## Assignment Goals

- Gain a deeper understanding of convolutional neural networks for regulatory genomics
- Resolve ambiguity in RNA-Seq quantification
- Use Gaussian processes to solve problems with temporal data

## Instructions

- To submit your assignment, log in to the biostat server **`mi1.biostat.wisc.edu`** or **`mi2.biostat.wisc.edu`** using your biostat username and password.
- Copy all relevant files to the directory **`/u/medinfo/handin/bmi776/hw3/<USERNAME>`** where **`<USERNAME>`** is your biostat username. Submit all of your DragoNN output files, your code for Part 1C, and any code or spreadsheets you created for Part 2. Do not run DragoNN on **`adhara.biostat.wisc.edu`**.
- Compile all of your written answers in a single file and submit as **`solution.pdf`**.
- Write the number of late days you used at the top of **`solution.pdf`**.
- Show your work for partial credit.

## Part 1: Deep RegulAtory GenOmic Neural Networks (DragoNN)

We will use the DragoNN Python package[1] to explore convolutional networks for regulatory genomics. DragoNN can create DeepSEA-like networks but is more user-friendly, makes it easier to test different network architectures, implements network interpretation strategies, and simulates DNA sequence training data for user-specified *cis*-regulatory modules.

The package has many dependencies, include old versions of other Python packages and unpublished packages in GitHub repositories. We will provide minimal support for installing it on your own machine and instead request that you perform all of your testing on **`mi1.biostat.wisc.edu`** or **`mi2.biostat.wisc.edu`**. Do not wait until the day or two before the homework due date to conduct your tests because the server load will make network training slow.

[1] http://kundajelab.github.io/dragonn/index.html

To run DragoNN on the biostat servers, first make sure you have the BMI 776 Python installation set as the default as described in HW0. To confirm this, the command:

```
type -a python
```

should show

```
python is /u/medinfo/bmi776-miniconda3/bin/python
```

in the first line of the output.

Once you are using the BMI 776 Python environment, switch to the special HW3 conda environment with:

```
source activate hw3
```

This environment has the DragoNN package installed. Test that DragoNN is available with:

```
dragonn -h
```

For the following exercises, copy the HW3 sequence data and `interpret.py` to your handin directory. Run everything within your handin directory and leave the output files there. Specific questions you should answered are **bolded**.

**1A**: *Training convolutional networks*
First you will train a convolutional neural network on data from a simulated ChIP-Seq experiment. You have been provided a FASTA formatted file of 5000 DNA sequences bound by some regulatory proteins, `positive_train.fa`, and a negative set of 5000 unbound sequences, `negative_train.fa`. Use the following DragoNN command to train a 1 layer network with a 5 hidden units (filters) and a convolutional window of 15 base pairs:

```
dragonn train --pos-sequences positive_train.fa --neg-sequences
negative_train.fa --prefix training_1_layer --num-filters 5 --conv-
width 15
```

This trains the neural network and saves the model architecture and learned weights to **`training_1_layer.arch.json`** and **`training_1_layer.weights.h5`**. DragoNN splits the input data into a training and validation set and reports several performance metrics at each epoch (iteration) of training.

**What are the training auPRC (area under the precision recall curve) and validation auPRC at epoch 1?**

**What are the training auPRC and validation auPRC at the final epoch?**

The 1 layer network is an extremely simple neural network. We can train a more complex network by adding more filters and convolutional width arguments to the **`dragonn train`** command. Try a 2 layer network with 15 filters per layer and a window size of 15 base pairs:

```
dragonn train --pos-sequences positive_train.fa --neg-sequences
negative_train.fa --prefix training_2_layer --num-filters 15 15 --
conv-width 15 15
```

**What are the training and validation auPRC at the first and last epochs?**

**Why is the 2 layer network's performance better than the simple 1 layer network?**

DragoNN supports other training strategies and network architectures. The command:

```
dragonn train -h
```

shows some of other options. **`--pool-width`** changes the size of the pooling layer. **`--L1`** and **`--dropout`** are different regularization strategies for learning the weights. **`--num-filters`** and **`--conv-width`** can also be extended to three or more layers as long as you provide the same number of integer arguments to both of them. However, we will not use these other features.

**1B**: *Using and interpreting trained convolutional networks*
You will now inspect and interpret the 2 layer convolutional neural network you trained above. Use the command:

```
python interpret.py --pos-sequences positive_test.fa --neg-sequences negative_test.fa --arch-file training_2_layer.arch.json --weights-file training_2_layer.weights.h5
```

This will load the trained network from the `training_2_layer.arch.json` and `training_2_layer.weights.h5` files, load positive and negative test sequences, predict the probability that the test sequences are bound (i.e., in the positive class), and visualize the trained network.

Examine the output file `training_2_layer_architecture.png`. This shows a graphical representation of the layers of the neural network and their sizes.

**What do the input and output dimensions of the first Convolution2D layer correspond to (ignore the Nones and 1s)?** ***Hint: If it is not obvious, try training different networks with different values of*** `--num-filters` ***and*** `--conv-width` ***to see how these dimensions change.***
**What do the input and output dimensions of the Dense layer correspond do?** ***Hint: A Dense layer in Keras, the framework DragoNN uses, is what DeepSEA refers to as a fully connected layer.***

The probabilities that the test sequences are bound by the transcription factors are printed to the screen for the positive and negative test sequences. Suppose we predict that all sequences with P(bound) ≥ 0.5 are bound (positive) and all others are not bound (negative).

**How many true positives, false positives, false negatives, and true negatives are predicted?**

`interpret.py` also visualizes the true motifs that were used to generate the positive training and test data. Examine these motifs in the output files `motif1.png` and `motif2.png`. The output file `training_2_layer_convolutional_filters.png` visualizes the filters learned in the first convolutional layer, that is, the weights for the hidden units in this layer.

**Do any of the filters resemble the true motifs? In a multi-layer network, why do the first layer filters not need to learn motifs to get good predictive performance?**

DeepLIFT[2] provides an improved way to interpret convolutional neural networks versus visualizing the filters. DeepLIFT computes a score for each input feature. Examine the DeepLIFT plots for each positive test sequence in the subdirectory **`training_2_layer_deeplift_positive`**. The top panel shows the summarized score at each position in the input sequence. The gray region is zoomed and shown in the bottom panel with nucleotide-specific scores.

**Do the DeepLIFT scores look more or less similar to the true motifs than the convolutional filter visualizations? Do they represent both true motifs equally well?**

**1C**: *Implementing a forward pass*

Assume that a network has been trained using DragoNN and the network weights are available. The network includes 2 convolutional layers, a max-pooling layer, and a fully connected layer in sequence. Each convolutional layer contains 5 filters, has window size and stride size equal to 15 and 1, respectively, and uses ReLU as its activation function. The max-pooling layer has window size and stride size both equal to 35. The fully connected layer uses the sigmoid function as its activation function.

Write a program **`forward_pass.py`** that reads in **`positive_test.fa`**, **`negative_test.fa`**, and weights files and outputs the probability of each test sequence being bound. Transform the sequences using one-hot encoding. For example, **`GAATTC`** is encoded as

```
A 0 1 1 0 0 0
C 0 0 0 0 0 1
G 1 0 0 0 0 0
T 0 0 0 1 1 0
```

[2] https://arxiv.org/abs/1704.02685

The weights files with the prefix **`conv1`** are for the first convolutional layer. There is one file for each channel (filter) and another for the bias terms, which has the first filter's bias parameter in the first row. Files with the prefix **`conv2`** are for the second convolutional layer, and the **`dense`** files are for the final fully connected layer. The first 13 weights of **`dense_wgts.txt`** are for the first filter in the second convolutional layer, the next 13 weights are for the second filter, and so on.

You are *strongly recommended* to compute the output at each layer using matrix operations. You may find **`NumPy`** functions **`matmul`**, **`reshape`**, **`vstack`** and **`hstack`** helpful. No padding is needed. At the max-pooling layer, discard as few positions at the tail of the input as possible so that the remaining input size is divisible by the window size. Do not forget the bias terms.

Your output will *not* match the DragoNN output exactly. We will discuss the output predictions from our reference implementation on **`positive_test.fa`** and **`negative_test.fa`** on Piazza so you can check your output.

Input files and Python files can be downloaded from
**`https://www.biostat.wisc.edu/bmi776/hw/hw3_files.zip`**

## Part 2: RNA-Seq Rescue Algorithm

The full RSEM algorithm is too complicated to execute manually, but we can use the RNA-Seq rescue method presented in class to approximate one iteration of expectation maximization. The bipartite graph (Figure 1) contains two types of nodes: transcripts and read groups. The transcript nodes contain a transcript id and the transcript length in base pairs (bp). The read nodes contain the read counts for a group of reads that all align to the same transcripts. Transcript-read group edges designate the transcripts to which each read group aligns.

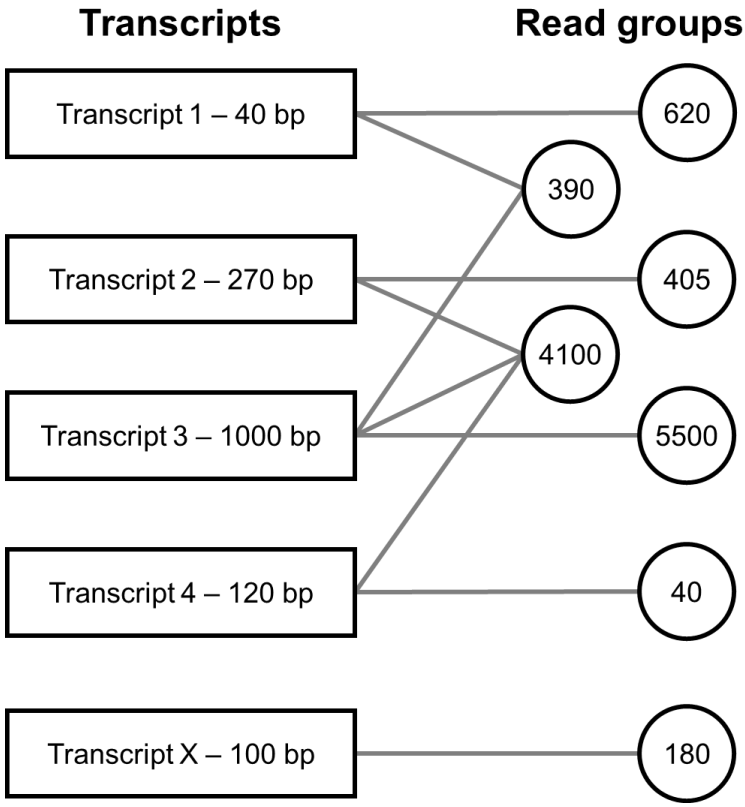

Figure 1: RNA-Seq bipartite graph

**2A**: *Estimating relative abundance*
Use the rescue method to calculate the relative abundance for the five transcripts to three decimal places. Show your work for partial credit.

**2B**: *Estimating absolute abundance*
Transcript X is a RNA spike-in. 1000 copies of transcript X were mixed into the experimental sample when preparing the sample for RNA-Seq, meaning its absolute abundance is 1000. Use the relative abundances you calculated above to calculate the absolute abundances for the other four transcripts, rounded to the nearest whole number. Show your work for partial credit.

*Hint: Review the "Issues with relative abundance measures" slide from the RNA-Seq lecture. Given the relative abundances for all six genes and the absolute abundance of Gene 6, you can derive the absolute abundances of Genes 1 through 5.*

**Part 3: Gaussian processes for time series data**

In a biological time series study, gene expression levels are collected at multiple time points. If the goal is to learn how cells react to an external stimulation, we can measure gene expression at 0 min (immediately before the stimulation) and at $t_i$ min after stimulation for $i \in \{1, \ldots T\}$. Gaussian processes with a squared exponential are well-suited for modeling biological data collected over time. The posterior mean is smooth over time, and the confidence intervals track uncertainty between the measured time points.

Suppose we are studying heat shock, a sudden temperature increase, and want a statistical test to assess which genes are differentially expressed over time. Specifically, we perform RNA-Seq on cells in normal growth conditions at 0, 5, 10, 15, 30, 60, and 120 min. We perform RNA-Seq at the same time points on cells that are heat shocked at 0 min (Figure 2).

Describe a Gaussian process-based test that can be applied separately to each gene to assess whether its temporal expression profile in the normal growth condition differs from the profile under heat shock.

*Hint: Recall that we can optimize the squared exponential kernel hyperparameters to maximize the posterior likelihood of some observed data and compute that posterior likelihood.*

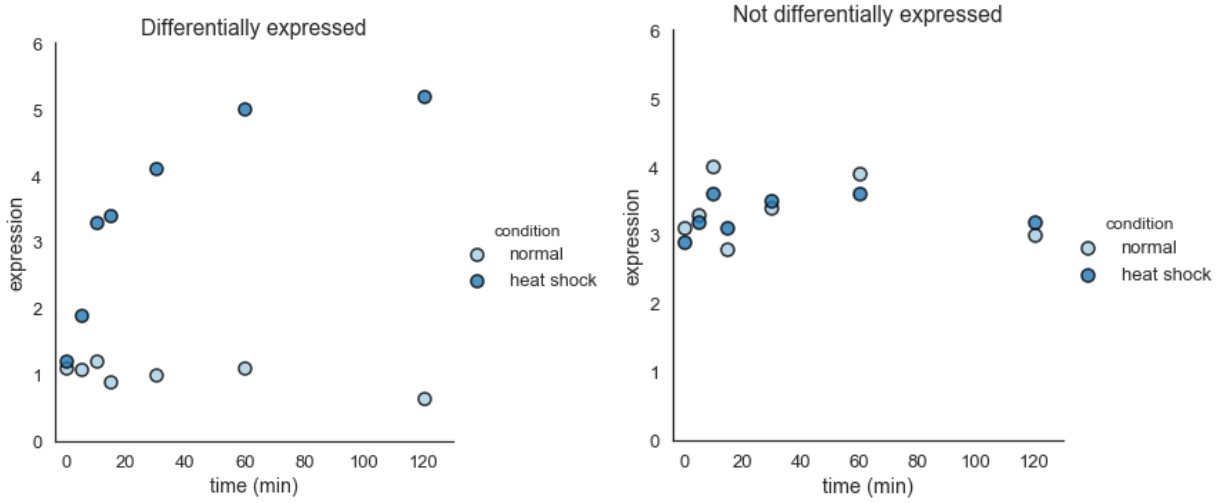


Figure 2: Examples of one gene that is differentially expressed and one that is not