such as Antimicrobial pepties.

Assumptions

The sequence data contains sufficient information for the language model to learn the biophysical properties that guide antiviral activity. This is based on the sequence-structure-function relationship and uses the many established sequence-based protein/peptide design tasks as precedent for the assumption.

Issues

Raw sequence data is limited in the information it carries, and SeqGAN do not have a precedent in its implementation for functional protein/peptide design

Risks

Antiviral_SeqGAN only outputs peptides based on language models. Any experimental testing of the outputs must be rigorously checked for safety.

3. Algorithm

Dataset

The AVPdb dataset (Qureshi *et. al*, 2013) was as positive examples of antiviral peptides for our GAN. This dataset is freely available for download at: http://crdd.osdd.net/servers/avpdb/

Only the sequences were retained from the dataset. This file named ‘AVPdb_data.csv’ is available at https://github.com/preetham-v/Antiviral_SeqGAN.

Pre-processing

To conform with the Problem Statement requirement of peptide molecular weight being < 2000 Da, the model was restricted to only deal with sequences that are at most 18 amino acids in length. (Average weight of an amino acid is 110 Da). Note that this can be easily modified by retraining a deeper model on longer amino acids.

Any sequences in the dataset longer than 18 amino acids were restricted to only the first 18 amino acids for the training. This step could be improved by incorporating the entire sequence using an overlap window – however, this was not done for this version of Antiviral_SeqGAN to limit the number of parameters to optimize.

A character dictionary was used which tokenizes all 20 amino acids, and sequences were converted to an array of tokens using this. The character dictionary is given below:

CHARACTER_DICT = {

'A': 1, 'C': 2, 'E': 3, 'D': 4, 'F': 5, 'I': 6, 'H': 7, 'K': 8, 'M': 9, 'L': 10, 'N': 11, 'Q': 12, 'P': 13, 'S': 14, 'R': 15, 'T': 16, 'W': 17, 'V': 18, 'Y': 19, 'G': 20}

An example of tokenization is given below.

Sequence: AHIGYTW

Token array: [1, 7, 6, 20, 19, 16, 17]

The array was prepended with a starting token and padded to a total sequence length of 18 using an END token.

Generator

A standard Recurrent Neural Network (RNN) is used as the generative model, which maps input embedding representations to a sequence of hidden states. The final output layer used softmax to output a token probability distribution. For generative tasks, an amino acid is picked using this probability distribution.

Discriminator

The discriminator is a Convolutional Neural Network (CNN) that predicts the probability of an input sequence being a positive example. A token embedding is used here as well. A max-over-time pooling operation is applied over the feature maps at the final step. The optimization target is to minimize the cross entropy between the ground truth label and the predicted probability.

Pre-training

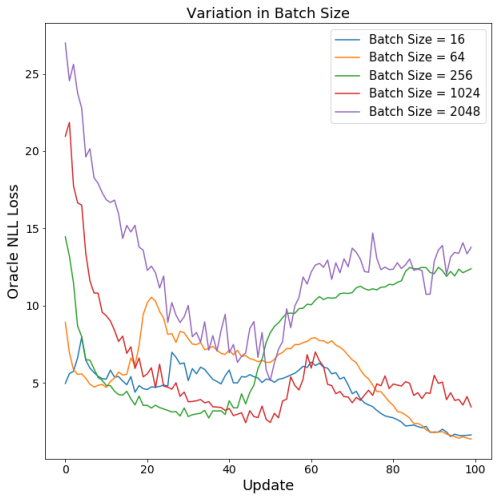
The generator is first pre-trained using Maximum Likelihood Estimate. Similarly, the discriminator is pre-trained using positive examples from our dataset and negative examples from the pre-trained generator.

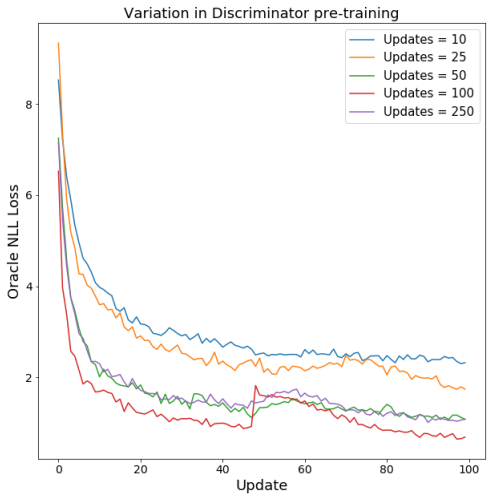
Adversarial Training

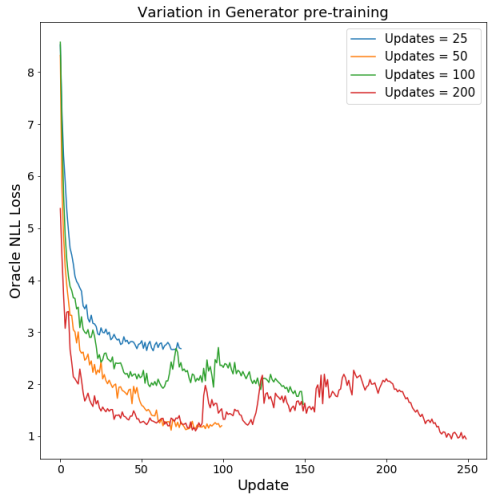
Finally, the generator and discriminator are simultaneously trained in an adversarial setting for a pre-determined number of epochs.

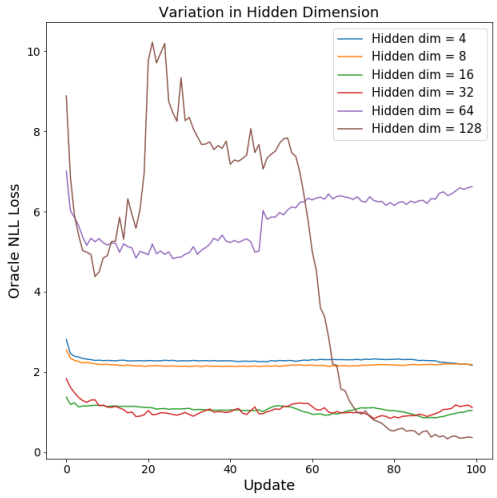
4. Hyperparameter tuning

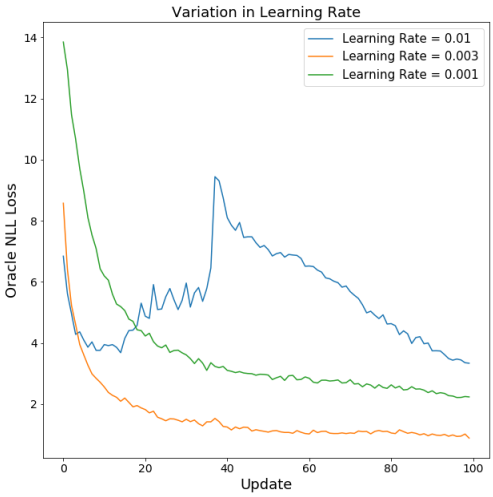
The hyperparameters were tuned using synthetic data generated by an Oracle model as described in the original SeqGAN paper (Yu *et. al*, 2017). Some example plots are shared below to demonstrate this tuning. The parameters were optimized one dimension at a time.

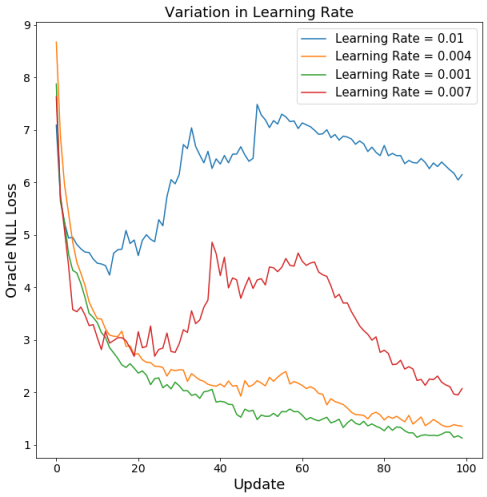












5. Final Hyperparameters

The following hyperparameters were used to generate the models and can be used to reproduce the results we obtained.

```
#Fixed Params
VOCAB_SIZE = 22 #Starting Letter + 20 AA + Padding
MAX_SEQ_LEN = 18 #2000 kDa / 110 kDa = 18
START_LETTER = 0
POS_NEG_SAMPLES = len(all_data) #Size of AVPDb dataset
torch.manual_seed(11)

#Variables
BATCH_SIZE = 16
ADV_TRAIN_EPOCHS = 5000

#Generator Parameters
MLE_TRAIN_EPOCHS = 50
GEN_EMBEDDING_DIM = 3
GEN_HIDDEN_DIM = 128
NUM_PG_BATCHES = 1
GEN_lr = 0.004

#Discriminator Parameters
DIS_EMBEDDING_DIM = 3
DIS_HIDDEN_DIM = 128
D_STEPS = 100
D_EPOCHS = 10
ADV_D_EPOCHS = 5
ADV_D_STEPS = 1
```

6. Testing Output

20 output sequences were generated using 3 different random seeds (chosen randomly) from the model trained for 5000 epochs. The sequences generated were submitted to AVPPred (threshold = 50 in all 4 methods), AVP-IC50-Pred (Hybrid model 1a), and Meta-iAVP (Instance-based classifier) to evaluate the likelihood of the sequences being an AVP. The results are described below:

1) Torch.manual_seed = 11

   AVPPred: 6/20 sequences classified as AVP
   AVP-IC50-Pred: IC50 range = 0.83 to 46.65 micromolar
   Meta-iAVP: 10/20 sequences classified as AVP

2) Torch.manual_seed = 49

   AVPPred: 6/20 sequences classified as AVP
   AVP-IC50-Pred: IC50 range = 0.83 to 38.72 micromolar
   Meta-iAVP: 10/20 sequences classified as AVP

3) Torch.manual_seed = 201

   AVPPred: 7/20 sequences classified as AVP
   AVP-IC50-Pred: IC50 range = 0.75 to 74.85 micromolar
   Meta-iAVP: 13/20 sequences classified as AVP

7. Sample Output

Sample outputs for manual seed 201 is shared below:

>pep0
KKKKVFAATYVLV
>pep1
RRKKA
>pep2
RRKKAAVALLPAVLLALL
>pep3
KKKKVVAATGVLV
>pep4
KKKKVVAAFYVLV
>pep5
KKKKVVAAFYVLV
>pep6
KKKKVVAAGYVLF
>pep7
VALDPIDISIELNKAKSD

>pep8

KKKKVVAAFLVLV

>pep9

KKKKVVAATYVLV

>pep10

KKKKVLAAFYVLV

>pep11

RRKKAAVALLPAVKLLLL

>pep12

RRRRRR

>pep13

KKKKLVLAFLFFF

>pep14

ILMCFSID

>pep15

ALDPIDISIELNKAKSDL

>pep16

ARKTRRRRR

>pep17

SWLRDIWDWICEV

>pep18

RRKKAVLLALLAP

>pep19

RRKKLLALLAP

While some of the sequences in any output are similar, sufficient diversity can be generated through multiple runs.

8. Requirements

The following versions were used in the code:

1) Python == 3.7.6
2) PyTorch == 1.4.0
3) NumPy == 1.18.1

4) Pandas == 1.0.3
5) Re == 2.2.1

The model was trained using a GPU on the Drug Discovery Tool Room.

9. Usage

The models folder provided has three models available as '.pth' files named after the number of adversarial epochs used when they were generated. The generator and discriminator are named respectively as follows:

1) gen_1000.pth and dis_1000.pth
2) gen_5000.pth and dis_5000.pth
3) gen_20000.pth and dis_20000.pth

The default version loaded is the gen_5000.pth and dis_5000.pth.

To use a pre-trained model, simply run:

python gan_generate.py

This will generate an output.txt file which lists the sequences. To modify the number of output sequences, or to change the model, open the code and scroll below to change the parameters.

To train your own model, first set your parameters by opening the 'gan.py' file and scrolling below to the list of parameters. Make any desired changes, including the name of the model.pth file you wish to save. Then, simply run:

python gan.py

This will save your named models in the 'models' directory, or any other path that you specify. This can then be used to generate outputs using the 'gan_generate.py' code.

10. Code Availability

The code is available at: https://github.com/preetham-v/Antiviral_SeqGAN under a CC-BY license.

11. Contact Details

For any questions or difficulties, please raise an issue on the github repo or contact preetham1104[at]gmail.com.