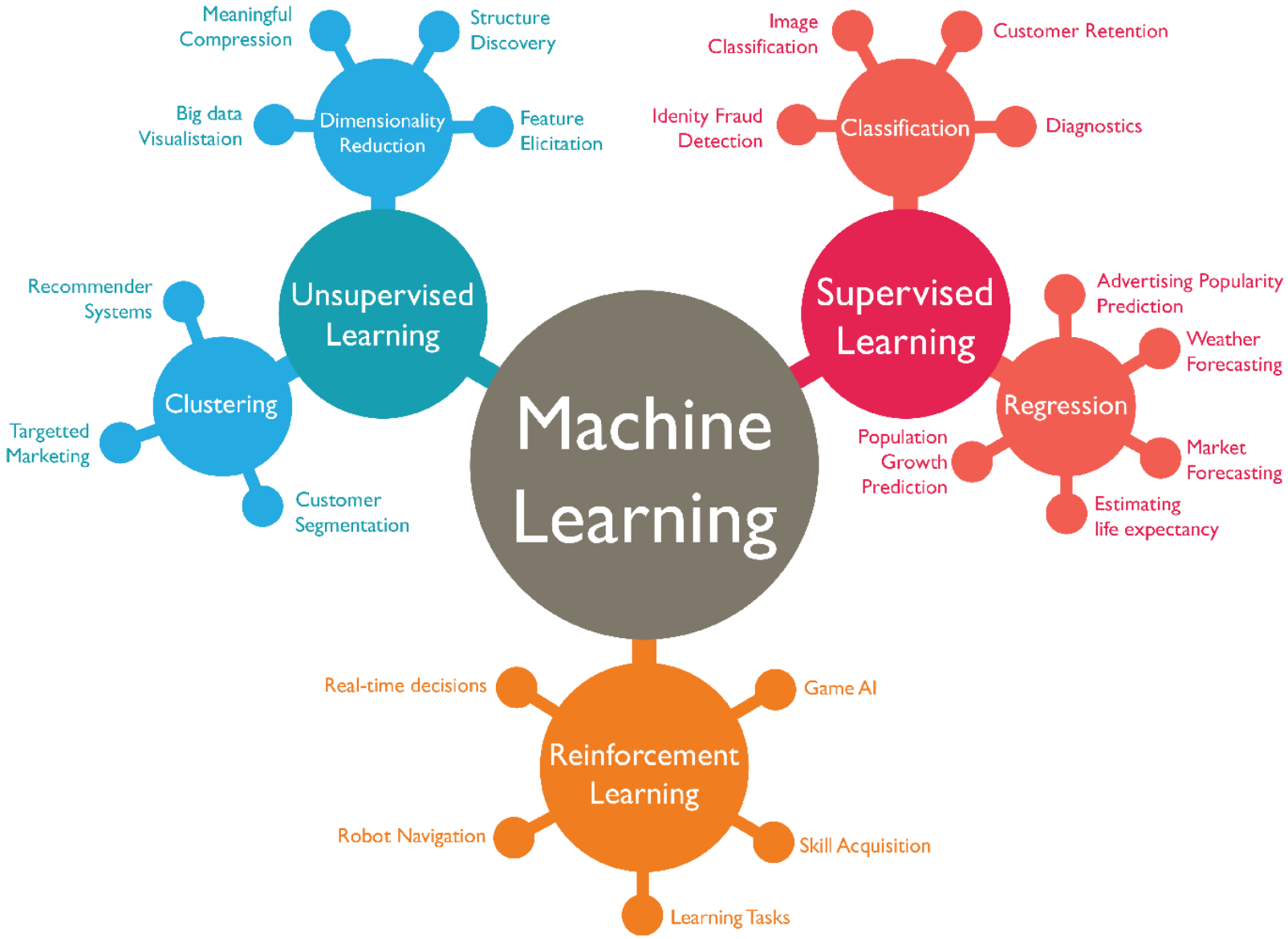
Machine Learning

+

Programme Grant Update Science Talk

Machine Learning Overview



Supervised Machine Learning

CCRF Cardiovascular Clinical Research Facility

General Idea:

- Given a dataset with known labels …
- “Train” a machine learning model using the dataset …
- Then used the trained model to make predictions for new data which do not have labels

Supervised Machine Learning

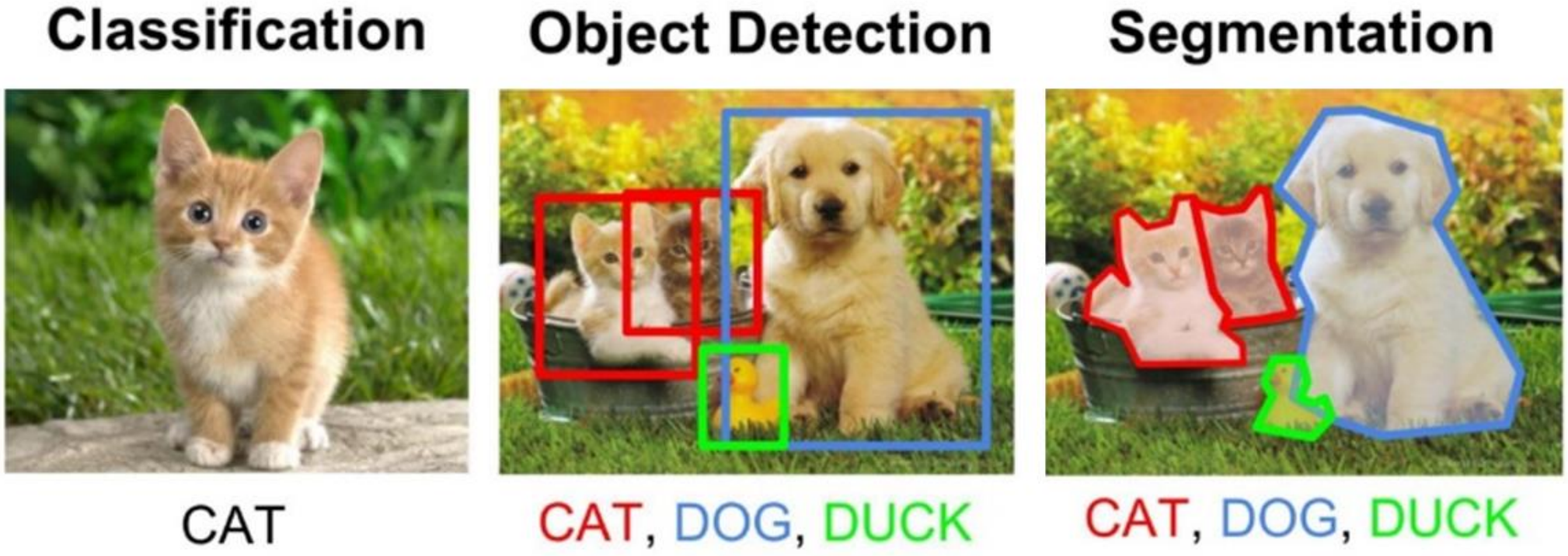
CCRF Cardiovascular Clinical Research Facility

Methods:

- Regression
- Random Forests, Support Vector Machines, K-Nearest Neighbor
- Deep Learning (Neural Networks)

Types of Supervised Learning Tasks

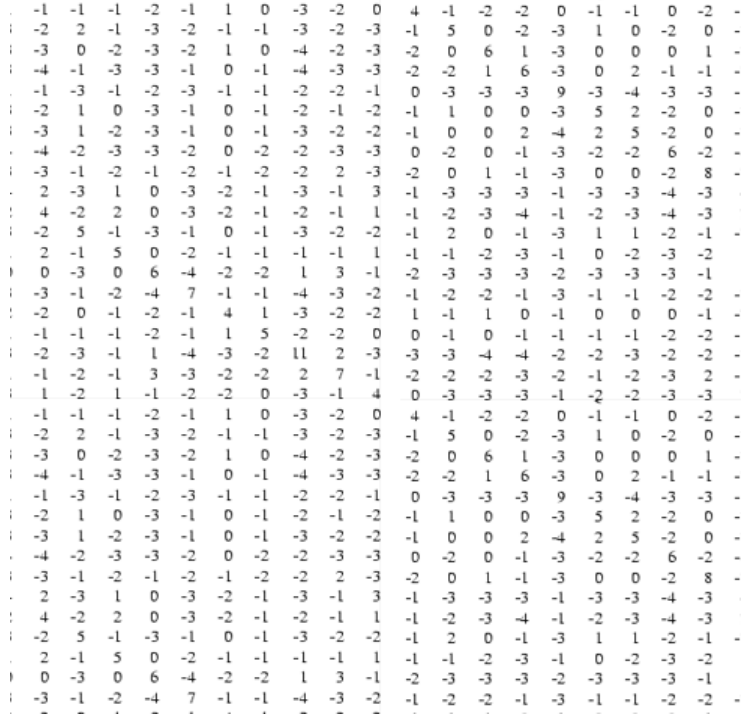
- Image related tasks are the most common problems for machine learning



Why These Tasks are Difficult?

What we see:

What the computer “sees”:

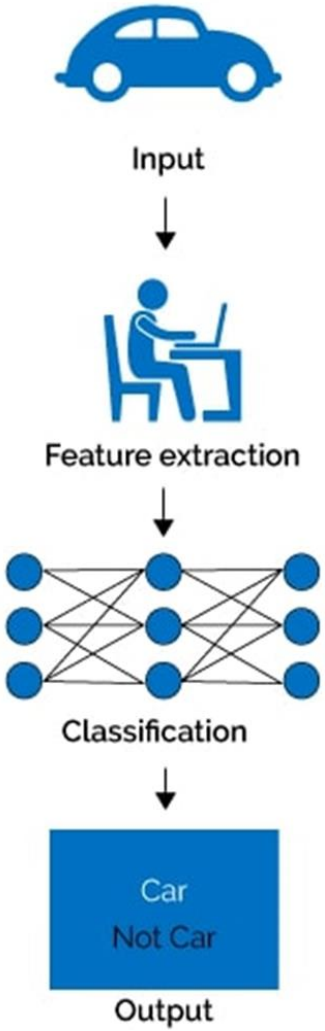


- It is difficult to come up with a set of “strict rules” to define image content
- Such tasks require the use of more sophisticated learning algorithms

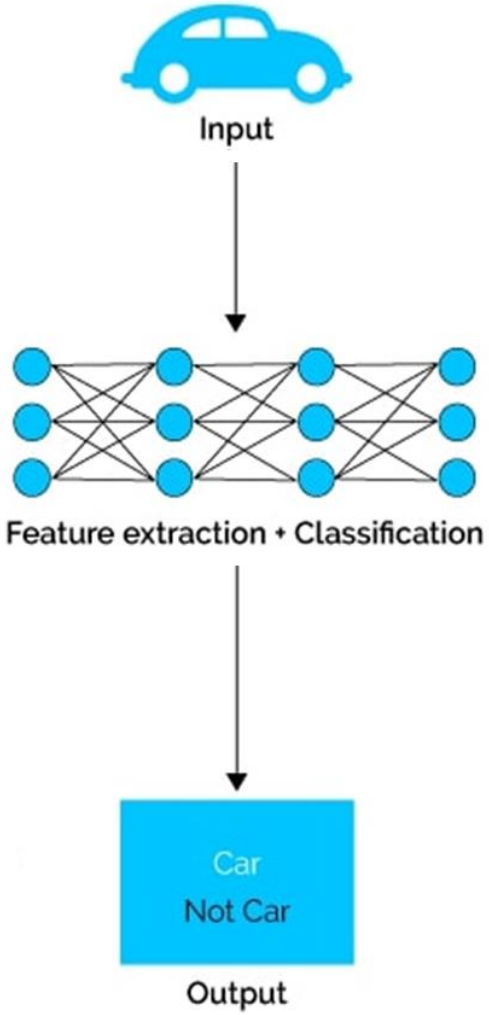
Machine Learning vs Deep Learning

CCRF Cardiovascular Clinical Research Facility

Machine Learning



Deep Learning



Neural Network Types

Simple Neural Networks:

- Used on 1-Dimensional Data

Convolutional Neural Networks:

- Used on 2D or 3D Data (Images, Volumes, Videos)

Recurrent Neural Networks:

- Used on Sequential Data (Text, Time-Series)

Neural Network Types

CCRF Cardiovascular Clinical Research Facility

Simple Neural Networks:

- Used on 1-Dimensional Data

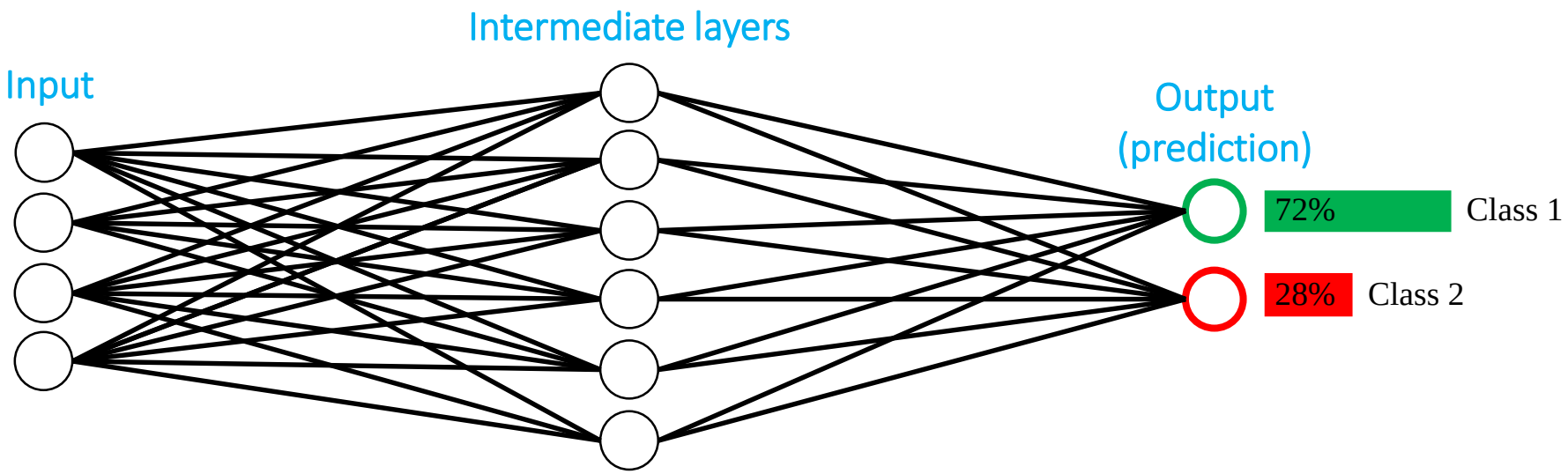
Convolutional Neural Networks:

- Used on 2D or 3D Data (Images, Volumes, Videos)

Recurrent Neural Networks:

- Used on Sequential Data (Text, Time-Series)

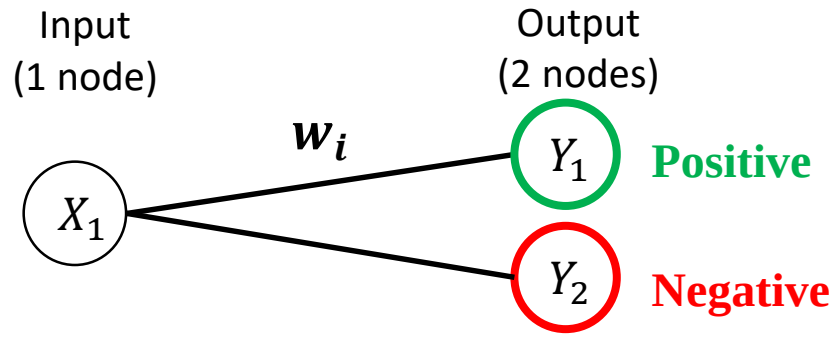
Introducing Neural Networks



- Given some **input** data,
- **Transform** the data through **intermediate** layers (number can be $N \geq 0$ layers) (we can also have as many nodes as we want for the intermediate layers),
- So the **output** becomes the probability of being in each class

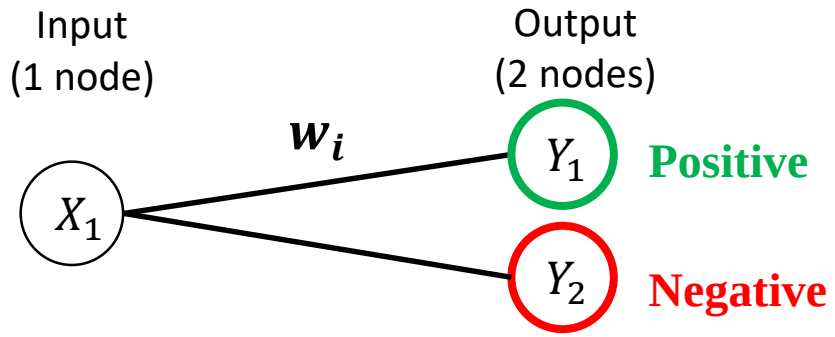
How Transformations are Performed?

Starting with a (very) simple neural network:



How Transformations are Performed?

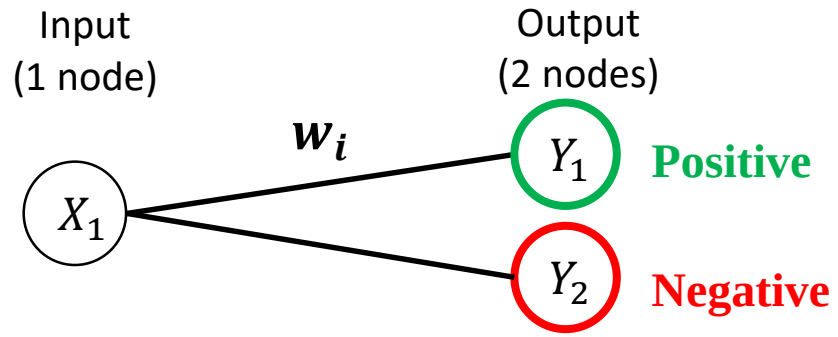
Starting with a (very) simple neural network:



TASK: given a single number at the input (X_1), predict if its **positive** or **negative** (at the output, Y).

How Transformations are Performed?

Starting with a (very) simple neural network:



TASK: given a single number at the input (X_1), predict if its **positive** or **negative** (at the output, Y).

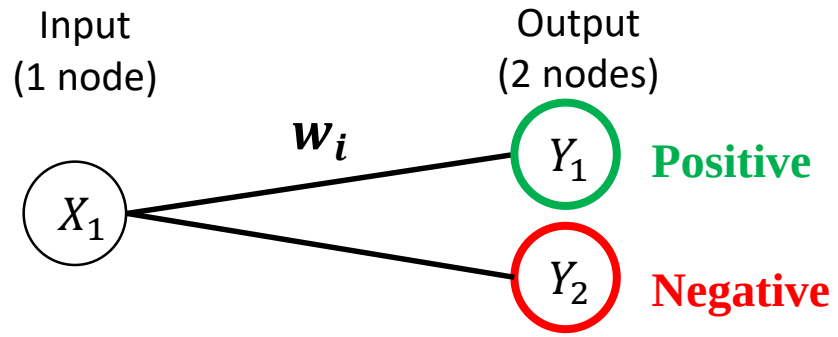
Ideally:

If $x \geq 0$, we would want $[Y_1 \quad Y_2] = [1 \quad 0]$ (or any $Y_1 > Y_2$)

If $x < 0$, we would want $[Y_1 \quad Y_2] = [0 \quad 1]$ (or any $Y_1 < Y_2$)

How Transformations are Performed?

Starting with a (very) simple neural network:



TASK: given a single number at the input (X_1), predict if its **positive** or **negative** (at the output, Y).

Ideally:

If $x \geq 0$, we would want $[Y_1 \quad Y_2] = [1 \quad 0]$ (or any $Y_1 > Y_2$)

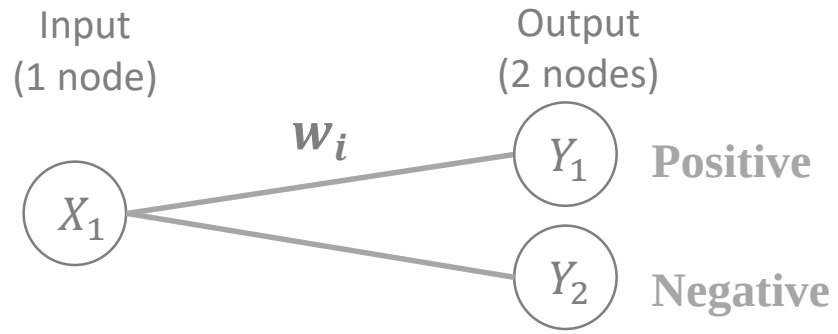
If $x < 0$, we would want $[Y_1 \quad Y_2] = [0 \quad 1]$ (or any $Y_1 < Y_2$)

To transform the input (1 node) to the output (2 nodes), we can perform a matrix multiplication:

$$[X_1][w_1 \quad w_2] = [Y_1 \quad Y_2]$$

How Transformations are Performed?

Starting with a (very) simple neural network:



TASK: given a single number at the input (X_1), predict if its **positive** or **negative** (at the output, Y).

Ideally:

If $x \geq 0$, we would want $[Y_1 \quad Y_2] = [1 \quad 0]$ (or any $Y_1 > Y_2$)

If $x < 0$, we would want $[Y_1 \quad Y_2] = [0 \quad 1]$ (or any $Y_1 < Y_2$)

To transform the input (1 node) to the output (2 nodes), we can perform a matrix multiplication:

$$[X_1][\boldsymbol{w_1} \quad \boldsymbol{w_2}] = [Y_1 \quad Y_2]$$

*How do we choose which **w**'s get our desired **Y**'s ?*

Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [Y_1 \quad Y_2]$$

- Set up an optimization problem to solve the values of ***w***'s.

Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [Y_1 \quad Y_2]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.

Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [Y_1 \quad Y_2]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.

Sample Training Set:

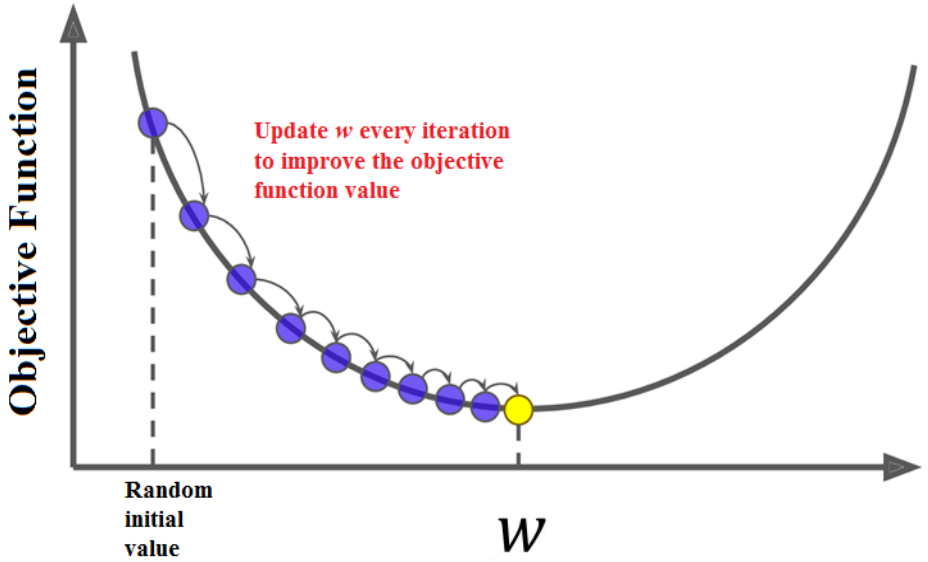
X:	$[3]$	$[-2]$	$[1]$	$[5]$	$[-1]$	$[-3]$	$[-2]$	$[2]$	$[1]$
Y:	$[1 \quad 0]$	$[0 \quad 1]$	$[1 \quad 0]$	$[1 \quad 0]$	$[0 \quad 1]$	$[0 \quad 1]$	$[0 \quad 1]$	$[1 \quad 0]$	$[1 \quad 0]$

Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [Y_1 \quad Y_2]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.

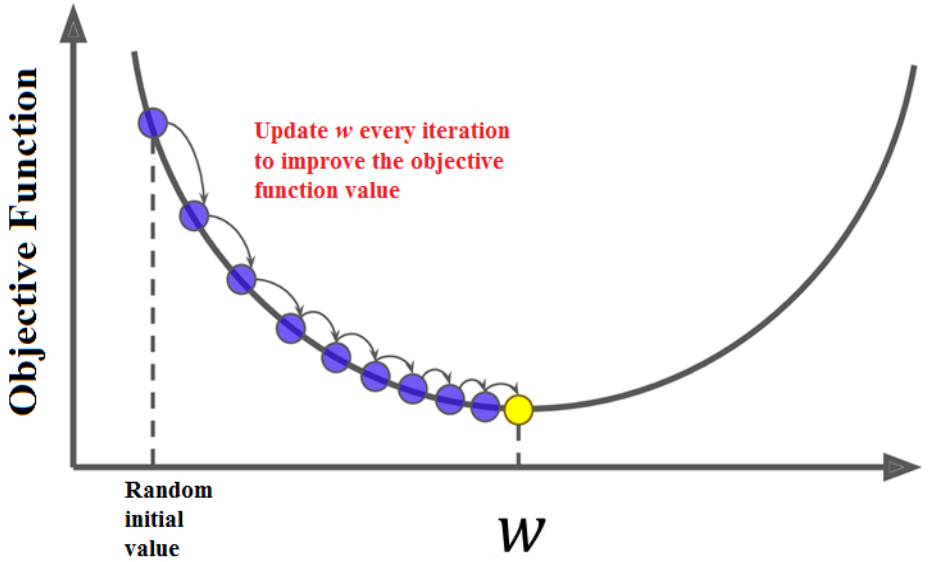


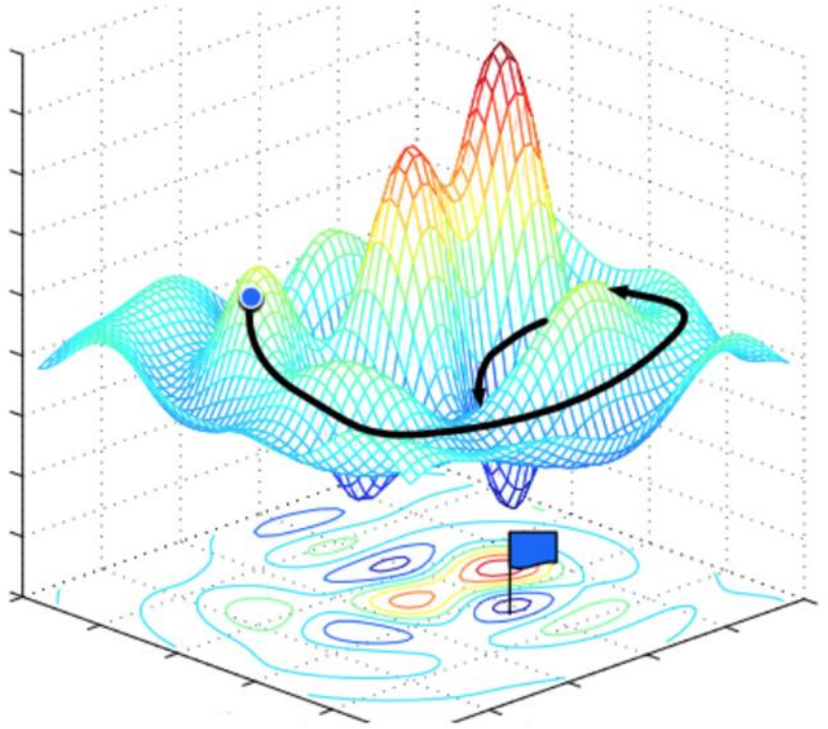
Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [Y_1 \quad Y_2]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.





Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [Y_1 \quad Y_2]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.

This simple problem has many solutions for ***w***, e.g. $[\boldsymbol{w_1} \quad \boldsymbol{w_2}] = [\boldsymbol{0.6} \quad \boldsymbol{0.4}]$(or any $w_1 > w_2$).

Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [\boldsymbol{Y_1} \quad \boldsymbol{Y_2}]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.

This simple problem has many solutions for ***w***, e.g. $[\boldsymbol{w_1} \quad \boldsymbol{w_2}] = [\boldsymbol{0.6} \quad \boldsymbol{0.4}]$(or any $w_1 > w_2$).

If we input $\boldsymbol{X = 1}$ and $\boldsymbol{X = -1}$:

$[1][\boldsymbol{0.6} \quad \boldsymbol{0.4}] = [\boldsymbol{0.6} \quad \boldsymbol{0.4}]$ $(Y_1 > Y_2$, higher value for **positive**)

$[-1][\boldsymbol{0.6} \quad \boldsymbol{0.4}] = [\boldsymbol{-0.6} \quad \boldsymbol{-0.4}]$ $(Y_1 < Y_2$, higher value for **negative**)

Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [\boldsymbol{Y_1} \quad \boldsymbol{Y_2}]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.

This simple problem has many solutions for ***w***, e.g. $[w_1 \quad w_2] = [0.6 \quad 0.4]$(or any $w_1 > w_2$).

If we input $\boldsymbol{X = 1}$:

$[1][\mathbf{0.6} \quad \mathbf{0.4}] = [\mathbf{0.6} \quad \mathbf{0.4}]$

Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [\boldsymbol{Y_1} \quad \boldsymbol{Y_2}]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.

This simple problem has many solutions for ***w***, e.g. $[w_1 \quad w_2] = [0.6 \quad 0.4]$(or any $w_1 > w_2$).

If we input $\boldsymbol{X = 1}$:

$[1][\boldsymbol{0.6} \quad \boldsymbol{0.4}] = [\boldsymbol{0.6} \quad \boldsymbol{0.4}]$

normalize: $\left[\frac{e^{0.6}}{e^{0.6}+e^{0.4}} \quad \frac{e^{0.4}}{e^{0.6}+e^{0.4}}\right] = [\boldsymbol{0.550} \quad \boldsymbol{0.450}]$

Training the Neural Network – An Optimization Problem

Equation from last slide:

$$[X_1][w_1 \quad w_2] = [\boldsymbol{Y_1} \quad \boldsymbol{Y_2}]$$

- Set up an optimization problem to solve the values of ***w***'s.
- Train the neural network with lots of ***X***'s and ***Y***'s so it can incrementally update ***w***'s with gradient descent each iteration.

This simple problem has many solutions for ***w***, e.g. $[w_1 \quad w_2] = [0.6 \quad 0.4]$(or any $w_1 > w_2$).

If we input $\boldsymbol{X = 1}$ and $\boldsymbol{X = -1}$:

$[1][\boldsymbol{0.6} \quad \boldsymbol{0.4}] = [\boldsymbol{0.6} \quad \boldsymbol{0.4}]$

normalize: $\left[\frac{e^{0.6}}{e^{0.6}+e^{0.4}} \quad \frac{e^{0.4}}{e^{0.6}+e^{0.4}}\right] = [\boldsymbol{0.550} \quad \boldsymbol{0.450}]$

$[-1][\boldsymbol{0.6} \quad \boldsymbol{0.4}] = [\boldsymbol{-0.6} \quad \boldsymbol{-0.4}]$

normalize: $\left[\frac{e^{-0.6}}{e^{-0.6}+e^{-0.4}} \quad \frac{e^{-0.4}}{e^{-0.6}+e^{-0.4}}\right] = [\boldsymbol{0.450} \quad \boldsymbol{0.550}]$

Question?

For our neural network:

$$[\boldsymbol{X_1}][\boldsymbol{w_1} \quad \boldsymbol{w_2}] = [\boldsymbol{Y_1} \quad \boldsymbol{Y_2}]$$

We have weights of:

$$[w_1 \quad w_2] = [0.6 \quad 0.4]$$

- What is our output if the input is $X_1 = 0$?

Question?

For our neural network:

$$[\boldsymbol{X_1}][\boldsymbol{w_1} \quad \boldsymbol{w_2}] = [\boldsymbol{Y_1} \quad \boldsymbol{Y_2}]$$

We have weights of:

$$[w_1 \quad w_2] = [0.6 \quad 0.4]$$

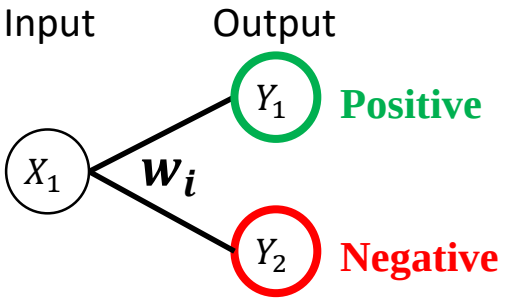
- What is our output if the input is $X_1 = 0$?

$$[\boldsymbol{X_1}][\boldsymbol{w_1} \quad \boldsymbol{w_2}] = [0][0 \quad 0]$$

$$[\boldsymbol{Y_1} \quad \boldsymbol{Y_2}] = \left[\frac{e^0}{e^0 + e^0} \quad \frac{e^0}{e^0 + e^0}\right] = [0.5 \quad 0.5]$$

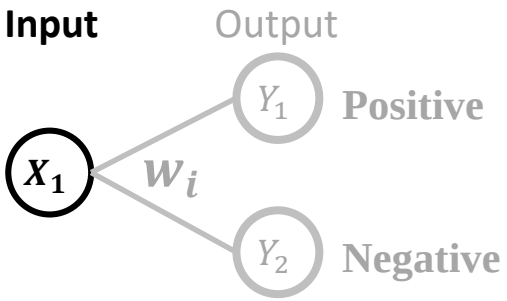
Coding the Example in Python

```
# import the packages we will be needing
import tflearn
```



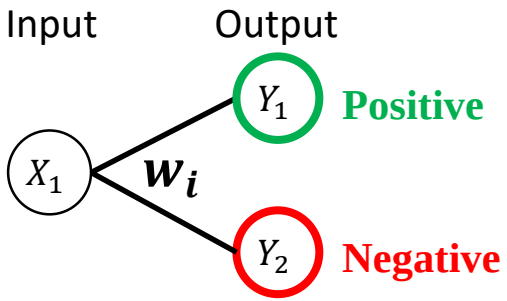
Coding the Example in Python

```
# import the packages we will be needing
import tflearn

# define the input layer (1 node)
X_i = tflearn.input_data(shape=[None, 1])
```



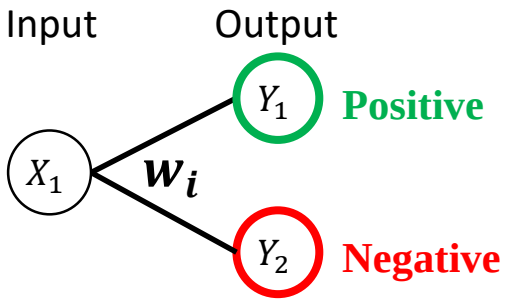
Coding the Example in Python

```
# import the packages we will be needing
import tflearn

# define the input layer (1 node)
X_i = tflearn.input_data(shape=[None, 1])

# define the output layer (2 nodes), softmax normalizes values between 0/1
Y_j = tflearn.fully_connected(X_i, n_units=2, activation='softmax')
```



Coding the Example in Python

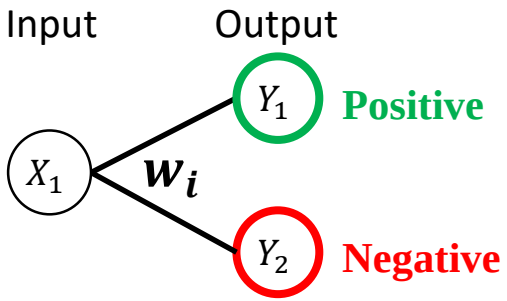
```
# import the packages we will be needing
import tflearn

# define the input layer (1 node)
X_i = tflearn.input_data(shape=[None, 1])

# define the output layer (2 nodes), softmax normalizes values between 0/1
Y_j = tflearn.fully_connected(X_i, n_units=2, activation='softmax')
```



We don't need to define w_i

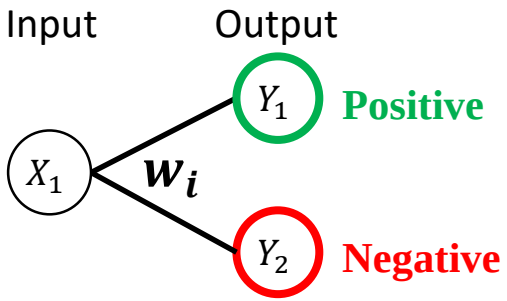
Coding the Example in Python

```
# import the packages we will be needing
import tflearn

# define the input layer (1 node)
X_i = tflearn.input_data(shape=[None, 1])

# define the output layer (2 nodes), softmax normalizes values between 0/1
Y_j = tflearn.fully_connected(X_i, n_units=2, activation='softmax')

# set up optimization problem (sgd = stochastic gradient descent)
optimization_problem = tflearn.regression(Y_j, optimizer="sgd")

# initialize variables w_i with random values to provide a starting point for optimization
model = tflearn.DNN(optimization_problem)
```

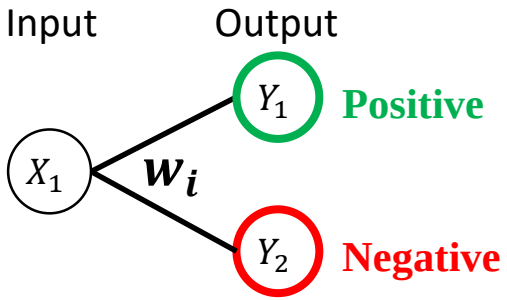


Coding the Example in Python

```
# import the packages we will be needing
import tflearn

# define the input layer (1 node)
X_i = tflearn.input_data(shape=[None, 1])

# define the output layer (2 nodes), softmax normalizes values between 0/1
Y_j = tflearn.fully_connected(X_i, n_units=2, activation='softmax')

# set up optimization problem (sgd = stochastic gradient descent)
optimization_problem = tflearn.regression(Y_j, optimizer="sgd")

# initialize variables w_i with random values to provide a starting point for optimization
model = tflearn.DNN(optimization_problem)

# train the model (assuming we have some data) for 100 iterations, update w every iteration
model.fit(data, label, n_epoch=100)
```

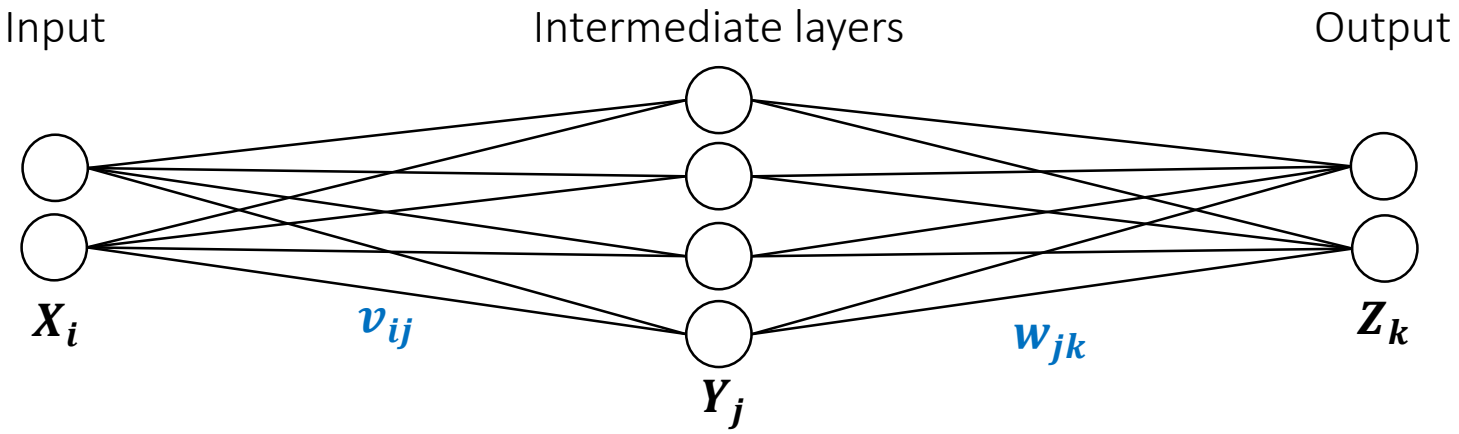


Coding the Example in Python




```
# import the packages we will be needing
import tflearn

# define the input layer (1 node)
X_i = tflearn.input_data(shape=[None, 1])

# define the output layer (2 nodes), softmax normalizes values between 0/1
Y_j = tflearn.fully_connected(X_i, n_units=2, activation='softmax')

# set up optimization problem (sgd = stochastic gradient descent)
optimization_problem = tflearn.regression(Y_j, optimizer="sgd")

# initialize variables w_i with random values to provide a starting point for optimization
model = tflearn.DNN(optimization_problem)

# train the model (assuming we have some data) for 100 iterations, update w every iteration
model.fit(data, label, n_epoch=100)

# make a prediction
print( model.predict([[1]]) )      # should get Y1 > Y2 (output = [Y1, Y2])
print( model.predict([[-1]]) )     # should get Y1 < Y2 (output = [Y1, Y2])
```

More Complicated Neural Network



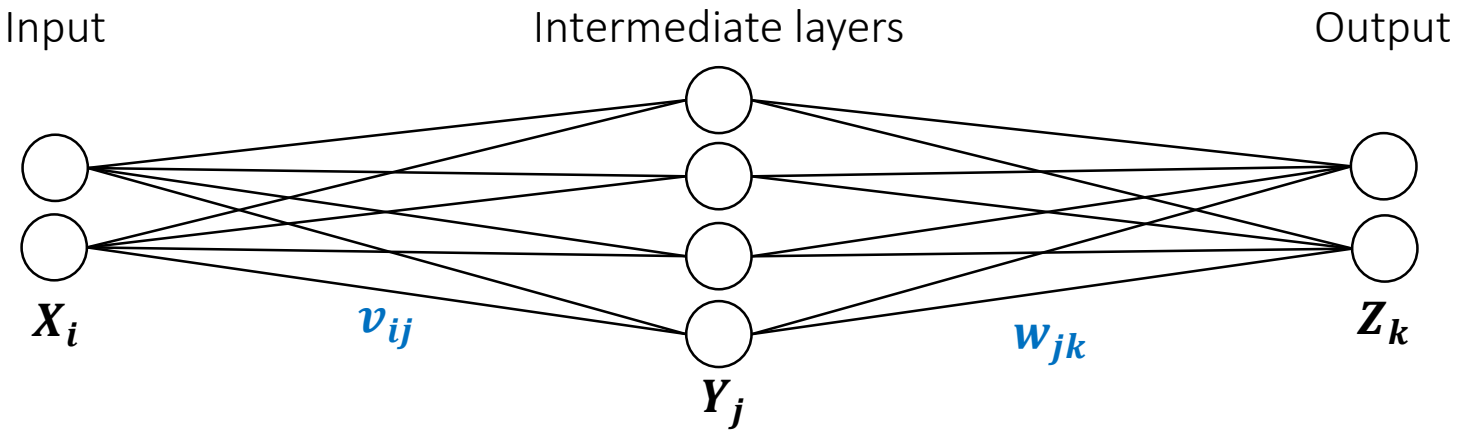
To map $X_i \rightarrow Y_j$:

$$[X_1 \quad X_2]\begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \end{bmatrix} = [Y_1 \quad Y_2 \quad Y_3 \quad Y_4]$$

To map $Y_j \rightarrow Z_k$:

$$[Y_1 \quad Y_2 \quad Y_3 \quad Y_4]\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = [Z_1 \quad Z_2]$$

More Complicated Neural Network



To map $X_i \rightarrow Y_j$:

$$[X_1 \quad X_2]\begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \end{bmatrix} = [Y_1 \quad Y_2 \quad Y_3 \quad Y_4]$$

Transformations are done with matrix multiplications

To map $Y_j \rightarrow Z_k$:

$$[Y_1 \quad Y_2 \quad Y_3 \quad Y_4]\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = [Z_1 \quad Z_2]$$

Mid Session Break

MRC – Hypertension Modelling

Background:

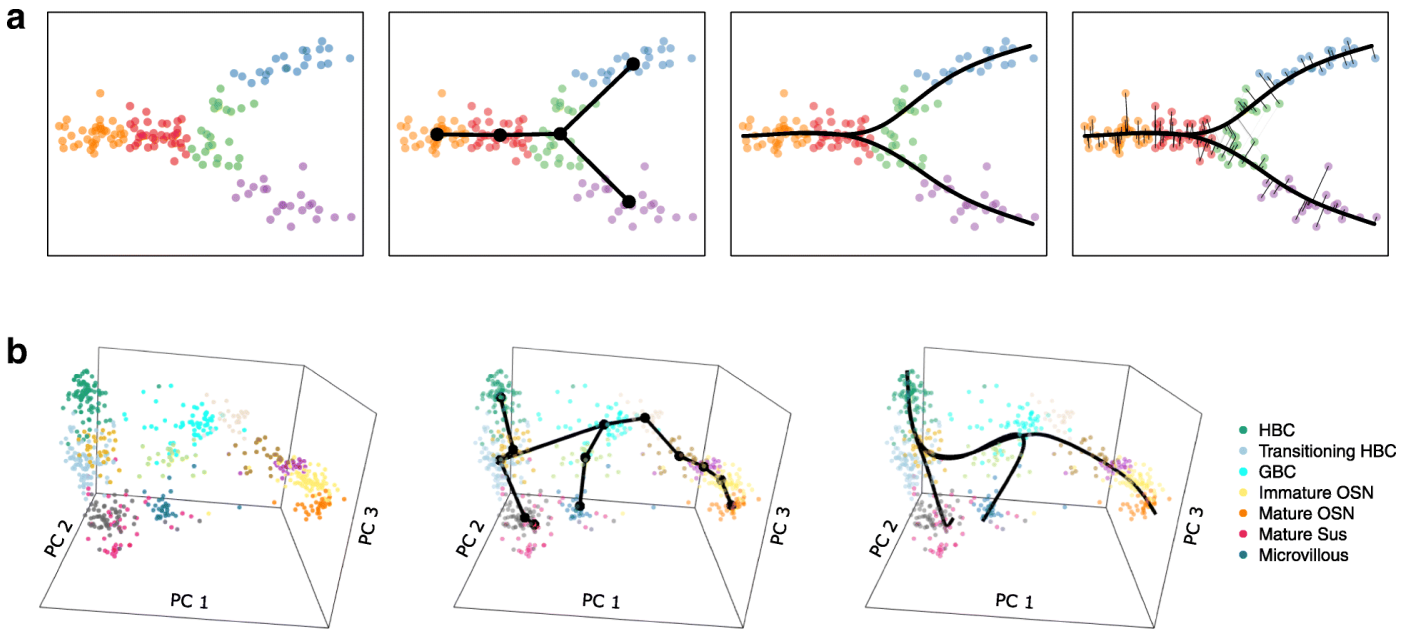
- Hypertension contributes to increased risk of stroke and heart attack
- Early detection of hypertension may aid the development of preventative measures
- We aim to identify trends and early indicators for hypertension to quantify progression in a population-wide dataset

Method – Pseudo Temporal Modelling

- In a population-level cohort (such as the UK Biobank), participants fit into different stages of disease progression and represent the entire healthy-to-disease scale
- Build a pseudo-temporal model using cross-sectional data to study the changes/progression of various biomarkers in patients from healthy to diseased
- Assign a score (0 to 1) to reflect the degree of disease, in our case the severity of hypertension
- “Trajectory inference” is the main method used

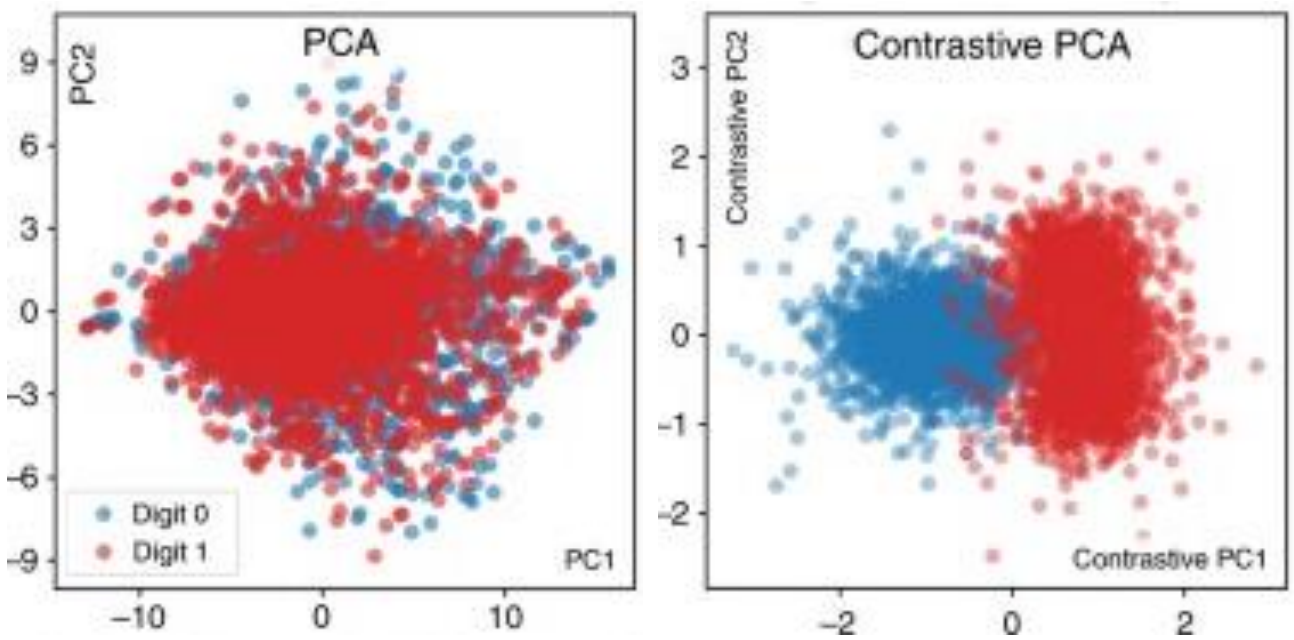
Trajectory Inference

- Pseudo-temporal modelling to score patients based on their disease progression
- Fits patients on “trajectories” of disease progression
- Each trajectory may relate to changes in different combinations of biomarkers
- Trajectory inference 2 main steps:
  - Dimensionality reduction of features
  - Connect each patient in the reduced dimensional space to form a minimum spanning tree



Contrastive Trajectory Inference (cTI)

- “Contrastive” trajectory inference method to improve separation of pre-defined groups
- Incorporate prior information (background/between/diseased) during the dimensionality reduction step of trajectory inference



Contrastive Trajectory Inference (cTI) – Dimensionality Reduction

- Contrastive PCA (cPCA) for dimensionality reduction:

PCA:

- Standard PCA is done by first computing the covariance matrix (*cov*)
- Then perform eigen decomposition on the covariance matrix to transform features into a reduced representation

cPCA:

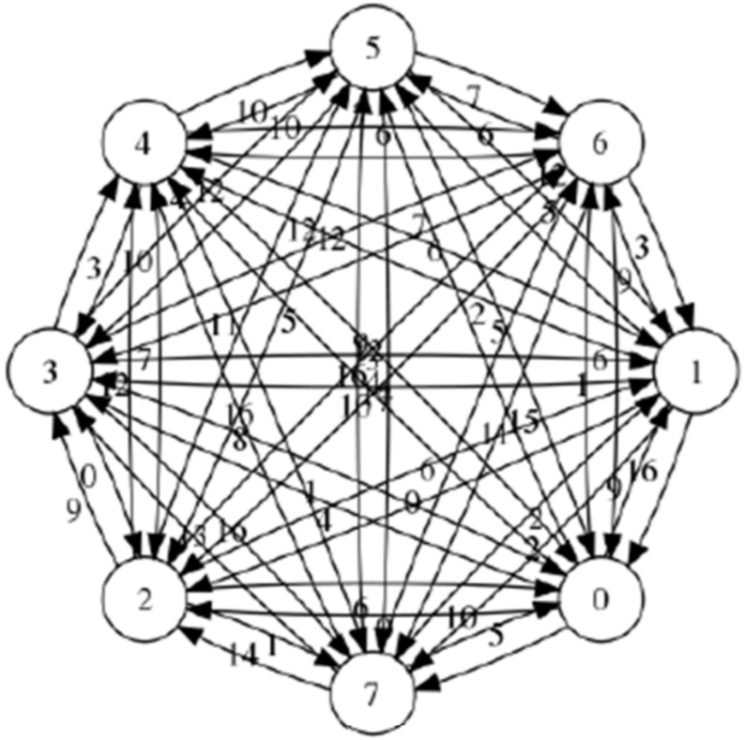
- cPCA is performed by computing a weighted sum between the covariance of the background (cov_b) and diseased (cov_d) group over a range of *α's* (~100 values from 10^{-2} to 10^{2}):

$$cov = cov_d - \alpha \times cov_b$$

- Perform eigen decomposition for each of these resulting covariance matrices
- Determine the optimal α using K-means clustering to find the value which produces the best clustering tendency of the background and disease groups in the reduced space

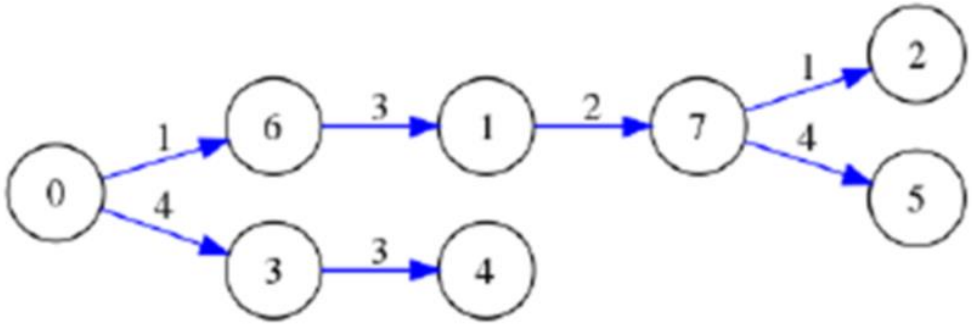
Contrastive Trajectory Inference (cTI) – Trajectory Construction

- Consider each individual as a “node” in a complete graph in the contrastive principle component space (cPC)
- Edge of the graph are defined by the Euclidean distance between nodes in the cPC space
- Define “*root node*” as the node with the smallest overall distance in the cPC space from all nodes in in the population



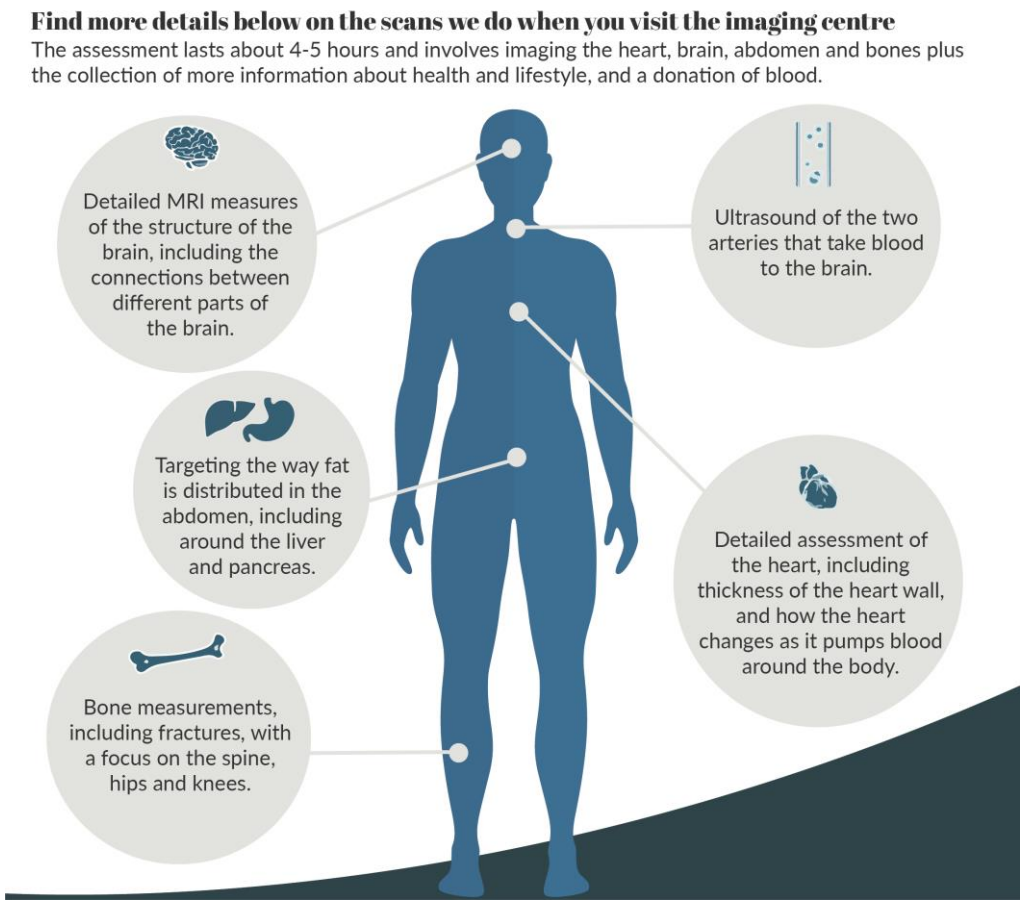
Contrastive Trajectory Inference (cTI) – Pseudo Time Score Calculation

- Construct a minimum spanning tree which minimizes the overall distance between all nodes
- Find the shortest path between each node and the root node
- The disease (pseudo time) score is the normalized distance along the shortest path



Study Population – UK Biobank

- UK Biobank has around 500,000 participants from the general population
- 100,000 of these participants are planned to undergo medical imaging (cardiac MRI + brain MRI + carotid ultrasound)
- UK Biobank contains ~15,000 features for each patient
- Imaging patient dataset contains **~30,000** participants with **~2,000** imaging features
- Aim:
  - Investigate the progression/change in features from the normotensive population to the hypertensive population



Data Pre-Processing

- Filter out patients without blood pressure readings
- Filter out variables which are not related to imaging/baseline measures
- Remove participants who have already experienced a heart attack, angina or stroke before the time of imaging
- Filter columns/rows with too many missing values
- Remove outliers
- Perform z-score normalization
- Adjust for age/sex

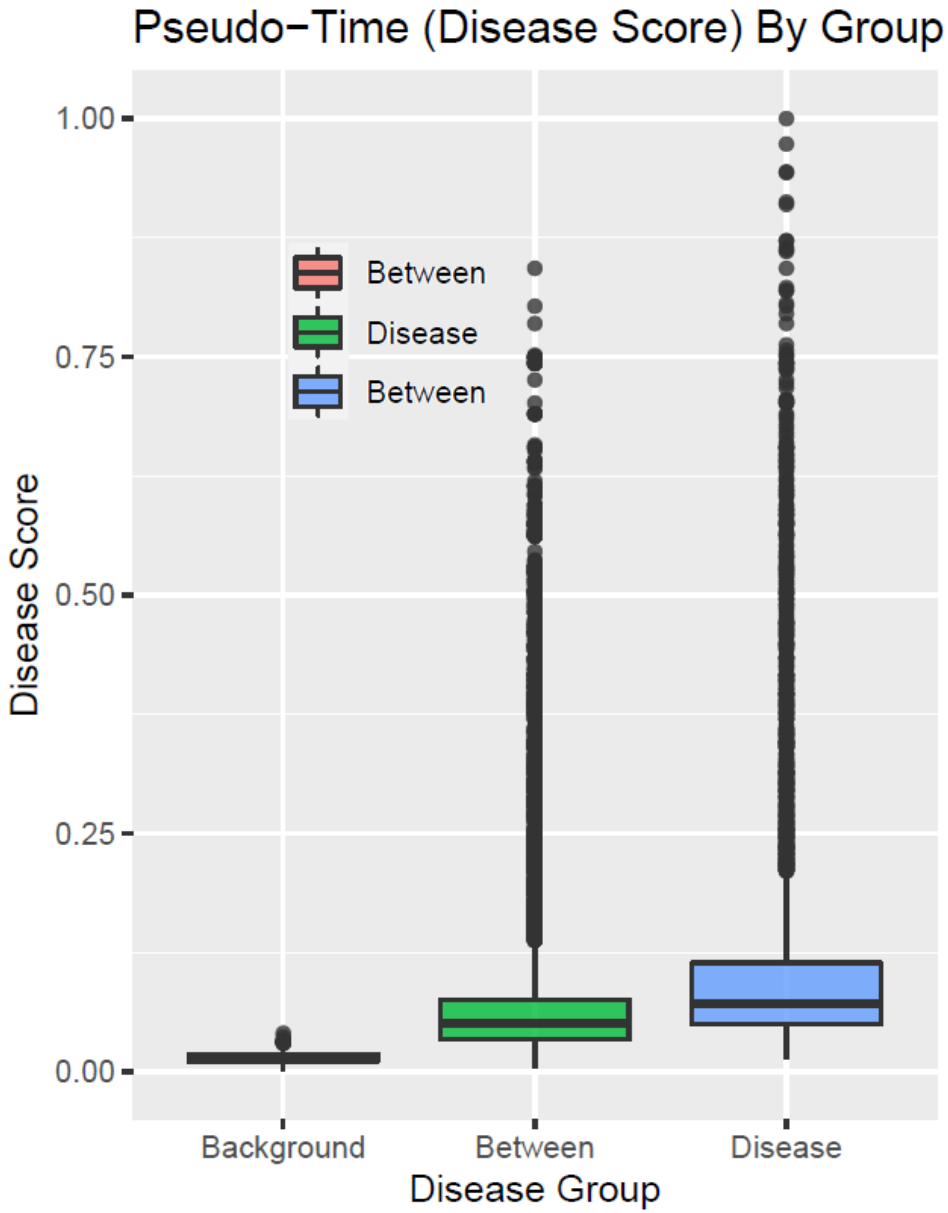
Data Label Definition

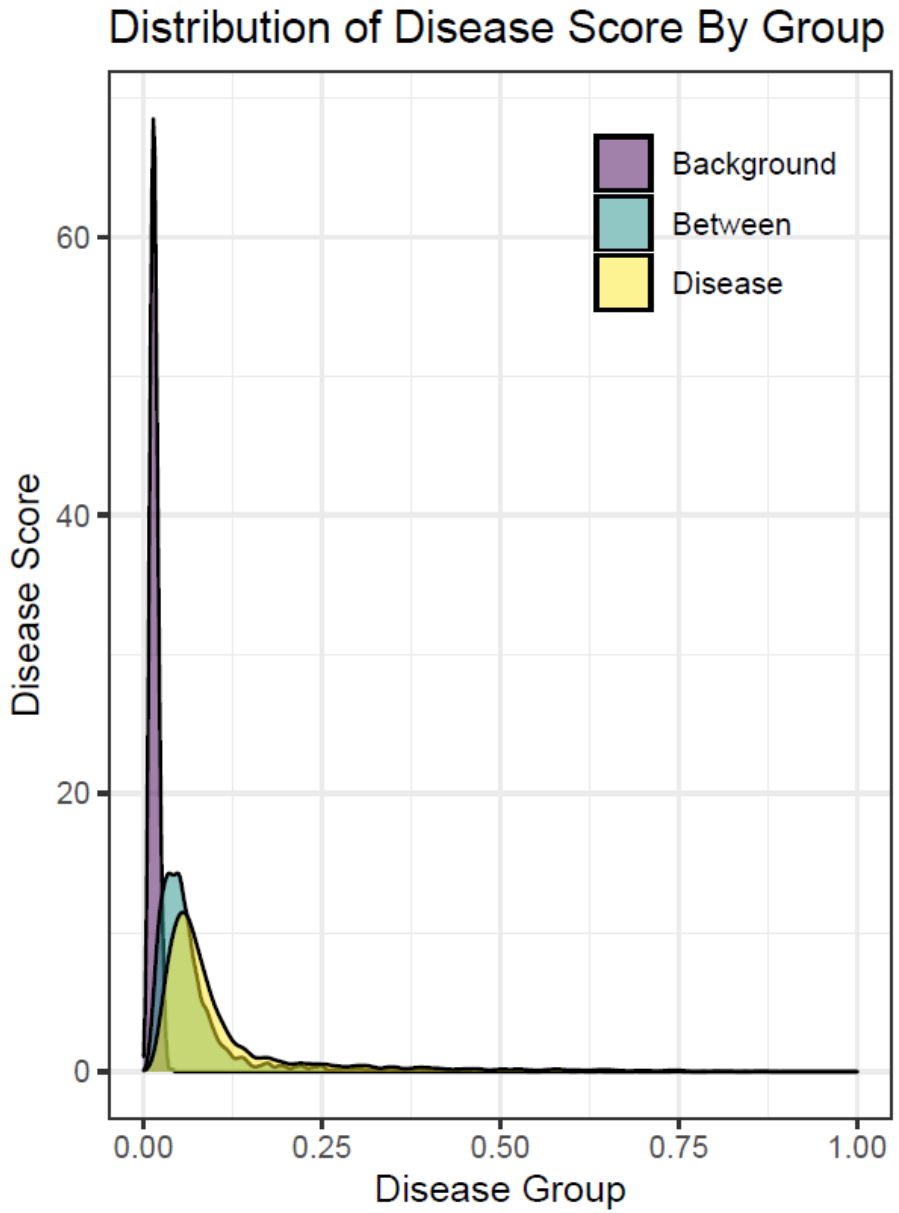
- Define blood pressure groups
  - Background: < 120/80 (both)
  - Disease: > 160/100 (either)
  - Between: Any patient otherwise

- Tightening “background” group definition
  - Remove patients already diagnosed with high blood pressure
  - Remove patients on blood pressure medication
  - Remove patients that had a high blood pressure reading at any stage (3 visits)

- The final data frame has **27,338** (participants) with **1,086** (features)
- Distribution of samples: background = 1,380, Between = 21,759, Disease = 4,199

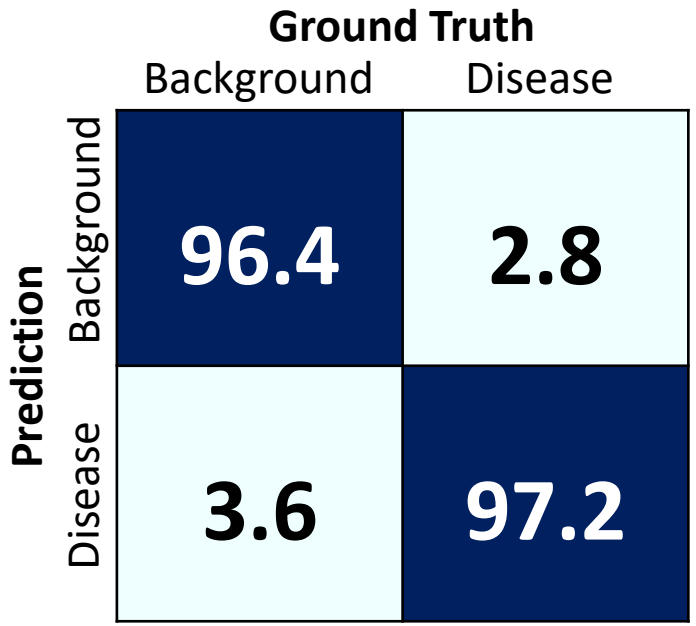
Main Results – Pseudotime Score

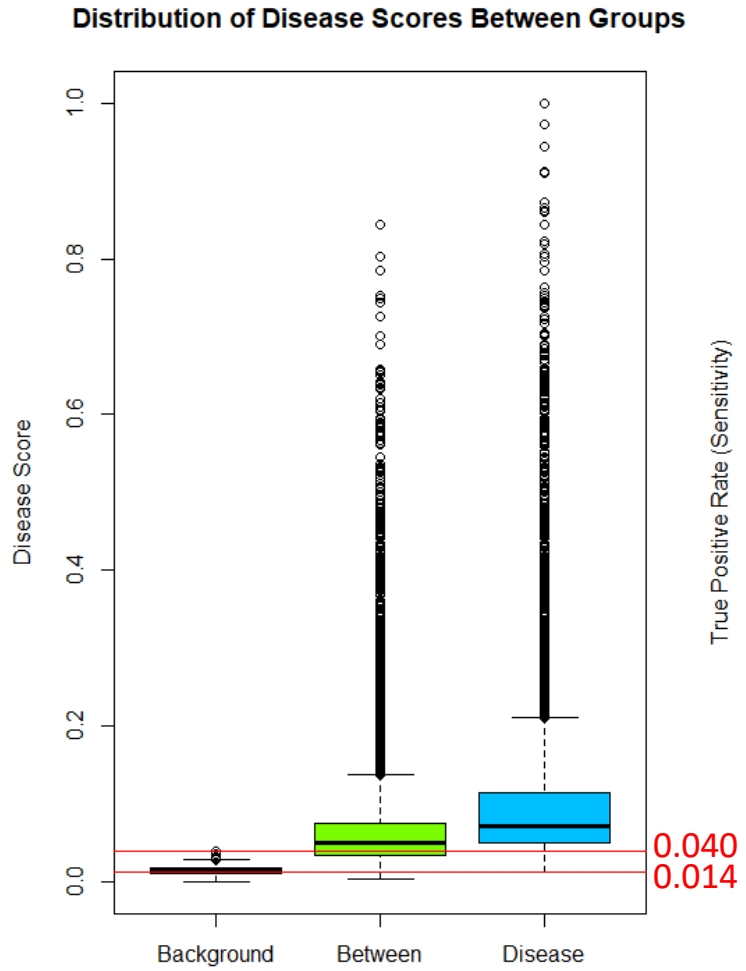


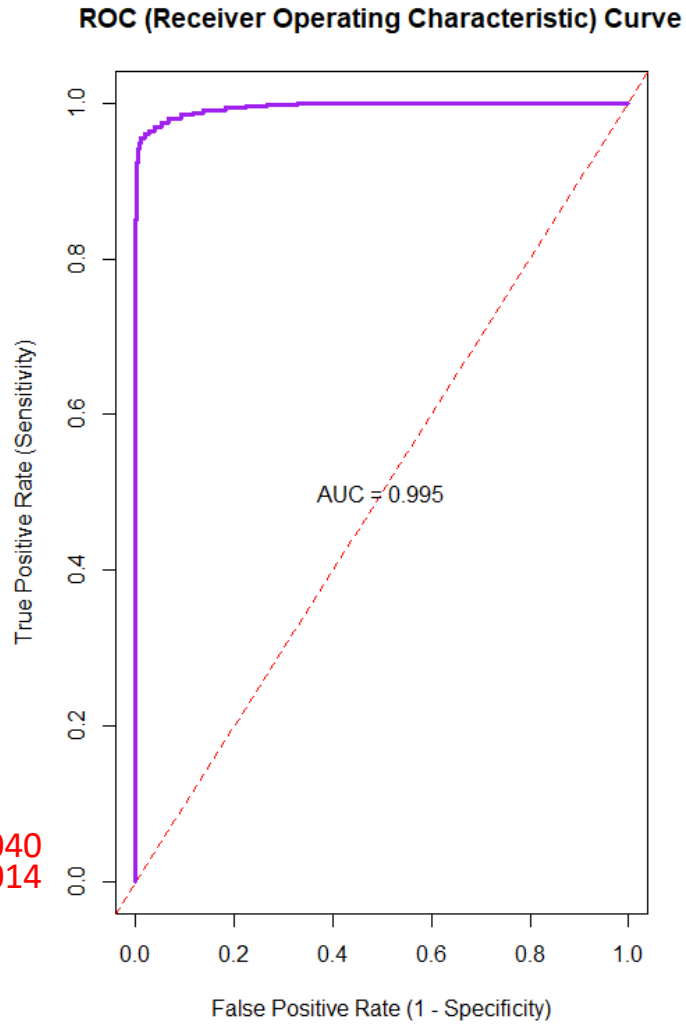


Evaluating the Accuracy of the cTI Model

- AUC of 99.5% for distinguishing between “background” and ”diseased”
- Sensitivity = 96.4%
- Specificity = 97.2%
- Optimal threshold = 0.027

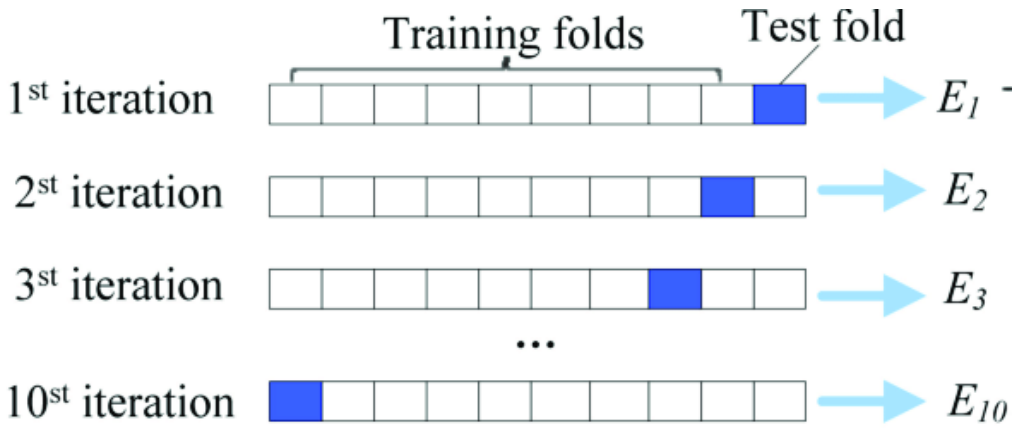






Evaluating the Stability of the cTI Model

- Run the cTI model on 100% of the data to produce the set of baseline disease scores
- Split dataset into 90:10 splits
- Run cTI on 90% of the data to generate disease, ignore the 10% left out
- Compare the scores generated from the subset dataset with those obtained from the full dataset



- Repeat this for every 90:10 split (10-fold cross validation)
- Overall 10-fold validation resulted in an **RMSE of 0.04 (sd = 0.004)**

Predicting Scores for New Patients

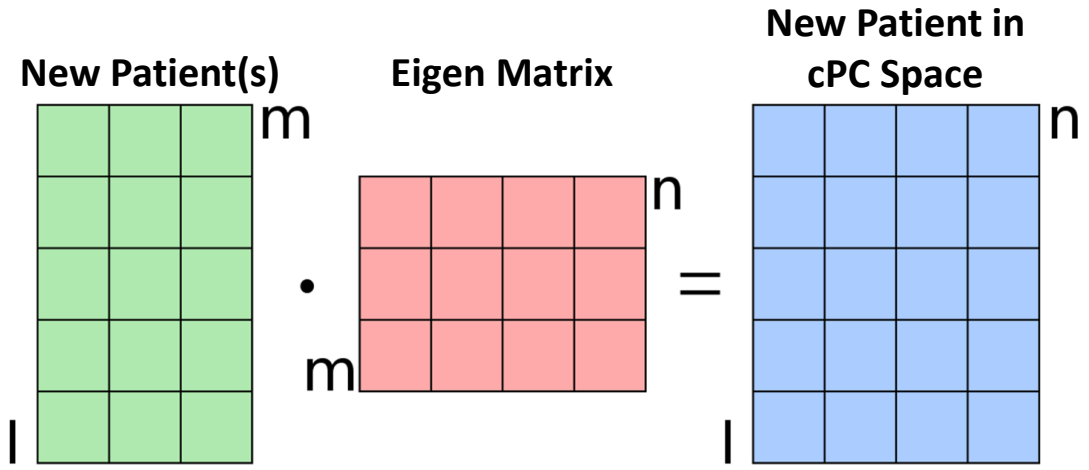
- Place new patient in the current trajectory map from the existing cTI model
  - cTI generates a reduced representation (cPC) for the existing data
  - Transform new patient into the existing cPC space of the current cTI model
  - “Infer” the disease score as the score of the patient in the existing model with the smallest distance from a new patient in the cPC space

- Advantages:
  - Don’t need to re-run cTI model
  - Nearest neighbor is easy to quantify when there are only ~15 cPCs

Implementation of cPC Transformation

- Equation of transformation in/out of the cPC space:
  - $X_{cPC} = X * Eig(cov_d - \alpha \times cov_b)$ (matrix multiplication with the most optimal α)
  - The eigen matrix above is generated when building the cTI model, and can be stored as an intermediary value
  - New patients can be mapped into the cPC space with:

$$X_{cPC\ New} = X_{New} * Eig(cov_d - \alpha \times cov_b)$$

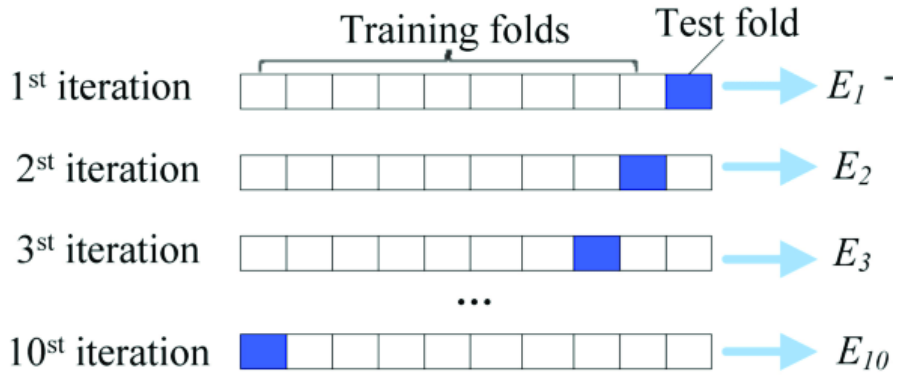


Implementation of cPC Transformation

- The transformed data for the new patient can then be compared with X_{cPC} to find the sample in the existing model with the closest Euclidean distance
- Disease score can then be inferred directly from the nearest existing sample

Experimental Setup for Evaluating the "Prediction" Method

- Run the cTI model on 100% of the data to produce the set of baseline disease scores
- Split the data into 90% (training) and 10% (testing)
- Use the 90% to train the cTI model
- Use the proposed prediction approach to predict the 10% left out

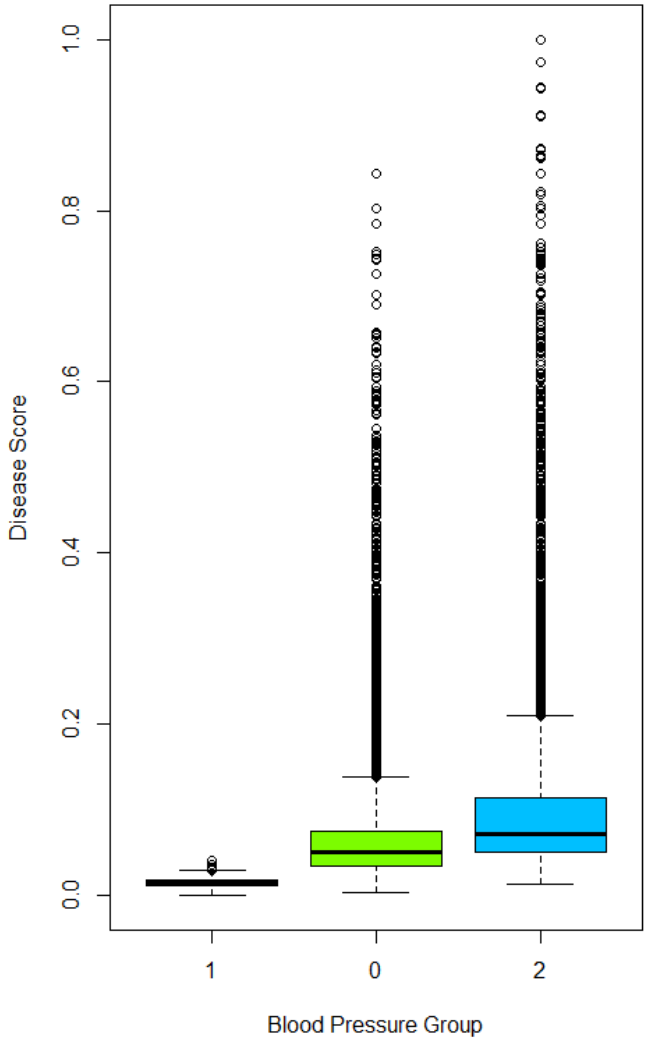


- Compare the predicted scores of the 10% with scores of the same patient in the 100% run
- Repeat this for every 90:10 split (10-fold cross validation)

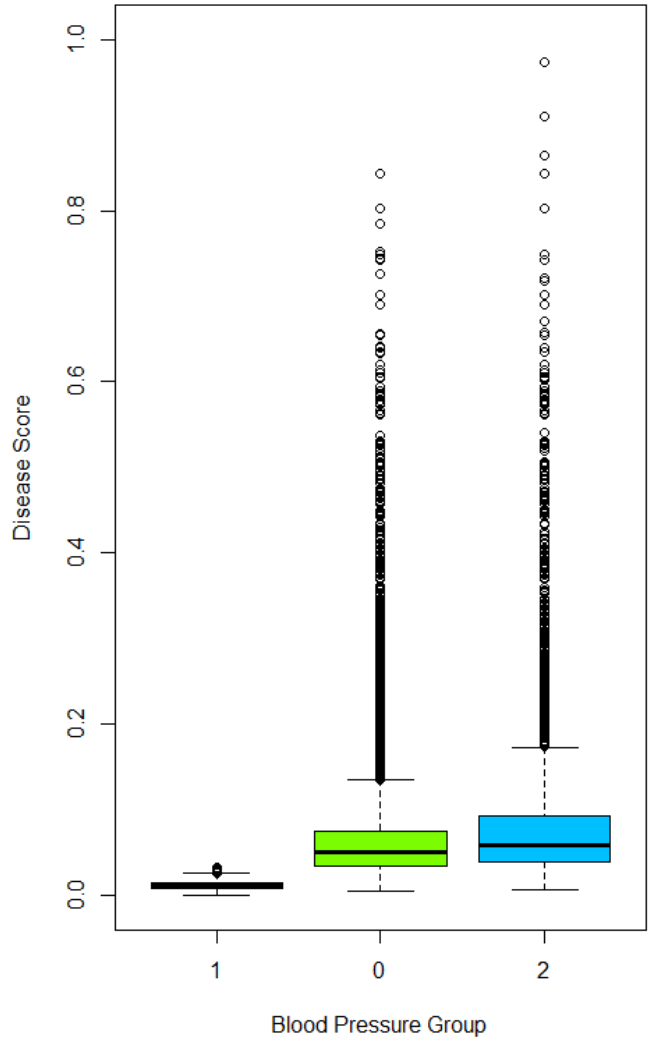
Experimental Results for “Prediction” Method

- 10-fold cross validation resulted in an **RMSE of 0.043**
- Errors for each group were:
  - Background = **0.018 RMSE**
  - Between = **0.085 RMSE**
  - Disease = **0.094 RMSE**

cTI Modelled Scores



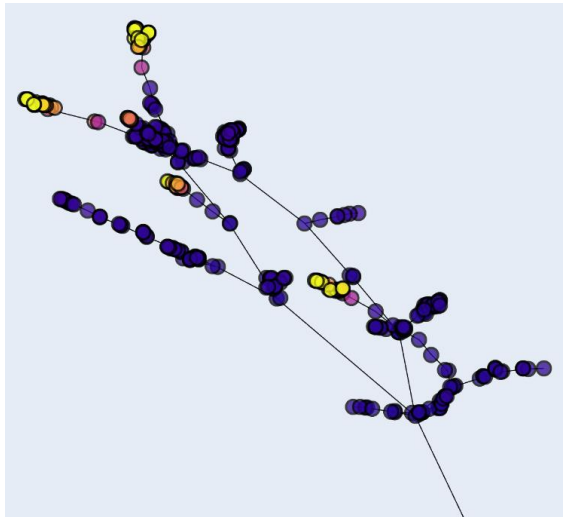
Predicted Scores Over 10-Folds of Cross-Validation

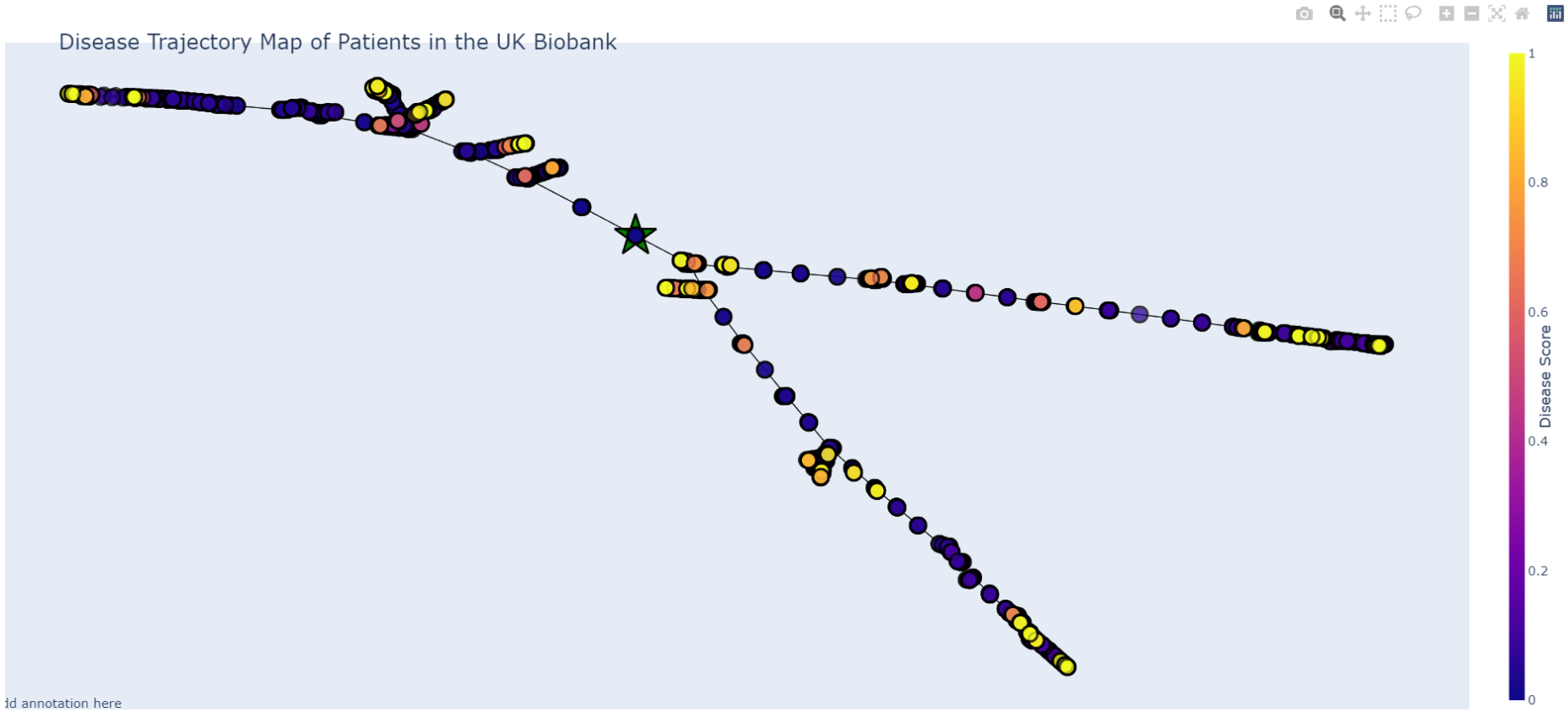


Trajectories – Preliminary Results

- Compute the Laplacian of the matrix representation of the minimum spanning tree
- Perform eigen decomposition on the Laplacian, the coordinates are thus the first two eigen vectors

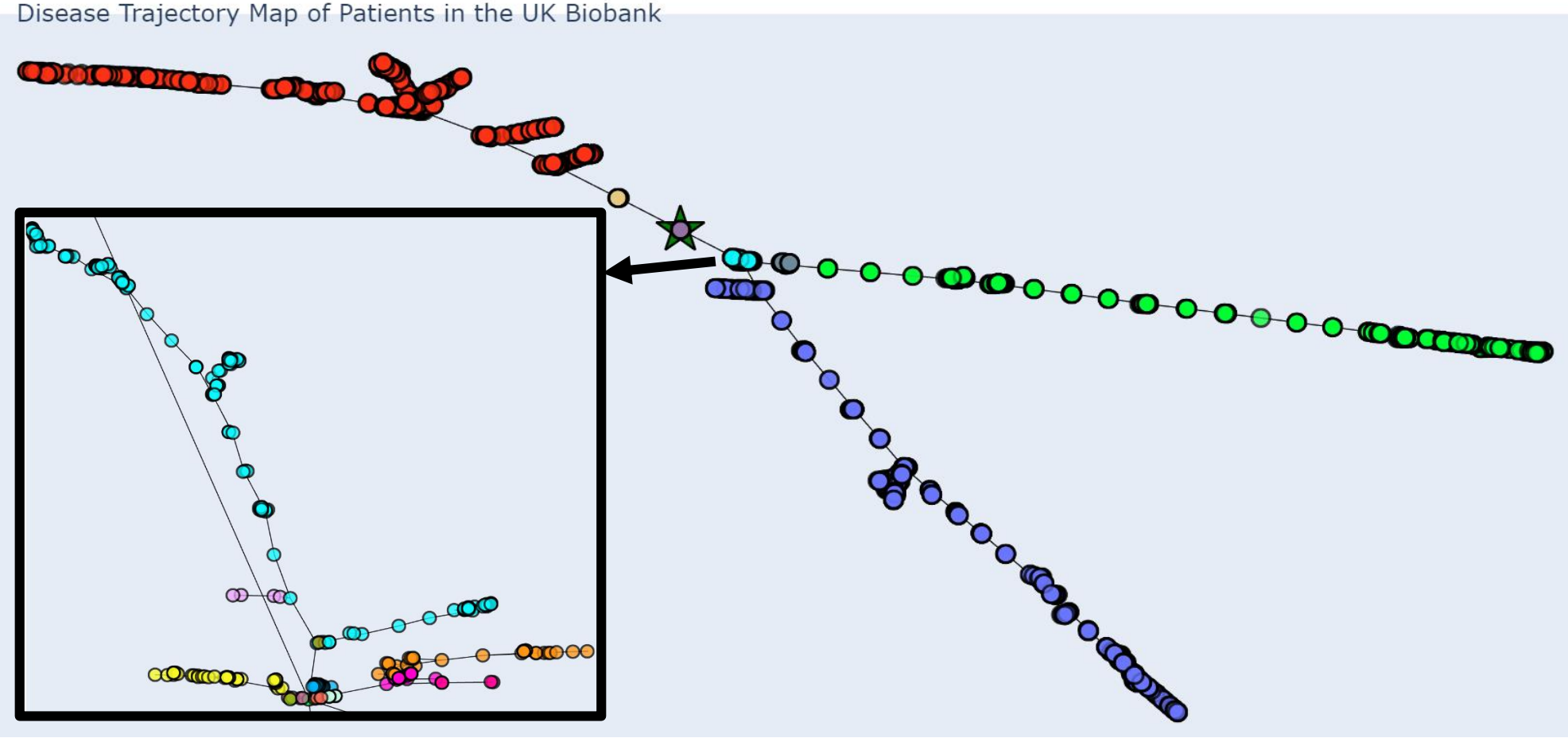
Zoomed in Section





Trajectories – Preliminary Results

- Color each unique trajectory
- Many “unique” trajectories close to the root node due to low overlap



Future Work

Contrastive Trajectory Inference Model:

- Improve score distribution in the model output

Understanding cTI Model:

- Explore which variables contribute to elevated scores
- Explore the change in these variables from healthy to disease

Trajectory Investigation:

- Develop method to assign patients to a small number of “main” trajectories
- Explore which variables are more significant in different trajectories