NIH NATIONAL CANCER INSTITUTE

Summer 2018 Internship Documentation: Introduction to Python, Machine Learning, and using the CANDLE Benchmarks

Madeleine Birdsell and Catherine Warnement
Dr. Eric Stahlberg

Table of Contents

Introduction **3**

Python and Installation **4**

Machine Learning **6**
- Introduction 6
- Creating Our Own Models 7

CANDLE Benchmarks **14**
- Running CANDLE 14
- P1B1 Test Cases 15
- NT3 Test Case 20
- Model for Data Generation 21

Biowulf **28**
- Introduction 28
- Running CANDLE Benchmarks 30
- Amazon EC2 35

Databases **36**
- Database Chart 36
- Specific Data Repositories 39
- Data for Models 40

Readings **47**
- Machine Learning for Cancer Data 47
- Machine Learning for Imaging 53
- Autoencoders 60
- Other Topics 64

Lectures and Conferences **68**
- ITCR Program meeting 68
- Portability of Deep Learning Models in Biomedical Applications Workshop 70
- NIH Workshop on Harnessing Artificial Intelligence and Machine Learning 78

to Advance Biomedical Research
Other Lectures 78

Code **81**
SEER Data Test Case 81
Function Optimizer 82
Patient No-Show 86
K-Means Clustering 88
Lymphoma Subtype Classification 90
Function Optimizor Model Data Generator: Part 1 92
Function Optimizor Model Data Generator: Part 2 96
P1B2 Model Data Generator: Part 1 98
P1B2 Model Data Generator - Version 2: Part 1 102
P1B2 Model Data Generator - Version 2: Part 2 105

Link Appendix **109**
Python and Installation 109
Machine Learning 109
CANDLE Benchmarks 111
Biowulf 112
Databases 113
Readings 116
Lectures and Conferences 119

Introduction

We are interns at the National Cancer Institute and have spent the summer working towards understanding and using machine learning on cancer datasets. We are both highschool graduates going into our freshman year of college. Madeleine has a background in physics and engineering and has taken introductory programming classes in Java and C++, but has never worked with machine learning. Catherine has a background in the medical field, including chemistry and biology, with minimal prior programming experience. The following document explains what we have done over the past three month at The National Cancer Institute. It covers everything from the installation of the necessary tools to our final test cases. We have included links to all the sources we have used, all of the code we wrote, summaries of all the articles we read, descriptions of the conferences and lectures we attended, and much more.

If any questions arise feel free to contact:

- Madeleine Birdsell: madeleinebirdsell@gmail.com
- Catherine Warnement: cwarnement@gmail.com
- Dr. Eric Stahlberg: eric.stahlberg@nih.gov

Python and Installation

The Basics

The first step to getting started with machine learning is to download Python. Python 2 and 3 work in similar ways, however there are a few syntactical differences between them. We recommend downloading 3 just because the libraries that will be needed later are formatted differently for the different versions of Python and it is the most up to date. After Python is downloaded, Anaconda is a good next step. Anaconda is a popular program for working with Python and is simple to download. Within Anaconda there are integrated development environments (IDE’s) like “spyder” or “jupyterlab”; “spyder” is what we have chosen to use.

Libraries

Libraries are inventories of different modules that contain variables and functions that can be used to perform different tasks. There are many different libraries that can be downloaded to use in Python and a few are specialized for machine learning. We have come across many different libraries through the examples we have done and working with the CANDLE benchmarks. Most of these libraries can be installed by going to a terminal and typing “conda install **library name**”. If that doesn't work, try “pip install **library name**”. However, some may require more commands and step by step instructions can be found online by searching the library name. For example, Theano provides online documentation for installation on all operating systems. In addition, installation instructions may vary for different operating systems. Below is a list of all the libraries we have installed throughout our work.

- keras
- Tensorflow
- theano
- matplotlib
- openCV
- pandas
- numpy
- scipy
- pyzmq
- data_utils
- tqdm
- hdf5=1.8.17

- git
- h5py
- ipython
- pip

Transferring Computers

In our second week we had to transfer from working on our personal computers to NIH computers. Switching computers after already getting one fully setup for machine learning can be tricky and requires the transfer of many different pieces. Keep in mind that when transferring computers that have different operating systems things will not be downloaded in the same ways. The first step in transferring is to redownload everything that will be needed to program, i.e. python, anaconda, and all the libraries. The next task is to transfer all files that have already been created in python. The easiest way to do this on a mac is to put all of the files in one folder, then view the folder in finder. From there you can share the folder through email to yourself then open it and download it on another computer. The final step is to transfer datasets. The datasets that are used for machine learning contain thousands of data points and are often times too big to send in an email. It is easiest to find them online again and redownload them. Depending on where you transfer your datasets and files too, you may have to change the pathways to them in the programs themselves. For example, if a file used to be opened in python with (‘Users/INSERT_USER/ Desktop/Datasets/my_data’), but got moved to a different folder and my username changed, it would have to be written as (‘Users/INSERT_USER/Desktop/Python/Datasets/my_data’). Also, depending on if the operating systems between computers are different, files may have be called with a different syntax. Mac is as stated before, windows it is ‘C:\\Users\INSERT_USER\desktop\folder\file.extension’.

Machine Learning

Introduction

Below are some websites that cover what machine learning is and how to get started. They go through a few of the Python libraries that will be used and some of what you will need to know to be successful with machine learning.

- 7 Steps to Mastering Machine Learning
- Get Started with Machine Learning Using Python

Datacamp is also a great website if you are looking for step by step tutorials on how machine learning functions work in python. They have courses about deep learning in python, pandas, data visualization, unsupervised learning, and many more topics. However, it does come at a price of $29 monthly.

Model Tutorials

Once everything is installed and you understand the basics of machine learning, you can start following some examples of machine learning models to test if everything installed is working and begin to understand neural networks. Tutorials for how to set up machine learning models can be found easily online, just ensure that you specify that you are using keras or whatever library you prefer. Some examples do not work for every computer/setup, have incorrect data, or require tweaking, but some are great. Ones that we found and used were:

- Linear SVC Machine Learning SVM example with python
- Use Keras Deep Learning Modules with Scikit-Learn in python
- Linear Regression in Theano
- Keras Tutorial: The Ultimate Beginner’s Guide to Deep Learning in Python
- Basics of Image Classification with Keras
- Image Classification Using Convolutional Neural Networks in Keras
- Simple Image Classification Using Convolutional Neural Network - Deep learning in Python
- Building Powerful Image Classification Models Using Very Little Data

Troubleshooting

In the beginning, when starting a new tutorial it would most likely be using software that we did not yet have downloaded. If this occurred, we looked at the error message then googled the software that we needed and figured out how to download it. The only one that we could not successfully install was “caffe”, which was needed for an image colorization program (link not included in examples). Another problem that we ran into quite often was trying to access datasets in python. We discovered that the data has to be traced from wherever python also is saved. In our case, we always have to upload data as (‘/Users/INSERT_USERDesktop/Datasets/training_set’), with the last term being the actual name of the data set or file you are trying to access. Another problem we ran into was labeling graphs of accuracy and loss data. The graph keys were covering the graph but we fixed this problem by adding “loc=4” to the plt.legend to move it. Most problems were solved by first, reading the error messages they produced and then tweaking code and seeing what changes had an effect on the output. Search engines are also a great tool to use when troubleshooting and debugging code. Most commonly, others will have come across the same issues when they were first learning, so there are many great forums and resources online to get your questions answered.

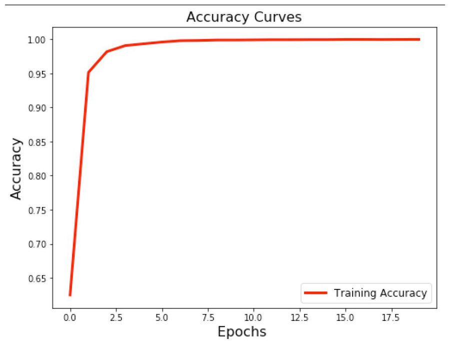
Creating Our Own Models

SEER Data Test Case

The SEER database contains broad datasets on patient outcomes including U.S. mortality and population data. The 1970-2012 expected life table was used to create a model. The first step was to look at the data and format it in a way that it can be transferred into python code. The data dictionary for the dataset was essential in understanding how to interpret the data and decide which data points to use. In order to be read by the code, the data was put into an excel file and split into columns that corresponded to age, sex, race, year, and percentage. The file was then imported and the data was split into testing and training datasets for the X (age, sex, race, year) and Y (percentage) components. The Y value data was cases out of 1,000,000, so in order to get the data into a more manageable percent format, the Y values were all divided by 1,000,000 and rounded to 2 decimal places. The model was then created using 4 inputs and 4 layers and 40,000 training data points. The activation, batch sizes, number of epochs, optimizer, and many other components were changed and tested but the accuracy remained ~50%. However, when the number of datapoints used for training was reduced to only 9,500 the accuracy jumped to 96.67%. The same factors as before(activation, number of epochs, etc.) were adjusted to attempt and bring the accuracy up but little to no improvement was made. One thing that stood out was that all accuracies were exactly 96.69% after the first epoch and did not changed as other factors were adjusted. In an attempt to understand why the predictions were not improving I wrote code that would take parameters from the user and print predictions based on those parameters. However, when I provided the model with inputs every output was the same (95.6%). I then went back to look at the data to see why this may be happening and for my future tests on the code jumped around between different metrics besides just accuracy to evaluate the effectiveness of the model. Returning to the data, I notice that it was organized by age, so when I was taking the first 9,500 data points, I was taking all the youngest cases. So, I switched the training set again to include every 10th data point which made the accuracy drop to ~50%. I then began the process of tweaking different parameters of the model once again and found that nothing was making much of a difference and that the accuracy stopped improving after the 2nd epoch once again. After hours spent on trying to improve the model the conclusion was reached that the dataset may not be the right fit for the model I was attempting to use and hyperparameter optimization capabilities would be useful in solving problems like these in the future. The code can be found on page 81.

Function Optimizer Test Case

Given a function, we were tasked with finding the most optimal model to reproduce the function. The function was "f(x1, x2, x3, x4, x5, x6, x7, x8) = x1*x2 + x2*x3 - x3*x4 + x5*x6 - x7*x8" over range -10 to +10". We created a random number generator to create the dataset and then a model with four layers to train the data. We followed a simple structure that we had used many times in the examples we had done. At first, the



accuracy was very low so we started tweaking characteristics to try and make it better. We first reduced the range of the random number generator to -1 to +1 which greatly increased the accuracy. We then adjusted the activations and optimizers for each layer and found that a combination of activations 'relu' and 'selu' with the 'adam' optimizer produce the best results.

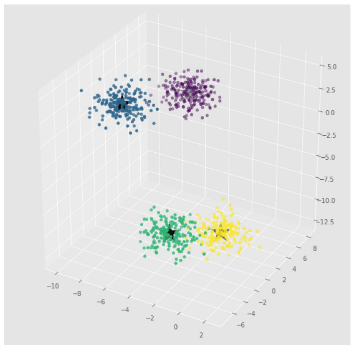
We then discovered that the values the program was outputting only included integers from -1 to 5 even though the function should produce integers from -5 to 5. It turns out that the 'relu' function only works in a range of [0, infinity). Another activation function with a broader range would be needed to accurately reproduce the function. Advanced activations are activations that function with ranges from (-infinity, infinity), making them very useful for this model. After replacing the activations with 'elu', the accuracy was increased to 100%. A graph was created of the accuracy of the model in order to ensure that we were not over/undertraining.

Accuracy is still very low when the range is reverted back to -10 to +10, and despite our best efforts, has not improved much. Much like the SEER data test case, we are unsure of the correct type of model to use to accomplish the goal of evaluating a function from -10 to 10 and hyperparameter optimization capabilities would be useful. The code for this model can be found on page 82.

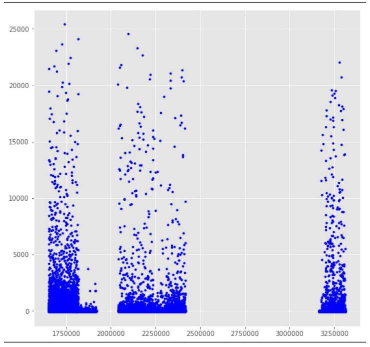
Patient No-Show Test Case

Another data set that we found online was one outlining patient data and whether or not they showed up to their appointment. The first step was to download the dataset, which was already in a compatible .csv format. The data was then imported into the code where it was altered based on the factors needed and changed to a certain naming convention. The neighborhood and appointment day were removed from the model and the gender and no-show data was converted from characters to numbers. A basic model was coded as well as a way to get prediction for a specific patient in the data set. The activation function was the main model parameter that was adjusted in order to get the accuracy up. The accuracy was in the high nineties but the predictions only had a range of 3%. With some tweaking the range was increased to 25% with the accuracy remaining high. The model was generating the likelihood that a patient would show up, not a yes or no answer like was provided in the Y values of the model. Ultimately, the model worked and provided the percentages, but with all the percentages being above 65% it is hard to say what patients would or would not show up. The main takeaway from this test case was the understanding that the real work is in gathering data and figuring out what type model that will be helpful in actually accomplishing the goal instead of just working. The code for this model can be found on page 86.

K Means Clustering



All of the machine learning examples previously discussed have used keras neural network models. Training data sets were used to train and test sets were used to predict and test outcomes. Another type of unsupervised machine learning is K means clustering. This program plots the data points in a scatter plot and then isolates the different clusters by assigning boundaries. Graphing the data is fairly easy but creating the algorithm to assign the centroids is much more difficult. Following a tutorial that used a random number generator, I was successfully able to plot and cluster the data points. See code on page 88.



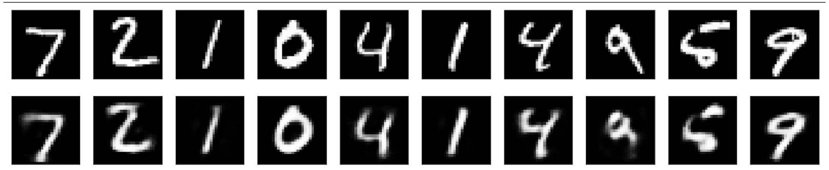
The GEO database contains thousands of datasets that can be used for machine learning. The data that is plotted in the scatterplot to the right is metastatic breast cancer data from sample GSM1599177 and clearly shows three clusters. The x axis is the REF ID number and the y axis is normalized expression. While the clusters do not represent correlated data, they are well defined and will make it easier to evaluate the placement of the centroids. After applying the same centroid algorithm for the previous clustering example, which worked, the graph to the right was given. With each additional run, the centroids moved to random points even though the data was not changing. Testing an alternate algorithm produced a completely empty graph. See code on page 88.

Lymphoma Subtype Classification

We decided to move forward with tissue classification using deep learning so, I attempted to recreate one of the use cases featured in Deep Learning for Digital pathology Image Analysis: A Comprehensive tutorial with Selected use Cases. The article provided a link to the datasets, models, source code, and a tutorial for the implementation. The introduction notes that Caffe must be installed and working in order to run the code. Thus, I attempted once again to download the software. I attempted to follow multiple tutorials however the lines of code in "opencv" that are supposed to be edited are not in the "opencv" file that I have. I looked more alternatives, but after many hours of trying, it was decided to move forward with a different approach. The dataset features over 300 images featuring 3 different lymphoma subtypes (CLL, MCL, and FL) which could be downloaded easily into separate folders for each subtype. I then downloaded the data and began creating code to transform it into a format that would be usable for a model of my own creation. As I began there was an error unrelated to this project. The console was not working properly and there was the following error "AttributeError: type object 'IOLoop' has no attribute 'initialized'". With some googling I discovered that this was due to a package being out of date and installing pyzmq fixed the issue. This tutorial provides an example of how to upload images into python code in a usable format for a machine learning model. I based the code off of the tutorial however many changes had to be made as it was originally for a cat versus dog classifier with different data organization. The folders of images were uploaded and then a list of labels to go along with the images was created (0=CLL, 1=MCL, 2=FL). The three separate lists of labels and collection of images were then group together to form 2 lists, one of all the labels and another or the corresponding images. The two lists were then zipped together in order to be shuffled and then unzipped. Lastly, the dataset was then split into 6 different groups: training images, training labels, testing images, testing labels, validation images, and validation labels. The code can be found on page 90 although it was never fully completed as we moved on to other topics.

Keras Autoencoder Example

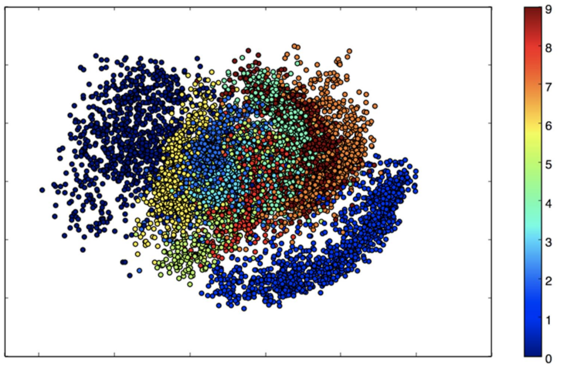
This example walks through how to create a basic autoencoder as well as how autoencoders can be altered to make them more accurate. I followed the code to create a basic autoencoder and was able to recreate the original input data from the MNIST data set.



Things like adding sparsity or adding layers to the autoencoder can increase its accuracy in recreating input data. The hidden layer in the basic autoencoder model learns using principal component analysis. Adding a sparsity constraint on the hidden layer creates results that are twice as sparse as the original model by constraining how many neurons are active at a time. Adding more layers to the model also achieves a lower test loss, making the recreated images look closer to the originals. Convolutional networks are often used for models that train with image data. Adding conv2D and UpSampling2D layers to this autoencoder made the model even more accurate than the autoencoder with the additional layers. Autoencoders can also be used to generate output data despite the input data being noisy. The example first generated synthetic images with a lot of background noise. Then they added more layers to the original basic autoencoder to make it to improve the quality of the reconstructed data. The model was still able to reconstruct the original, clean, images when fed the dirty data.

Variational autoencoders are more modern versions of autoencoders that plot input data in a 2D space. Instead of learning through a function, a variational autoencoder learns by turning input samples into two parameters in latent space and then sampling points. This latent space is two dimensional, meaning that it is possible to graphically show the distribution of the points. In addition to the reconstruction loss function that is used in all the previous autoencoders, KL divergence between the learned latent distribution and previous

```
ImportError: Failed to import pydot. You must install pydot and graphviz
for `pydotprint` to work.
```



distribution is also used.This helps the model learn well-formed latent spaces and prevents overtraining. Sampling point from the latent spaces also enables variational autoencoders to create new data points, making it a generative model. When I tried to run this example in python, I received the following error to the right.

After trying to download pydot and graphviz using the commands “pip install appname” and “conda install appname” in terminal, I googled a different way to install pydot. Every way I tried, the kernel returned that all the packages were up to date but even after restarting Spyder, the code still threw the same error. The output of the code should look like the image to the right. It is a plot of all the numbers arranged by how similar they are.

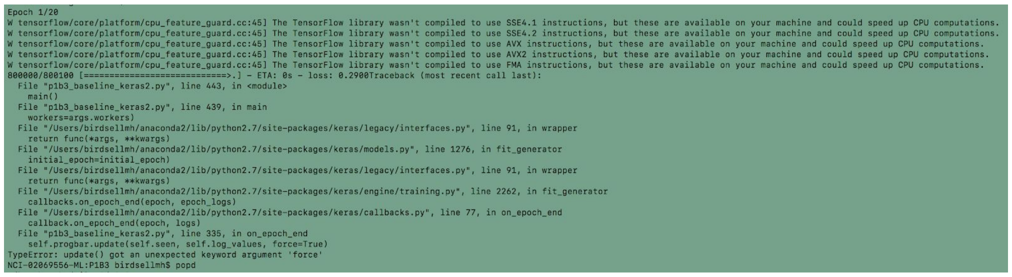
CANDLE Benchmarks

Running CANDLE

Once all set up in python, you can begin to download the CANDLE pilots. Follow the instructions on "README.setup.mac" and download all of the libraries. In the CANDLE instruction, below the documentation to install the libraries, is the code to install the pilots. The benchmarks that have millions of data points will take a very long time, around 20 hours. There are a few errors when installing the pilots:

Pilot 1

Benchmarks 1 and 2 install without any errors, but benchmark 3 does not and gives the following error after running through one epoch:

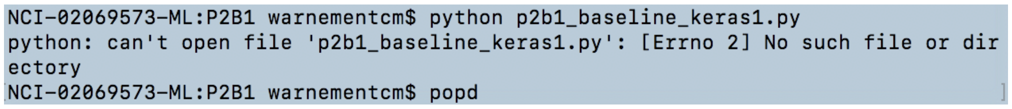


```
Epoch 1/20
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, but these are available on your machine and could speed up CPU computations.
800000/800100 [============================>.] - ETA: 0s - loss: 0.2900Traceback (most recent call last):
  File "p1b3_baseline_keras2.py", line 443, in <module>
    main()
  File "p1b3_baseline_keras2.py", line 439, in main
    workers=args.workers)
  File "/Users/birdsellmh/anaconda2/lib/python2.7/site-packages/keras/legacy/interfaces.py", line 91, in wrapper
    return func(*args, **kwargs)
  File "/Users/birdsellmh/anaconda2/lib/python2.7/site-packages/keras/models.py", line 1276, in fit_generator
    initial_epoch=initial_epoch)
  File "/Users/birdsellmh/anaconda2/lib/python2.7/site-packages/keras/legacy/interfaces.py", line 91, in wrapper
    return func(*args, **kwargs)
  File "/Users/birdsellmh/anaconda2/lib/python2.7/site-packages/keras/engine/training.py", line 2262, in fit_generator
    callbacks.on_epoch_end(epoch, epoch_logs)
  File "/Users/birdsellmh/anaconda2/lib/python2.7/site-packages/keras/callbacks.py", line 77, in on_epoch_end
    callback.on_epoch_end(epoch, logs)
  File "p1b3_baseline_keras2.py", line 335, in on_epoch_end
    self.progbar.update(self.seen, self.log_values, force=True)
TypeError: update() got an unexpected keyword argument 'force'
NCI-02069556-ML:P1B3 birdsellmh$ popd
```

Pilot 2

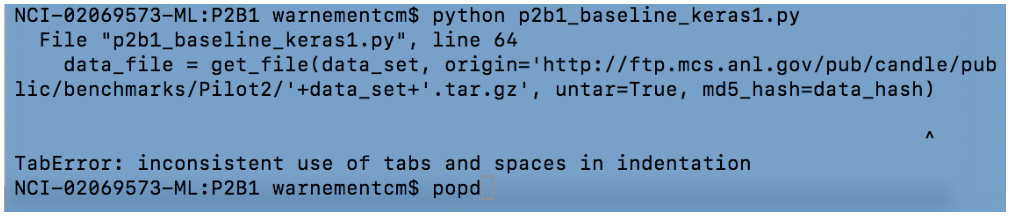
Both benchmarks for pilot 2 create errors during installation.

The following error message occurs for Pilot 2 without altering anything:



```
NCI-02069573-ML:P2B1 warnementcm$ python p2b1_baseline_keras1.py
python: can't open file 'p2b1_baseline_keras1.py': [Errno 2] No such file or dir
ectory
NCI-02069573-ML:P2B1 warnementcm$ popd
```

The following error message for Pilot 2 occurs after changing the file name from keras1 to keras2



```
NCI-02069573-ML:P2B1 warnementcm$ python p2b1_baseline_keras1.py
  File "p2b1_baseline_keras1.py", line 64
    data_file = get_file(data_set, origin='http://ftp.mcs.anl.gov/pub/candle/pub
lic/benchmarks/Pilot2/'+data_set+'.tar.gz', untar=True, md5_hash=data_hash)

                                                                          ^
TabError: inconsistent use of tabs and spaces in indentation
NCI-02069573-ML:P2B1 warnementcm$ popd
```

Pilot 3

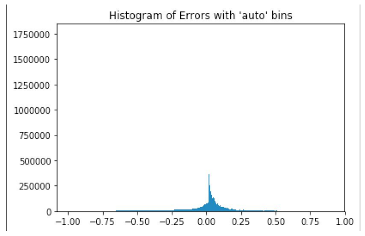
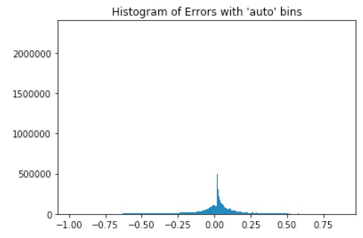
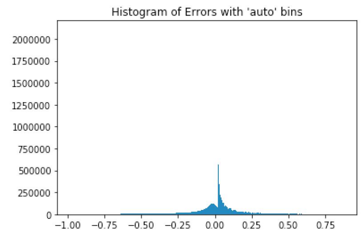
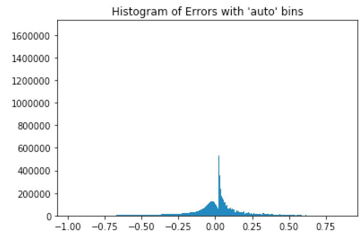
Both benchmarks for pilot 3 install without any errors. The second benchmark contains 20 iterations that each take around 1 hour to complete, so be prepared to wait.

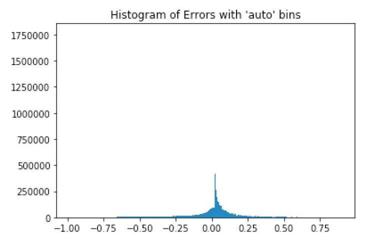
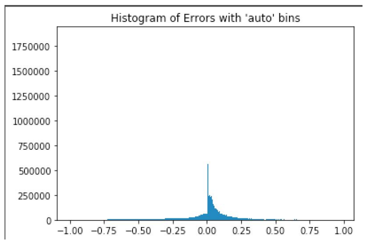
Running all benchmarks run successfully on Biowulf (see Biowulf section for instructions). If Biowulf is not yet installed, read through the “Read Me” files for each Pilot to understand the code and see example outputs.

<u>**Pilot 1 Benchmark 1 Test Case**</u>

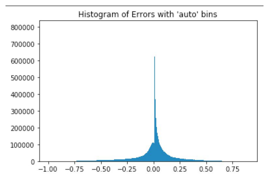
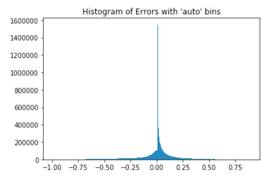
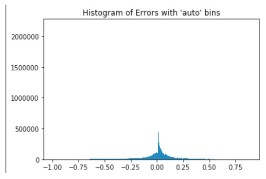
The first benchmark of pilot 1 features an “Autoencoder Compressed Representation for Gene Expression.” The work with this benchmark began with reading over the documentation and becoming familiar was the dataset of RNA sequencing data from the GDC. All components of the code were then reviewed with the goal of understanding the code and getting it running in Spyder in order to run tests and make adjustments (if Biowulf account is set up, running in Biowulf is prefered) The P1B1 folder was stored on the computer from running all the CANDLE benchmarks a couple weeks ago. A couple problems were encountered when trying to get P1B1 running successfully in Spyder. First, when running p1b1.py there was an error that said “cannot find data_utils”, this was solved by entering “pip install data_utils” in the terminal. Next, an error that read, “cannot import name get_file” was encountered. This error did not occur when running the CANDLE benchmark the first time, so the installation guide was followed and repeated and “from data_utils import get_file” was replaced with “from keras.utils.data_utils import get_file” in the code. After the syntax error was solved a runtime error was next. The error “module compiled against API version 0xb but this version of numpy is 0xa” was solved by upgrading numpy by entering “pip install numpy --upgrade” in the terminal. The next step was to run the P1B1 baseline implementation in the terminal with the commands “cd Pilot1/P1B1” and “p1b1_baseline_keras2.py,” this returned an error that directory was not found, but when the command was changed to “cd benchmarks/Pilot1/P1B1” it ran successfully. After the baseline implementation “p1b1_baseline_keras2.py” was run in Spyder successfully. Then, p1b1_4layer.py was run and the error “No module names p1b1” was encountered. This was solved by moving the file out of the Fangfang folder and into the P1B1 folder. After fixing that issue all three files were running successfully and the next step was to read through the code line by line and look up things that were not understood. The code was not commented and approached the making of a model slightly different to what I was seen in the past so comments were added to describe what each line of the code did. After reading through the code it was discovered which elements had to be adjusted in order to test with different levels of reduction. Different values were tested for the 2 layer autoencoder (p1b1_baseline_keras2.py) and the 4 layer autoencoder (p1b1_4layer.py). Each run took anywhere between 10 minutes to a couple hours and the results are recorded in the tables below. Note that these tests are preliminary and a more methodical testing approach will be taken later.

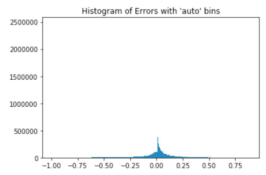
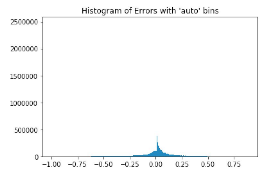
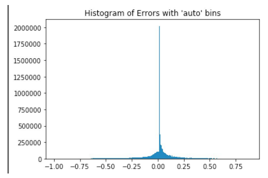
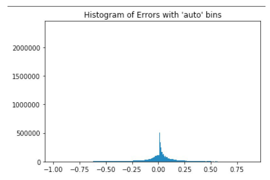
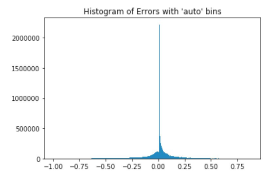
2 Layer (p1b1_baseline_keras2.py)

Size of Input/ Output	**N1: Size of Transition Layer**	**NE: Size of Encoded Layer**	**Loss**	**Val_loss**	**Histogram**
60483	2000	600	.0376	0.0380	
60483	1000	500	.0374	.0377	
60483	2000	100	.0374	.0378	
60483	4000	500	.0378	.0379	

60483	4000	100	.0375	.0378	
60483	4000	1	.0484	.0481	

4 Layer (p1b1_4layer.py)

Size of Input /Output	N1: Size of 1st Transition Layer	N2: Size of 2nd Transition Layer	N3: Size of 3rd Transition Layer	NE: Size of Encoded Layer	Loss	Val_loss	Histogram
60483	1000	500	250	100	.0369	.0371	
60483	500	250	100	50	.0368	.0373	
60483	250	100	50	25	.0368	.0373	

60483	100	50	25	10	.0367	.0372	
60483	50	25	10	5	.0367	.0372	
60483	25	10	5	1	.0366	.0372	
60483	4000	1000	100	10	*Will take approximately 10 hours, will be completed over the weekend		
60483	2000	1000	500	250	.0368	.0373	
60483	4000	2000	1000	500	.0369	.0374	
60483	8000	4000	2000	1000	*Will take approximately 10 hours, will be completed over the weekend		
60483	16000	8000	4000	2000	*Would take approximately 60 hours to run, need to be connected to biowulf before able to run test this large		

This test case has served as a way to become familiar with the way the benchmarks work and how to run them, first in Spyder and then in Biowulf. The above tests were more random testing in order to learn how to make adjustments to the code and read the data that was output. Now, benchmark 1 is being run on Biowulf with more epochs than were used previously (10 vs. 2) and with more methodical changing of parameters. This test will have one independent variable, the smallest bottleneck point. Each test will run with a input/output of 60,483 and dropout layers of 2,000. The original benchmark had a encoded layer of 600 and the tests will be run changing this by increments of 50. Other than the number of epochs and the size of the bottleneck layer the code is the same as what was downloaded off of GitHub. To edit the file in Biowulf the following command must be entered in the terminal:

```
vim benchmarks/Pilot1/P1B1/p1b1_baseline_keras2.py
```

Once the edits are made and saved the test can be run using this command:

```
singularity exec --nv /data/classes/candle/candle-gpu.img python
benchmarks/Pilot1/P1B1/p1b1_baseline_keras2.py
```

The tests were run in an interactive node (follow instructions above in “Interactive Node” section to set up) and each took approximately 5 minutes.

Size of Encoded Layer (NE)	**Loss**
50	0.0366
100	0.0360
150	0.0368
200	0.0369
250	0.0369
300	0.0369
350	0.0369

400	0.0369
450	0.0369
500	0.0369
550	0.0368
600	0.0368
650	0.0367
700	0.0366
750	0.0367
800	0.0366
850	0.0359
900	0.0366
950	0.0362
1,000	0.0367

NT3

NT3 is another of the CANDLE benchmarks. The ultimate goal of the code is to graphically cluster data into two categories, neutral and tumor. The proposed methodology for this is to use the first half of an autoencoder to encode the data. Then, transfer that encoded data to be represented by only two data points. Those two data points are then to be graphed in order to visualize the difference between the tumor data and the neutral data. In order to do this the first step is to get the autoencoder working. Different bottleneck points will need to be tested with regards to the loss to find that best number of points to then be passed on to the next step.

The files that compose the benchmark NT3 are in a different branch than we have previously been working in. To access them we have to switch branches in Biowulf. To do so, first enter the benchmarks folder. We can then use the following command to view all of the branches:

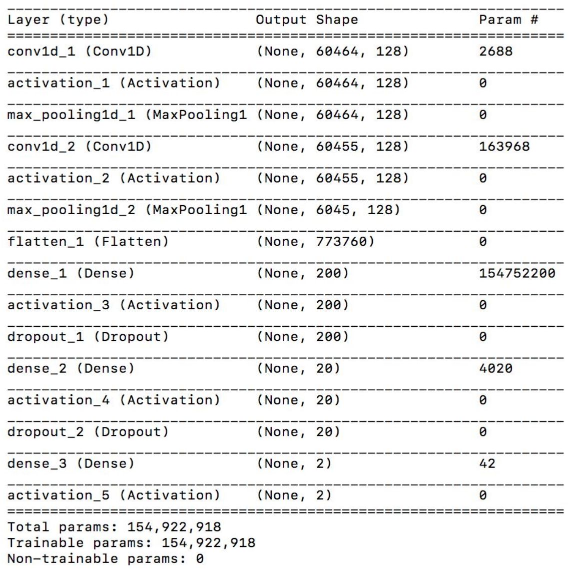
```
git branch -a
```

To get to the NT3 folder we need to be in the frameworks branch. To do this use the following command:

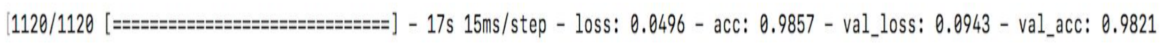
git checkout remotes/origin/frameworks

NT3 can be found inside of Pilot1. Next, to ensure that all the files were working correctly, to get a sample of the time they took, and see an example output the files in NT3 were run. See Biowulf section for instructions on how to set the node up/submit the job.

Nt3_baseline_keras2.py
400 Epoches, run takes approximately 4 hours. Run by submitting job because of length. Code has 15 layers, see image for description. The metrics for the run include loss and accuracy, metrics for final epoch are below.



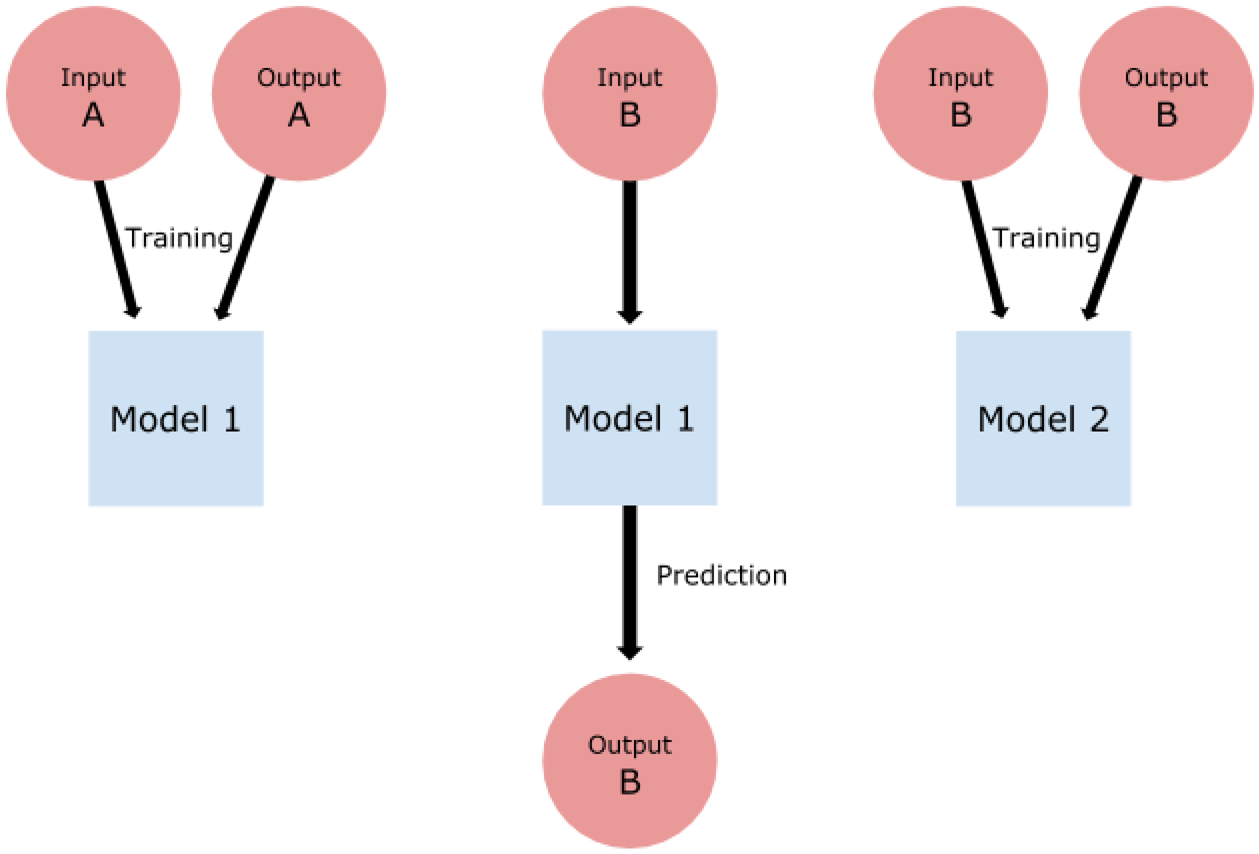
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 60464, 128)        2688
_________________________________________________________________
activation_1 (Activation)    (None, 60464, 128)        0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 60464, 128)        0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 60455, 128)        163968
_________________________________________________________________
activation_2 (Activation)    (None, 60455, 128)        0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 6045, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 773760)            0
_________________________________________________________________
dense_1 (Dense)              (None, 200)               154752200
_________________________________________________________________
activation_3 (Activation)    (None, 200)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                4020
_________________________________________________________________
activation_4 (Activation)    (None, 20)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 42
_________________________________________________________________
activation_5 (Activation)    (None, 2)                 0
=================================================================
Total params: 154,922,918
Trainable params: 154,922,918
Non-trainable params: 0
```



```
1120/1120 [==============================] - 17s 15ms/step - loss: 0.0496 - acc: 0.9857 - val_loss: 0.0943 - val_acc: 0.9821
```

<u>Model for Data Generation</u>

Finding and formatting public data has been one of the most difficult challenges in creating machine learning models in the medical field. On proposed solution to overcoming this is to train a model on real medical data, used fake/altered/noiser medical data to generate a prediction and then use the fake data and the prediction to train another model. This is depicted in the graphic below. “Input A” and “Output A” are real medical data “A” being the variables and “B” being the result. “Input B” depicts data that is of a similar format to “Input A” but is altered to not contain any real medical data. The predictions that the model trained on real medical data (“Model 1”) gives are “Output B”. “Input B” and “Output B” should then be of an identical format to “Input A” and “Output A” except containing no real medical data. “Input B” and “Output B” can then be used to train another model, “Model 2”. After both models are trained data can be given to both models and their predictions, accuracy, and loss can then be compared.

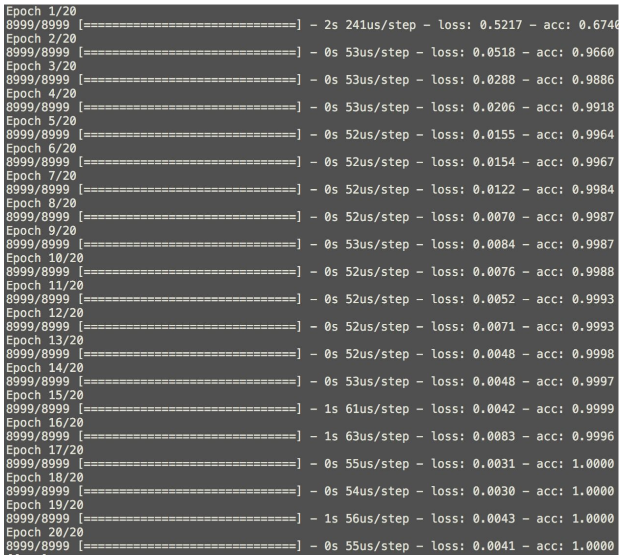


This method can potentially be a way to create models without using any patient data, however testing on the method must be done first. To see the effectiveness of this method models of a format depicted above will be created using the data from Pilot One benchmark of the CANDLE benchmarks. However, first to practice, this method will be applied to the function optimizer model.

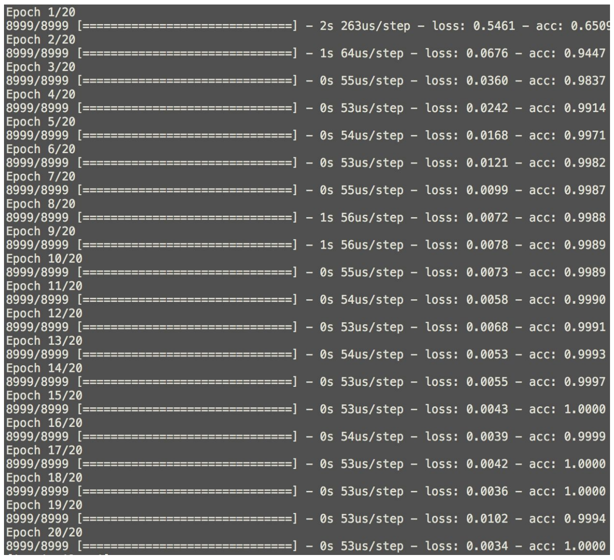
Function Optimizer Alterations

A new goal is to create a model that would be able to generate data with which to train another model (see below for details). To begin learning how to do this a simple model from week 2 was revisited. The function optimizer code takes an input of eight whole numbers between -1 and 1, puts the numbers in a function, and the eight inputs and the output are used to train a model. Once the model is trained it is able to make predictions based on any input of eight numbers within the constraints.

For this project random data was generated by altering the prediction method from previous versions of the code to generate a larger number of input and output pairs (10,000). These pairs were then transferred over and used to train another model. Since the original model was trained on random data and the model had an accuracy near 100%. There was not much difference in the performance of the two models. However, when the rounding feature of the code was removed the accuracy of the 2nd model dropped significantly. The accuracy can be looked into in the weeks to come to figure out why the change is so significant and to adjust the model in a way that will result in a higher accuracy. The code can be found on page 92, part 1 features the original model and the generates the predictions and part 2 receives the predictions and features the 2nd model.



```
Epoch 1/20
8999/8999 [==============================] - 2s 241us/step - loss: 0.5217 - acc: 0.674
Epoch 2/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0518 - acc: 0.9660
Epoch 3/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0288 - acc: 0.9886
Epoch 4/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0206 - acc: 0.9918
Epoch 5/20
8999/8999 [==============================] - 0s 52us/step - loss: 0.0155 - acc: 0.9964
Epoch 6/20
8999/8999 [==============================] - 0s 52us/step - loss: 0.0154 - acc: 0.9967
Epoch 7/20
8999/8999 [==============================] - 0s 52us/step - loss: 0.0122 - acc: 0.9984
Epoch 8/20
8999/8999 [==============================] - 0s 52us/step - loss: 0.0070 - acc: 0.9987
Epoch 9/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0084 - acc: 0.9987
Epoch 10/20
8999/8999 [==============================] - 0s 52us/step - loss: 0.0076 - acc: 0.9988
Epoch 11/20
8999/8999 [==============================] - 0s 52us/step - loss: 0.0052 - acc: 0.9993
Epoch 12/20
8999/8999 [==============================] - 0s 52us/step - loss: 0.0071 - acc: 0.9993
Epoch 13/20
8999/8999 [==============================] - 0s 52us/step - loss: 0.0048 - acc: 0.9998
Epoch 14/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0048 - acc: 0.9997
Epoch 15/20
8999/8999 [==============================] - 1s 61us/step - loss: 0.0042 - acc: 0.9999
Epoch 16/20
8999/8999 [==============================] - 1s 63us/step - loss: 0.0083 - acc: 0.9996
Epoch 17/20
8999/8999 [==============================] - 0s 55us/step - loss: 0.0031 - acc: 1.0000
Epoch 18/20
8999/8999 [==============================] - 0s 54us/step - loss: 0.0030 - acc: 1.0000
Epoch 19/20
8999/8999 [==============================] - 1s 56us/step - loss: 0.0043 - acc: 1.0000
Epoch 20/20
8999/8999 [==============================] - 0s 55us/step - loss: 0.0041 - acc: 1.0000
```



```
Epoch 1/20
8999/8999 [==============================] - 2s 263us/step - loss: 0.5461 - acc: 0.650
Epoch 2/20
8999/8999 [==============================] - 1s 64us/step - loss: 0.0676 - acc: 0.9447
Epoch 3/20
8999/8999 [==============================] - 0s 55us/step - loss: 0.0360 - acc: 0.9837
Epoch 4/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0242 - acc: 0.9914
Epoch 5/20
8999/8999 [==============================] - 0s 54us/step - loss: 0.0168 - acc: 0.9971
Epoch 6/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0121 - acc: 0.9982
Epoch 7/20
8999/8999 [==============================] - 0s 55us/step - loss: 0.0099 - acc: 0.9987
Epoch 8/20
8999/8999 [==============================] - 1s 56us/step - loss: 0.0072 - acc: 0.9988
Epoch 9/20
8999/8999 [==============================] - 1s 56us/step - loss: 0.0078 - acc: 0.9989
Epoch 10/20
8999/8999 [==============================] - 0s 55us/step - loss: 0.0073 - acc: 0.9989
Epoch 11/20
8999/8999 [==============================] - 0s 54us/step - loss: 0.0058 - acc: 0.9990
Epoch 12/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0068 - acc: 0.9991
Epoch 13/20
8999/8999 [==============================] - 0s 54us/step - loss: 0.0053 - acc: 0.9993
Epoch 14/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0055 - acc: 0.9997
Epoch 15/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0043 - acc: 1.0000
Epoch 16/20
8999/8999 [==============================] - 0s 54us/step - loss: 0.0039 - acc: 0.9999
Epoch 17/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0042 - acc: 1.0000
Epoch 18/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0036 - acc: 1.0000
Epoch 19/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0102 - acc: 0.9994
Epoch 20/20
8999/8999 [==============================] - 0s 53us/step - loss: 0.0034 - acc: 1.0000
```

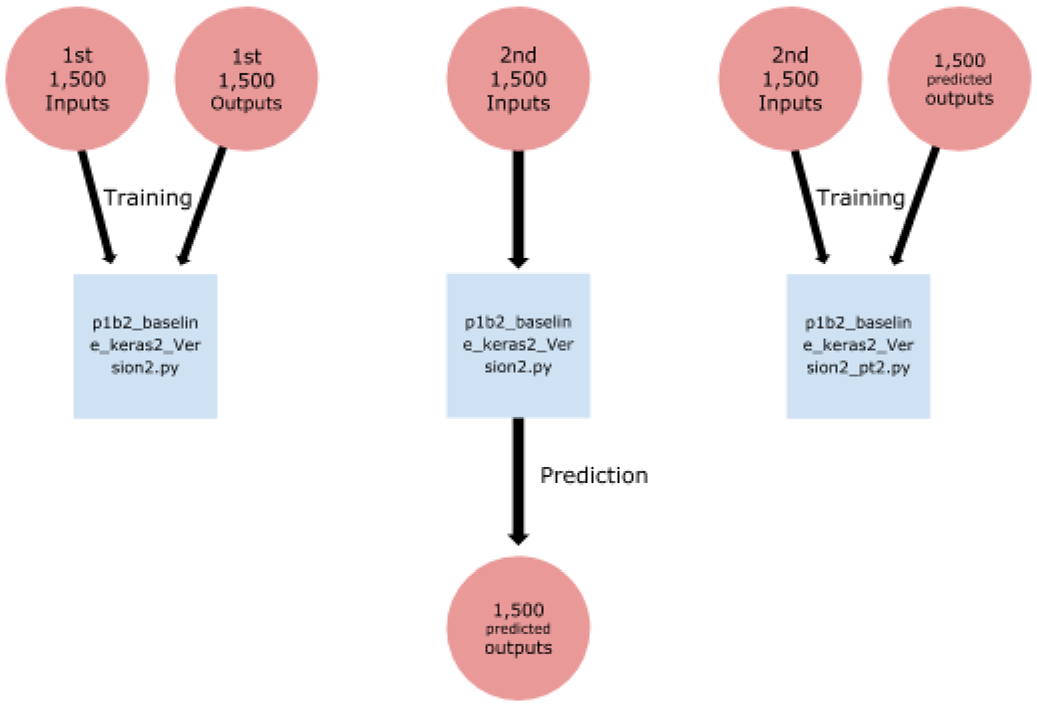
Original Model **2nd Model**

Pilot 1, Baseline 2 Alterations

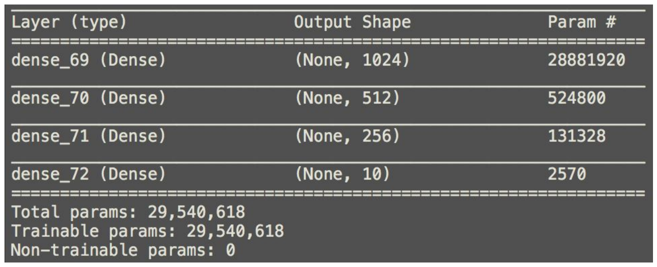
The method discussed above was implemented using Pilot 1, baseline 2 (p1b2) of the CANDLE benchmarks. The benchmark uses 4,000 data points (3,000 training and 1,000 testing). The data is SNP (single nucleotide polymorphisms) data from GDC (Genomic Data Commons) MAF (Mutation Annotation Format) files. You can learn for about the data and formatting at the following links: SNP database, NCI GCD summary, GCD website, and NCI MAF summary. The input dimensions are 28,205 and the output is 10 class probabilities. The raw training data and raw testing data can be found at these links: training and testing. The original model resulted in a best loss (best_val_loss) of 1.31111 and best accuracy (best_val_acc) of 0.59500.

Preliminary Testing

The proposed method requires data in addition to what is used to originally train and test the model. This is to prevent actual patient data from being used. However, for the preliminary testing on p1b2 the dataset of 4,000 data points was split into two data sets, each with 1,500 training data points. Both models were tested on the same 1,000 testing data points. A graphic demonstrating the method is below:



Both models had the same construction as the original p1b2_baseline_keras2.py, the only alterations to the model were the datasets used. The first model (p1b2_baseline_keras2_Version2.py) had a best loss of 1.41667 and best accuracy of 0.55000. The model that was trained on the predicted values (p1b2_baseline_keras2_Version2_pt2.py) had a best loss of 0.13260 and best accuracy of 0.57000. The code for both of the models can be found under "Week 10: Code".



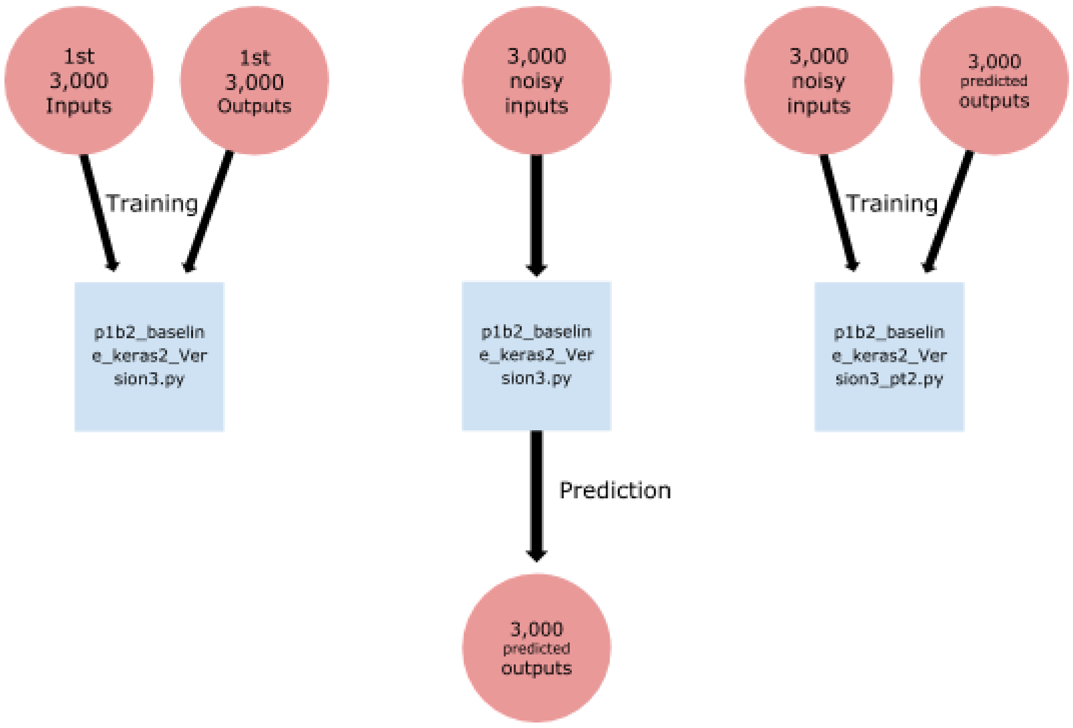
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_69 (Dense)             (None, 1024)              28881920
_________________________________________________________________
dense_70 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_71 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_72 (Dense)             (None, 10)                2570
=================================================================
Total params: 29,540,618
Trainable params: 29,540,618
Non-trainable params: 0
```

Predictions from Noisy Data

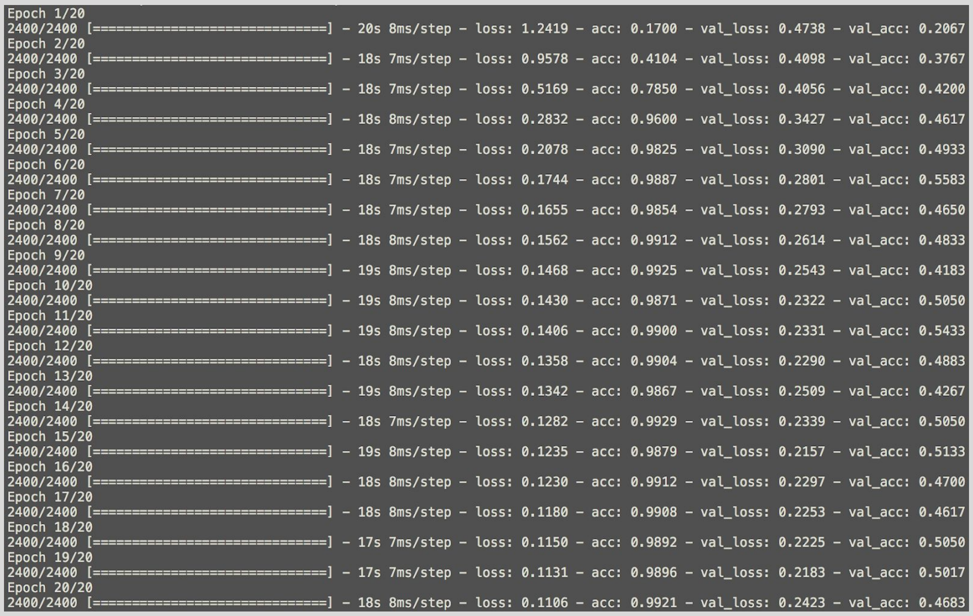
The next test of the model was done by adding noise to the original 3,000 training points to make predictions on and train the second model on. The noise was added using the following lines of code:

```
noise = np.random.normal(0, 0.1, X_train.shape)
x_pred = noise + X_train
```

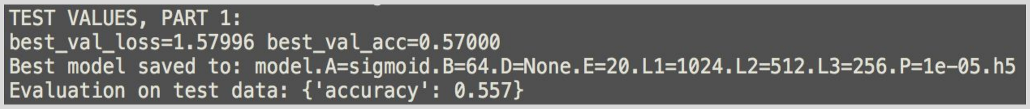
The first number following "np.random.normal" is the "loc" or center of the distribution. The second number is the "scale" or standard deviation of the distribution and the last number is the "size" or output shape. These numbers can be adjusted depending on the dataset and how much noise is desired. The models followed a similar to format to the model used for preliminary testing. The graphic below depicts the data that was used and produced from each model.



Testing was done with the same 1,000 unaltered data points for both models. Below is the evaluation metrics of each epoch, and the results of the testing. The part 1 and part 2 code can be found in "Week 10: Code".

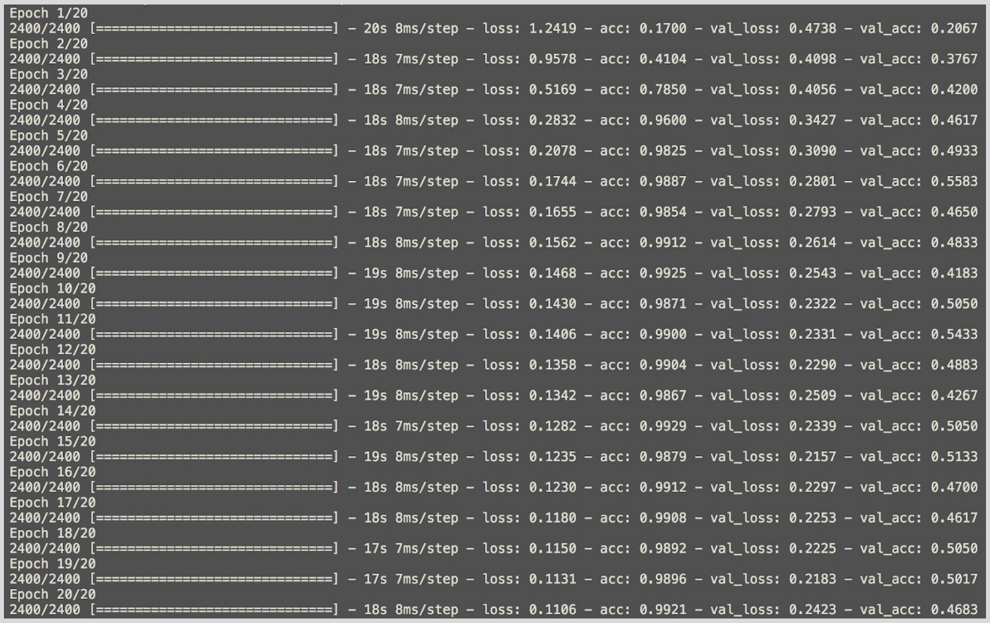

```
Epoch 1/20
2400/2400 [==============================] - 20s 8ms/step - loss: 1.2419 - acc: 0.1700 - val_loss: 0.4738 - val_acc: 0.2067
Epoch 2/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.9578 - acc: 0.4104 - val_loss: 0.4098 - val_acc: 0.3767
Epoch 3/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.5169 - acc: 0.7850 - val_loss: 0.4056 - val_acc: 0.4200
Epoch 4/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.2832 - acc: 0.9600 - val_loss: 0.3427 - val_acc: 0.4617
Epoch 5/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.2078 - acc: 0.9825 - val_loss: 0.3090 - val_acc: 0.4933
Epoch 6/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.1744 - acc: 0.9887 - val_loss: 0.2801 - val_acc: 0.5583
Epoch 7/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.1655 - acc: 0.9854 - val_loss: 0.2793 - val_acc: 0.4650
Epoch 8/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1562 - acc: 0.9912 - val_loss: 0.2614 - val_acc: 0.4833
Epoch 9/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1468 - acc: 0.9925 - val_loss: 0.2543 - val_acc: 0.4183
Epoch 10/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1430 - acc: 0.9871 - val_loss: 0.2322 - val_acc: 0.5050
Epoch 11/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1406 - acc: 0.9900 - val_loss: 0.2331 - val_acc: 0.5433
Epoch 12/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1358 - acc: 0.9904 - val_loss: 0.2290 - val_acc: 0.4883
Epoch 13/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1342 - acc: 0.9867 - val_loss: 0.2509 - val_acc: 0.4267
Epoch 14/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.1282 - acc: 0.9929 - val_loss: 0.2339 - val_acc: 0.5050
Epoch 15/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1235 - acc: 0.9879 - val_loss: 0.2157 - val_acc: 0.5133
Epoch 16/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1230 - acc: 0.9912 - val_loss: 0.2297 - val_acc: 0.4700
Epoch 17/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1180 - acc: 0.9908 - val_loss: 0.2253 - val_acc: 0.4617
Epoch 18/20
2400/2400 [==============================] - 17s 7ms/step - loss: 0.1150 - acc: 0.9892 - val_loss: 0.2225 - val_acc: 0.5050
Epoch 19/20
2400/2400 [==============================] - 17s 7ms/step - loss: 0.1131 - acc: 0.9896 - val_loss: 0.2183 - val_acc: 0.5017
Epoch 20/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1106 - acc: 0.9921 - val_loss: 0.2423 - val_acc: 0.4683
```

Model 1 Training



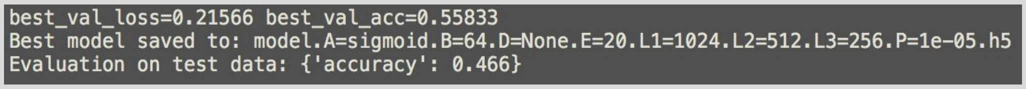
```
TEST VALUES, PART 1:
best_val_loss=1.57996 best_val_acc=0.57000
Best model saved to: model.A=sigmoid.B=64.D=None.E=20.L1=1024.L2=512.L3=256.P=1e-05.h5
Evaluation on test data: {'accuracy': 0.557}
```

Model 1 Testing



```
Epoch 1/20
2400/2400 [==============================] - 20s 8ms/step - loss: 1.2419 - acc: 0.1700 - val_loss: 0.4738 - val_acc: 0.2067
Epoch 2/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.9578 - acc: 0.4104 - val_loss: 0.4098 - val_acc: 0.3767
Epoch 3/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.5169 - acc: 0.7850 - val_loss: 0.4056 - val_acc: 0.4200
Epoch 4/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.2832 - acc: 0.9600 - val_loss: 0.3427 - val_acc: 0.4617
Epoch 5/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.2078 - acc: 0.9825 - val_loss: 0.3090 - val_acc: 0.4933
Epoch 6/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.1744 - acc: 0.9887 - val_loss: 0.2801 - val_acc: 0.5583
Epoch 7/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.1655 - acc: 0.9854 - val_loss: 0.2793 - val_acc: 0.4650
Epoch 8/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1562 - acc: 0.9912 - val_loss: 0.2614 - val_acc: 0.4833
Epoch 9/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1468 - acc: 0.9925 - val_loss: 0.2543 - val_acc: 0.4183
Epoch 10/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1430 - acc: 0.9871 - val_loss: 0.2322 - val_acc: 0.5050
Epoch 11/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1406 - acc: 0.9900 - val_loss: 0.2331 - val_acc: 0.5433
Epoch 12/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1358 - acc: 0.9904 - val_loss: 0.2290 - val_acc: 0.4883
Epoch 13/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1342 - acc: 0.9867 - val_loss: 0.2509 - val_acc: 0.4267
Epoch 14/20
2400/2400 [==============================] - 18s 7ms/step - loss: 0.1282 - acc: 0.9929 - val_loss: 0.2339 - val_acc: 0.5050
Epoch 15/20
2400/2400 [==============================] - 19s 8ms/step - loss: 0.1235 - acc: 0.9879 - val_loss: 0.2157 - val_acc: 0.5133
Epoch 16/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1230 - acc: 0.9912 - val_loss: 0.2297 - val_acc: 0.4700
Epoch 17/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1180 - acc: 0.9908 - val_loss: 0.2253 - val_acc: 0.4617
Epoch 18/20
2400/2400 [==============================] - 17s 7ms/step - loss: 0.1150 - acc: 0.9892 - val_loss: 0.2225 - val_acc: 0.5050
Epoch 19/20
2400/2400 [==============================] - 17s 7ms/step - loss: 0.1131 - acc: 0.9896 - val_loss: 0.2183 - val_acc: 0.5017
Epoch 20/20
2400/2400 [==============================] - 18s 8ms/step - loss: 0.1106 - acc: 0.9921 - val_loss: 0.2423 - val_acc: 0.4683
```

Model 2 Training



```
best_val_loss=0.21566 best_val_acc=0.55833
Best model saved to: model.A=sigmoid.B=64.D=None.E=20.L1=1024.L2=512.L3=256.P=1e-05.h5
Evaluation on test data: {'accuracy': 0.466}
```

Model 2 Testing

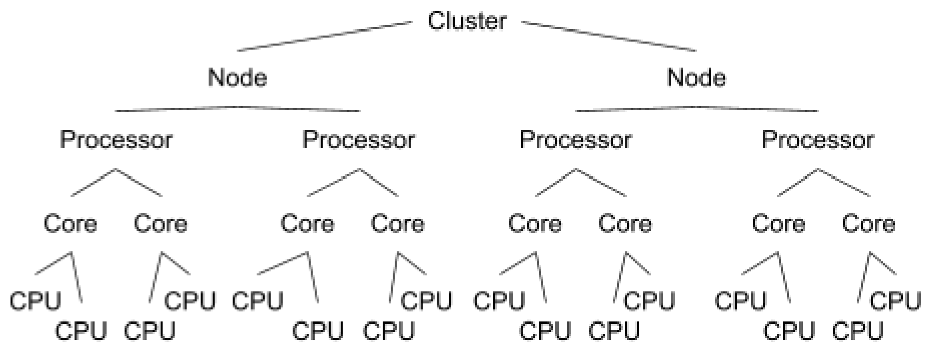
Biowulf

Introduction

Biowulf is a high performance computing system for the NIH community. Users connect to it through the terminal on their desktop or laptop computer and submit jobs to run remotely on the high performance computer, which then get returns the results back to their own computer. It allows users to run programs that require much more computing power than their personal computers are capable of. NIH offers online resources for getting started in Biowulf, including an online course and individual websites for things like modules, and using Python in Biowulf. They also offer shorter and more in depth tutorials for things like connecting to biowulf as well as presentations used in previous classes.

The Computer

The Biowulf computer has 94,000 processor cores, 25 petabytes of storage and supports over 500 applications. The cluster contains nodes, each of which has two processors which have two cores, which have two CPUs. The overall computer structure looks like this (except with many more nodes):



Connecting to Biowulf

In order to get a Biowulf account, you must submit an application and have your PI approve you. Once you have an account, you may log in to Biowulf. There is no sharing of accounts and each account must be renewed each year. The first thing to do when connecting to Biowulf is opening a terminal window and starting a secure shell. This can be done on a mac by typing ssh -X username@biowulf.nih.gov. It will prompt a list of warnings, type 'yes' and it will finish opening the secure shell with in Biowulf. The command ‘logout’ logs the user out of Biowulf.

Submitting Jobs

The default space on the computer alloted is 1 core and 4GB of memory (2 CPUs) but can be modified. The command ‘sbatch --cpus-per-task=# --mem=#g jobscript’ allows users to specify how many CPUs and how many GB of memory they wish to assign to a job. When submitting multiple jobs, the command ‘sbatch --exclusive --cpus-per-task=# jobscript’ ensures that no other jobs will run on that node. Users can also set up job dependencies which will link jobs and only run the second job after the first one has finished. Users can monitor their current jobs and their past jobs using the user dashboard.

Single jobs should be submitted using the sbatch function. This can be done by entering ‘sbatch job.sh’. This will return a string of numbers, which is the job ID. When users submit jobs, they get placed into a queue where they wait until the resources are free to run the job. Using the command ‘squeue -u $USER’ will show the queue of jobs, and ‘ls’ will show the completed jobs. There are many different ways to cancel a job. To cancel a single job, type ‘scancel jobID’, to cancel all jobs by you, type ‘scancel -u username’, and to cancel a job by name, type ‘scancel jobname.sh’. It is impossible to cancel jobs by other users. Set up an interactive session by typing the command ‘sinteractive’ and exit the session with the command ‘exit’.

Submitting a swarm is useful for submitting a large number of similar jobs. A swarm command file is a list of all the subjobs that need to be completed where each line is its own separate subjob. To submit a swarm subjob, use the command ‘swarm -f filename’. There is one job ID, but one number changes for each subjob. There are many ways to customize swarm functions including changing how long they will run for and how much memory they will get. It is also possible to create a swarm bundle which will run jobs back to back after they complete. If one swarm has more than 1000 subjobs the program will automatically bundle it.

Modules

Biowulf provides different programs and applications that are already loaded into modules for users to work with. Using the command ‘module avail’ will print a list of all the modules that are available, including the modules that have multiple versions. To load an application into a module, simply type the command ‘module load (appname)’. Only after an app is loaded can it be run. Unloading an app is just as simple: ‘module unload (appname)’ or ‘module purge’ to unload all the apps at once.

Transferring Data

There are a few ways to transfer data in and out of Biowulf. The first only applies if you own a Mac. Go to ‘Finder’, ‘Go’, ‘Connect to Server’, and type ‘smb://helixdrive.nih.gov/_____’. Fill in the blank with nothing, ‘data’, or the name of a shared data drive and hit connect to get connected to the data drive. From here you can drag and drop files to or from the directory. The second way to transfer data is using a secure copy. In the Biowulf secure shell type the command “scp username@biowulf.nih.gov: /file/path/” which will copy the file to your desktop. To transfer data from the internet, you first have to open an interactive session. Then use the command ‘wget url/filename’ to transfer the data. The final way to transfer data is using Globus, on online data transfer tool. Go to globus.org and login using your NIH login information. Select endpoints for the data transfer and then select which files you wish to transfer. You can also share files with others using email and encrypt files for transfer.

Running CANDLE Benchmarks

The CANDLE benchmarks must be able to be run on Biowulf in order to efficiently test varying chokepoint sizes for pilot one, benchmark one and eventually pilot 2, benchmark one. A description of Biowulf, its various features, and how to setup an account can be found above. There is extra setup and steps that must be done in addition to what was previously described in order to run the CANDLE benchmarks. The steps are as follows:

Initial Setup

First thing is to login to your Biowulf account using the following command line:

```
ssh -X <user>@biowulf.nih.gov
```

You will then be prompted to enter your password. Next, since Biowulf is not connected to the files on your home computer you will need to install the CANDLE benchmarks. You can do so by entering the following command line:

git clone https://github.com/ECP-Candle/benchmarks

You can also copy files from your home directory into Biowulf by using the following command:

scp <filename> <user>@biowulf.nih.gov:/home/<user>

Creating and Using Bash Scripts

A bash script contains a series of commands to be run by the computer. To create a bash script in your home directory open TextEdit which can be found in the applications folder on a Mac. To make the file a bash script the following command should be the first line:

#!/bin/bash

The remainder of the script should be the commands you want completed. Once done click "Format" in the upper left corner and then select "Make Plain Text." You can then save the file (make sure you remember the name and location). Additionally, make sure the file is unlocked, you can do so by right clicking the file and selecting "Get Info." To run the file type the following command:

bash <filename.extension>

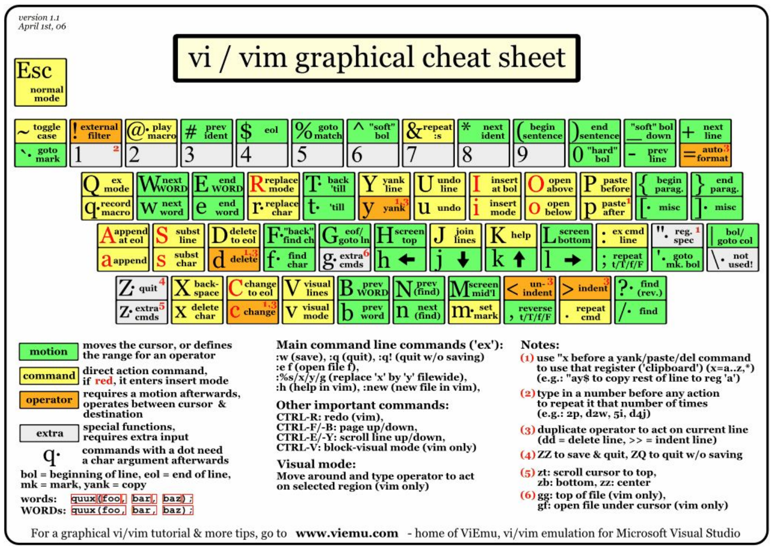
You can then use the last command in the "Initial Setup" section to transfer the script to Biowulf. Alternatively, you can create a bash script using VIM. This method is preferred, especially in Biowulf, to avoid repeatedly copying files. To do this type the following in the terminal.

vim <filename>

The command will open a file, to make this file a bash script include the #!/bin/bash header. See below for how to use VIM to write, save, and run the script.

VIM

VIM is a text editor that will be used when running the CANDLE benchmarks in biowulf. It was created as an efficient way to edit and write files that does not require the use of a mouse. You can follow this tutorial to learn the VIM basics. While you are first learning it is recommended to print out a VIM "cheat sheet" to have for reference. An example is below:



motion: moves the cursor, or defines the range for an operator
command: direct action command, if red, it enters insert mode
operator: requires a motion afterwards, operates between cursor & destination
extra: special functions, requires extra input
q· commands with a dot need a char argument afterwards
bol = beginning of line, eol = end of line, mk = mark, yank = copy
words: quux(foo, bar, baz);
WORDs: quux(foo, bar, baz);

Main command line commands ('ex'):
:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file f),
:%s/x/y/g (replace 'x' by 'y' filewide),
:h (help in vim), :new (new file in vim),

Other important commands:
CTRL-R: redo (vim),
CTRL-F/-B: page up/down,
CTRL-E/-Y: scroll line up/down,
CTRL-V: block-visual mode (vim only)

Visual mode:
Move around and type operator to act on selected region (vim only)

Notes:
(1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*) (e.g.: "ay$ to copy rest of line to reg 'a')
(2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
(3) duplicate operator to act on current line (dd = delete line, >> = indent line)
(4) ZZ to save & quit, ZQ to quit w/o saving
(5) zt: scroll cursor to top, zb: bottom, zz: center
(6) gg: top of file (vim only), gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to **www.viemu.com** - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

Singularity Container

In order to run the CANDLE benchmarks in Biowulf there is certain API's and software that is needed. Rather than installing it all individually a singularity container combines it all together so only one install is necessary. The path for the Singularity container is as follows:

/data/classes/candle/candle-gpu.img

Running CANDLE Benchmarks in Interactive Node

An interactive node is used when you want to be able to run your code interactively as opposed to submitting a file to be executed and run in the background. In order to allocate an interactive node enter the following command:

```
sinteractive --constraint=gpup100 --ntasks=2
        --gres=gpu:p100:1 --mem=20g
```

This command can and should be adjusted according to what you are planning on using the node for. In the command gres tells which GPU and the amount of disk space needed. You can substitute the "p100" with other GPU hardware, p100 and v100 typically perform the best. The "1" following "p100" tells how many GPUs to allocate. "Mem" is the amount of memory. You can make other specifications in the allocation, you can find examples and their descriptions on the "HPC @ NIH" website. Next is to load singularity:

```
ml singularity
```

To create an interactive shell in the container enter the following:

```
singularity shell --nv /data/classes/candle/candle-gpu.img
```

In order to run the benchmark you also need to bind paths. You can use a bashrc to do this. A bashrc script will run automatically every time you log into the node. To create the file enter:

```
vim ~/.bashrc
```

This will open a file in which you should type:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
      . /etc/bashrc
fi

# User specific aliases and functions
```

export SINGULARITY_BINDPATH="/gs3,/gs4,/gs5,/gs6,/gs7,/gs8,/gs9,/gs10/,/gs11,/gpfs,/spin1,/data,/scratch,/fdb,/lscratch"

Now everything is setup to run the benchmarks! To do so enter the following:

singularity exec --nv /data/classes/candle/candle-gpu.img python <path>/<file.extension>

Running CANDLE Benchmarks by Submitting a Job

Submitting jobs is an alternative to the interactive node. It will allow you to put all of your code in a bash file or something similar to be run. This process can be done remotely and you do not have to enter commands or be with your computer while the job is running. To submit a job you first have to create the file to submit. You can do this using a bash file. You can create the file by using the **vim** command and adding the bash header. In order to run the job you must load singularity, bind the paths, and run the desired code all in the bash script. To load singularity insert the following command:

ml singularity

The command to bind the paths:

export SINGULARITY_BINDPATH="/gs3,/gs4,/gs5,/gs6,/gs7,/gs8,/gs9,/gs10/,/gs11,/gpfs,/spin1,/data,/scratch,/fdb,/lscratch"

To run the code:

singularity exec --nv /data/classes/candle/candle-gpu.img python <path>/<file.extension>

After the bash script is created and saved it is time to submit the job. This can be done by writing the following:

sbatch --partition=gpu --gres=lscratch:50,gpu:p100:1 --mem=50g --time=48:00:00 <filename>

Like the interactive node allocation, the specifications can be adjusted. After the above command is enter a number should be returned. This is the job number. In order to monitor all jobs you have submitted you can use the following command:

```
squeue -u $USER
```

In order to view the output of a specific job, similar to what would otherwise be displayed in the terminal, you can use the following command:

```
more slurm-<job_ID_number>.out
```

Another way to view the progress of all submitted jobs is to go to the “User Dashboard” tab of the HPC @ NIH website, login, and go to the job info tab.

Amazon Elastic Compute Cloud (Amazon EC2)

Before we began using Biowulf we looked into other HPCs (High performance computers). The first one we looked into was Amazon EC2. In order to learn about this service we first did preliminary research and reading on Amazon’s own website and other sources around the internet. After understanding the function and the basics of how cloud computing works we created free accounts and went through Amazon’s 10 minute tutorials. For CANDLE we will be using EC2 and S3, so we focused on the tutorials involving those subjects. The tutorials we completed were:

- Launch Linux Virtual Machine (EC2)
- Launch Windows Virtual Machine (EC2) → required Microsoft Remote Desktop download
- Store and Receive File (S3)
- Launch an AWS Deep Learning AMI (EC2)
- Batch Upload Files to the Cloud (S3)

After we completed the tutorials we began looking into the specifics of how machine learning can be applied through the service.

Databases

Introduction/Database Chart

While reading the cancer and machine learning articles, listening to the NIH informatics technology meeting, and creating our own machine learning code databases were always a topic of discussion. The biggest struggle we have faced in out machine learning attempts was to find appropriate data for the models and get that data in a usable format. In our search for data, readings, and the presentations we have seen we have come across many databases and this week began recording them. We created a table that featured the name, summary, data type, link, a list of studies we have read that use the database, and other relevant features, such as how to request the data.

Link	Database	Summary	Data Type/Format	Other	Studies using this database
GEO	Gene Expression Omnibus	International public repository that archives and freely distributes microarray, next-generation sequencing, and other forms of high-throughput functional genomics data submitted by the research community.	DataSets and Profiles (see organization)	Database organization	Gene expression inference with deep learning
SEER	Surveillance, Epidemiology , and End Results Program	U.S. Mortality and Population data publically available, must request access to other data	.csv files, available in binary or Ascii table values	Data dictionaries for public data or text descriptions for private data	Breast Cancer Data Analysis for Survivability Studies and Prediction
SRA	Sequence Read Archive	Source: DNA, RNA Type: Exome, Genome	See example	Linked to related GEO, BioSample,	

				and BioProject	
wwPDB	Worldwide Protein Data Bank	Single protein data bank archive of macromolecular structural data that is freely and publicly available to the global community	PDB and mmCIF format	Each wwPDB site provides its own view of the primary data (RCSB PDB, PDBe, and PDBj)	
ECO	Evaluation of Cancer Outcomes	Regional records information on demographics, primary tumour and metastatic tumors, cancer stage, tumour size, lymph nodes, breast tumour-specific information, treatment type, outcomes, and recurrence information	Unknown	Must request data through VCR (Victorian Cancer Registry)	Machine-learning prediction of cancer survival
TCIA	The Cancer Imaging Archive	Archive of medical images of tumors (MRI, CT, etc.)	Primary Format: DICOM	Most all data is available for public download (Data Usage Policies and Restrictions)	Deep Learning Convolutional Neural Networks Accurately Classify Genetic Mutations in Gliomas
TCGA	Cancer Genome Atlas	Search, submit, and download cancer informatics data	Varies depending on project. Can sort by data type, format, category, platform, access, etc.	Can be found through GDC (Genomic data Commons)	Deep Learning Convolutional Neural Networks Accurately Classify Genetic Mutations in Gliomas
WHO	World Health	Health related			

	Organization	statistics to monitor progress of the Sustainable Development Goals including mortality, disease, drug control and abuse, and health care.			
COSMIC	Catalog of Somatic Mutations In Cancer	Search cancer tissue, or genomes. Each gene has the sequences for it, information about drug resistance, the shape of it, and how often it gets mutated	Can export some data as .csv or .tsv files, others are text boxes (gene sequences)		Classification of Cancer Primary Sites Using Machine Learning and Somatic Mutations
ICGC	International Cancer Genome Consortium	Lets users view and download a variety of different cancer data types. Data is released quarterly so it is always relatively up to date	Different data comes in different formats. Includes xlxs, BAM, and VCF.		
CCLE	Cancer Cell Line Encyclopedia	View cell lines and their mutations. It lists the gene mutation, the protein that changed, what type of mutation it is, like a deletion or insertion, and which allele it is associated with.	Excel files	Have to register to download datasets	
MIAS	Mammographic Image Analysis Society	This is a central website that provides links to other digital mammogram	Most of the images are in JPEG format	A few of the links are just overviews of projects	Detection of microcalcification in digitized mammograms with

		databases and projects. There are five current projects and two older projects.		using mammogram image data, not links to the databases themselves.	multistable cellular neural networks using a new image enhancement method: automated lesion intensity enhancer (ALIE)
TCPA	The Cancer Proteome Atlas	A data portal that was made to provide researchers with a more user friendly way to access, visualize, and analyze proteomic data	All of the files are zipped Specific data type unknown	Must enable javascript in browser to view	
cBioPortal	cBioPortal	Visualization, analysis, and download of large-scale cancer genomic data sets.	tsv		
GDC Data Portal	NCI Genomic Data Commons Data Portal	A unified data portal for cancer genomic data to support the development of precision medicine	VCF		

Specific Data Repositories

TCGA

The TCGA is The Cancer Genome Atlas and contains The Cancer Digital Slide Archive. Data from the TCGA has been used in many of the machine learning and pathology articles that we have read and has proven to be a good resource for pathological data. The data on the TCGA is in JSON format, which is JavaScript Object Notation. JSON data can be used in python, but not in the way that we would like to. The JSON data downloads as text that classifies the case ID, diagnosis, project ID, and other information that is not the actual slide image. For example:

```
e7a1cbe2-793c-4747-8412-8be794f2382b      TCGA-G7-6790
TCGA-KIRP male  1950  white not hispanic or latino    --      not
reported    not reported      C64.1stage i      21169      alive
8260/3      --      --      --      not reported      C64.9-21169.0
not reported      not reported      C64.91575.0
        --      --      --
```

When the images is added to the cart, there is an option for the data to be downloaded as an image. However, the image is not of the slide section, it is of the QR code that is used to identify the slide. While the actual image of the slide can be viewed on the digital cancer archive, there seems to be no way to download the image to use for training a model.

TCIA

The Cancer Imaging Archive is another online database where images of cancer radiology are stored. To view this data users have to download the TCIA Data app, the 3.0 version works best. To download data, users can search the repository and then add whatever data they want to their cart. The data can then be downloaded from the cart to the computer. The images will be downloaded in the DICOM format.

DICOM images can be used in python with the pydicom module, which can be downloaded by using the command 'pip install pydicom' in your computers terminal. I followed this tutorial to work with the DICOM data in python. However, I ran into trouble when importing the pydicom module into my program. Occasionally, programs will not work with the older version of the 'pydicom' module, which was just called 'dicom'. I installed both 'dicom' and 'pydicom' but I repeatedly get an error saying neither can be found. I tried to install pydicom with another method but it still could not be found.

Data for Models

CANDLE P1B2

CANDLE pilot 1 uses gene expression data to build autoencoders. Using biowulf, we have been exploring optimizing the chokepoints of the P1B1 autoencoder. Moving forward, we hope to be able to create two models: one that will be trained on a set of data, and a second that will be trained on a similar set of noisier data in order to verify the first model (see “Model for Data Generation” in CANDLE benchmarks section).

A resource that we can use to get data sets similar to those used in P1B1 is the 1000 Genome project. This is an online database of real human data, organized into three different pilots. This database is continually updated by phasing in more individuals and populations. They provide a data portal which allows you to filter through data by population, analysis group, and which project the data was collected under. It also allows you to view the gender of the individual their unique sample ID number. For each individual, you can access their data from each different phase that they were a part of. For example, the individual HG00119 has been a part of the project since Phase 1, and has data sets for Geuvadis, Phase 1, Phase 3, and GRCh38. From these datasets, anyone is able to access all different analysis groups including exomes, low coverage whole genome sequences (WGS), integrated variant call sets, and HD genotype chips. Exomes are a specific part of the genome that only includes the exons, which are the parts that of RNA that code for proteins and will remain after it is transcribed. Low coverage WGS is a form of analyzing genomes for variants and is a very popular way of extracting information from genomes. The higher the coverage, the more specific information is being gathered from the genome. This also means that it takes longer and more layers of analyses. Variant calling is when the variants found in the sequences are identified and recorded in a separate document. These documents are usually in VCF format.

Users can also view different data types including alignments, sequences, and variants. Alignments are when genome data is organized so that it can easily be compared to other genomes. Individual alignments can be downloaded directly from the website data portal, but cation, downloading a single dataset takes hours. Another option is to download the entire International Genome Sample Resource (IGSR) through a portal that is created on your computer. Using this link, a server will be opened up and added to your desktop. Users can log in for free using a guest login which will give them access to all of the data files. Data is organized by population and then by individual. Each individual has folders for exome alignment data and plain alignment data. This data will be very helpful in the future when creating models similar to P1B1. We will continue to explore the data provided on this database and search for other similar databases for additional resources.

FASTQ

All of the data in the 1000 Genome database is in FASTQ format. FASTQ is a text based format that stores biological sequence data and its quality scores. Both pieces of information are coded as a single ASCII character to be as brief as possible. FASTQ files consist of multiple four line sequences:

Line 1	@ (identifier and description)
Line 2	sequence
Line 3	+ (same identifier and description)
Line 4	quality values for line 2

The quality values are the probabilities that the nucleotide listed is the actual nucleotide that is in the sequence. After obtaining the raw data, it is best to perform an overall quality control check on it. A popular program for doing so is FastQC, which is a downloadable program that scans the data and alerts the user to any problems that may cause issues with analysis. After importing your data, the program will assess it and provide a full graphical report that can be exported as its own individual HTML site for future reference. In some areas where the quality is subpar, it is best to trim the data and remove those bad areas. This will increase the overall quality of the data.

FASTA

FASTA format is very similar to FASTQ format except that it is used to represent nucleotide sequences instead of genomic data. This data only uses the first line of each sequence as a header or identifier, followed by the entire nucleic acid code.

Biopython

The Biopython package is needed in order to open FASTQ files in python. Included in Biopython is Bio.SeqIO which is an interface for python that allows users to work with sequence data, like FASTQ. Installing Biopython is free and easy, just type "pip install biopython" into the terminal. There are many different ways you can analyze the code in python, some are outlined here. To simply print the data into the kernel, use the following code:

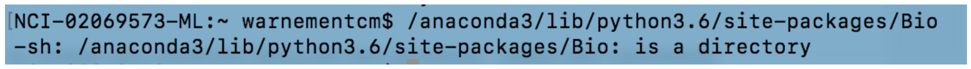
```
from Bio import SeqIO
For record in SeqIO.parse('file_name.fastq','fastq'):
print "%s %s" % (record.id, record.seq)
```

The biopython website also walks through how to iterate through records, index them, change the file formats, and filter through the data.

This Bio.SeqIO interface was created to be similar to BioPerl's Bio:SeqIO, a more advanced system for processing sequence data. BioPerl is an open source resource for developers to create and share bioinformatics tools in the Perl programming language. The Bioperl website also has many tutorials and examples that demonstrate all of the available functions as explanations for handling a few errors.

Pip Install

I am currently unable to install any modules through the terminal. Using the command "pip install biopython" does not work on my computer but does work on others. After trying multiple times to install biopython on my computer using "pip install biopython", "brew install biopython", and "conda install biopython", I searched through my directories to see if I could find Bio. It is installed and was in the correct folder.



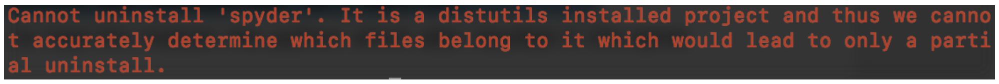
```
[NCI-02069573-ML:~ warnementcm$ /anaconda3/lib/python3.6/site-packages/Bio
-sh: /anaconda3/lib/python3.6/site-packages/Bio: is a directory
```

I also tried other ways to load the module into python which also resulted in the same error (see code below).

```
import sys
sys.path.append('/usr/local/lib/python3.4/site-packages')
from Bio.Seq import Seq
```

Maddie ran into a similar issue earlier and suggested that I update numpy by using "pip install numpy -- update". My numpy was out of data but updating it did not get rid of the error.

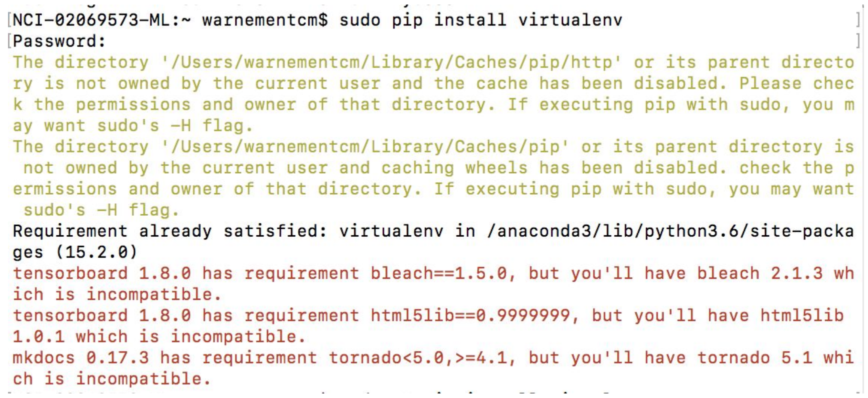
In addition, I tried to uninstall and reinstall biopython, anaconda, and spyder. Reinstalling biopython did not fix the problem. I then updated anaconda from 4.4.10 to 4.5.8 using "conda update --all", which also did not fix the problem. When I



```
Cannot uninstall 'spyder'. It is a distutils installed project and thus we canno
t accurately determine which files belong to it which would lead to only a parti
al uninstall.
```

tried to update spyder from 3.1.2 to 3.3.0 using "pip install --upgrade spyder", I got the following error:

I copied this error into google and found a source that I suggested I open a virtual environment in order to install and



```
NCI-02069573-ML:~ warnementcm$ sudo pip install virtualenv
Password:
The directory '/Users/warnementcm/Library/Caches/pip/http' or its parent directo
ry is not owned by the current user and the cache has been disabled. Please chec
k the permissions and owner of that directory. If executing pip with sudo, you m
ay want sudo's -H flag.
The directory '/Users/warnementcm/Library/Caches/pip' or its parent directory is
 not owned by the current user and caching wheels has been disabled. check the p
ermissions and owner of that directory. If executing pip with sudo, you may want
 sudo's -H flag.
Requirement already satisfied: virtualenv in /anaconda3/lib/python3.6/site-packa
ges (15.2.0)
tensorboard 1.8.0 has requirement bleach==1.5.0, but you'll have bleach 2.1.3 wh
ich is incompatible.
tensorboard 1.8.0 has requirement html5lib==0.9999999, but you'll have html5lib
1.0.1 which is incompatible.
mkdocs 0.17.3 has requirement tornado<5.0,>=4.1, but you'll have tornado 5.1 whi
ch is incompatible.
```

uninstall libraries without interfering with packages. However, when I tried to open a virtual environment, I got the following error:

The last thing that I checked was to make sure that I had the latest version of pip installed, 10.0.1, using "pip --version". My pip is the most up to date version currently. I am continuing to search other ways to fix the above errors but have not had much luck.

Annotated Pathology Slide Data

After reading articles about models using whole slide images to train, I learned that the data must be annotated for the model to be able to learn from it. Many of these articles used to data from online databases like TCIA or TCGA but I was unable to find any annotated slides in these repositories. Many articles also have professional pathologists annotate the slides for them before they train them, which is not an option for us seeing as we are not professional pathologists.

The Digital Pathology Association published a website titled the Whole Slide Imaging Repository which contains a list of many different resources for obtaining whole slide images. They offer a wide ranges of images from still images and whole slide images, to oil immersion and immunofluorescence. Some of the sources require logins but many allow you to create free logins. There are a only a few that require you to be affiliated with the company in order to gain access. The data sets that I explored are listed below:

- Iowa virtual slide box
  - In order to gain access to this data, I had to reach out to Dr. Fred Dee, the creator of the data repository. His contact information is listed on the site. This data set contains about 1000 slides and is intended for teaching purposes. Dr. Dee responded saying that I must be an educator with the intention of distributing the data to students using a password protected server in order to gain access to the data. If I did gain access to the data, the software biolucida would be required to view the data.
- NYU virtual microscope
  - This source allows the public to explore a few smaller datasets. However, to view the entire collection of slides or to be able to annotate and manipulate the slides, a login is required. Logins are only granted to those with NYU School of Medicine accounts.
- PathPresenter
  - Users are able to view individual slides and collections without creating a login. The website links to many different collections from other countries as well, mainly Germany. Some of the data sets are available in English while others are only available in German.
- Pathorama
  - This site requires users to create a free login to gain access to the data. Users are allowed to view slides that are labeled with the pathology type, for example, Atypical duct hyperplasia, and annotate them. However, the slides to not come pre annotated.

Many of theses resources are more aimed at education and giving people practice annotating pathology slides. Most of the sources provide slides with the diagnosis and also the ability to annotate the slides yourself, however, they lack the annotations for the diagnosis.

There were two other sites that I found that contained whole slide pathological data: Histowiz and Brain Tumor Segmentation, but they were not good resources. Histowiz makes users pay to view the slides and annotate them and the Brain Tumor Segmentation website would not load. These websites might have very useful data but they are not available for public use. In our experiences, it has been very difficult to find public data resources that provide quality data in large quantities

Readings

Over the course of our summer we came across and read many scholarly articles while learning about machine learning, databases, cancer research, etc. We searched the PubMed database to find articles that were related to machine learning, AI, cancer research, and any other topic we were interested in. Some of the first research papers we read were studies on how machine learning can be applied to cancer data. These sources highlighted what problems still need solved in cancer research and are helped us in discovering what specific areas we wanted to focus on in our own project. Additionally, we took note of how machine learning can be applied to problems in cancer, for example what models they used, what their comparison metrics were, how they gathered their data, what problems did they come across, etc. Next, we continued our research and narrowed our focus to machine learning used specifically for medical imaging. Lasty, as we were working with the autoencoders in the CANDLE benchmarks we sought out examples of other researchers using autoencoders for medical data. Along the way we encountered other papers that informed the work we were doing more indirectly, these can be found under "Other Topics"

<u>Machine Learning for Cancer Data</u>

Applications of Machine Learning in Cancer Prediction and Prognosis

This article uses machine learning for cancer prediction and prognosis. Previously, machine learning algorithms were primarily used to detect and diagnosis cancer, but there has been a recent shift in using AI for predictive purposes. There are three main focuses of cancer prediction and prognosis:

1. Prediction of cancer susceptibility
2. Prediction of cancer recurrence
3. Prediction of cancer survivability

This study collects and analyzes many published research articles on pubmed featuring keywords like "cancer and machine learning", "cancer prediction and machine learning", "cancer prognosis and machine learning", and "cancer risk assessment and machine learning". The researchers looked at the different machine learning methods, the types of training data, the kinds of endpoint predictions, the varieties of cancer being studied, and the overall performance of the methods. The study finds that the most common machine learning methods for cancer prediction were supervised learning methods. These include artificial neural networks (ANNs), decisions grees (DTs), genetic algorithms (GAs), linear discriminant analysis (LDA), and k-nearest neighbor algorithms prognosis.

Machine-learning Prediction of Cancer Survival

This is an Australian study featuring data from the ECO and EAR databases. The goal of the study is to create a model to predict the survival rates of cancer patients. The study compares four sets of predictions: predictions from a model trained on just the ECO data, predictions from a model trained just the EAR data, predictions from a model trained on both databases, and predictions from a clinician panel. The models are support vector machines and details of their composition can be found in the "Comparing predictions by machine-learning models and clinician" section of the paper. The study finds that the model trained on both datasets preformed better than the models trained on the individual sets and the predictions of the clinician panel.

Assessing Breast Cancer Risk with an Artificial Neural Network

This study uses an artificial neural network to distinguish between benign and malignant tumors and predict the probability of breast cancer in a patient. The study finds that machine learning models are helpful tools for informing medical decisions. Mammography data and clinical/ demographic data collected from Shahid Motahari breast clinic was used to train the model. The data was divided into tenths, one tenth is withheld to test the model with later, the nine remaining tenths are used to train the data in batches. Using an artificial neural network with a back propagation algorithm, the model achieved 95.5% accuracy. The study shows that machine learning algorithms similar to this one can be used to help make important medical determinations. For example, to predict the probability of tumors and to increase the positive predictive value (PPV) of doing a biopsy. However, while they can be great tools to help calculate accurate probabilities of breast cancer, they should not be used as the sole determining factor for diagnosing or treating cancer.

Instead, they should be used alongside doctors' research and expertise to make more accurate predictions in less time.

Predictive Models for Breast Cancer Susceptibility from Multiple Single Nucleotide Polymorphisms

This article (http://clincancerres.aacrjournals.org/content/10/8/2725.long) uses a support vector machine (SVM), a form of unsupervised learning, to predict non-familial breast cancer. The researchers use single nucleotide polymorphisms (SNPs) as markers of breast cancer. There were over 3 million SNPs identified but only 98 are used in this study. Information on the SNPs can be obtained from the Human Genome Variability Database. The SNPs are given a code of 1, 2, or 3 that classifies them based on their alleles. This also allows the data to be used in machine learning algorithms. Patient data was collected from those with cancer and control data from those without cancer and used to train and test three different machine learning models. Naive Bayes is a method that naturally handles missing data, so it was the first method used. It produces an accuracy of 63 +/- 2%. Support vector machines, SVMs, do not handle missing data so, the data set was modified to remove patients who had missing data. It produces an accuracy of 69 +/- 4%. The last machine learning algorithm tested was a decision tree, which only examines one single feature of the model at a time. It produces an accuracy of 68 +/- 1%. Overall, all three of the methods perform similarly when predicting if a patient had cancer or not, but the decision tree produces a more even distribution of errors between cancerous and noncancerous patients.

Machine Learning Approaches for Predicting Radiation Therapy Outcomes: A Clinician's Perspective

This paper highlights the gap between the makers of machine learning models and the clinicians that deal with the problems that these tools are trying to solve. Thus, the goal of the paper is to bridge the gap and educate clinicians about machine learning methods and the principles of machine learning as well as provide examples of studies where the methods are used for radiation oncology. Machine learning is used for image processing, tumor motion localization, survival analysis, and normal tissue toxicity in the field of radiation oncology. However, the paper states the best use of machine learning in the field is in NTCP (normal tissue complication probability). The paper then goes on to describe what the researchers think are the seven principles of modeling:

1. Consider both dosimetric and non-dosimetric predictors
2. Manually curate predictors before automated analysis
3. Select a method for automated predictor selection
4. Consider how predictor multicollinearity is affecting the model
5. Correctly use cross-validation to improve prediction performance and generalization to external data
6. Provide model generalizability with external data sets when possible
7. Assess multiple models and compare results with established models

The paper then describes and provides examples of specific machine learning methods. The first method discussed is logistic regression. The authors describe it as a binary classification and "ideal for clinical questions where there are a relatively few, unrelated predictors" with the goal to "create a linear 'decision boundary,' that is 1 dimension lower than the data set." The paper then discusses the 2006 paper, Multivariable modeling of radiotherapy outcomes, including dose–volume and clinical factors as an example of linear regression used in radiation oncology. Next is the support vector machine (SVM). SVM's are described as another binary classification with the "ability to find complex patterns that are nonlinear through the use of 'similarity functions' that alter the definition of how data points are compared to each other." The paper used to demonstrate SVM's is Investigation of the support vector machine algorithm to predict lung radiation-induced pneumonitis. Lastly, artificial neural networks (ANN) are discussed. ANN's are described as being able to learn nonlinear data and as having support for multi-class classification. The paper gives a brief description of neurons, weights, hidden layers, and deep learning (which had not yet been used for radiation therapy modeling when the paper was published in 2015). Use of artificial neural networks to predict biological outcomes for patients receiving radical radiotherapy of the prostate is the paper used in demonstrating ANN in radiation oncology.

Machine Learning Algorithms for Outcome Prediction in (Chemo)Radiotherapy: An Empirical Comparison of Classifier

This paper tests a data set using many different classifiers using the same dataset. The classifiers used are: decision tree, random forest, neural network, support vector machine, elastic net logistic regression, and LogitBoost. 12 datasets were used, totaling 3,496 patients from studies regarding post-(chemo)radiotherapy toxicity, survival, tumor control, etc. The study runs tests of all six classifiers on all 12 datasets. It finds that on half of the datasets random forest yielded the best results and on four of the datasets elastic net logistic regression is the best.

Machine Learning for Biomarker Identification in Cancer Research

This article is a review of current work in predictive oncology and machine learning that highlights the P4 healthcare concept (predictive, personalized, preventive, and participatory). The author describes the major machine learning algorithms, artificial neural networks, support vector machines, decision trees, and random forest, and surveys articles utilizing these algorithms to determine the most popular methods for classifying data through machine learning. Breast cancer is the most studied type of cancer and support vector machines are the most utilized predictive algorithm. Despite a majority of machine learning cases being on breast cancer using SVM algorithms, there is still a high amount of variability in the way that data is reported, making it difficult to analyze all the data available. The solution to this problem that is proposed is to create an overarching set of rules that must be followed by researchers regarding data cleaning and preprocessing before publication. This would make all data available to be used and analyzed by everyone, which is an important to accelerate developments in precision oncology. The article mentions that cancer needs to be classified on a molecular level rather than a phenotypic level, but in order to do this, more data needs to be available.

Classification of Cervical Cancer Using Artificial Neural Networks

This paper discusses using neural network to classify cervical cells as normal, abnormal, or cancerous. The old methods of classification include having a pathologist look at cell samples and liquid cytology based (LCB). Both of these methods are said to have issues. For the first method there are few pathologist trained to classify the cells and the classification can take a long time and is prone to human error due to complicated cell structure and the overlapping of cells. As for LCB, it is said to have low accuracy. As a result of the imperfections of these methods, a machine learning method of classification is proposed. Many different methodologies are tested including a multi-layer perceptron (MLP), cascaded multi-layer perceptron, a hybrid multi-layer perceptron, back propagation neural network, gene expression, radial basis function network, fuzzy RBF network, convolutional neural network, multi-fractal analysis, feed-forward neural network, knowledge based networks and a modular neural network. The paper concludes that "a Gene feedforward neural network with a combination genetic algorithm with feedforward neural network will be proposed for detecting cervical cancer."

Classification of Cancer Primary Sites Using Machine Learning and Somatic Mutations

The goal of this paper is to use machine learning in finding a relationship between cancer sites and mutations. The researchers use the COSMIC (Catalog of Somatic Mutations in Cancer) database which holds patient information, mutation-associated genes, and mutation-associated chromosomes. The researchers collect 990,529 samples, 25,660 genes, 1,292,597 coding mutations, 1,528,225 noncoding variations, and 11,330 references from the database and then filtered to 230,255 patients, 22,111 unique genes, and 1,760,846 mutations. They then filtered the data once again by the number of sites and commonality of those sites and got the data down to down to 6,751 patients and 17 sites. They represent all of the data in binary. The model used is a support vector machine (SVM). The researchers adjusted the feature set and used cross-validation to find the best accuracy which was ~60%. Then evaluated the model at each cancer site using precision (~.7), recall (~.5), and F-measure. They found that the model work best in the large intestine, liver, skin, pancreas, and lung cancer site. These sites happened to have the largest sample size so they infer that increasing the sample size will also increase the performance of the model.

Machine Learning for Imaging

Deep Learning for Digital pathology Image Analysis: A Comprehensive tutorial with Selected use Cases

This article split deep learning for digital pathology image analysis into three categories: detection and counting, segmentation, and classification. Some current problems in deep learning in these areas are variances in slide preparation, stain and scan, vendor platforms, and in the biological variance. When this paper was published (2016) the current tools did not provide help in selecting the suitable magnification, dealing with errors in the annotations of the images, and selecting an appropriate training set. The paper goes into more depth on how the models typically are made for each of the three categories. Detection is used to identify the center of something and requires unannotated as well as annotated images for training. Segmentation is more difficult as it requires the same data as detection but delineates the boundary, not just the center. Lastly, classification only requires the images and labels to train the model and has the goal of identifying the issue the image depicts. Next, the paper provides descriptions of multiple use cases. The classification use case involved lymphoma subtype classification. The data has a training set of 300 images and 75 test images that represented 3 different subtypes of lymphoma. The model uses caffe to crop smaller images from the dataset in order to create more training data, totaling about 825,000 training patches. The model's mean average is 96.58% (2.6 misclassified images in 75 tests). The source code, step by step tutorial, the models, and the datasets can be found online.

Image Analysis and Machine Learning in Digital Pathology: Challenges and Opportunities

This article discusses the benefits of digitizing glass pathology slides in the hopes of making pathology more quantitative. Digitizing slides would allow for more remote consult without shipping slides, reduce material and the need for storage, and potentially teach medical students. There is also some current interest in automated learning algorithms and deep learning with pathology. Some past projects included identifying epithelial and stromal tissue regions, detecting and classifying individual nuclei in colon cancer images, and identifying and quantifying the number of mitoses on cancer pathology images. Histopathology images are well suited for deep learning approaches given their complexity. Advantages of digital pathology include assisting clinicians when making medical decisions by providing a “second opinion”, and allowing for the combination of radiologic, histologic, and molecular measurements. While deep learning has been criticized for its dependence on large sets of data and lack of intuition associated with the image features generated, there is still interest in developing algorithms for automated tissue classification.

Quantitative feature modeling is comprised of two different categories: handcrafted features, which can be connected to specific measurable attributes, and unsupervised features, which rely on filtered responses from large training sets. Within handcrafted features, there are domain agnostic features and domain inspired features. Domain agnostic features have an intuitive meaning and can be applied to image characterization across difference diseases, like gland size, shape and color. Domain inspired features represent a class of hand crafted feature for a specific reaction, disease, or organ site and can not be applied to other diseases. Handcrafted features are beneficial because they may provide more transparency and be more intuitive, but they require comprehensive knowledge of diseases being studied. On the other hand, unsupervised features are beneficial because they can be applied quickly to any domain or problem, but are harder to interpret.

Digital pathology has the potential to bridge the gap between molecular pathways and digital biomarkers associated with disease progression and severity. However, there are some challenges with digital pathology. These include color variations in the samples due to mounting differences that could make learning algorithms harder to create, the lack of 3D representation in images, and when 3D scanning capabilities are created, there will be an increase in data complexity that GPU technology might not be able to keep up with.

Machine Learning in Digital Pathology: A Journey from Handcrafted Feature Descriptor to Deep Learning Approaches

Machine learning is already being used in histology and some pathology and has been shown to be an influential factor in medical decisions. This article discusses different algorithms to use and considerations to keep in mind when creating a model. The three deep learning algorithms mentioned are random forest, convolution neural networks, and deep neural decision forest. Random forests select the best set of representative features on infinite and generic feature spaces and have shown autonomous adaptation to different visualization problems. An example of this is detecting nuclear reactions in histology images in addition to classification of 3D ultrasound. Convolutional neural networks, CNNs, are networks of layers for local feature extraction, dimensionality reduction, and non-linear activation functions. Local connections within each layer share weights to reduce the amount of weights that need to be optimized. This has been used for mitosis detection, nucleus segmentation, gland segmentation, and metastatic detection. Deep neural decision forests embed forest decision functions with the nodes of fully connected output layer of CNN which enables representation learning. Another topic that is covered is feature engineering, which is when the model is trained to discover mapping between feature values and known output values. Feature extraction is when a problem is transferred into a feature space where the mapping can be more easily learned. Features can be complex and require large efforts to find, thus were machine learning comes in. When designing models, it is important to consider to the amount of data available because it can be hard to get accurate predictions if the data set is not large enough or there is not enough computing power to run the model. Pairing knowledge driven and data driven approaches that focus on training what is unknown, rather than what is known, could make training models with smaller sets of data easier. For example, a model for nuclei identification would focus on identifying what it knows is not a nucleus rather than what it knows is one.

Spatial Organization and Molecular Correlation of Tumor Infiltrating Lymphocytes Using Deep Learning on Pathology Images

This article uses data from the TCGA which they argue is an underutilized resource. This is because it is not systematically characterized and has to be manually mined. The data in this article are whole slide images containing tumor infiltrating lymphocytes (TIL). The researchers created two different models to characterize patterns on the slides; one for tumor infiltration and the other for necrosis segmentation. Creating two different models is necessary because different imaging scales are needed in order to classify them accurately. The lymphocyte infiltration classification model is semi supervised to categorize tiny patches of an input image into those with or without lymphocyte infiltration. The necrosis segmentation model segments regions of necrosis and accounts for false positive of sites with lymphocyte infiltration. The training data for these models comes from datasets that are annotated by pathologists. A convolutional autoencoder that had two branches, one recognized high resolution features like cell nuclei, and the other recognized one low resolution features is used to train the model. The unsupervised autoencoder features from CAE go into the supervised CNN model. Data is also augmented in three different ways to train (color, orientation, and cropped). The model outperforms other CNNs and experienced pathologists.

Overall, this model correlated TIL spatial structures with survival rate. Structural patterns are associated with survival rate indicating that immune response can be encoded in the patterns of TILs. This article proposes a scalable and cost effective method for computationally staining and extracting lymphocytes and shows how combining spatial structure with genomics and molecular assessment can be used to characterize tumors. In addition, a contact is given to direct any questions about the data resources: Vésteinn Thorsson (Vesteinn.Thorsson@systemsbiology.org). This may be a good person to reach out to about using data from the TCGA.

Automatic Detection of Invasive Ductal Carcinoma in Whole Slide Images with Convolution Neural Network

Machine learning models either train on annotations made on the slides by pathologists or they learn their own "annotations" or most important features for classification. Using annotated data requires pathologists to annotate all the data before training the model which is time consuming. The latter method requires multiple non-linear transformation of the data to create more abstract, and thus more useful, representations. In this article, the whole slide images were split into patches of 100 x 100 pixels. Data from the University of Pennsylvania and The Cancer Institute of New Jersey was used to train a 3 layer CNN. This is another private data set. The model classified the image patches as non-cancerous or cancerous and labeled as either 0 or 1, respectively. The learn-from-data approach had a specificity of 88.86%. To test the model, pathologists annotated images and them compared them to what the model had predicted. The learning network was also compared against a set of handcrafted features. The model created in this article was able to accurately predict cancerous regions and outperformed the set of handcrafted features. This model shows that it is possible to train an accurate model with data that is not annotated. However, it requires a much more complex model to do so.

Deep Learning for Classification of Colorectal Polyps on Whole-Slide Images

Sessile serrated polyps develop into more aggressive forms of colorectal cancer than other types of polyps and thus need to be identified early. The only way to diagnose these polyps is through histopathology. This article created a model to characterize polyps in pathology on whole slide images. The data was gathered from their own private data repository from patients of their clinic, therefore it was not public data. The images were annotated by doctors at their facility to used to train model. 90% data was used for training and the other 10% was used for testing. The images were preprocessed by cropping around the polyps, and having another pathologist outline the regions. At first, the scientists were limited to how many layers they could add to the model because of the vanishing gradient problem, which occurs when there are too many layers and the model can not optimize properly. This was fixed by using a residual network architecture (ResNet), which allows for the addition of up to 152 layers. This model was used to help pathologists identify at risk patients earlier improve patient outcomes.

Automated Grading of Glioma Using Deep Learning in Digital Pathology Images: A Module Approach with Ensemble of Convolutional Neural Networks

Currently, gliomas are diagnosed and categorized by pathologists through visual inspection of histological slides, but this process could be transitioned into a computer algorithm. There are three types of pathology focused on in this article: astrocytoma, oligodendroglioma, and oligodendroglioma of nucleus. To identify these, pathologists look for specific morphological features including mitosis, nuclear atypia, microvascular proliferation, and necrosis. Data for this project was gathered from the TCGA but it was too big, so the images were compressed into tiles. The neural network used to train the data was created from convolutional layers and pooling layers. The ReLU activation was used and the final loss layer used softmax. A back propagation algorithm, Caffe, was used to train the algorithm and increase accuracy. The model was tested on an individual test set separate from the training set; 80% of the data was used a training data and 20% was used as testing data. Two models were created, one to classify tumor type and another to classify the grade of the tumor. The type model consisted of 8 layers and had an accuracy of 96%, and the grade model had 19 layers and was only 71% accurate. Possibilities for the lower accuracy include poor data quality and the subtle difference between grade 2 and grade 3 tumors. This article created a multiple network pipeline that allows researchers to gauge measures of diagnostic quality and performance of models, make it easier to analyze each unit independently, use smaller amount of data to train the individual models rather than creating one big model to classify both type and grade, explore difference model architectures, and enabled users to use specific parts of the pipeline to integrate into other methods. Despite the time it takes to train the models, once they are trained they allow for faster processing and updating times that can assist pathologies when making medical decisions.

Chest Pathology Detection Using Deep Learning With Non-Medical Training

Chest radiographs are the most common examination performed and therefore contain a lot of data that can be used in machine learning algorithms to uncover new patterns. This article trained a model to analyze images with data that did not contain medical data. Then they tested the accuracy of their model on chest radiographs. The most common abnormalities found in chest x rays are lung infiltrates, catheters, and abnormalities of the size and contour of the heart, though most computer and diagnostic tools focus on identifying lung nodules. The pretrained model was a Decaf CNN and used images from ImageNet. There were two different depictors tests: GIST and Bag of Visual Words (BOVW). GIST resizes the images and iterates them over different scales, orientation, color, and intensity and creates histograms for each cell. BOVW is usually used for documents and it extracts patches of each training images, applies principal component analysis, adds patch center, and then clusters the patches into representatie visual words creating a dictionary. After, each images can be represented by a unique distribution of generated dictionary words. The medical images used to test the model were DICOM images from Diagnostic Imaging Department of Sheba Medical Center. The images were analyzed by two radiologists and their reports were used as the golden standard. The images depicted three chest pathology conditions: right pleural effusion, cardiomegaly, and abnormal mediastinum. The model achieved the best accuracy when combining depictors of Decaf 5, Decaf 6, and GIST. They also increased the accuracy by applying late fusion of the descriptors.

Automated Nuclear Pleomorphism Scoring in Breast Cancer Histopathology Images using Deep Neural Networks

This article dealt with both classification and detection in determining the grade of grade/aggressiveness of breast cancer. The grade is determined based off of the degree of tubule or gland formation in the tumor, the nuclear pleomorphism, and mitotic nuclei count. The model used to do this was a DBN-DNN. Each layer independently trained restricted boltzmann machine (RBM: "generative artificial neural network that learns the probability distribution over its set of inputs"). The RBM is unsupervised and trained using a contra divergence algorithm (CD). The RBMs are stacked to form a DBM and a is DNN constructed with final layer. The DBN-DNN was then fine-tuned with a classical back-propagation algorithm (supervised). The data set used featured 80 H&E stained breast cancer histopathology images of nuclear pleomorphism scores, from the MITOS-ATYPIA dataset. The images were of 10x magnification and were assigned to a score of either a 1, 2, or 3 based on the evaluations of 3 pathologists. The main modules were nuclei detection, feature set preparation, DBN training and classification and the attributes extracted include area, solidity, eccentricity, equiv diameter, average gray value, average contrast, smoothness, skewness, uniformity measure, entropy. The model took 20 inputs, had 2 hidden layers of 13, and had 3 outputs to tell the score of the sample. The model recorded the root mean square error which was .2407.

Autoencoders

Autoencoders, Unsupervised Learning, and Deep Architectures

Autoencoders can be linear or nonlinear, the most nonlinear autoencoder being the Boolean autoencoder. Autoencoders were first created to learn from input data instead of teaching data through backpropagation, They are now used for deep learning, most popularly in the form of Restricted Boltzmann Machines (RBMs), just like the one discussed in the previous article. Layers of autoencoders are trained using unsupervised algorithms and then stacked. The last output layer is trained and refined using supervised learning algorithms.

This article compared features of linear autoencoders and Boolean autoencoders including group invariance, problem complexity, landscape of E, and clustering, among other things. A basic autoencoder framework follows the pattern n/p/n where $p < n$. n is the original number of data points and p is the reduced dataset according to function A. n is then recreated using function B. However, the vertical composition of autoencoders can change from n/p/n to many other combinations like $n/p_1/p/p_1/n$ or $n/p/n/p_1/n$. In another case, the hidden layer, p, can contain more data points than the input or output layers. Architectures like these require more constraints than other autoencoders but can still provide beneficial outcomes. Autoencoders like these can occur when there are two or more autoencoders: $n/p_1/n$ and $n/p_2/n$. p_1 and p_2 can come from different learning algorithms, different initializations, different training samples, different learning rates or different distortion measures. Once they are both trained, p_1 and p_2 can be combined, altering the horizontal structure and creating one autoencoder.

There are also other autoencoders that can be considered. There are mixed autoencoders that have different constraints on different functions, A and B. There are also linear autoencoders that exist for non real numbers like complex numbers or infinite numbers. Autoencoders are different from other neural networks because they cluster the data and label it with one function. Autoencoders can be altered in both their vertical and horizontal structures to create models that can obtain better results.

Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions

Autoencoders can be used to classify many different types of data including images and word documents. This article used a hierarchical structure and compositional semantics to understand sentiment better than bag-of-words algorithms. This model can be trained using both labeled and unlabeled data and the sentiment is not limited to just positive and negative.

The model represents words as continuous vectors of parameters, so each sentence can be represented as an ordered list of vectors like (x_1, …, x_n). This is done by embedding words into a vector space and then capturing syntactic and semantic information form oc-occurrence statistics. Representing words as vectors is better than representing them as binary. Unlike other autoencoders, the model does not require a tree structure. Tree structures are used to repeatedly compute parent representations until the whole tree is reconstructed. This model, however, computes a cross entropy error score for the first pair of vectors, then shifts one space and computes the score for that pari. This continues until each pair has a score. The pairs with the lowest scores are replaced with the parent vector and the process starts again until the tree is recovered by unfolding the collapsed decisions. The resulting tree structure captures as much single word information as possible. The weights for the model are adjusted to account for some vectors representing different amounts of collapsed words. The cross entropy of the final softmax layer is minimized by back propagating error and influencing the RAE parameters and the word representations. This changes the model from focusing on more syntactic characteristics to semantic.

The model was trained on data from Experience Project and other common datasets. Experience Project is a place where people can post short stories and other people can add up to five reactions to them: shock, understanding, sympathetic, amusement, and approvement. The model was accurately able to predict the emotion conveyed in the stories. It was also tested against three other methods: majority phrase voting using sentiment and reversal lexica, rule based reversal using a dependency tree, Bag-of-features and their full tree-CRF model. This model outperforms them and works well even with small amounts of training data.

The autoencoder created in this article is more advanced than other popular state of the art autoencoders. RAAMs, recursive auto-associative memories which learn vector representations for only fixed recursive data structures. RNNs, recursive neural networks, require labeled trees structures and use supervised cross entropy error scores at each node. Naive Bayes, Maxent models, and SVMs are also all good for document classification but not as good at predicting sentiment. The model in this article outperforms many other autoencoders at predicting sentiment in word documents.

Deep Learning of Part-based Representations of Data Using Sparse Autoencoders with Non Negativity Constraints

Deep learning architectures are hard to train since they have so many hidden layers. This article outlines a greedy layer-wise training algorithm where each layer is separately initialized by unsupervised learning algorithms, then stacked and fine tuned using supervised algorithms. Popular unsupervised learning algorithms include restricted Boltzmann machines (RBMs), and autoencoders. An autoencoder is a neural network that is trained with an unsupervised learning algorithm which reconstructs inputs from their coded representations. Many autoencoders are similar but the one in this articles uses one general autoencoder with trainable weights for both hidden and output layers, uses a nonlinear function for the nodes making the model more flexible, and can easily be scaled up for larder deep learning networks.

To train this model, limitations were imposed on parameters so the autoencoder learns compressed representations of the input to discover the structure of the data in a high dimensional space. A part based representation method was used which is when the data is decomposed into parts that reproduce the original data when combined. They created a non negative constrained autoencoder in which every layer was individually optimized with the hidden activities of the previous stage being used as the input for the next. They used non negativity because it promotes additive features and captures part based representation of data. The last stages used a softmax regression classifier to fine tune the model. Only the weights of the softmax were altered in the fine tuning of the final stage.

The datasets used in this article were all public. They used the MNIST handwritten number set, the ORL face set, and the NORB object recognition set. Compared against other autoencoder methods, this one reproduced the best data on all three datasets before fine tuning the last layer, and better in two after fine tuning.

Deep Features Learning for Medical Image Analysis with Convolutional Autoencoder Neural Network

This research paper proposes an alternative to the previous methods of identifying nodules in CT images of lungs. The current methods include region of interest (ROI) definition, segmentation, hand-crafted feature, and categorization. Previously research have used convolution neural networks (CNN) for the issue, however this method requires a large amount of labeled data. Thus, the researchers proposed the use of a convolutional autoencoder neural network (CANN) for the classification of pulmonary nodules. This method was primarily unsupervised using a large amount of unlabeled data and using a small amount of labeled data for fine-tuning the network. The network feature 9 layers: the input layer, 3 convolution layers, 3 pooling layer, a fully connected layer, and the output layer. The specifications of each layer as described by the authors are as follows:

- Input: 64×64 patch captured from CT image.
- C1: convolution kernel is 5 × 5, the step 1, the number of convolution kernel is 50, nonlinear function is Relu.
- P1: max pooling is used, the size of pooling area is 2 × 2.
- C2: convolution kernel is 3×3, the step 1, the number of convolution kernel 50, non-linear function is Relu.
- P2: max pooling is used, the size of pooling area 2 × 2.
- C3: convolution kernel is 3×3, the step 1, the number of convolution kernel 50, non-linear function: Relu.
- P3: max pooling with 2 × 2 size of pooling area.
- Full: fully connected layer, 500 neurons.
- Output: softmax classifier, 2 classes.

The data set originated from a Chinese hospital and featured 4500 patient lung CT scans. The data was split into 3 sections, the first being 50,000 unlabeled 64x64 patches. Additionally, there was 3,662, 64 × 64 patches of which 1,754 contained nodules and 500 pairs of labeled patches for similarity judgements. The results showed the CANN method had significant over the other methods with an accuracy of 92%, precision of 91%, recall of 91%, F1 of 91%, and AUC if 0.97.

Convolutional Auto-Encoder for Image Denoising of Ultra-Low-Dose CT

Normal and even low dose CT scans often put patients at risk because of the increased exposure to radiation. As a result, ultra low dose CT (ULDCT) scans are sometimes used, however this scans are highly affected by noise. This paper proposes post-processing technique for reducing the noise in ULDCT scans, a convolutional autoencoder. The goals of the study were to "validate a patch-based, neural-network-trained image-denoising method for ULDCT images," "to train the neural network with CAE and pairs of SDCT and ULDCT image patches," and "to investigate the performance of our proposed method using a chest phantom." The data used was one set of SDCT images and two sets of ULDCT images, one of which which was used for testing and the other for training. The data was formatted, the noise was added, the image with noise and the image without noise were paired. The network tested with different numbers of layers using the sigmoid activation function and the same weights for encoding and decoding. Then, 3 technologists and 5 radiologists evaluated the images after going through the model by recording streak artifacts, other noise, visualization of pulmonary vessels, and overall image quality. The researchers found the best model to have 9 layers and 20 feature maps. The models evaluation in streak artifacts, other noise, and visualization of pulmonary vessels were an improvement over block-matching and 3D filtering. However, the overall image quality did not show an improvement.

<u>Other Topics</u>

Deep Learning Convolutional Neural Networks Accurately Classify Genetic Mutations in Gliomas

This article is about a convolutional neural network that was created to identify mutations in images of gliomas. They created a 34 layer model and trained it with preprocessed images of gliomas. The imaging data was from the VASARI project in The Cancer Imaging Archive and corresponding genomic data was from The Cancer Genome Atlas. 80% of the data was used to train the model and the remaining 20% was used to validate the model. The three different mutations that were evaluated were IDH1 mutation, 1p/19q codeletion, and MGMT promoter methylation. Predictions for the IDH1 mutation had the highest accuracy at 94%, and the MGMT methylation mutation had the lowest accuracy at 84%. The final layer of the model used principal component analysis, which is a form of data analysis typically used in predictive models when learned features are highly correlated and can not be separated. This was the case with features learned from the 1p/19q codeletion. This article is a good example of how machine learning models can be created to identify patterns in data sets to draw conclusions. This specific example used imaging data but convolutional neural networks are not limited to just images, they can be created to analyze data in many different formats.

Gene Expression Inference with Deep Learning

This paper highlights the NIH LINCS program and their work that found that ~1000 specific genes can capture 80% of data on CMap (~22000 genes). The LINCS researches then developed L1000 Luminex bead technology that would create an expression profile from the ~1000 “landmark genes”. If a user was interested in one of the other ~21000 genes, “target genes” they could use the landmark genes to infer the expression. However, LINCS currently uses a linear regression which is computationally challenging as it ignores all nonlinearities. In order to effectively demonstrate the gene expression the model used must be scalable and able to represent complex data. D-GEX was then developed as a machine learning method for gene expression interface. The study compared the performance of the linear regression, D-Gex, and KNN (k-nearest neighbor) on GEO data and RNA-Sequence expression data. The GEO data contained 129,158 gene expression profiles that each featured 2,978 landmark genes and 21,290 target genes. This data was normalized to be in a numerical range of 4-15 and duplicate profiles were removed, leaving 111,009 gene expression profiles. The RNA-sequencing data contained 2,191 profiles of tissue samples. The GEO and RNA data used different units and had to be normalized, leaving 943 landmark genes and 9520 target genes. Additionally, all data was standardized by subtracting the mean and dividing by the standard deviation. As for the model used, D-GEX is a multi-task, multi-layer feedforward neural network with a hyperbolic tangent activation function, a linear activation function applied to the output, and a loss function of mean squared error. The model was too large if all the data was used, so the target genes were randomly split in half and 2 models were made. The model used Theono and Pylearn2 and the primary training techniques were dropout, momentum method, normalized initialization, with the learning rate initialized to $5*10^{-4}$ or $3*10^{-4}$. The D-Gex model was a

15.33% improvement over the linear regression on the GEO data and a 6.57% improvement on the independent RNA-Sequenced data.

A Deep Learning Framework for Gene Network Inference by Aggregating Biological Information

Researchers in this article created new deep learning framework to incorporate gene expression data and GI network topological structure information and tested it against competitors to predict new gene interactions. The model used 1st and 2nd order proximity to maintain topological space and make their predictions more accurate. 1st order proximity is the interactions between genes, assuming that closely related genes will interact. However, genes that aren't similar could still interact but not be connected, so a 2nd order is needed to account for these additional interactions. Topological structure is the structure imposed on X that characterizes it as topological space. The framework, Gene Network Embedding (GNE), incorporates both gene network structure and gene expression data to learn gene representations. Embedding the gene network in a lower dimensional space preserves the structure and their statistical relationships. It also encodes the expression data from microarray experiments to the dense representation using nonlinear transformation which is helpful in capturing the nonlinearities of the data. GNE uses an early fusion approach that combines the heterogeneous data at the beginning then creates and trains one single model. Most other models use a late fusion approach which combines the data after they have been trained in separate models. The model was optimized using Adam, an activator ELU, and only one hidden layer using a tanh activator. Gene interaction network data was from BioGRID database and gene expression data was from DREAM5 challenge. The performance of the GNE was evaluated against other popular methods: Correlation Isomap, LINE, and Node2vec. GNE out performed the other methods in structure preserving, predicting gene interactions between unseen genes.

Breast Cancer Data Analysis for Survivability Studies and Prediction

The SEER database was used to gather the data in this study. The dataset of breast cancer cases included 740,506 records that contained 146 variables. The dataset was then reduced to 85,189 cases and only the most impactful 29 variables were used in the creation of the model. The machine learning model used was a SOM-DBSCAN (self organizing map and density based clustering algorithm). SOM is a type of unsupervised ANN used for clustering, dimensionality reduction, and visualization of high dimensional data. A 2D, high-resolution SOM was chosen for the model in order to form clusters on data with missing values. DBSCAN identifies the clusters formed by SOM. The DBSCAN does not require a predetermined number of clusters, can detect arbitrary shaped clusters, is resistant to outliers and noise, and only requires only two algorithmic parameters. This methodology allows for predictions to be made for individual clusters, cohorts. By doing this rather than training on the dataset as a whole the accuracy can get as high as 90%. In addition to providing valuable information about one example of a new machine learning tool and the data they used, this study also highlights many other relevant studies and their findings.

Lectures and Conferences

<u>The NCI Informatics Technology for Cancer Research (ITCR) Program</u>

The NIH held an informatics technology meeting showcasing the newest advances in data management tools. There were many talks about new platforms for data analysis or updates made to existing ones. There were also tool demonstrations for the platforms and a post session to allow for more in depth conversation. We took notes on most every talk we attended and learned about new tools that we plan on looking into or have already added to our list of databases. Some talks of special interest included:

"The cBioPortal for Cancer Genomics" - Dana Farber Cancer Institute

cBioPortal is an open sourced web based platform that allows researchers to visualize and analyse large scale cancer genomics data sets. It is an open sourced platform that allows users to use it without charge but it can also be made private for company specific data sets. The data sets include both clinical and genomic data. They recently made improvements to the site that now allow users to view the most commonly mutated genes for specific cancers as well as survival rates and clinical variables. It also includes a view for specific patient data that shows the initial genome sequencing, all treatments the patient underwent, and any more recent genome sequences so that the researcher can view how the treatment altered the genome. Additionally users are now able to visualize and analyze large-scale cancer genomic datasets, store genomic and clinical data and use interactive analysis tools.

"TCIA Sustainment and Scalability Platforms for Quantitative Imaging Informatics in Precision Medicine" - Emory University

The Cancer Imaging Archive, TCIA, is an online collection of de-identified medical images. However, it does not only contain imaging data, it also has clinical data, radiotherapy data, and digital pathology data. Developers have recently added features that allow users to curate, annotate, and manage their data, as well as cloud sharing capabilities. PRISM (Platform for Imaging in Precision Medicine) will containerize the TCIA technology stack and integrate the imaging and non-imaging data and make query and analysis more readily available. TCIA is continuing to improve as they are working on interactive deep learning and visual analytics and they would like to continue to improve TCIA and add more features but they need more guidance from the community to determine what needs to be changed.

"Advanced Development of an Open-Source Platform for Web Based Integrative Digital Image Analysis in Cancer" - Emory University

The Digital Slide Archive, DSA, is an online resource for searching and viewing digital cancer pathology data. One of the goals of the DSA has been to create a better way to manage the data. Unlike DICOM, there is no universal way to store digital pathology images. A recent project they have been working on is training a machine learning model to identify features of the images, like blood stains, air bubbles, or even actual areas of interest in the samples. They are doing this by enlisting the help of about 20 scientists around the world and annotating thousands of individual images. These images are used to train the model to identify the annotated features. Eventually, users will be able to say what they are looking for and the machine will pull out images with what they find. Then the user can refine the search by confirming or denying what they machine pulls. What they have found was that the convolutional neural network, CNN, is actually statistically better at classifying histology compared to humans.

Talk from NCI Director

Dr. Norman Sharpless highlighted how important big data science is becoming in modern cancer research. He emphasized how important it is to make standards for recording information on online public datasets and that it is every researchers' job to make their data public and usable for other people. One initiative that has been started in order to achieve these goals is the NCI Genomic Data Commons Portal which is an public online data sharing site. With more unified data resources, researchers can begin to understand the trends in treatment to begin to make medicine more personal and precise. Another initiative that has been started is NCI-MATCH, a nationwide clinical trial where patients are assigned treatment based on their tumor genomes. Dr. Sharpless also discussed the NCI's goals to accelerate cancer diagnoses with high performance computing. He mentioned the three pilots that are functioning within CANDLE: Pilot 1 - Preclinical model development, Pilot 2 - RAS therapeutic targets, and Pilot 3 - Precision oncology surveillance. However, there are some complications that need to be worked out in order to continue to moving forward with high performance computing. These include ensuring that patient data is secure, streamlining and organizing data storage with in the cloud, and building a workforce and leadership team to push what is possible with high performance computing.

Portability of Deep Learning Models in Biomedical Applications Workshop

This workshop spanned Wednesday and Thursday and was comprised of professionals who specialize in machine learning and data specifically for biomedical applications. During the meeting concerns with the current state of machine learning in medicine were brought up and then solutions and next steps were posed. Topics focused on included, representing and describing data input and output, defining deep learning models for predictive oncology, and transfer learning and how to share already trained models for other researchers to use. Below is a summary of the notes from each section of the meeting.

Panel Session 1: Creating Deep Learning Models

Panel Session 1 featured presentations from Maulik Shukla with CANDLE, Ian Foster with DLHub, and Fernanda Foeretter with Nvidia. The CANDLE presentation covered the 3 pilot projects and their purposes and shared the project's goal of supporting hyperparameter optimization and creating a repository of the most effective models to be used later. The different components of CANDLE (metadata store, model store, and data API) were discussed and other similar resources such as ModelHub and Knowledge Grid were mentioned in the discussion. DLHub is also a data and learning hub but rather than the medical field, its primary focus was on models for material sciences. DLHub features image classification models and is a place to collect, publish, and categorize models. They also work to create models that auto-train as new data is made available, storing the most accurate models for reuse. Nvidia's presentation discussed GPU computing and GPU accelerated libraries for deep learning, signal, image & video, linear algebra, and parallel algorithms. One of their goals was implement machine learning technologies in hospitals in order to accelerate image recreation from MRI scans.

Panel Session 2: Applications of Deep Learning Models

The second panel session featured presentations from Kayhan Batmanghelich, Frank Alexander, and Hema Chamraj. Batmanghelich's presentation highlighted big data and the importance of interpretability, improving biomarkers, flexability, incorporating feedback and knowledge, and most importantly, data integration. Alexander's presentation emphasised that the theoretical and mathematical foundations of deep learning is limited and discussed the repercussions. He also highlighted the applications for deep learning in biology such as image analysis, time series analysis, general multimodal analysis, model reduction of high-fidelity models, and transfer learning. Lastly, he went over some of the challenges deep learning faces including social, organization, and scientific challenges. Chamraj present on Intel and their AI and deep learning use cases. She highlighted that AI can be used on images, text, genomes, to assist clinicians, and to cut drug development time and cost. The use cases mentioned were: tumor detection in large 3D medical images, disease risk prediction using genetic variation data (MLP), diagnostic imaging: thyroid detection in ultrasound, diagnostic imaging: body part classification in CT scans, predicting billing codes with EHR, AI accelerates high-content screening in drug discovery, and classify and predict protein sequences using DL.

Breakout Session: Initial Development (Group 1)

This group chose to focus a lot more on how to increase data access and make sure data is consistent so that it is usable. There is a lot of data that can be assessed, but people are either not willing to share their data or do not have the rights to do so. Data also comes in many different forms and is required to meet different requirements depending on the organization the research group is associated with. Most data currently reside in data silos with lots of patient identifying information attached to it. Before this data can be used and shared, it must be cleaned in order to protect data confidentiality. This can be a very challenging task to accomplish because there is not easy way to make sure a data set containing information from thousands of patients is completely deidentified. If there is an error rate greater than 0 for de identifying data, the study will not get approved.

In order to make all data usable to everyone, there needs to be a set of guidelines that all researchers must follow to ensure that data is in a universal form. Standards are really difficult to set because it requires all parties to agree on one thing that is no longer unique the them. While the NIH or NCI can organize a community that can decide how best to record and share data, it is more beneficial if the guidelines do not come from within the NIH. Integrated data from sensors have been used to train models showing that it is possible, it will just take more regulations to allow for data integration in the biomedical fields. Some potential sources that already have existing guidelines could be DBGAP, GenBank, and BioBank.

Another issue is how to make and train models that then can be used by others so that they don't have to start from scratch. What does the model need to include in order to make it useable to everyone? What needs to be included in containers for models? In addition, there can be issues of security when people start sharing models because there is the possibility that confidential, personal data can be recreated from a trained model even though the data itself was not released. There is also the problem of how to ensure that the data people put into the model is in the same format that it was trained on. One solution to this problem could be creating a universal set of terminologies that are acceptable for metadata.

Breakout Session: Initial Development (Group 3)

The breakout session discussed challenges and questions about data sources and specifications, model descriptions and representations, and interoperability, portability, and accessibility. In area 1 (data sources and specifications) the group discussed the challenges that face machine learning repositories in terms of data. The group proposed that when models are uploaded they must clearly define input data, data format, original data source, type of measurements, data processing and transformation steps, normalization, missing value imputations, and data distribution statistics. Additionally, the importance of uploading the data set the model was trained on was highlighted and the question of what to do if a model was trained on a non-public dataset was brought up. Other questions that were posed were how to make people provided the right metadata early on, who the target audience for models and transfer learning is, and how to handle different data distributions. The first thing that was discussed in area 2 (model descriptions and representations) was the current approach: keras model description (JSON), models weights as HDF5, and container for running software. This approach was said to be useful for reusing Keras, may not work with other deep learning frameworks. The question was then brought up of how to support cross-framework usage. Non-deep-learning models were then discussed and the questions of how to capture model descriptions and weights were posed. Metadata was brought up again and it was said that it should included a description of the model, the input data, the scientific problem/application, the type of model, and the general applicability of model. Lastly, in area 3 (interoperability, portability, and accessibility) containers and open API specification were highlighted. The idea of a website or API for submitting models and that would allow users to search/browse published models and related metadata and download them was brought up. Additionally, throughout the three areas examples of machine learning repositories were brought up, some of them include MoleculeNet, ModelHub, and Kipoi.

Top Challenges

These were the top challenges that stand in the way of furthering machine learning in biomedical research. There are many problems must be overcome in order to share models featuring different types of data with many institutions. The goal of this brainstorming session was to record all of the problems that are associated with building better models, sharing models with peers, and disseminating the models for reuse by nonexperts. The issues were then labeled to determine if they related to a technical issue, policy issue, or infrastructure issue. After much discussion, it was determined that the first issue that needs to be tackled was creating a micro environment that allows users to test their models. This goal was set to be accomplished by September, when the next mini-workshop is scheduled to take place. See below for a full list of the issues, organized by main topic and labeled with their area of focus.

Labeling instructions:
[t] technical
[p] policy
[i] infrastructure
Building better models

A. [t] Theoretical underpinnings for learning and dynamical models
B. [t] ID'ing when models are out of bounds, UQ *
C. [t] Creating hybrid statistical/DL models with interpretability
D. [t] Experiment with biocompute object as framework
E. [t] Understanding performance issues as part of annotation **
F. [t] Using DL to understand new knowledge (why labels are correct)
G. [t] Problem: More training data that fills in gaps that has the right properties
H. [t] Quickly searching model space for better models from giving starting points. **
    a. Solution.. lowering barriers for data sharing and data finding — solution.. tools for cleaning/characterizing data — solution.. tools for understanding distributions of models and data — solution.. automated methods for model construction, training and evaluation .. datarobot type stuff for deep learning models that run on exascale systems.
I. [t] methods for using UQ to have the model identify when it is out of bounds for training *
J. [t] ID and note where problems or variation in performance is seen (e.g., by registry, hospital, person, year, model, training data selection criteria, conditions of workflow where it is tested)? *
K. [t] How does the model topology change across hardware generations? Do you need to write different topologies to achieve the same performance?
L. [t, p] Test for classes of adversarial examples, class sensitivity, data augmentation – creation of best practices **
M. [t, p] characterization of data sets: what makes it hard, why are you using it, key features of the data distribution (e.g. imbalance) *
N. [t,p] Making object computable and scalable
O. [t,p] *Defining models and evaluation criteria – develop evaluation criteria to gauge effectiveness/performance

P. [t,i] Understanding how model topology changes across generations
Q. [t,i] ID'ing how tricks cumulatively affect the model
R. [t,i] Automating the process for characterizing models
S. [t,i] Automating process of breaking models & defining model breaking
    a. Data augmentation/munging
    b. Checking data vs. checklists – ensuring models are annotated
    c. Model breaking (moving outside acceptance criteria, performance under adverse conditions)
    d. Provide external gauntlet of tests
T. [t,p,i] Look at mechanisms for supporting transfer learning exploration
    a. Applying reference for experimental protocol
    b. Looking at values experimented with and values obtained
U. [p] Release notes about model *
V. [p] Description of intended purposes *
W. [p] Domains of applicability *
X. [p, t] Building models with an option to self-explain – inferencing, generating new layer of metadata to analyze
Y. [p] Informing how models are validated that are referenced in publications
Z. [p] Having an automated taxonomy of models describing hierarchy/interconnectedness
AA. [p] Building models that are 'good enough' – defining good enough, and focusing efforts on toolkits to get there
BB. [p] key structural features in the network, and why it was developed, and how in improved the model (e.g. release notes with reference to key work)
    a. atrous convolutions / atrous pyramid pooling, image segmentation fine grained local features, +5% over reference model
CC. [p] statistical range of quality of the model given multiple runs and possibly multiple tools
DD. [p] Clearly defining problem that model is designed to solve

Sharing with peers

A. [t,i] Statistical description of quality of the model *

B. [t,p,i] Characteristics of input dataset, description by important features *
C. [t] How to stitch learning and dynamical models together, or various dynamical model scales together
D. [t] *Moving models across silos (potentially easier than breaking down silos)
E. [t] *Ensure models fit into user data pipeline
F. [t, p] If models are 'good enough', can be reproduced and shared more quickly
G. [t] Opening up black boxes
H. [t] Solutions.. tools for characterizing models .. CV, UQ, sweet spot analysis, impacts of model simplification..
I. [t] Like the biocompute object for clinical data, starting with NAACCR elements and machine learning
J. [t] Like the biocompute object for clinical data, starting with NAACCR elements and machine learning
K. [p] *Independent model description
L. [p] *Include model metadata
M. [p] Create a standard DUA or DUA template to enable sharing of models across silos (similar to UBMTA model)
N. [p] Characterizing barriers to having standard DUA
O. [p] Having published tiers of models to better inform users
P. [p] *DUA-free data – creation of synthetic data; reliably automating and making broadly available
Q. [p] Other metadata attached to model (e.g., coding rules and standards)
R. [p] Vendor: could model metadata be included in a way that any training/inference optimization can persist when a new practitioner grabs that shared model.
S. [p] Within a limited set of silos or silos that meet certain conditions (e.g., standards like NAACCR XML, DUAs, team leadership and responsiveness), can be reduce friction/barriers to moving models
T. [p] Standard DUAs for sharing models were built on identified datasets
U. [p] Mapping Pilot 3 models and data (registries, TIES) to NAACCR XML
V. [p] ID'ing individual metadata/search queries for your own models/applications, to disseminate to non-peers

        a. High level summary breaking down usage of model for non-peers
W. [i] Model clearinghouses
    a. Defining approval authority (tools)
    b. Describing model behavior without bias
X. [i] *Need a sharing platform (e.g. Dockstore) - the sharing platform could connect to CI in the backend for automated validation/benchmarking, etc.
Y. [i] Providing secure compute
Z. [i] Solutions.. mechanism for sharing/repos for both models, data and containers to reproduce — solutions.. automated methods for breaking models.
AA. [i] *Searchable catalog of models, data used, constraints

Disseminating models for reuse by nonexperts

A. [t] *Domains of applicability
B. [t] Understand how model topology changes through optimizations
C. [t] Having compiler that would make model more portable
D. [t] [i] Provide visualization of AIDL methods with documentation, and define how to document (e.g. workflows)
    a. Graphical representation of workflows
E. [t] when is it out of bounds
F. [t] Vendors: would like to understand how model topology changes (aka optimizations) will affect outcomes/inference (aka TensorRT)
G. [t] Vendors: would like understand how floating point precision and or hardware casting/accumulates/compiler optimizations may affect weights, outcomes
H. [p] *Education
I. [p] *Description of intended purposes
J. [p] *broad description of the model's domain
K. [i] Solutions.. packaging of models to run as function calls that support inference, updating (active and online learning), — solutions.. check-lists for publishing, requirements for self explanation and to output confident (validated) with predictions

Harnessing Artificial Intelligence and Machine Learning to Advanced Biomedical Research Workshop

This workshop, hosted by the NIH, brought in speakers from Amazon, Microsoft, MIT, Stanford, University of Cincinnati, and IBM to discuss what they were working on in artificial intelligence. The workshop started with an address from the NIH director, Dr. Francis Collins, who discussed the NIH's goal to pursue more AI and machine learning projects. The keynote speaker was Craig Mundie from Mundie and Associates, previously Microsoft. He talked about models that are being trained to compete in board games and video games that have no input from humans. They learn through trial and error and are able to outplay humans that are experts in the games. He proposed that training models without human input can be more accurate than those that are trained with data already analyzed by scientists.

A few of the presenters include Dr. Dina Katabi, Dr. Ronald Summers, and Dr. Judith Dexheimer. Dr. Katabi shared her current research at MIT with measuring electromagnetic waves emitted from wifi router like devices to detect patients breathing, heart rate, and daily activities. Dr. Summers discussed how machine learning has impacted radiology. Some current projects he talked about included detecting tumors in MRI images and measuring the volume of them, detect small and large polyps, and measure muscle segmentation. Dr. Dexheimer is from University of Cincinnati and Cincinnati Children's Hospital and talked about how machine learning applications are different when they are relating to pediatrics, mainly because medicine is drastically different in pediatrics. She also discussed electronic health records and how they will have to be altered considerably in order to be effective input for machine learning models. Natural language processing algorithms are not sophisticated enough to understand everything written in these documents.

Overall, this workshop was very comprehensive and covered a lot of different topics in the scope of machine learning. We have attended many conferences for machine learning in cancer research so it was interesting to see machine learning being used in many different subsets of biomedical research.

Other Lectures

Artificial Intelligence and Oncology Seminar

This week I also attended a seminar by Regina Barzilay titled "Learning to Cure Cancer". She talked about the work that she is doing at MIT to incorporate machine learning methods into cancer research. She discussed how her personal journey with breast cancer inspired her to do something about the lack of computer scientists who are working on curing cancer. Doctors continue to use a rather primitive method of diagnosing cancer despite the fact that machine learning has been used to predict outcomes for years in many other areas including business and economics. Decisions on how to diagnose people are what treatment to use are made by condensing the patient's information and making generalizations that aren't always accurate. Barzilay argues that medical decisions need to be made using all of the available information that comes from the raw data. She has created an artificial intelligence program that characterises patient records to reduce manual labor and allow scientists to easily search for information they need for their models. She also created a model that classifies mammogram images as fatty, dense, or scattered with a much greater accuracy than human radiologists. In addition, she is working on a model that will predict outcomes of ductal carcinoma diagnosis which can appear in all different forms, some benign and others extremely aggressive. Barzilay has been working to find ways to use her expertise to help classify cancer data and increase the accuracy with which doctors diagnose cancer. While she continues her work, she is also encouraging others to join her and help create more models that will extract more precise conclusions from data.

Modelhub: Plug & Predict solutions for medical AI research

There was a WebX meeting this week about Model Hub, which is an online, open source, site that allows users to search machine learning models and test them out. They call it a plug and predict solutions for machine reproducible AI research. There are AI applications that are starting to reach the same level as human analysis, specifically in image classification. Researchers working with medical images are starting to lean more towards machine learning algorithms for segmentation, reconstruction, and registration. It is becoming easier for them to do so due to the rise in open source tools like caffe, keras, theano, tensorflow, etc. Having these open source tools are great but there also needs to be a focus on the reproducibility of models. There are some current solutions to this problem like github, or other online repositories like niftynet, DLTK, Tomat, and Deepinfer. Modelhub provides another solution to this problem. The goals of model hub is to create a place where users can share their models and test out others. It is all open source and is made to be as intuitive as possible so that a wide range of users can utilize it. User are able to take models on the site, and put them through a framework which supplies a runtime environment, and frontend. This allows user to test out models with their own data and explore the model. Output data types include vectors, mask images, heat maps, and images, all in numpy format, and label lists in JSON formatting. Modelhub will help the public be able to create and share machine learning models so that they can explore models and find which one fits their data best.

Code

SEER Data Test Case

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: birdsellmh

Given age, sex, race, and year predicts likelihood of expected survival based on model trained on SEER database
"""

import pandas as pds
import numpy as np
from keras.models import Sequential
from keras.layers import Dense


### Data
xValues = pds.read_csv('/Users/birdsellmh/Code/Expected Survival/SurvivalData.csv', usecols=[1, 2, 3, 4])
yValues = pds.read_csv('/Users/birdsellmh/Code/Expected Survival/SurvivalData.csv', usecols=[5])
print(xValues)
yValues = round(yValues/1000000, 2)
print(yValues)

xTrain = xValues[1:43000]
yTrain = yValues[1:43000]
xTest = xValues[1:43000]
yTest = yValues[1:43000]

xTrain = xTrain[5::50]
yTrain = yTrain[5::50]
xTest = xValues[1::100]
yTest = yValues[1::100]


### Model
```

```
np.random.seed(128)

model = Sequential()
model.add(Dense(50, input_shape=(4,), init='uniform', activation='sigmoid'))
model.add(Dense(50, init='uniform', activation='sigmoid'))
model.add(Dense(50, init='uniform', activation='sigmoid'))
model.add(Dense(1, init='uniform', activation='sigmoid')
model.summary()

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(xTrain, yTrain, batch_size=50, nb_epoch=5)


### Predict
age = int(input('Enter age: '))
sex = int(input('Enter sex (1=Male 2=Female): '))
race = int(input('Enter race (1=White, 2=Black, 3=Other(American Indian/AK Native, Asian/Pacific Islander), 7=Other unspecified, 9=Unknown): '))
year = int(input('Enter year: '))

toPredict1 = np.array([age, sex, race, year])
toPredict = np.array([toPredict1])
prediction = model.predict(np.array(toPredict))
print('Survival rate: ', prediction)
```

<u>Function Optimizer Test Case</u>

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 16 10:16:18 2018

@author: birdsellmh

Reads in 8 numbers between -1 and 1 and predicts the result of f(x1, x2, x3, x4, x5, x6, x7, x8) = x1*x2
+ x2*x3 - x3*x4 + x5*x6 - x7*x8
```

```
"""

import random
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
import  keras

### Data setup
i = 0

x1 = random.randint(-1, 1)
x2 = random.randint(-1, 1)
x3 = random.randint(-1, 1)
x4 = random.randint(-1, 1)
x5 = random.randint(-1, 1)
x6 = random.randint(-1, 1)
x7 = random.randint(-1, 1)
x8 = random.randint(-1, 1)
xTrain = np.array([x1, x2, x3, x4, x5, x6, x7, x8])

y = x1 * x2 + x2 * x3 - x3 * x4 + x5 * x6 - x7 * x8
yTrain = np.array([y])

while i < 10000:

        x1 = random.randint(-1, 1)
        x2 = random.randint(-1, 1)
        x3 = random.randint(-1, 1)
        x4 = random.randint(-1, 1)
        x5 = random.randint(-1, 1)
        x6 = random.randint(-1, 1)
        x7 = random.randint(-1, 1)
        x8 = random.randint(-1, 1)
        xValues = np.array([x1, x2, x3, x4, x5, x6, x7, x8])

        xTrain = np.vstack((xTrain, xValues))

        y = x1 * x2 + x2 * x3 - x3 * x4 + x5 * x6 - x7 * x8
```

```
    yValues = np.array([y])

    yTrain = np.vstack((yTrain, yValues))

    i = i + 1

xTest = xTrain[9000:10000]
yTest = yTrain[9000:10000]
yTrain = yTrain[0:8999]
xTrain = xTrain[0:8999]

### Model
model = Sequential()
act = keras.layers.advanced_activations.ELU(alpha=6.0)
model.add(Dense(50, input_shape=(8,), init='uniform'))
model.add(act)
model.add(Dense(50, init='uniform'))
model.add(act)
model.add(Dense(50, init='uniform'))
model.add(act)
model.add(Dense(1, init='uniform'))
model.add(act)
model.summary()

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
history = model.fit(xTrain, yTrain, batch_size=50, nb_epoch=20)

##plot data
import matplotlib.pyplot as plt

print (history.history.keys())
plt.figure(figsize=[8,6])
plt.plot(history.history['acc'],'r',linewidth=3.0)
plt.legend(['Training Accuracy'],fontsize=12, loc=4)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=16)

```

```
### Prediction
x1 = int(input('Enter 1st X value: '))
x2 = int(input('Enter 2nd X value: '))
x3 = int(input('Enter 3rd X value: '))
x4 = int(input('Enter 4th X value: '))
x5 = int(input('Enter 5th X value: '))
x6 = int(input('Enter 6th X value: '))
x7 = int(input('Enter 7th X value: '))
x8 = int(input('Enter 8th X value: '))
print()

toPredict1 = np.array([x1, x2, x3, x4, x5, x6, x7, x8])
toPredict = np.array([toPredict1])
prediction = model.predict(np.array(toPredict))

if -4.5 > prediction[0]:
    print('Prediction: -5')
if -3.5 > prediction[0] > -4.5:
    print('Prediction: -4')
if -2.5 > prediction[0] > -3.5:
    print('Prediction: -3')
if -1.5 > prediction[0] > -2.5:
    print('Prediction: -2')
if -1.5 < prediction[0] < -0.5:
    print('Prediction: -1')
if -0.5 < prediction[0] < 0.5:
    print('Prediction: 0')
if 1.5 > prediction[0] > 0.5:
    print('Prediction: 1')
if 1.5 < prediction[0] < 2.5:
    print('Prediction: 2')
if 2.5 < prediction[0] < 3.5:
    print('Prediction: 3')
if 3.5 < prediction[0] < 4.5:
    print('Prediction: 4')
if 4.5 < prediction[0]:
    print('Prediction: 5')

Actual = x1 * x2 + x2 * x3 - x3 * x4 + x5 * x6 - x7 * x8
```

```
print('Actual:', Actual)
```

Patient No-Show Test Case

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: birdsellmh

Given patient data model predicts likelihood of a patient to show-up to an appointment based on model trained by data from kaggle.com
"""

import pandas as pds
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense

dataframeX = pds.read_csv(r'C:\Users\sjbir\Downloads\anaconda\data.csv', usecols=[2, 5, 7, 8, 9, 10, 11, 12])
dataframeY = pds.read_csv(r'C:\Users\sjbir\Downloads\anaconda\data.csv', usecols=[13])

def genderToInt(gender):
    if gender == 'M':
        return 0
    else:
        return 1

def NoShowToInt(NoShow):
    if NoShow == 'No':
        return 1
    else:
        return 0

dataframeX.Gender = dataframeX.Gender.apply(genderToInt)
dataframeY.NoShow = dataframeY.NoShow.apply(NoShowToInt)
```

```

#print(dataframeX.head())
#print(dataframeY.head())

trainX = dataframeX[50000:100000]
testX = dataframeX[100001:110000]
trainY = dataframeY[50000:100000]
testY = dataframeY[100001:110000]
print(trainX)
print(testX)
print(trainY)
print(testY)
#print(dataframeX.head())
#print(dataframeY.head())

np.random.seed(128)

model = Sequential()
model.add(Dense(87, input_shape=(8,), init='uniform', activation='linear'))
model.add(Dense(87, init='uniform', activation='linear'))
model.add(Dense(1, init='uniform', activation='linear'))
model.summary()

tbCallBack = keras.callbacks.TensorBoard(log_dir='/tmp/keras_logs', write_graph=True)

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(trainX, trainY, batch_size=50, nb_epoch=10, verbose=1, callbacks=[tbCallBack])
#score = model.evaluate(testX, testY, verbose=0)

toPredict = dataframeX[110001:110528]

prediction = model.predict(np.array(toPredict))
print(prediction)
```

K Means Clustering

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri May 18 08:47:25 2018

@author: warnementcm
"""

##Code for graphing data in scatter plot

from copy import deepcopy
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

plt.rcParams['figure.figsize'] = (9, 9)
plt.style.use('ggplot')

# Importing the dataset
data = pd.read_csv('/Users/warnementcm/Datasets/Breast_cancer_data.csv')
print(data.shape)
data.head()

# Getting the values and plotting it
f1 = data['ID'].values
f2 = data['Expression'].values
X = np.array(list(zip(f1, f2)))
plt.scatter(f1, f2, c='blue', s=10)
```

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri May 18 08:22:33 2018

@author: warnementcm
"""
```

```

##alternate clustering algorithm that produced empty graph

from copy import deepcopy
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize'] = (9, 9)
plt.style.use('ggplot')

# Importing the dataset
data = pd.read_csv('/Users/warnementcm/Datasets/Breast_cancer_data.csv')
print(data.shape)
data.head()

# Getting the values and plotting it
f1 = data['ID'].values
f2 = data['Expression'].values
X = np.array(list(zip(f1, f2)))
plt.scatter(f1, f2, c='blue', s=10)

##KMeans algorithm
K = 3
kmeans_model = KMeans(n_clusters=K).fit(X)

print(kmeans_model.cluster_centers_)
centers = np.array(kmeans_model.cluster_centers_)

plt.plot()
plt.title('k means centroids')

for i, l in enumerate(kmeans_model.labels_):
    plt.plot(f1[i], f2[i],ls='None')
    plt.xlim([0, 10])
    plt.ylim([0, 10])

plt.scatter(centers[:,0], centers[:,1], marker="x", color='r')
plt.show()
```

Lymphoma Subtype Classification

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Thu Jun  7 08:25:42 2018
@author: birdsellmh
"""

from random import shuffle
import glob
hdf5_path = '/Users/birdsellmh/Code/Lymphoma Subtype Classification/dataset.hdf5'

### Import images from 3 files and create list of images ###
CLLPath = '/Users/birdsellmh/Desktop/lymphoma/CLL/*.tif'
MCLPath = '/Users/birdsellmh/Desktop/lymphoma/MCL/*.tif'
FLPath = '/Users/birdsellmh/Desktop/lymphoma/FL/*.tif'

imagesCLL = glob.glob(CLLPath)
imagesMCL = glob.glob(MCLPath)
imagesFL = glob.glob(FLPath)


### Create labels corresponding to image data ###
labelsCLL = [0]
i = 1
while i < 113:
    labelsCLL.append(0)
    i = i + 1

labelsMCL = [1]
i = 1
while i < 122:
    labelsMCL.append(1)
    i = i + 1

labelsFL = [2]
```

```
i = 1
while i < 139:
    labelsFL.append(2)
    i = i + 1


### Join lists for images and labels ###
images = imagesCLL
images.extend(imagesMCL)
images.extend(imagesFL)
labels = labelsCLL
labels.extend(labelsMCL)
labels.extend(labelsFL)


### Zip lists, shuffle data, unzip lists ###
data = list(zip(images, labels))
#print(images)
#print(labels)
#print(data)

#shuffle data
shuffle(data)
#print(data)

images, labels = zip(*data)
#print(images)
#print(labels)


#Divide the data into 60% train, 20% validation, and 20% test
trainImages = images[0:int(0.6*len(images))]
trainLabels = labels[0:int(0.6*len(labels))]
valImages = images[int(0.6*len(images)):int(0.8*len(images))]
valLabels = labels[int(0.6*len(images)):int(0.8*len(images))]
testImages = images[int(0.8*len(images)):]
testLabels = labels[int(0.8*len(labels)):]

#print(trainImages)
```

```
#print(trainLabels)
#print(valImages)
#print(valLabels)
#print(testImages)
#print(testLabels)
:, 0:P] / F_MAX

#test  =  (pd.read_csv('breast.test.csv').values).astype('float32')  #get
and format testing data
#X_test = test[:, 0:P] / F_MAX

ae.compile(optimizer='rmsprop', loss='mean_squared_error')

ae.fit(X_train, X_train,
       batch_size=BATCH,
          epochs=EPOCH,
          validation_split=0.2)
```

DLModel_Version4

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Jul  3 13:06:25 2018

@author: birdsellmh
"""

import random
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
import  keras

### Data setup
i = 0

x1 = random.randint(-1, 1)
x2 = random.randint(-1, 1)
x3 = random.randint(-1, 1)
```

```
x4 = random.randint(-1, 1)
x5 = random.randint(-1, 1)
x6 = random.randint(-1, 1)
x7 = random.randint(-1, 1)
x8 = random.randint(-1, 1)
xTrain = np.array([x1, x2, x3, x4, x5, x6, x7, x8])

y = x1 * x2 + x2 * x3 - x3 * x4 + x5 * x6 - x7 * x8
yTrain = np.array([y])

while i < 10000:

        x1 = random.randint(-1, 1)
        x2 = random.randint(-1, 1)
        x3 = random.randint(-1, 1)
        x4 = random.randint(-1, 1)
        x5 = random.randint(-1, 1)
        x6 = random.randint(-1, 1)
        x7 = random.randint(-1, 1)
        x8 = random.randint(-1, 1)
        xValues = np.array([x1, x2, x3, x4, x5, x6, x7, x8])

        xTrain = np.vstack((xTrain, xValues))

        y = x1 * x2 + x2 * x3 - x3 * x4 + x5 * x6 - x7 * x8
        yValues = np.array([y])

        yTrain = np.vstack((yTrain, yValues))

        i = i + 1

xTest = xTrain[9000:10000]
yTest = yTrain[9000:10000]
yTrain = yTrain[0:8999]
xTrain = xTrain[0:8999]

### Model
model = Sequential()
act = keras.layers.advanced_activations.ELU(alpha=6.0)
```

```
model.add(Dense(50, input_shape=(8,), init='uniform'))
model.add(act)
model.add(Dense(50, init='uniform'))
model.add(act)
model.add(Dense(50, init='uniform'))
model.add(act)
model.add(Dense(1, init='uniform'))
model.add(act)
model.summary()

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
history = model.fit(xTrain, yTrain, batch_size=50, nb_epoch=20)

### Compiling data to transfer to second model
def load_data():

    x1_predict = random.randint(-1, 1)
    x2_predict = random.randint(-1, 1)
    x3_predict = random.randint(-1, 1)
    x4_predict = random.randint(-1, 1)
    x5_predict = random.randint(-1, 1)
    x6_predict = random.randint(-1, 1)
    x7_predict = random.randint(-1, 1)
    x8_predict = random.randint(-1, 1)
    x_predict = np.array([x1_predict, x2_predict, x3_predict, x4_predict, x5_predict, x6_predict, x7_predict, x8_predict])

    toPredict1 = np.array([x1_predict, x2_predict, x3_predict, x4_predict, x5_predict, x6_predict, x7_predict, x8_predict])
    toPredict = np.array([toPredict1])

    y_predict = np.array([model.predict(np.array(toPredict))])

    i = 0

    while i < 10000:

        x1_predict = random.randint(-1, 1)
```

```
            x2_predict = random.randint(-1, 1)
            x3_predict = random.randint(-1, 1)
            x4_predict = random.randint(-1, 1)
            x5_predict = random.randint(-1, 1)
            x6_predict = random.randint(-1, 1)
            x7_predict = random.randint(-1, 1)
            x8_predict = random.randint(-1, 1)
            xValues = np.array([x1_predict, x2_predict, x3_predict, x4_predict, x5_predict, x6_predict, x7_predict, x8_predict])
            x_predict = np.vstack((x_predict, xValues))

            #toPredict1 = np.array([x1_predict, x2_predict, x3_predict, x4_predict, x5_predict, x6_predict, x7_predict, x8_predict])
            #toPredict = np.array([toPredict1])
            #yValues = np.array([model.predict(np.array(toPredict))])
            #y_predict = np.vstack((y_predict, yValues))

            i = i + 1

    y_predict = np.array([model.predict(x_predict)])
    y_predict = y_predict[0]

    j = 0
    while j < 10001:
        if -4.5 > y_predict[j]:
            y_predict[j] = -5
        if -3.5 > y_predict[j] > -4.5:
            y_predict[j] = -4
        if -2.5 > y_predict[j] > -3.5:
            y_predict[j] = -3
        if -1.5 > y_predict[j] > -2.5:
            y_predict[j] = -2
        if -1.5 < y_predict[j] < -0.5:
            y_predict[j] = -1
        if -0.5 < y_predict[j] < 0.5:
            y_predict[j] = 0
        if 1.5 > y_predict[j] > 0.5:
            y_predict[j] = 1
        if 1.5 < y_predict[j] < 2.5:
            y_predict[j] = 2
        if 2.5 < y_predict[j] < 3.5:
            y_predict[j] = 3
        if 3.5 < y_predict[j] < 4.5:
            y_predict[j] = 4
        if 4.5 < y_predict[j]:
            y_predict[j] = 5
        j = j + 1

    print(y_predict)
    print(x_predict)

    return x_predict, y_predict
```

DLModel_Version4_pt2

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Jul  3 12:30:49 2018

@author: birdsellmh
"""

import numpy as np
from keras.models import Sequential
from keras.layers import Dense
import  keras

import DLModel_Version4

xValues, yValues = DLModel_Version4.load_data()


xTest = xValues[9000:10000]
yTest = yValues[9000:10000]
yTrain = yValues[0:8999]
xTrain = xValues[0:8999]

```

```
#print(xTest)
#print(xTrain)
#print(yTest)
#print(yTrain)

### Model
model = Sequential()
act = keras.layers.advanced_activations.ELU(alpha=6.0)
model.add(Dense(50, input_shape=(8,), init='uniform'))
model.add(act)
model.add(Dense(50, init='uniform'))
model.add(act)
model.add(Dense(50, init='uniform'))
model.add(act)
model.add(Dense(1, init='uniform'))
model.add(act)
model.summary()

model.compile(loss='mean_squared_error',          optimizer='adam',
metrics=['accuracy'])
history = model.fit(xTrain, yTrain, batch_size=50, nb_epoch=20)

##plot data
import matplotlib.pyplot as plt

print (history.history.keys())
plt.figure(figsize=[8,6])
plt.plot(history.history['acc'],'r',linewidth=3.0)
plt.legend(['Training Accuracy'],fontsize=12, loc=4)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=16)

### Prediction
x1 = int(input('Enter 1st X value: '))
x2 = int(input('Enter 2nd X value: '))
x3 = int(input('Enter 3rd X value: '))
x4 = int(input('Enter 4th X value: '))
x5 = int(input('Enter 5th X value: '))
```

```
x6 = int(input('Enter 6th X value: '))
x7 = int(input('Enter 7th X value: '))
x8 = int(input('Enter 8th X value: '))
print()

toPredict1 = np.array([x1, x2, x3, x4, x5, x6, x7, x8])
toPredict = np.array([toPredict1])
prediction = model.predict(np.array(toPredict))

if -4.5 > prediction[0]:
    print('Prediction: -5')
if -3.5 > prediction[0] > -4.5:
    print('Prediction: -4')
if -2.5 > prediction[0] > -3.5:
    print('Prediction: -3')
if -1.5 > prediction[0] > -2.5:
    print('Prediction: -2')
if -1.5 < prediction[0] < -0.5:
    print('Prediction: -1')
if -0.5 < prediction[0] < 0.5:
    print('Prediction: 0')
if 1.5 > prediction[0] > 0.5:
    print('Prediction: 1')
if 1.5 < prediction[0] < 2.5:
    print('Prediction: 2')
if 2.5 < prediction[0] < 3.5:
    print('Prediction: 3')
if 3.5 < prediction[0] < 4.5:
    print('Prediction: 4')
if 4.5 < prediction[0]:
    print('Prediction: 5')

Actual = x1 * x2 + x2 * x3 - x3 * x4 + x5 * x6 - x7 * x8
print('Actual:', Actual)
```

P1B2 Model Data Generator: Part 1

```
from __future__ import print_function

```

```
import numpy as np

from keras.models import Model, Sequential
from keras.layers import Dense, Dropout, Input
from keras.callbacks import Callback, ModelCheckpoint
from keras.regularizers import l2

import p1b2


BATCH_SIZE = 64
NB_EPOCH = 20                 # number of training epochs
PENALTY = 0.00001             # L2 regularization penalty
ACTIVATION = 'sigmoid'
FEATURE_SUBSAMPLE = None
DROP = None

L1 = 1024
L2 = 512
L3 = 256
L4 = 0
LAYERS = [L1, L2, L3, L4]


class BestLossHistory(Callback):
    def on_train_begin(self, logs={}):
        self.best_val_loss = np.Inf
        self.best_val_acc = -np.Inf
        self.best_model = None

    def on_epoch_end(self, batch, logs={}):
        if float(logs.get('val_loss', 0)) < self.best_val_loss:
            self.best_model = self.model
            self.best_val_loss = min(float(logs.get('val_loss', 0)), self.best_val_loss)
            self.best_val_acc = max(float(logs.get('val_acc', 0)), self.best_val_acc)


```

```
def extension_from_parameters():
    """Construct string for saving model with annotation of parameters"""
    ext = ''
    ext += '.A={}'.format(ACTIVATION)
    ext += '.B={}'.format(BATCH_SIZE)
    ext += '.D={}'.format(DROP)
    ext += '.E={}'.format(NB_EPOCH)
    if FEATURE_SUBSAMPLE:
        ext += '.F={}'.format(FEATURE_SUBSAMPLE)
    for i, n in enumerate(LAYERS):
        if n:
            ext += '.L{}={}'.format(i+1, n)
    ext += '.P={}'.format(PENALTY)
    return ext


def main():
    (X_train, y_train), (X_test, y_test) = p1b2.load_data(n_cols=FEATURE_SUBSAMPLE)

    X_train = np.vsplit(X_train, 2)
    X_train1 = X_train[0]

    X_test = np.vsplit(X_test, 2)
    X_test1 = X_test[0]

    y_train = np.vsplit(y_train, 2)
    y_train1 = y_train[0]

    y_test = np.vsplit(y_test, 2)
    y_test1 = y_test[0]

    X_predict = X_train[1]

    input_dim = X_train1.shape[1]
    output_dim = y_train1.shape[1]

    model = Sequential()

    model.add(Dense(LAYERS[0], input_dim=input_dim,
                    activation=ACTIVATION,
                    kernel_regularizer=l2(PENALTY),
                    activity_regularizer=l2(PENALTY)))

    for layer in LAYERS[1:]:
        if layer:
            if DROP:
                model.add(Dropout(DROP))
            model.add(Dense(layer, activation=ACTIVATION,
                            kernel_regularizer=l2(PENALTY),
                            activity_regularizer=l2(PENALTY)))

    model.add(Dense(output_dim, activation=ACTIVATION))

    model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    print(model.summary())

    ext = extension_from_parameters()
    checkpointer = ModelCheckpoint(filepath='model'+ext+'.h5', save_best_only=True)
    history = BestLossHistory()

    model.fit(X_train1, y_train1,
              batch_size=BATCH_SIZE,
              epochs=NB_EPOCH,
              validation_split=0.2,
              callbacks=[history, checkpointer])

    y_pred = history.best_model.predict(X_test1)

    print('best_val_loss={:.5f} best_val_acc={:.5f}'.format(history.best_val_loss, history.best_val_acc))
    print('Best model saved to: {}'.format('model'+ext+'.h5'))

    scores = p1b2.evaluate(y_pred, y_test1)
    print('Evaluation on test data:', scores)

    y_predict = history.best_model.predict(X_predict)

    submission = {'scores': scores,
                  'model': model.summary(),
                  'submitter': 'Developer Name' }

    # print('Submitting to leaderboard...')
    # leaderboard.submit(submission)

    return X_predict, y_predict, X_test1, y_test1

if __name__ == '__main__':
    main()
```

P1B2 Model Data Generator - Version 2: Part 1

```
from __future__ import print_function

import numpy as np

from keras.models import Model, Sequential
from keras.layers import Dense, Dropout, Input
from keras.callbacks import Callback, ModelCheckpoint
from keras.regularizers import l2

import p1b2
import p1b2_baseline_keras2_Version2


BATCH_SIZE = 64
NB_EPOCH = 20                # number of training epochs
PENALTY = 0.00001             # L2 regularization penalty
ACTIVATION = 'sigmoid'
FEATURE_SUBSAMPLE = None
DROP = None

L1 = 1024
L2 = 512
```

```
L3 = 256
L4 = 0
LAYERS = [L1, L2, L3, L4]


class BestLossHistory(Callback):
    def on_train_begin(self, logs={}):
        self.best_val_loss = np.Inf
        self.best_val_acc = -np.Inf
        self.best_model = None

    def on_epoch_end(self, batch, logs={}):
        if float(logs.get('val_loss', 0)) < self.best_val_loss:
            self.best_model = self.model
            self.best_val_loss = min(float(logs.get('val_loss', 0)), self.best_val_loss)
            self.best_val_acc = max(float(logs.get('val_acc', 0)), self.best_val_acc)


def extension_from_parameters():
    """Construct string for saving model with annotation of parameters"""
    ext = ''
    ext += '.A={}'.format(ACTIVATION)
    ext += '.B={}'.format(BATCH_SIZE)
    ext += '.D={}'.format(DROP)
    ext += '.E={}'.format(NB_EPOCH)
    if FEATURE_SUBSAMPLE:
        ext += '.F={}'.format(FEATURE_SUBSAMPLE)
    for i, n in enumerate(LAYERS):
        if n:
            ext += '.L{}={}'.format(i+1, n)
    ext += '.P={}'.format(PENALTY)
    return ext


def main():

    X_train, y_train, X_test, y_test = p1b2_baseline_keras2.main()

    input_dim = X_train.shape[1]
    output_dim = y_train.shape[1]

    model = Sequential()

    model.add(Dense(LAYERS[0], input_dim=input_dim,
                    activation=ACTIVATION,
                    kernel_regularizer=l2(PENALTY),
                    activity_regularizer=l2(PENALTY)))

    for layer in LAYERS[1:]:
        if layer:
            if DROP:
                model.add(Dropout(DROP))
            model.add(Dense(layer, activation=ACTIVATION,
                            kernel_regularizer=l2(PENALTY),
                            activity_regularizer=l2(PENALTY)))

    model.add(Dense(output_dim, activation=ACTIVATION))

    model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    print(model.summary())

    ext = extension_from_parameters()
    checkpointer = ModelCheckpoint(filepath='model'+ext+'.h5', save_best_only=True)
    history = BestLossHistory()

    model.fit(X_train, y_train,
              batch_size=BATCH_SIZE,
              epochs=NB_EPOCH,
              validation_split=0.2,
              callbacks=[history, checkpointer])

    y_pred = history.best_model.predict(X_test)

```

```
        print('best_val_loss={:.5f} best_val_acc={:.5f}'.format(history.best_val_loss, history.best_val_acc))
    print('Best model saved to: {}'.format('model'+ext+'.h5'))

    scores = p1b2.evaluate(y_pred, y_test)
    print('Evaluation on test data:', scores)

    submission = {'scores': scores,
                  'model': model.summary(),
                  'submitter': 'Developer Name' }

    # print('Submitting to leaderboard...')
    # leaderboard.submit(submission)

if __name__ == '__main__':
    main()
```

P1B2 Model Data Generator - Version 2: Part 2

```
from __future__ import print_function

import numpy as np

from keras.models import Model, Sequential
from keras.layers import Dense, Dropout, Input
from keras.callbacks import Callback, ModelCheckpoint
from keras.regularizers import l2

import p1b2
import p1b2_baseline_keras2_Version3


BATCH_SIZE = 64
NB_EPOCH = 20                # number of training epochs
PENALTY = 0.00001             # L2 regularization penalty
ACTIVATION = 'sigmoid'
FEATURE_SUBSAMPLE = None
DROP = None

```

```
L1 = 1024
L2 = 512
L3 = 256
L4 = 0
LAYERS = [L1, L2, L3, L4]


class BestLossHistory(Callback):
    def on_train_begin(self, logs={}):
        self.best_val_loss = np.Inf
        self.best_val_acc = -np.Inf
        self.best_model = None

    def on_epoch_end(self, batch, logs={}):
        if float(logs.get('val_loss', 0)) < self.best_val_loss:
            self.best_model = self.model
            self.best_val_loss = min(float(logs.get('val_loss', 0)), self.best_val_loss)
            self.best_val_acc = max(float(logs.get('val_acc', 0)), self.best_val_acc)


def extension_from_parameters():
    """Construct string for saving model with annotation of parameters"""
    ext = ''
    ext += '.A={}'.format(ACTIVATION)
    ext += '.B={}'.format(BATCH_SIZE)
    ext += '.D={}'.format(DROP)
    ext += '.E={}'.format(NB_EPOCH)
    if FEATURE_SUBSAMPLE:
        ext += '.F={}'.format(FEATURE_SUBSAMPLE)
    for i, n in enumerate(LAYERS):
        if n:
            ext += '.L{}={}'.format(i+1, n)
    ext += '.P={}'.format(PENALTY)
    return ext


```

```
def main():
    X_train, y_train, X_test, y_test = p1b2_baseline_keras2_Version3.main()

    print('MODEL/TRAINING, PART 2: ')

    input_dim = X_train.shape[1]
    output_dim = y_train.shape[1]

    model = Sequential()

    model.add(Dense(LAYERS[0], input_dim=input_dim,
                    activation=ACTIVATION,
                    kernel_regularizer=l2(PENALTY),
                    activity_regularizer=l2(PENALTY)))

    for layer in LAYERS[1:]:
        if layer:
            if DROP:
                model.add(Dropout(DROP))
            model.add(Dense(layer, activation=ACTIVATION,
                            kernel_regularizer=l2(PENALTY),
                            activity_regularizer=l2(PENALTY)))

    model.add(Dense(output_dim, activation=ACTIVATION))

    model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    print(model.summary())

    ext = extension_from_parameters()
    checkpointer = ModelCheckpoint(filepath='model'+ext+'.h5', save_best_only=True)
    history = BestLossHistory()

    model.fit(X_train, y_train,
              batch_size=BATCH_SIZE,
              epochs=NB_EPOCH,
              validation_split=0.2,
                  callbacks=[history, checkpointer])

        print('TEST VALUES, PART 2: ')
        y_pred = history.best_model.predict(X_test)

                                                               print('best_val_loss={:.5f}
    best_val_acc={:.5f}'.format(history.best_val_loss,
    history.best_val_acc))
        print('Best model saved to: {}'.format('model'+ext+'.h5'))

        scores = p1b2.evaluate(y_pred, y_test)
        print('Evaluation on test data:', scores)

        submission = {'scores': scores,
                      'model': model.summary(),
                      'submitter': 'Developer Name' }

        # print('Submitting to leaderboard...')
        # leaderboard.submit(submission)

    if __name__ == '__main__':
    main()
```

Link Appendix

Python and Installation:

Anaconda

https://www.anaconda.com/download/#macos

CANDLE

https://github.com/ECP-CANDLE/Benchmarks/blob/master/README.setup.mac

Python

https://www.python.org/downloads/release/python-365/

Theano

http://deeplearning.net/software/theano/install.html

Machine Learning:

Activations

https://keras.io/activations/

Activation function

https://en.wikipedia.org/wiki/Activation_function

Adding “Color” to MRI (“preliminary research”)

https://www.nih.gov/news-events/nih-research-matters/adding-color-mri

Advanced activations

https://keras.io/layers/advanced-activations/

Alternate algorithm

https://pythonprogramminglanguage.com/kmeans-clustering-centroid/

Another data set

https://www.kaggle.com/joniarroba/noshowappointments

Basics of Image Classification with Keras

https://towardsdatascience.com/basics-of-image-classification-with-keras-43779a299c8b

Building Powerful Image Classification Models Using Very Little Data

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Datacamp

https://www.datacamp.com/

Data dictionary (SEER)

https://seer.cancer.gov/expsurvival/esdatadic.html

Deep Learning for digital pathology image analysis: A comprehensive tutorial with selected use cases
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4977982/
Deep Learning for digital pathology image analysis: A comprehensive tutorial with selected use cases ("link")
http://www.andrewjanowczyk.com/use-case-7-lymphoma-sub-type-classification/
Deep Learning for digital pathology image analysis: A comprehensive tutorial with selected use cases ("tutorial")
http://machinelearninguru.com/deep_learning/data_preparation/hdf5/hdf5.html
GEO database
https://www.ncbi.nlm.nih.gov/sites/GDSbrowser?acc=GDS5819#details
Getting Started with Machine Learning Using Python
https://opensource.com/article/17/5/python-machine-learning-introduction
GSM1599177
https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?view=data&acc=GSM1599177&id=4192&db=GeoDb_blob123
Image Classification Using Convolutional Neural Networks in Keras
https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/
Keras Autoencoder Example
https://blog.keras.io/building-autoencoders-in-keras.html
Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python
https://elitedatascience.com/keras-tutorial-deep-learning-in-python
K-Means Clustering
https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html
Linear Regression in Theano
https://roshansanthosh.wordpress.com/2015/02/22/linear-regression-in-theano/
Linear SVC Machine Learning SVM example with python
https://pythonprogramming.net/linear-svc-example-scikit-learn-svm-python/
Optimizers

https://keras.io/optimizers/

Pydot

https://askubuntu.com/questions/917030/how-to-install-pydot-and-graphviz

SEER Database

https://seer.cancer.gov/data/

Simple Image Classification Using Convolutional Neural Network - Deep learning in Python

https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

Tutorial (K-Means Clustering)

https://mubaris.com/2017/10/01/kmeans-clustering-in-python/

Use Keras Deep Learning Modules with Scikit-Learn in python

https://machinelearningmastery.com/use-keras-deep-learning-models-scikit-learn-python/

1970-2012 expected life table

https://seer.cancer.gov/expsurvival/US.1970thru2012.individual.years.txt

7 Steps to Mastering Machine Learning

https://www.kdnuggets.com/2015/11/seven-steps-machine-learning-python.html

CANDLE Benchmarks:

CANDLE Benchmarks

https://github.com/ECP-CANDLE/Benchmarks/blob/master/README.setup.mac

GDC Website

https://gdc.cancer.gov/about-gdc

NCI GDC Summary

https://docs.gdc.cancer.gov/Data/Data_Model/GDC_Data_Model/

NCI MAF Summary

https://docs.gdc.cancer.gov/Data/File_Formats/MAF_Format/

NT3

https://github.com/ECP-CANDLE/Benchmarks/tree/frameworks/Pilot1/NT3

Raw training data (“training”)

http://ftp.mcs.anl.gov/pub/candle/public/benchmarks/P1B2/P1B2.train.csv

Raw testing data (“testing”)

http://ftp.mcs.anl.gov/pub/candle/public/benchmarks/P1B2/P1B2.test.csv

SNP Database

https://www.ncbi.nlm.nih.gov/snp

Biowulf:

Amazon EC2

https://aws.amazon.com/ec2/

Amazon Machine Learning

https://aws.amazon.com/machine-learning/

Amazon 10 Minute Tutorials

https://aws.amazon.com/getting-started/tutorials/

Bash script

https://ryanstutorials.net/bash-scripting-tutorial/bash-script.php

Batch Upload Files to the Cloud (Amazon EC2)

https://aws.amazon.com/getting-started/tutorials/backup-to-s3-cli/

Biowulf

https://hpc.nih.gov/docs/userguide.html

Connecting to Biowulf

https://hpc.nih.gov/docs/connect.html

Globus

https://www.globus.org

“HPC @ NIH” Website

https://hpc.nih.gov/docs/userguide.html

Launch an AWS Deep Learning AMI (Amazon EC2)

https://aws.amazon.com/getting-started/tutorials/get-started-dlami/

Launch a Linux Virtual Machine (Amazon EC2)

https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/

Launch a Windows Virtual Machine (Amazon EC2)

https://aws.amazon.com/getting-started/tutorials/launch-windows-vm/

Modules

https://hpc.nih.gov/apps/modules.html

Online Course
https://hpc.nih.gov/training/intro_biowulf/
Presentations
https://hpc.nih.gov/training/handouts/171121_python_in_hpc.pdf
Python in Biowulf
https://hpc.nih.gov/docs/python.html
Singularity container
https://singularity.lbl.gov
Store and Retrieve a File (Amazon EC2)
https://aws.amazon.com/getting-started/tutorials/backup-files-to-amazon-s3/
Swarm function
https://hpc.nih.gov/apps/swarm.html
Tensorflow install guide
https://www.tensorflow.org/install/install_linux
User Dashboard
https://auth.nih.gov/CertAuthV2/forms/NIHPivOrFormLogin.aspx?TYPE=33554433&REALMOID=06-effe824d-683e-408e-962c-86fed36d3317&GUID=&SMAUTHREASON=0&METHOD=GET&SMAGENTNAME=-SM-aKmoe5cvpd5WMc7bZS8DqazGggT0l50j5WjH2pgUXDKXvzHEQOnbTYJNejNHZ%2bzw&TARGET=-SM-https%3a%2f%2fhpc%2enih%2egov%2fdashboard%2f
VIM basics ("this tutorial")
https://www.openvim.com

Databases:

Bioperl
https://bioperl.org
Bioperl BioseqIO
https://bioperl.org/howtos/SeqIO_HOWTO
Biopython
https://biopython.org
Biopython qualityIO ("here")

http://biopython.org/DIST/docs/api/Bio.SeqIO.QualityIO-module.html

Breast Cancer data analysis for survivability studies and prediction
https://www.sciencedirect.com/science/article/pii/S0169260717307551?via%3Dihub

cBioPortal

  http://www.cbioportal.org

CCLE

  https://portals.broadinstitute.org/ccle_legacy/home

CCLE register

  https://portals.broadinstitute.org/ccle_legacy/toa/termsOfAccess/23/

Classification of Cancer Primary Sites Using Machine Learning and Somatic Mutations

  https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4619847/

COSMIC

  https://cancer.sanger.ac.uk/cosmic

Deep-Learning Convolutional Neural Networks Accurately Classify Genetic Mutations in Gliomas

  http://www.ajnr.org/content/early/2018/05/10/ajnr.A5667.long

Detection of microcalcification in digitized mammograms with multistable cellular neural networks using a new image enhancement method: automated lesion intensity enhancer (ALIE)

  http://journals.tubitak.gov.tr/elektrik/issues/elk-15-23-3/elk-23-3-17-1303-139.pdf

DICOM data in python

  https://pyscience.wordpress.com/2014/09/08/dicom-in-python-importing-medical-image-data-into-numpy-with-pydicom-and-vtk/

ECO

  https://www.cancervic.org.au/research/projects/project_victorian_cancer_outco.html

ECO data request

  https://www.cancervic.org.au/research/registry-statistics

FASTA

  https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp

FastQC

  http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

GDC Data Portal

  https://portal.gdc.cancer.gov

Genomic Data Commons

  https://portal.gdc.cancer.gov

Gene expression inference with deep learning https://academic.oup.com/bioinformatics/article/32/12/1832/1743989

GEO

https://www.ncbi.nlm.nih.gov/geo/

GEO database organization

https://www.ncbi.nlm.nih.gov/geo/info/overview.html

Handling conflicting distutils libraries (“source”)

https://stackoverflow.com/questions/49916736/how-to-properly-handle-conflicting-distutils-libraries-with-pip?rq=1

ICGC

https://dcc.icgc.org

JSON

https://docs.python.org/3/library/json.html

Machine Learning Prediction of cancer survival: a retrospective study using electronic administration records and a cancer registry

https://bmjopen.bmj.com/content/4/3/e004007.long

MIAS

http://www.mammoimage.org/databases/

PDB

http://www.wwpdb.org

PDB PDBe

https://www.ebi.ac.uk/pdbe/

PDB PDBj

https://pdbj.org

PDB RCSB

https://www.rcsb.org

pyDICOM (“another method”)

https://pydicom.github.io/pydicom/stable/getting_started.html

SRA

https://www.ncbi.nlm.nih.gov/sra

SEER

https://seer.cancer.gov

TCGA

https://cancergenome.nih.gov

TCIA

http://www.cancerimagingarchive.net

TCIA Data app

https://docs.python.org/3/library/json.html

TCIA Data usage policies and restrictions https://wiki.cancerimagingarchive.net/display/Public/Data+Usage+Policies+and+Restrictions

TCPA

http://tcpaportal.org/tcpa/

VCR (Victorian Cancer Registry)

https://www.cancervic.org.au/research/registry-statistics

WHO

http://www.who.int/gho/database/en/

1000 Genome project

http://www.internationalgenome.org/data/

1000 Genome project server ("link")

ftp.1000genomes.ebi.ac.uk/vol1/ftp/

Readings:

A Deep Learning Framework for Gene Network Inference by Aggregating Biological Information

https://www.biorxiv.org/content/biorxiv/early/2018/04/13/300996.full.pdf

Applications of Machine Learning in Cancer Prediction and Prognosis

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2675494/

Assessing Breast Cancer Risk with an Artificial Neural Network

https://www.ncbi.nlm.nih.gov/pubmed/29693975

Autoencoders, Unsupervised Learning, and Deep Architectures

http://proceedings.mlr.press/v27/baldi12a/baldi12a.pdf

Automated Grading of Glioma Using Deep Learning in Digital Pathology Images: A Module Approach with Ensemble of Convolutional Neural Networks

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4765616/pdf/2243353.pdf

Automatic Detection of Invasive Ductal Carcinoma in Whole Slide Images with Convolution Neural Network

http://engineering.case.edu/centers/ccipd/sites/ccipd.case.edu/files/Automatic_detection_of_invasive_ductal_carcinoma_in_whole.pdf

Automated Nuclear Pleomorphism Scoring in Breast Cancer Histopathology Images using Deep Neural Networks

https://link.springer.com/chapter/10.1007/978-3-319-26832-3_26

BioGRID

https://thebiogrid.org/
Breast Cancer Data Analysis for Survivability Studies and Prediction
https://www.sciencedirect.com/science/article/pii/S0169260717307551?via%3Dihub
Chest Pathology Detection Using Deep Learning With Non-Medical Training
https://ieeexplore.ieee.org/document/7163871/
Classification of Cancer Primary Sites Using Machine Learning and Somatic Mutations
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4619847/
Classification of Cervical Cancer Using Artificial Neural Networks
https://www.sciencedirect.com/science/article/pii/S187705091631170X
Convolutional Auto-Encoder for Image Denoising of Ultra-Low-Dose CT
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5577435/
Correlation Isomap
https://www.worldscientific.com/doi/abs/10.1142/9789814447331_0040
COSMIC
https://cancer.sanger.ac.uk/cosmic
Deep Features Learning for Medical Image Analysis with Convolutional Autoencoder Neural Network
https://ieeexplore.ieee.org/document/7954012/
Deep Learning Convolutional Neural Networks Accurately Classify Genetic Mutations in Gliomas
http://www.ajnr.org/content/early/2018/05/10/ajnr.A5667.long
Deep Learning for Digital pathology Image Analysis: A Comprehensive tutorial with Selected use Cases
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4977982/
Deep Learning of Part-based Representations of Data Using Sparse Autoencoders with Non Negativity Constraints
https://arxiv.org/pdf/1601.02733.pdf
D-GEX
https://github.com/uci-cbcl/D-GEX
Digital Pathology Source Code
http://www.andrewjanowczyk.com/deep-learning/
DREAM5
http://dreamchallenges.org/project/dream-5-network-inference-challenge/

Gene Expression Inference with Deep Learning
https://academic.oup.com/bioinformatics/article/32/12/1832/1743989
Image Analysis and Machine Learning in Digital Pathology: Challenges and Opportunities
https://www.ncbi.nlm.nih.gov/pubmed/27423409
Investigation of the support vector machine algorithm to predict lung radiation-induced pneumonitis
https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.2776669
Learning for Classification of Colorectal Polyps on Whole-Slide Images
http://www.jpathinformatics.org/article.asp?issn=2153-3539;year=2017;volume=8;issue=1;spage=30;epage=30;aulast=Korbar
LINCS program
http://www.lincsproject.org/
LINE
https://dl.acm.org/citation.cfm?id=2741093
Machine Learning Algorithms for Outcome Prediction in (Chemo)Radiotherapy: An Empirical Comparison of Classifier
https://www.ncbi.nlm.nih.gov/pubmed/29763967
Machine Learning Approaches for Predicting Radiation Therapy Outcomes: A Clinician's Perspective
https://www.sciencedirect.com/science/article/pii/S0360301615030783
Machine Learning for Biomarker Identification in Cancer Research
https://www.futuremedicine.com/doi/pdf/10.2217/pme.15.5
Machine Learning in Digital Pathology: A Journey from Handcrafted Feature Descriptor to Deep Learning Approaches
http://info.definiens.com/blog/machine-learning-in-digital-pathology-a-journey-from-handcrafted-feature-descriptors-to-deep-learning-approaches
Machine-learning Prediction of Cancer Survival
http://bmjopen.bmj.com/content/4/3/e004007.long
MNIST
http://yann.lecun.com/exdb/mnist/
Multivariable modeling of radiotherapy outcomes, including dose–volume and clinical factors

https://www.sciencedirect.com/science/article/pii/S0360301605029718

Node2vec

https://dl.acm.org/citation.cfm?id=2939754

NORB

https://cs.nyu.edu/~ylclab/data/norb-v1.0/

ORL

http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

Predictive Models for Breast Cancer Susceptibility from Multiple Single Nucleotide Polymorphisms

http://clincancerres.aacrjournals.org/content/10/8/2725.long

Principal Component Analysis

https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

PubMed

https://www.ncbi.nlm.nih.gov/pubmed/

Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions

https://nlp.stanford.edu/pubs/SocherPenningtonHuangNgManning_EMNLP2011.pdf

Spatial Organization and Molecular Correlation of Tumor Infiltrating Lymphocytes Using Deep Learning on Pathology Images

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5943714/pdf/nihms958989.pdf

Use of artificial neural networks to predict biological outcomes for patients receiving radical radiotherapy of the prostate

https://www.sciencedirect.com/science/article/pii/S016781400300272X

Lectures and Conferences:

BioBank

http://www.ukbiobank.ac.uk/

cBioPortal

http://www.cbioportal.org/

dbGaP

https://www.ncbi.nlm.nih.gov/gap

DeepInfer

http://www.deepinfer.org/

DLTK
    https://github.com/DLTK/DLTK
GenBank
    https://www.ncbi.nlm.nih.gov/genbank/?
Kipoi
    http://kipoi.org/
ModelHub
    https://arxiv.org/abs/1611.06224
MoleculeNet
    http://moleculenet.ai/
NiftyNet
    http://niftynet.io/
NVIDA
    http://www.nvidia.com/content/global/global.php