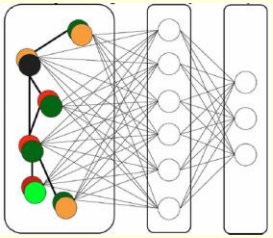
Deep Learning by Example on Biowulf

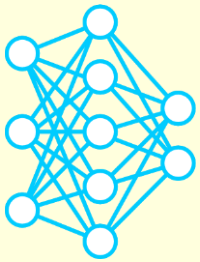
Class #6. Graph Convolutional Networks, handling imbalanced data and their application to classification of cancer types

Gennady Denisov, PhD



Intro and goals

graph, nodes / vertices, links / edges, feature, node degree



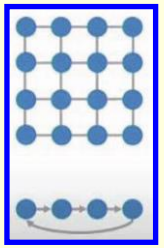
Question: how do we <u>represent</u> and <u>learn</u> the structure of data?

Representation

Data with regular structure (classes #1,2,4,5)

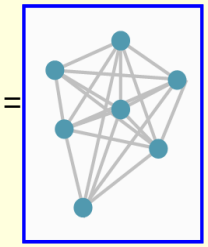
image

sequence



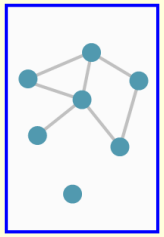
Gene expression data (class #3) - could be represented by a complete graph

degree = N-1



Data with irregular structure: should be represented by a graph

degree - variable



Learning

Convolutional or Recurrent layers

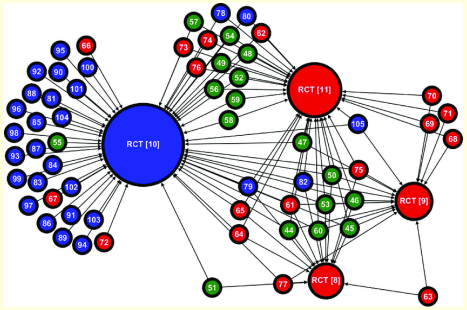
Dense layers

Graph Convolutional (+ Pooling) layers

Non-bio examples:

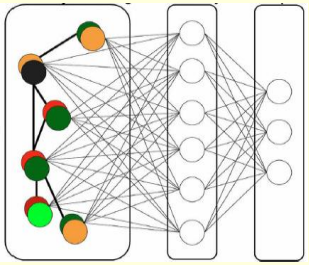
Social network: nodes = individuals, edges = connections

Citation network: nodes = documents, (directed) edges = citations



Bio example: GCN_Cancer

nodes = genes; edges = associations between genes; data similar to those of class #3, but the tasks / approaches are different



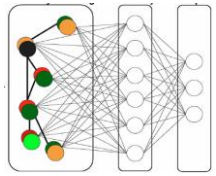
Examples overview

https://hpc.nih.gov/training/deep_learning_by_example.html

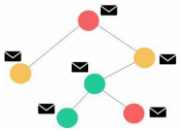
Class #	1	2	3	4	5	6	7
Bio app	Bioimage segmentation / fly brain connectome	Genomics / prediction of function of non-coding DNA	Genomics / reduction of dimensionality of cancer transcriptome	Bioimage synthesis / developmental biology	Drug molecule design	Genomics / classification of cancer types	Drug molecule property prediction
Neural network type	Convolutional	Recurrent or 1D-Convolutional	Autoencoder	Generative Adversarial	Reinforcement Learning	Graph Convolutional	Message Passing
ML type	Supervised	Supervised	Unsupervised	Unsupervised	Reinforcement	Supervised	Supervised

- **Supervised ML** like in classes #1 and #2, but the data are similar to those in #3
- Primary purpose: **demystifying** the Graph Neural Networks
- **"Transition" examples**: the more "traditional" MLP / DenseNet can still be used
- Graph imposes **constraints /** provides **additional knowledge** about the data, which may allow for **more accurate predictions** as compared to the constraint-agnostic models.

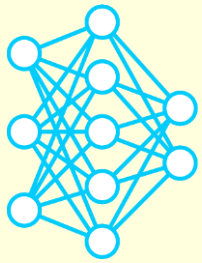
GCN



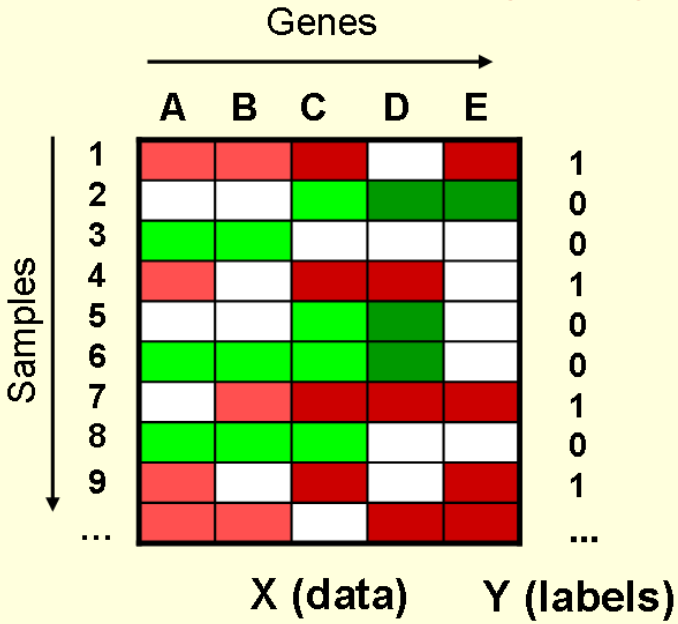
MPN



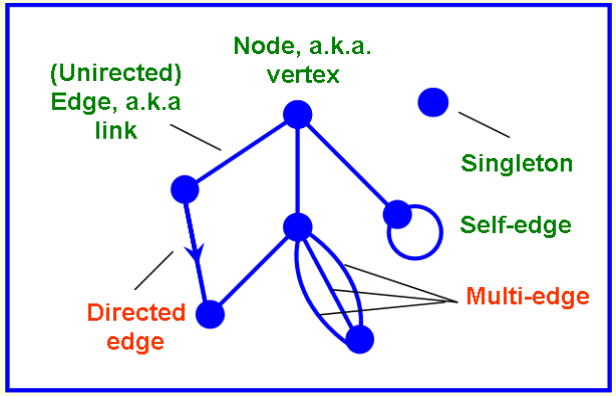
Prototype example #1: the graph classification task



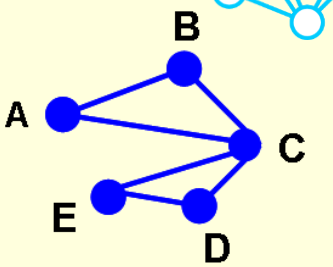
gene expression matrix, node features, undirected graph, singletons, adjacency matrix, graph classification task



Modules = [[0,0,0,1,1], [0,0,1,1,1]]



Graph notation



Undirected graph:

Nodes = genes
Node feature = gene expression level
Edges = associations between genes

	A	B	C	D	E
A		1	1		
B	1		1		
C	1	1		1	1
D			1		1
E			1	1	

Adjacency matrix:

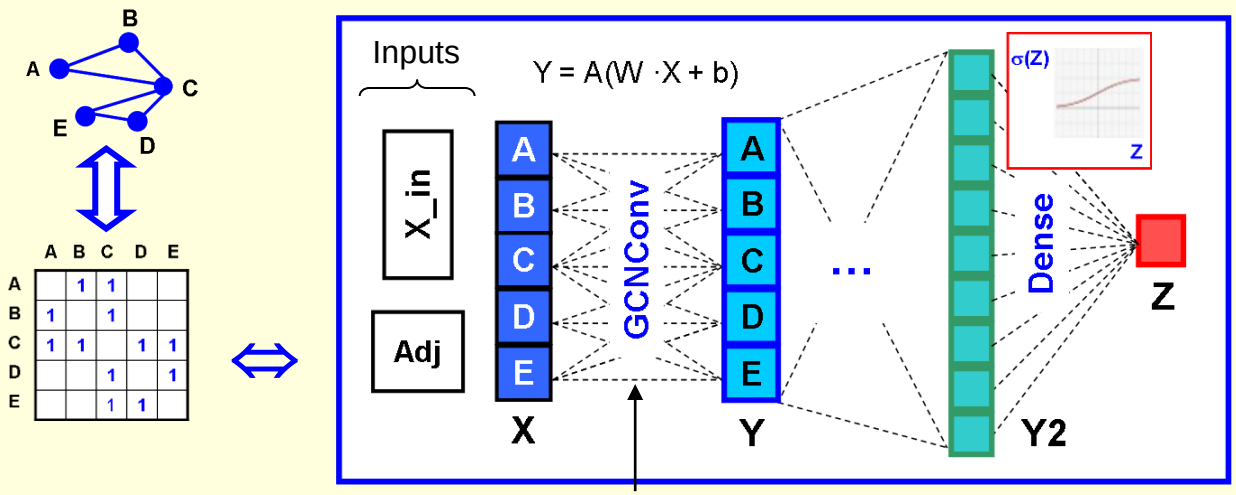
- symmetric (⇔ no directed edges)
- values ≤ 1 (⇔ no multi-edges)

Input:
Gene expression data matrix generated randomly "on the fly", with rows = **samples**, columns = **genes** and **values =1** (gene is expressed, shown in color) **or =0** (gene is unexpressed, shown as white). The samples can be of two types: **"normal"** (label=0) or **"tumor"** (label=1). In each the type of samples, genes are associated into **two modules,** designated 0 and 1, with the same probability of expression for all genes in a module, but different probabilities across different modules.

Task:
Train a graph convolutional **network model** on this data, so that it could **predict the class labels** for new, previously unseen samples.

Prototype example #1 (cont.): GCNConv vs Dense layer

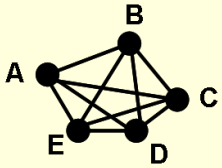
Vanilla Graph Convolution: Kipf, T.N., Welling, M.: arXiv preprint (2016)



Missing links: A-D, A-E, B-D, B-E,D-A, E-A, D-B, E-B.
The corresponding weights are constrained to 0

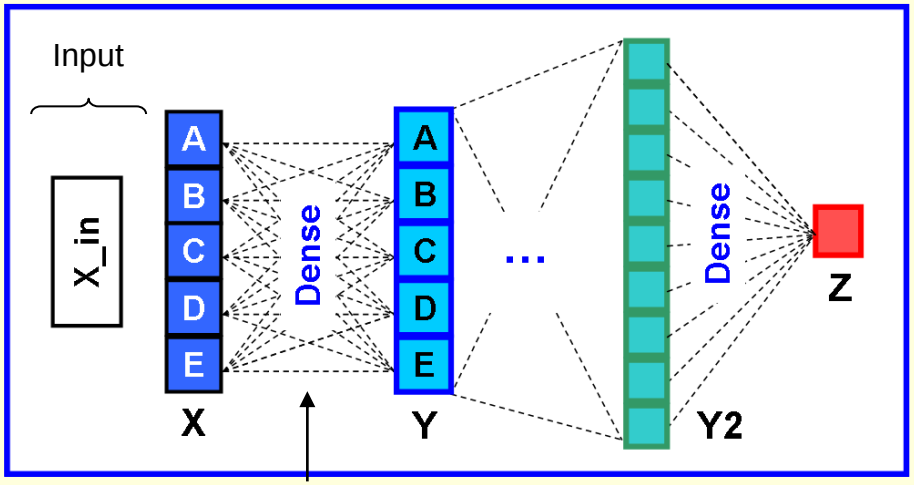
Graph Convolutional Network

Complete graph



Adjacency matrix of ones

	A	B	C	D	E
A	1	1	1	1	1
B	1	1	1	1	1
C	1	1	1	1	1
D	1	1	1	1	1
E	1	1	1	1	1



No missing links / zero weights in the fully connected layer.

MultiLayer Perceptron (MLP)

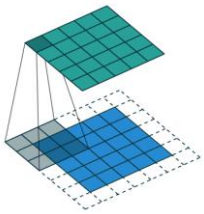
CONCLUSION: Dense layer can be regarded as GCNConv layer with adjacency matrix of ones.

Prototype example #1 (cont.): GCNConv vs Conv1D / Conv2D layers

filtering, k-hop neighborhood, pooling, supernode, parent-child relationship

2D Convolution with 'same' padding

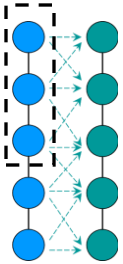
Output image



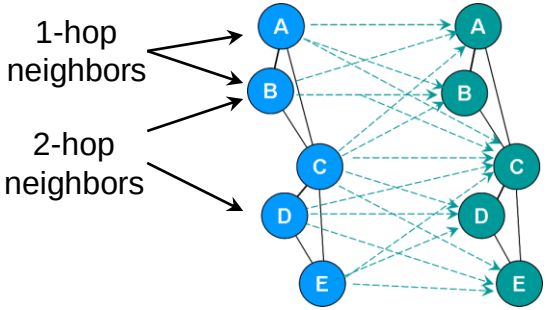
Input image

$$O = \sum_{kernel} w_i * I_i + b$$

1D Convolution with 'same' padding



Filtering, a.k.a. Graph Convolution in the narrow sense

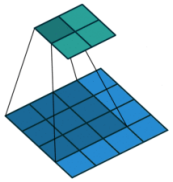


$$O = \sum_{\text{1-hop neighborhood}} w_i * I_i + b$$

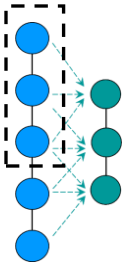
(only reassigns the node features, but does not change the number of nodes)

CONCLUSION: the filtering performed by the GCNConv layer is analogous to the Conv1D / Conv2D with 'same' padding and kernel of size = 3.

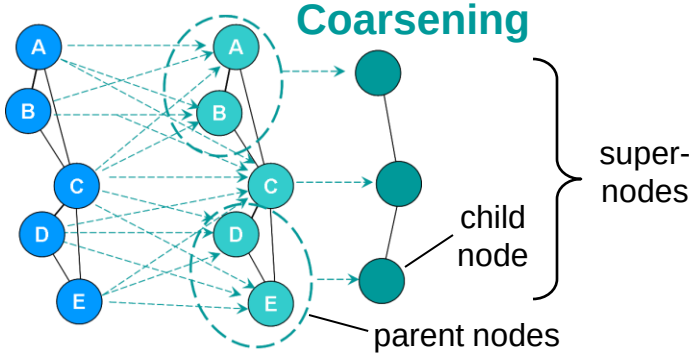
2D Convolution with 'valid' padding



1D Convolution with 'valid' padding

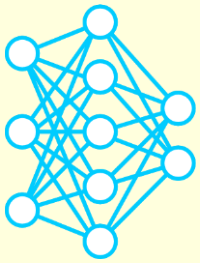


1) Filtering 2) Pooling, a.k.a. Coarsening



NOTE: an analog of the Conv1D / Conv2D with kernel of size > 3 would be a ChebConv layer, to be discussed later in this class, which employs a k-hop neighborhood of a node to update its value, with k > 1.

Prototype example #1 (cont.): training code



GCN: Kipf, T.N., Welling, M.: arXiv preprint (2016)

Spektral: *https://graphneural.network*

Header
- tensorflow
- spektral
- graph_net

Get data
- modules
- adj matrix
- samples and labels

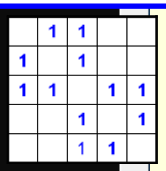
Define a model:
- Functional API
- GCNConv vs Dense
- loss and optimizer

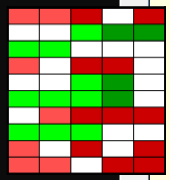
Run the model

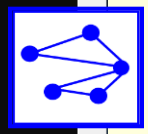
denisovga@biowulf:/usr/local/apps/DLBio/class6/bin

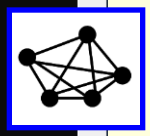
```
#!/usr/bin/env python
import tensorflow as tf, spektral, numpy as np, scipy.stats as stats
graph_net, n_g, n_train, bsize, epochs, prob=1, 5, 700, 50, 100, 0.7
np.random.seed(1); tf.random.set_seed(1)

modules = [[0,0,0,1,1], [0,0,1,1,1]]
adj = np.zeros([n_g,n_g])
for k in range(2):
    for i in range(n_g):
        for j in range(n_g):
            if not i == j and modules[k][i] == modules[k][j]:
                adj[i][j] = 1
probs = [[prob,1.-prob],[prob,1.-prob]]
x_train,y_train = [],[]
for s in range(n_train):
    sample, t = [], np.random.choice([0,1], p=(0.5, 0.5))
    for g in range(n_g):
        sample.append(np.random.choice([0,1], 1,
            p=[probs[t][modules[t][g]],1.-probs[t][modules[t][g]]])[0])
    x_train.append(sample)
    y_train.append(t)                          # 0 = green, 1 = red
x_train, y_train = np.array(x_train,dtype=float), np.array(y_train,dtype=float)
adj_train = np.array([adj for s in range(n_train)])

X     = tf.keras.layers.Input(shape = (n_g,1))
A     = tf.keras.layers.Input(shape = (n_g, n_g,), sparse=False)
if graph_net == 1:
    Y = spektral.layers.GCNConv(n_g, activation='relu')([X,A])
else: # mlp
    Y = tf.keras.layers.Dense(  n_g, activation='relu')( X)
Y_1   = tf.keras.layers.Flatten()(Y)
Y_2   = tf.keras.layers.Dropout(0.1)(Y_1)
Z     = tf.keras.layers.Dense(1, activation='sigmoid')(Y_2)
model = tf.keras.models.Model(inputs=[X, A], outputs=Z)
model.compile(loss='bce', optimizer='adam')
model.summary()

model.fit([x_train, adj_train], y_train, epochs=epochs, batch_size=bsize)
                                                          38,76          Top
```

graph_net = { 1 ⇒ GCNNet; 0 ⇒ MLP }

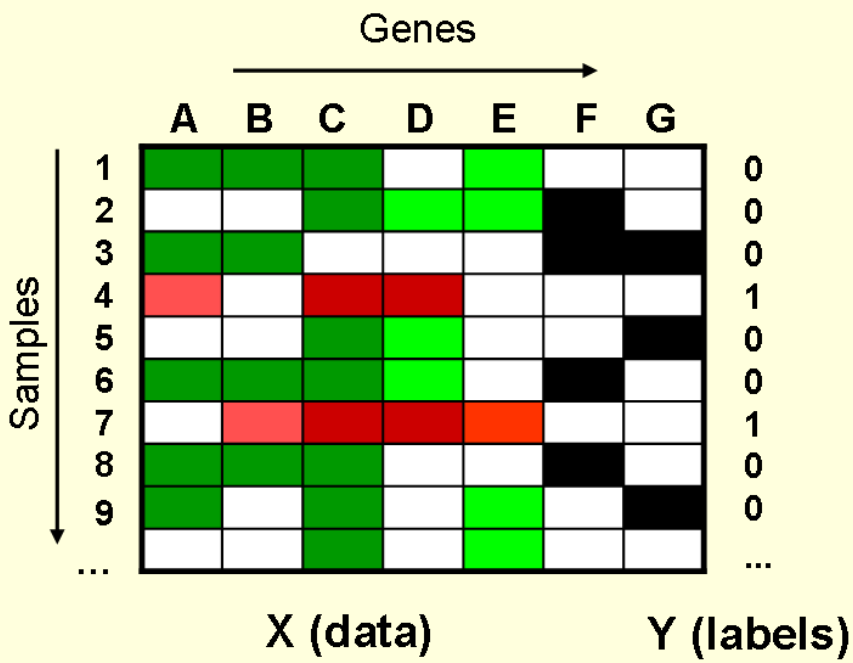








Prototype example #2: imbalanced input data containing singlentons

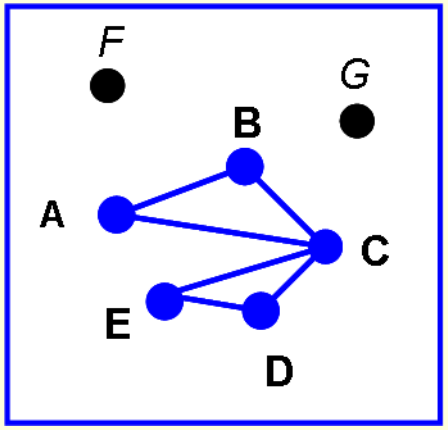


Two additional hyperparameters:

IR (imbalance ratio)
= # green samples / # red samples

n_s (# of singletons),
= genes that are not associated / co-expressed with other genes

Undirected graph with singleton genes

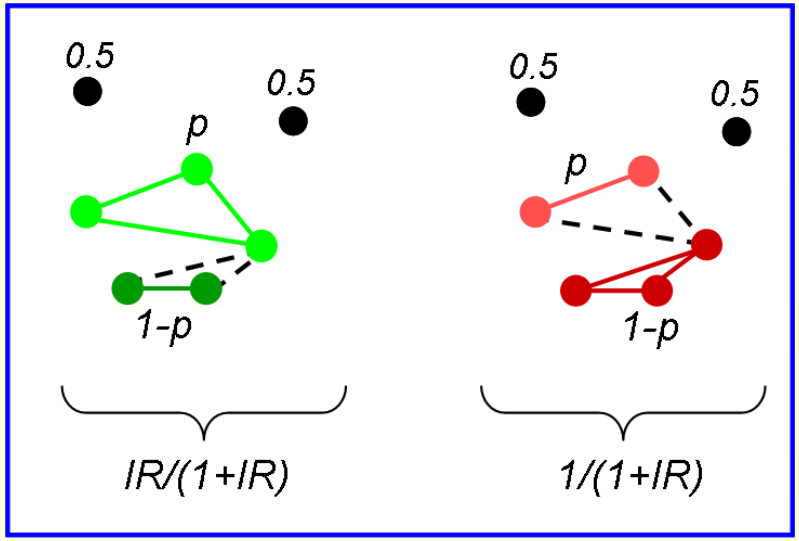


	A	B	C	D	E	F	G
A		1	1				
B	1		1				
C	1	1		1	1		
D			1		1		
E			1	1			
F							
G							

Adjacency matrix with singleton genes

Predictions from the prototype example #2

Parameterization of the gene expression probabilities



% correct predictions for *p = prob = 0.7*

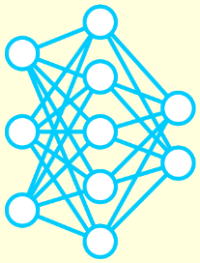
n_s	IR=1		IR=10	
	GCN	MLP	GCN	MLP
2	76	76	50.8	50.8
10	72	72	48.7	48.7
50	70.9	67.1	51.8	53.1
100	69.4	64.9	50.0	51.0
200	69	63.2	49.3	54.2

CONCLUSIONS:

- the constraints imposed on singletons by the adjacency matrix in GCN allow for attenuation of the effect of "noise" introduced by the presence of singletons, and hence for a better performance of the GCN over the constraint-agnostic MLP on <u>balanced data</u>

- since MLP possesses more adjustable parameters than GCN, it provides more flexibility in handling the challenge of data imbalance, and therefore can overperform the GCN on <u>imbalanced data</u>

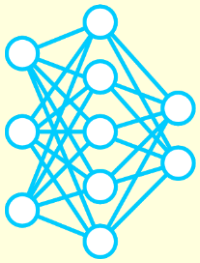
How to run the simple/prototype models on Biowulf?



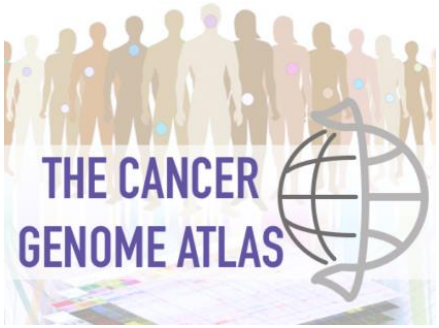
denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

```
sinteractive --gres=gpu:p100:1

module load DLBio/class6
...

ls $DLBIO_BIN
gcn_basic.py   gcn_imbalanced,singletons.py


gcn_basic.py
...
Epoch 500/500
20/20 [==============================] - 0s 1ms/step - loss: 0.6122
...
correct= 696 / 1000
...

gcn_imbalanced,singletons.py
...
Epoch 500/500
20/20 [==============================] - 0s 2ms/step - loss: 0.2572
...
correct= 508 / 1000
...
                                                    23,35          Top
```

Biological example #6:
GCN_Cancer: Classification of Cancer Types Using Graph Convolutional Networks



R.Ramirez et al., Frontiers in Physics (2020)

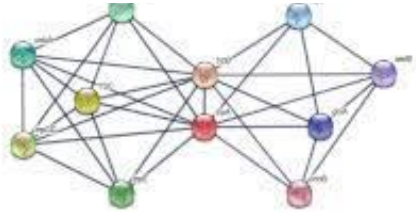


NIH program led by the NCI and NHGRI

GCN_Cancer input: RNA- seq data for
- 731 normal samples and
- 10,340 tumor samples representing 33 types of cancer

Goal: classify each sample as one of 34 types

The STRING database



STRING = Search Tool for the Retrieval of INteracting Genes/proteins

Protein-protein interaction (PPI) / association confidence scores
- used to generate adjacency matrices for the settings involving only protein coding genes

GCN_Cancer pipeline (reimplemented in Keras from Tensorflow):

Input data:
- **GE levels**
- **adjacency matrices**

→ **prepocess.py** → **train.py** → **predict.py** → **visualize.R**

prepocess.py: Balance training data; optionally, compute custom adjacency matrices

train.py: Use ~80% of ground truth data

predict.py: Predict sample type using ~20% of data

visualize.R: Bar plots of prediction error by sample type

Overview of the GCN_Cancer code

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

Header
- imports, incl. Spektral
- parsing command line options

Define model(s)
- GCN_Cancer model
- GCNConv
- ChebConv
- MinCutPool
- DiffPool

Get data
- GCE, GCES,
- PPI, PPIS

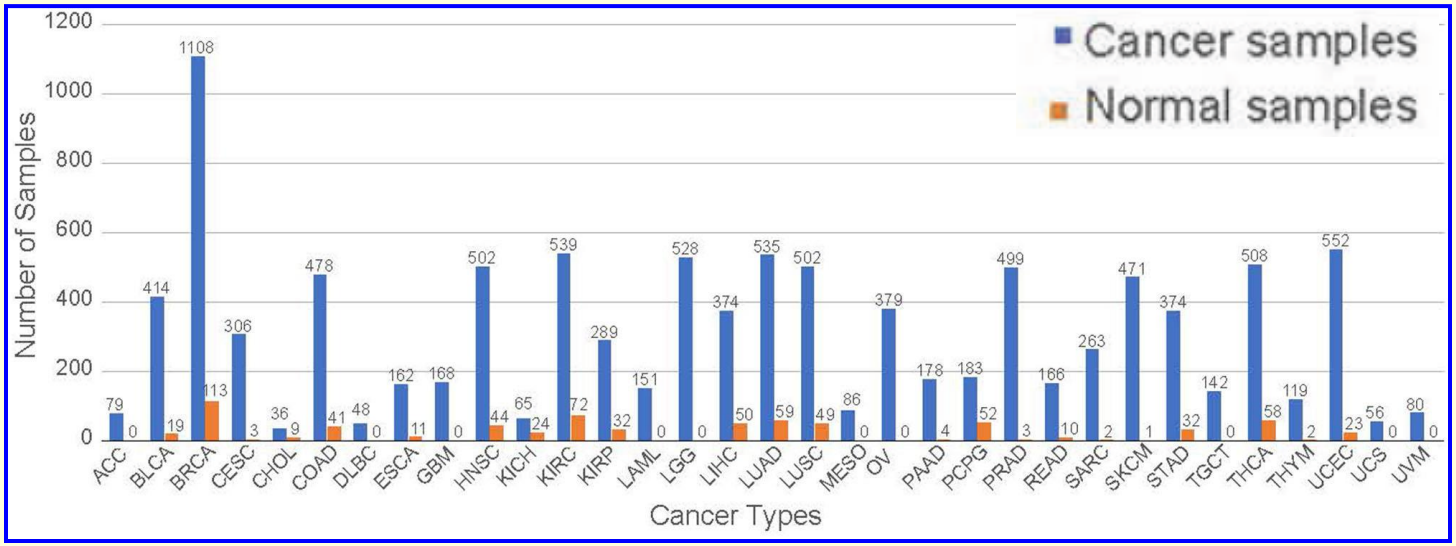
Run the models
- data balancing, SMOTE variants
- classification error

denisovga@biowulf:/data/denisovga/1_DL_Course/6_GCNs

```
#!/usr/bin/env python

import os, sys
import tensorflow as tf
import spektral
import options, data, models

if __name__ == '__main__':
    opt = options.parse_training_arguments()
    opt = options.parse_command_line_arguments("train")

    os.environ['CUDA_VISIBLE_DEVICES'] = "0"
    for j in range(1, opt.num_gpus):
        os.environ['CUDA_VISIBLE_DEVICES'] += "," + str(j)
    strategy = tf.distribute.MirroredStrategy()
    with strategy.scope():
        model = models.get_model(opt)

    opt, data_size, data_train = data.get_data(opt)
    if opt.load_weights:
        try:
            model.load_weights(opt.checkpoint_file)
        except:
            sys.exit("\nCannot read weights from: " + opt.checkpoint_file)

    checkpointer = tf.keras.callbacks.ModelCheckpoint(filepath=\
                        opt.checkpoint_file, save_weights_only=True)
    num_steps = int(round(float(data_size)/float(opt.batch_size)))
    loader = spektral.data.BatchLoader(data_train, batch_size=opt.batch_size)
    model.fit(loader.load(), epochs=opt.num_epochs,steps_per_epoch=num_steps,
              batch_size=opt.batch_size, shuffle=True, callbacks=[checkpointer])
                                                                    31,82          All
```

The GCN_Cancer data

R.Ramirez et al., Frontiers in Physics (2020)



RNA-seq data

- highly **imbalanced**
- **normal** samples from **23 tissues only**

Data matrices

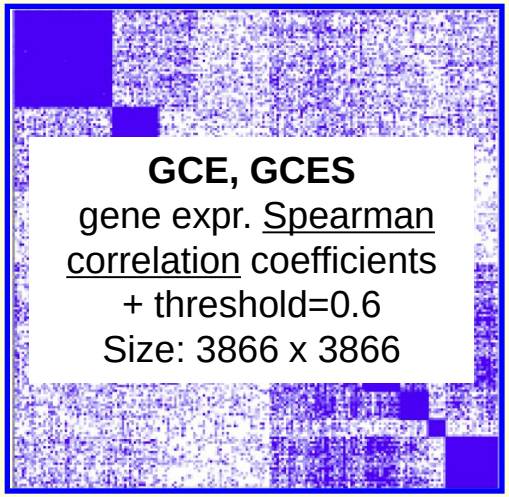
GCE: genes of any type, no singletons 8850 samples x 3866 genes

GCES: genes of any type, **including singletons**; 8850 samples X 7091 genes

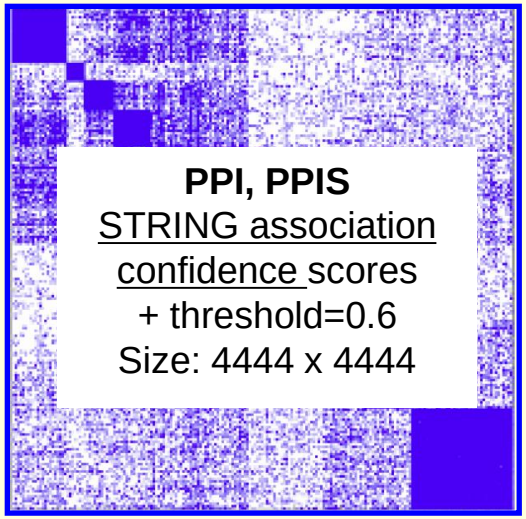
PPI: protein coding genes, no singletons; 8896 samples X 4444 genes

PPIS: protein coding genes, **including singletons**; 8850 samples x X 7091 genes

Adjacency matrices



GCE: gene co-expression

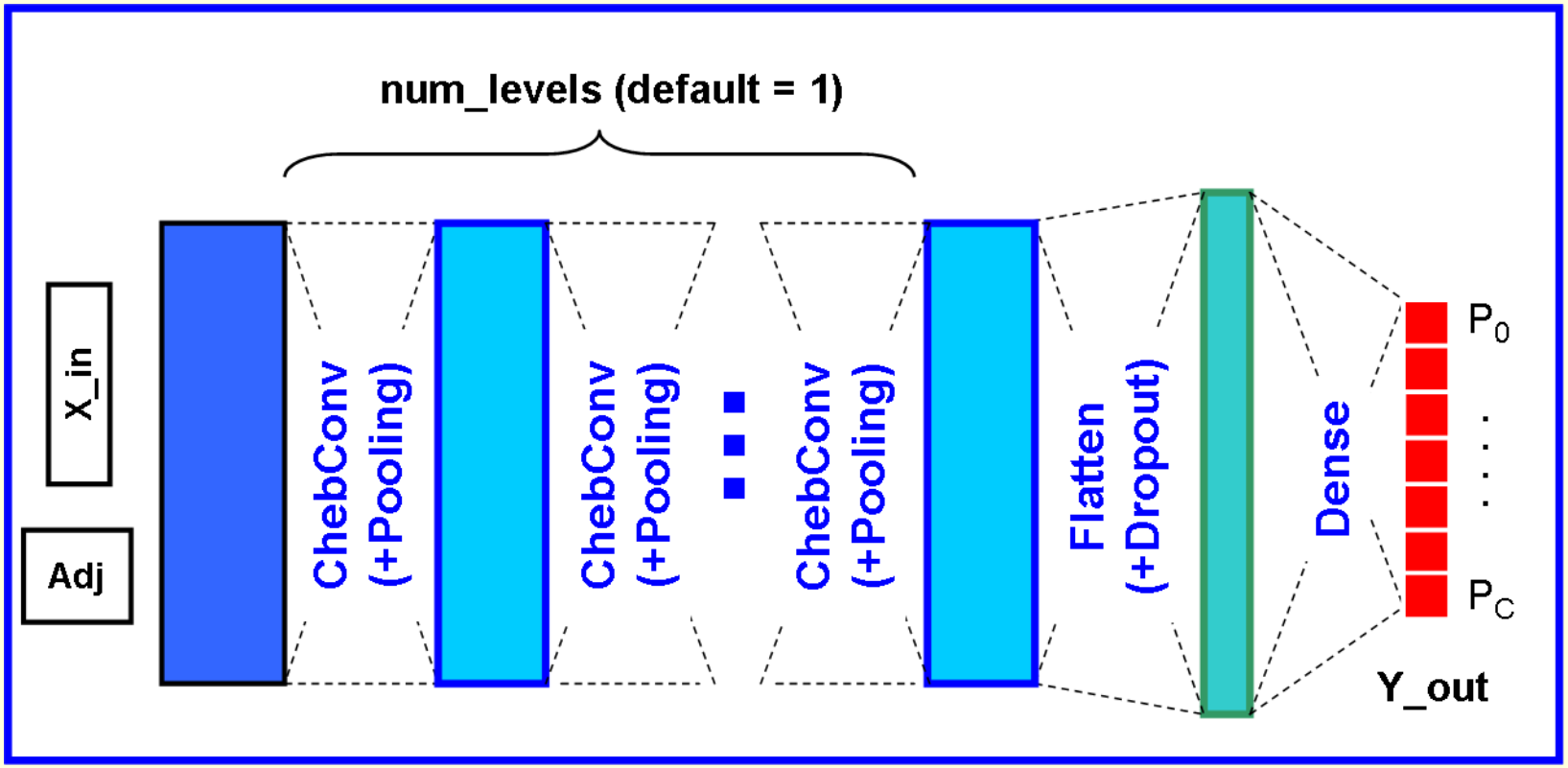


PPI: protein-protein interaction

The GCN_Cancer model

Spektral: ***https://graphneural.network***

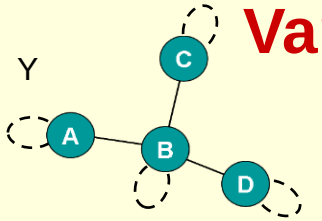
https://hpc.nih.gov/apps/GCN_Cancer.html



Features of the Keras implementation:

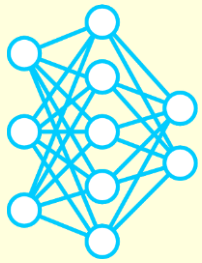
- classifies each sample as one of **C=34 types**
- supports GCNConv, **ChebConv** (=default) and Dense as the 1st layer in the network model
- **optionally**, allows for **balancing** of the number of training samples across different classes
- **optionally,** allows for **Pooling,** with two supported types of layers: MinCutPool and DiffPool
- **optionally,** allows for **multiple levels** of Filtering (+ Pooling)

Vanilla Graph Convolution: the GCNConv layer

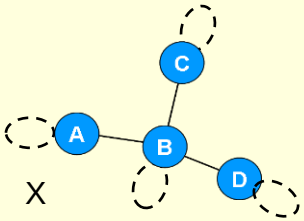


degree matrix, normalized adjacency matrix

Kipf, T.N., Welling, M.: arXiv preprint (2016)



GCNConv filtering ⇑



	A	B	C	D
A	0	1	0	0
B	1	0	1	1
C	0	1	0	0
D	0	1	0	0

$A = \{a_{ij}\}$
Adjacency matrix

	A	B	C	D
A	1	0	0	0
B	0	3	0	0
C	0	0	1	0
D	0	0	0	1

$D = \mathrm{diag}(\Sigma_j\, a_{ij})$
Degree matrix

denisovga@biowulf:/data/denisovga/1_DL_Course/6_GCNs

```
module load GCN_Cancer
python
>>> import numpy as np
>>> import scipy.linalg as sl
>>> A = np.array([[0,1,0,0],[1,0,1,1],
...               [0,1,0,0],[0,1,0,0]])
>>> D = np.array([[1,0,0,0],[0,3,0,0],
...               [0,0,1,0],[0,0,0,1]])
>>> I = np.array([[1,0,0,0],[0,1,0,0],
...               [0,0,1,0],[0,0,0,1]])
>>> A_tilde = A + I
>>> D_tilde = D + I
>>> D_tilde_minus_one_half = \
... sl.fractional_matrix_power(D_tilde,-0.5)
>>> A_hat = D_tilde_minus_one_half * A_tilde \
...      * D_tilde_minus_one_half
                                  16,37          Top
```

$I =$

	A	B	C	D
A	1	0	0	0
B	0	1	0	0
C	0	0	1	0
D	0	0	0	1

	A	B	C	D
A	1	1	0	0
B	1	1	1	1
C	0	1	1	0
D	0	1	0	1

$\tilde{A} = A + I$

	A	B	C	D
A	2	0	0	0
B	0	4	0	0
C	0	0	2	0
D	0	0	0	2

$\tilde{D} = D + I$

including self-edges

GCNConv transformation

$$\boxed{Y_i = \frac{1}{\tilde{d}_i} \sum_{\text{one-hop neighbors and self}} X_j \cdot w_j + b_i}$$

$\tilde{d}_i$ = number of immediate neighbors of the node i, incl. itself

$$= \boxed{Y = \hat{A} \cdot W \cdot X + b}$$

GCNConv in matrix form

where: $\hat{A} = \tilde{D}^{-1/2} \cdot \tilde{A} \cdot \tilde{D}^{-1/2} =$

Normalized adjacency matrix

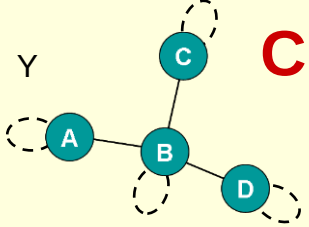
	A	B	C	D
A	0.5	0	0	0
B	0	0.25	0	0
C	0	0	0.5	0
D	0	0	0	0.5

spektral.layers.GCNConv(num_channels, activation=None, …)

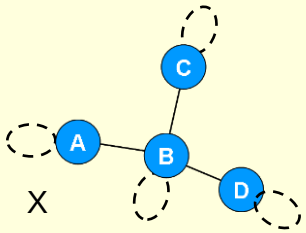
Chebyshev Convolution: the ChebConv layer

spectral approach, Convolution theorem, Fast Fourier transform, Graph Laplacian, eigenvectors, Chebyshev polynomials

M.Defferard et al, arXiv preprint (2016)



ChebConv filtering ⇑



Main ideas from convolution of discrete 1D signals:
"Regular" convolution in the time / spatial domain: $(f * g)[n] = \sum_{k=-\infty}^{\infty} f[n-k] \cdot g[k]$

From the **Convolution theorem**: $f * g = F^{-1}\{ F\{ f \} \cdot F\{ g \}\}$, where $F\{\ \}$ = Fourier transform

Fast Fourier transform: decomposing a signal into **eigenfunctions of Laplacian L** = $- d^2/dt^2$

$x[k] = \sum_n [\ X_c[n] \cdot cos(2\pi k\, n / N) + X_s[n] \cdot sin(2\pi k\, n / N)\ ]$

Graph Convolution:

X = vector of node features

Graph Fourier transform:

$\hat{X} = F(X) = U^T \cdot X,$

inverse: $X = F^{-1}(\hat{X}) = U \cdot \hat{X}$

	A	B	C	D
A	1	-1	0	0
B	-1	3	-1	-1
C	0	-1	1	0
D	0	-1	0	1

Graph Laplacian:
L = D - A

4
0
1
1

Eigenvalues of L:
$\lambda = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$

-0.5	3e-4	-0.8	0.3
-0.5	0	0	-0.9
-0.5	-0.7	0.4	0.3
-0.5	0.7	0.4	0.3

Eigenvectors of L:
$U = \{u_1, u_2, u_3, u_4\}$

denisovga@biowulf:/data/denisovga/1_DL_Cou...

```
>>> L = D - A
>>> lambd = np.linalg.eigvals(L)
>>> _, U = np.linalg.eigh(L)
>>> D_minus_one_half = \
    sl.fractional_matrix_power(D,-0.5)
>>> L_hat = D_minus_one_half * L * \
    D_minus_one_half
>>> L_tilde = (2/max(lambd))*(I-L_hat)-I
                                8,42          All
```

$$Y = \sum_{k=0}^{K} T^{(k)}(\tilde{L}) \cdot W \cdot X + b$$

$\tilde{L} = (2/\lambda_{max}) \cdot L - I$ = normalized Graph Laplacian

$T^{(k)}(x)$ = Chebyshev polynomials:

$T^{(0)}(x) = I \qquad T^{(1)}(x) = x$

$T^{(n+1)}(x) = 2x \cdot T^{(n)}(x) - T^{(n-1)}(x)$

employs K-hop neighborhood of a node to update its value

spektral.layers.ChebConv(channels, K=1, …)

Classification error: using the original / imbalanced input data

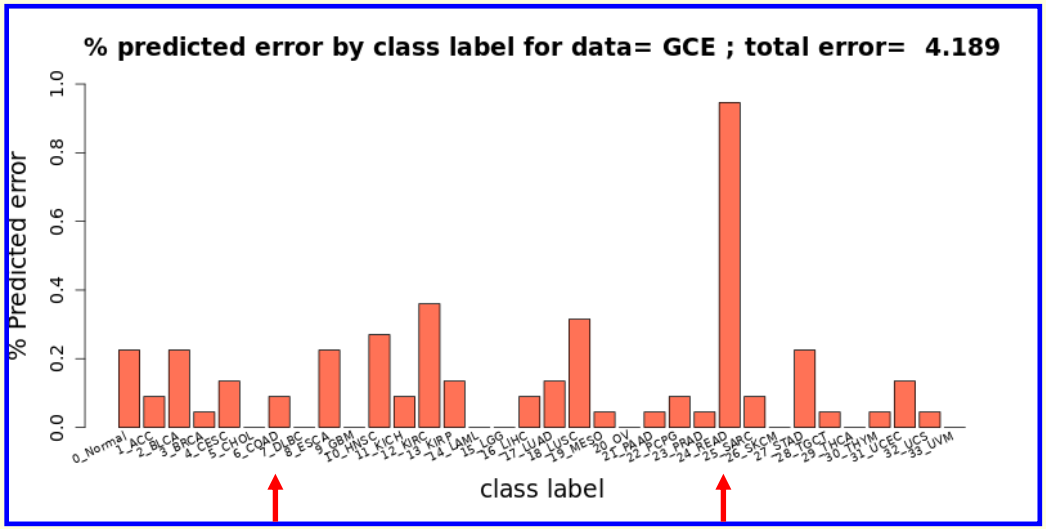
R.Ramirez et al., Frontiers in Physics (2020)

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

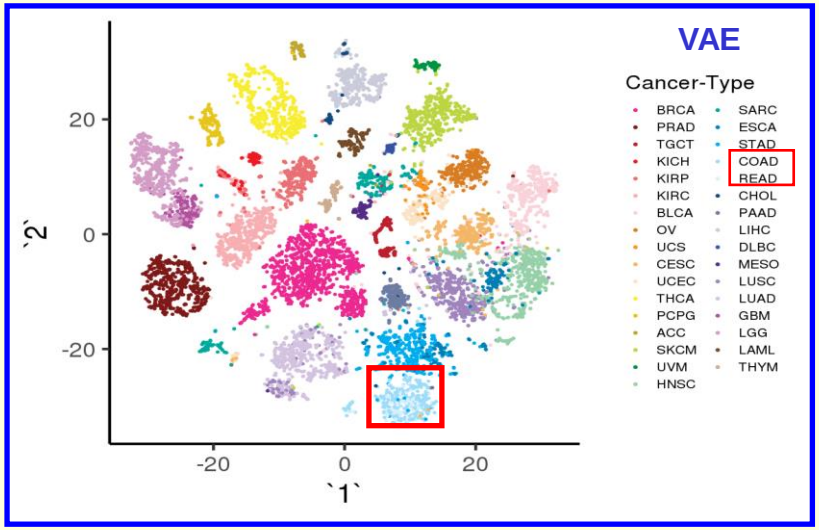
Data type	GCNNet	DenseNet	ChebNet, K=1	ChebNet, K=1 (orig. publication)
GCES	9.28%	5.7%	4.5%	5.76±0.251%
GCE	6.4%	4.32%	4.19%	5.77±0.146 %
PPIS	7.79%	5.09%	4.37%	5.39±0.107%
PPI	6.44%	5.06%	5.02%	11.02±0.883%

CONCLUSIONS:

1) **DenseNet overperforms GCNNet**
2) **ChebNet is the best**
3) **The ChebNet error is primarily due to misclassification of COAD / READ**



6_COAD Colon adenocarcinoma
24_READ Rectum adenocarcinoma



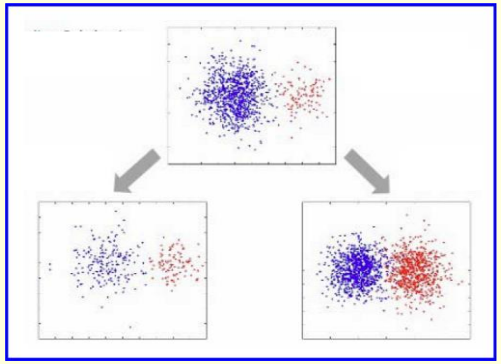
tSNE results from the class #3: COAD and READ are not well separable

Classification error: using preprocessed / balanced input data

naïve balancing, synthetic minority oversampling technique (SMOTE)

SMOTE variants: G.Kovacs, SMOTE variants – Neurocomputing (2019)

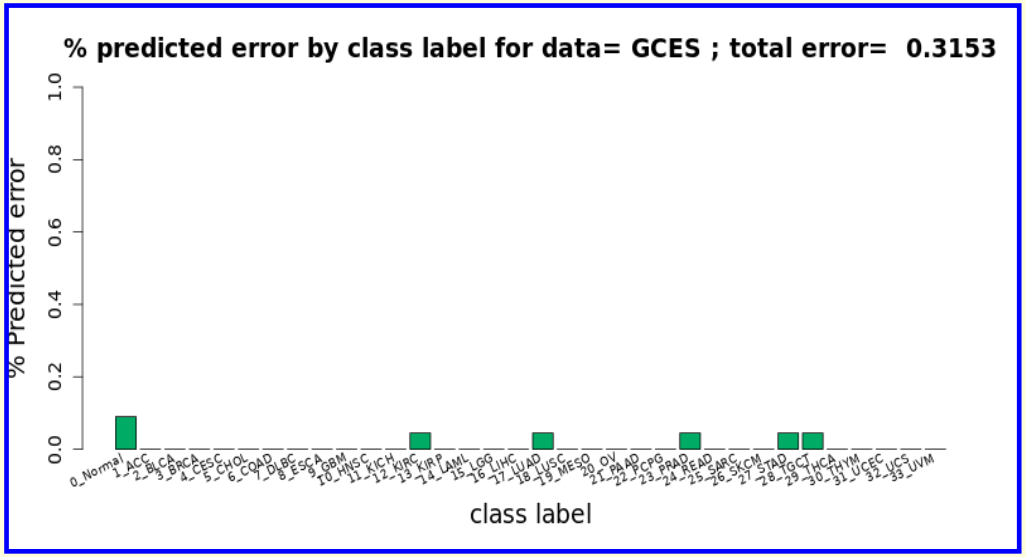
Imbalanced data



Undersampling **Oversampling**

- **naïve balancing**: duplicating randomly selected minority samples
- using **SMOTE variants** (total = 85): **MWMOTE** and **LLE_SMOTE** perform well

Data type	ChebNet, K = 1	ChebNet, K=10	ChebNet, K=20	ChebNet, K=100	
GCE	0.09%	0%	0%	0%	**0 errors**
GCES	0.54%	0.32%	0.32%	0.32%	
PPI	0.18%	0.14%	0.046%	0.046%	**1 error**
PPIS	0.54%	0.41%	0.32%	0.14%	



CONCLUSION:

1) balancing the # of training samples across classes and

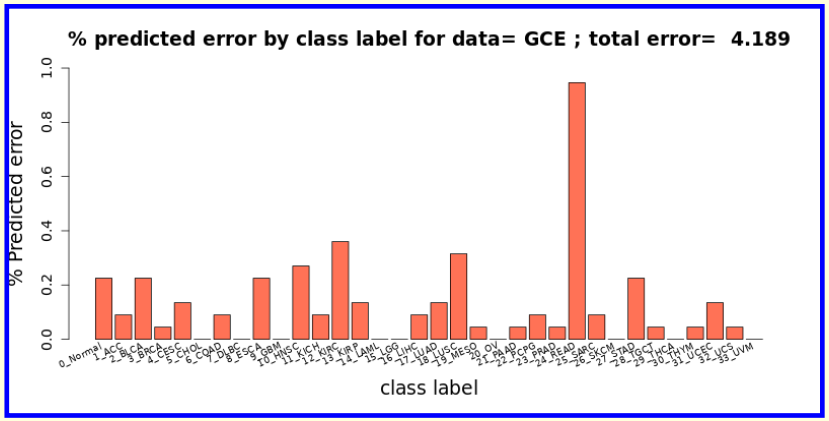
2) using ChebNet at higher K can dramatically improve the accuracy of class predictions

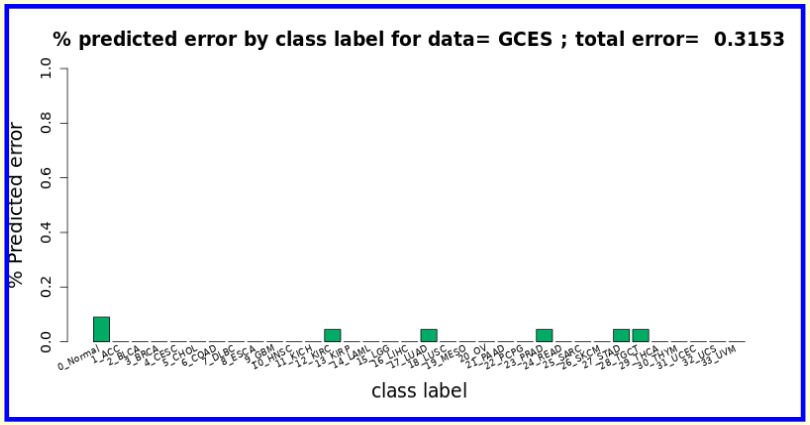
How to run the GCN_Cancer app on Biowulf

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

denisovga@biowulf:/data/denisovga/1_DL_Course/6_GCNs

```
sinteractive  --mem=120g --gres=gpu:p100:1,lscratch:100 \
              --cpus-per-task=14
module load gcn_cancer

ls $GCN_CANCER_SRC
data.py models.py options.py predict.py preprocess.py train.py visualize.R

cp -r $GCN_CANCER_DATA/* .

preprocess.py -h
preprocess.py -d GCES -B -S MWMOTE
preprocess.py -d PPI  -B

train.py   -h
train.py   -d GCE  -D 0.1
train.py   -d GCES -K 100 -D 0.1 -B -S MWMOTE
train.py   -d PPI  -m DenseNet -D 0.1 -B

predict.py -h
predict.py -d GCE  -D 0.1
predict.py -d GCES -K 100 -D 0.1 -B -S MWMOTE
predict.py -d PPI  -m DenseNet -D 0.1 -B

visualize.R  results/ChebNet.GCE.D_0.1.tsv
visualize.R  results/chebnet.K_100.gces.D_0.1.B.S_MWMOTE.tsv
                                                        25,64         All
```





Summary

1) Intro using simple / prototype examples
   - intro to the graph classification task and graph-related terminology
   - GCNNet model requires a second input - the adjacency matrix
   - GCNNet vs MLP: Dense layer ≈ GCNConv layer with adjacency matrix of ones
   - GCNConv vs Conv1D and Conv2D: Filtering and Pooling
   - GCNConv is an analog of Conv1D / Conv2D with filter of size = 3
   - imbalanced input data and the presence of singletons may reduce the classification accuracy
2) The GCN_Cancer application:
   - two types of gene association in the GCN_Cancer data: gene co-expression (GCE) and protein-protein interaction (PPI)
   - GCN_Cancer model outputs a vector of class probabilities
   - the GCNConv layer implements a vanilla graph convolution
   - the meaning of hyperparamreter K in the ChebConv layer
   - the techniques for data balancing: naïve balancing vs SMOTE variants
   - data balancing dramatically reduces the classification error
   - the ChebConv with higher K allows further reduction in the classification error

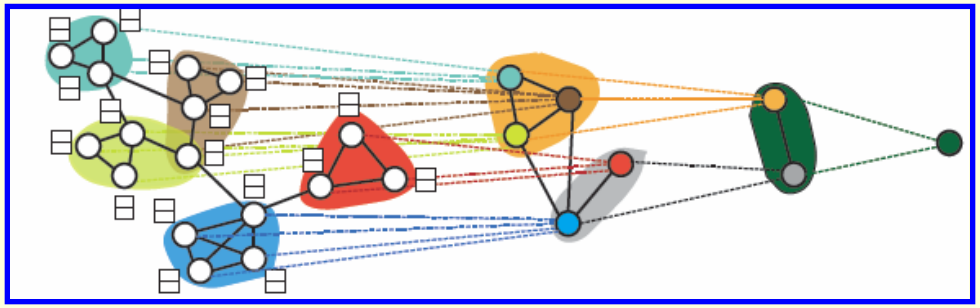
BACKUP SLIDES

Pooling layers: DiffPool and MinCutPool

DiffPool: R.Ying et al, arXiv:1806.08804 (2019)

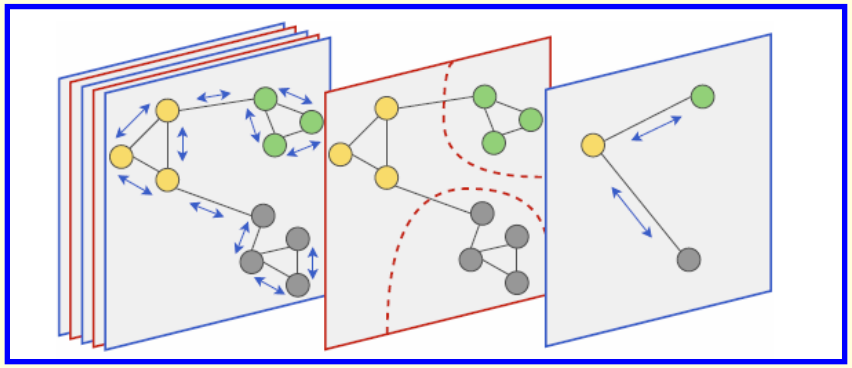
MinCutPool: F.M.Blanchi et al, arXiv:1907.00481 (2020)

Differentiable Pooling (agglomerative)



- iteratively **aggregate** “close” nodes
- compute a hierarchical representation of the graph
- stop when the target number of clusters is reached

MinCutPool (divisive)

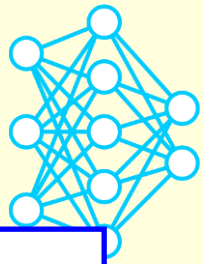


- **partition** nodes into a specified number C of clusters by removing the minimum # of links:

$$\frac{1}{K}\sum_{k=1}^{K} \frac{(\#\ \textit{links within cluster } k)}{\boxed{(\#\ \textit{links between cluster } k \textit{ and the rest of the graph})}} \rightarrow \textit{max}$$

Graph convolution with polynomial filters

https://distill.pub/2021/understanding-gnns
https://csustan.csustan.edu/~tom/Clustering/GraphLaplacian-tutorial.pdf



1) **Polynomials of Laplacian** $p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \ldots + w_d L^K = \sum_{i=0}^{K} w_i L^i.$
can be though of as the **equivalents of filters in CNNs**

2) More specifically, if X = a vector of features of all nodes in a graph, then the convolved vector Y will be: $Y = p_w(L)\, X$. In particular, when $K = 1$, the v-th component of Y will be computed based on X_v and its **one-hop neighbors**:

$$Y_v = D_v X_v - \sum_{u \in \mathcal{N}(v)} X_u$$

3) Likewise, it can be shown that for any K, the v-th component of Y will be computed based on the features of the nodes located at distance **no more then K-hops away** from the node v. This means that **polynomial filters are localized.**

4) ChebNet further refines this idea of polynomial filters by looking at polynomial filters of the form

$$p_w(L) = \sum_{i=1}^{d} w_i T_i(\tilde{L}) \qquad \tilde{L} = \frac{2L}{\lambda_{\max}(L)} - I_n.$$

5) Eigenvalues of L are all non-negative, and one of them is always zero. $\tilde{L}$ is effectively a scaled-down version of L, with **eigenvalues guaranteed to be in the range [-1, 1]**

The SMOTE variants LLE_SMOTE and MWMOTE

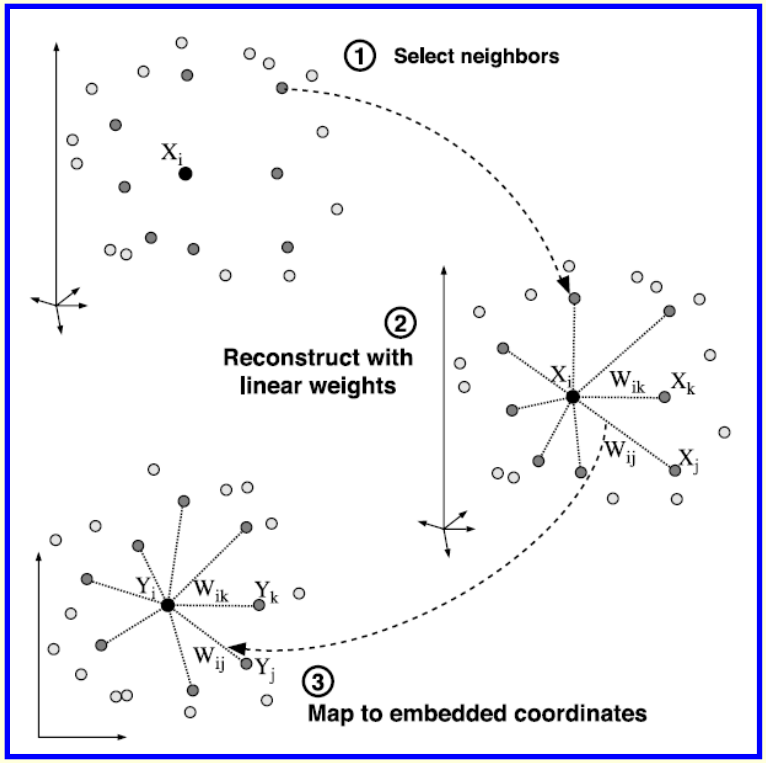
LLE SMOTE: J.Wang et al., ICSP 2006

MWMOTE: S.Barua et al, IEEE Trans. On Knowledge and Data Eng. (2014)

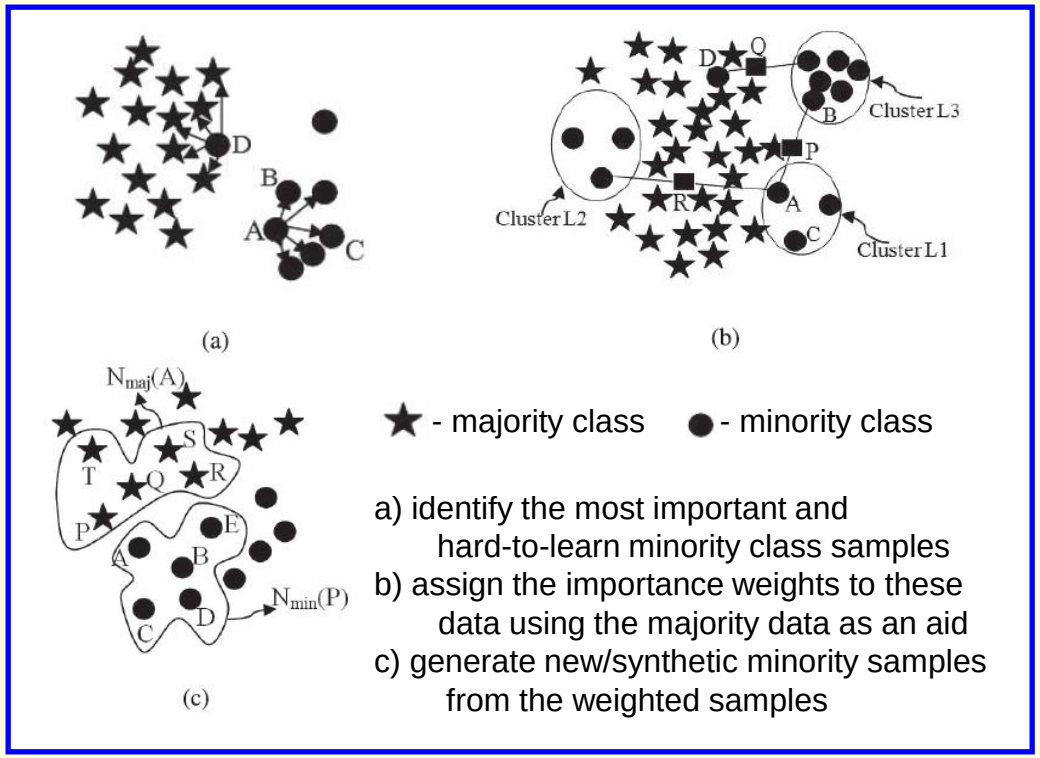
SMOTE variants: SMOTE, distance_SMOTE, SMOTE_D, SMOTE_TomekLinks, **LLE_SMOTE**, **MWMOTE,** NT_SMOTE, OUPS, Gazzah, ROSE, …

(total = 85)

LLE SMOTE: Locally Linear Embedding SMOTE



MWMOTE: Majority Weighted Minority Oversampling Technique



a) identify the most important and hard-to-learn minority class samples
b) assign the importance weights to these data using the majority data as an aid
c) generate new/synthetic minority samples from the weighted samples