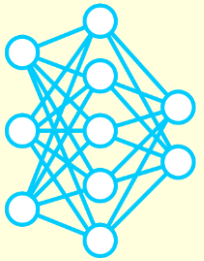
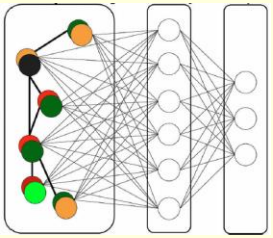


Deep Learning by Example on Biowulf

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

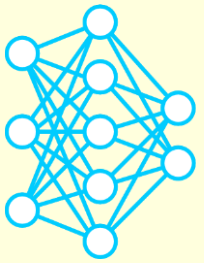
Gennady Denisov, PhD



Handouts: https://hpc.nih.gov/training/handouts/DL_by_Example6_20250225.pdf

Intro and goals

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



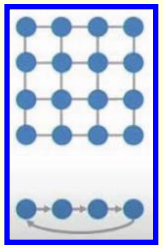
Question: how do we represent and learn the structure of data?

Representation

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

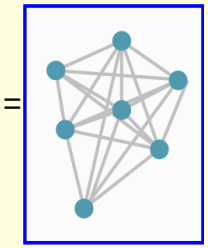
image

sequence



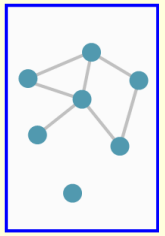
Gene expression data (class #3) - could be represented by a complete graph

degree = N-1



Data with irregular structure: should be represented by a graph

degree - variable



Learning

Convolutional or Recurrent layers

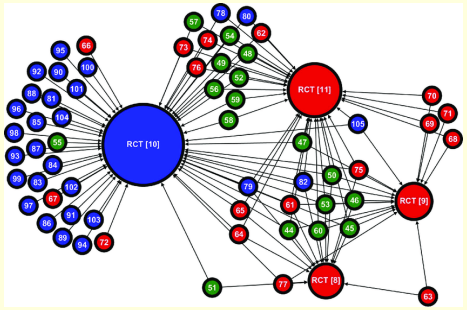
Dense layers

Graph Convolutional (+ Graph Pooling) layers

Non-bio examples:

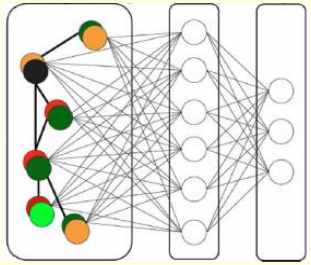
Social network: nodes = individuals, edges = connections

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



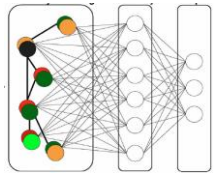
Examples overview

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

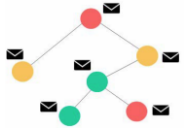
Class #	1	2	3	4	5	6	7
Bio app	Bioimage segmentation / fly brain connectome	Genomics / prediction of function of non-coding DNA	Genomics / reduction of dimensionality of cancer transcriptome	Bioimage synthesis / developmental biology	Drug molecule design	Genomics / classification of cancer types	Drug molecule property prediction
Neural network type	Convolutional	Recurrent or 1D-Convolutional	Autoencoder	Generative Adversarial	Reinforcement Learning	Graph Convolutional	Message Passing, Transformer
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
- 1st of two classes to discuss GNNs; **primary purpose: to demystify the GCNs**
- **"Transition" examples**: they **can be handled by both GCNs and MLP/DenseNet,** so their performances can be compared
- Why Graph? It **imposes constraints / provides additional knowledge** about the data, which allow for **more accurate model predictions** as compared to the constraint agnostic models, e.g. MLP, as will be **illustrated with a toy example**

GCN

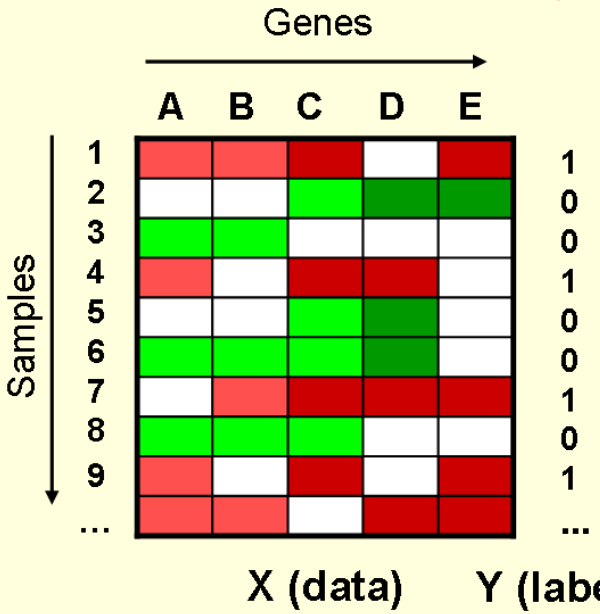


MPN

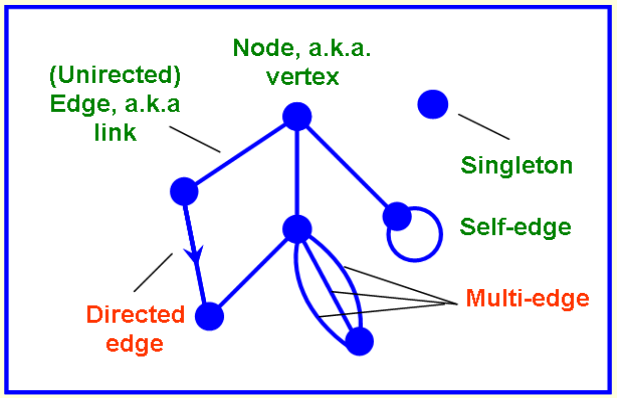


Prototype/toy example #1: the graph classification task

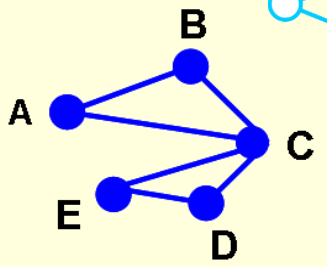
gene expression matrix, node features, undirected graph, singletons, adjacency matrix, graph classification task



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



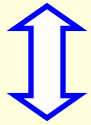
Graph notation



Node = gene
Node feature = gene expression level (=0 or 1)
(Undirected) Edge = association between couple of genes
Module: includes genes with equal probability of being expresses

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.



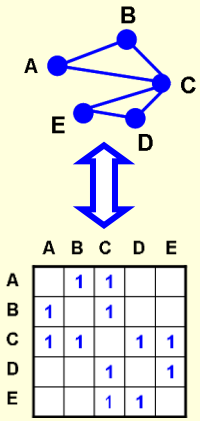
	A	B	C	D	E
A		1	1		
B	1		1		
C	1	1		1	1
D			1		1
E			1	1	

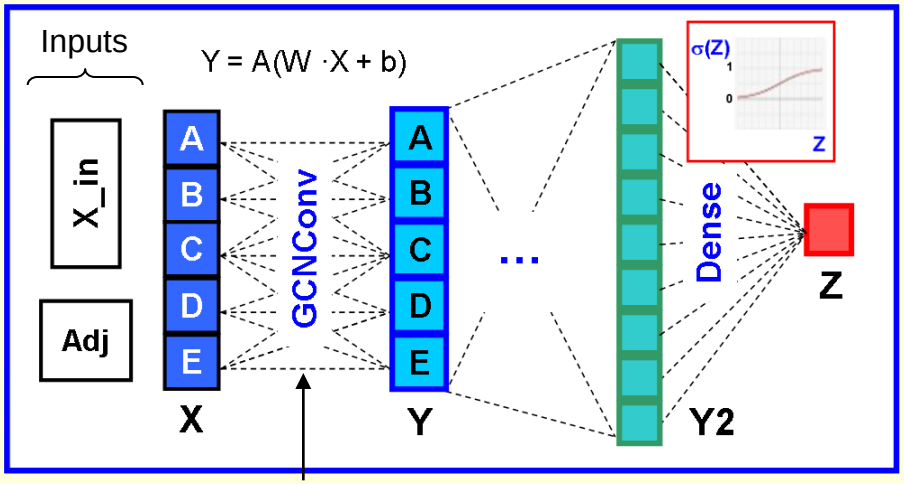
Adjacency matrix:

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

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

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

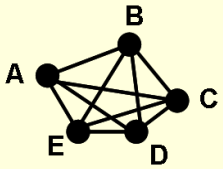




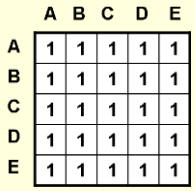
Graph Convolutional Network (GCN)

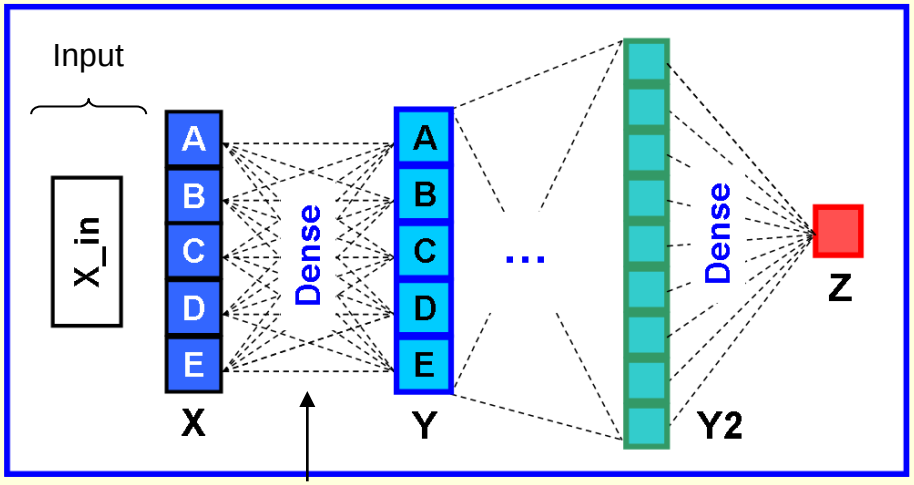
Missing links: A-D, A-E, B-D, B-E,D-A, E-A, D-B, E-B.
The corresponding weights are constrained to 0

Complete graph



Adjacency matrix of ones





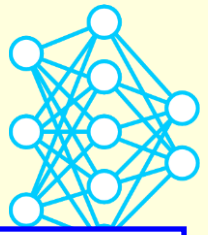
MultiLayer Perceptron (MLP)

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

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

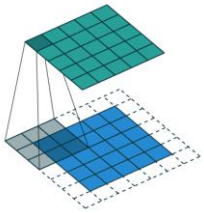
Prototype example #1 (cont.): GCNConv vs Conv1D / Conv2D layers

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



2D Convolution with 'same' padding

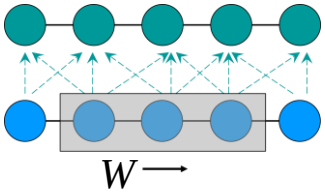
Output image Y



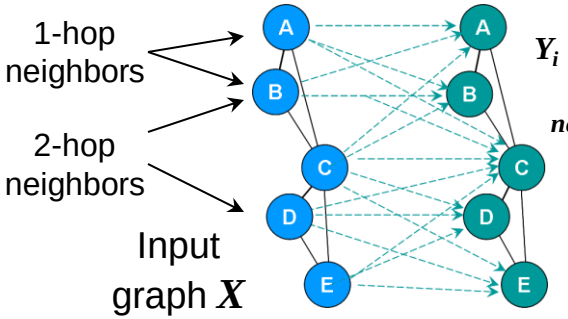
Input image X

1D Convolution with 'same' padding

$$Y_i = \sum_{kernel} w_{ij} * X_{ij} + b$$

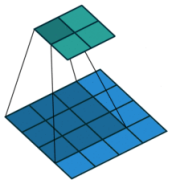


Filtering, a.k.a. Graph Convolution in the narrow sense

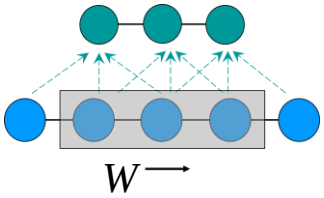


$$Y_i = \sum_{1\text{-}hop\ neighborhood} w_{ij} * X_{ij} + b$$

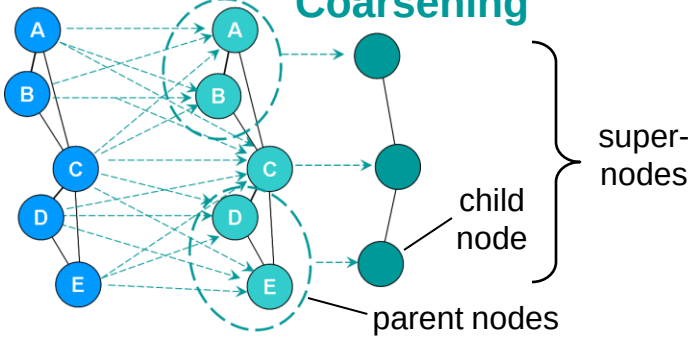
2D Convolution with 'valid' padding



1D Convolution with 'valid' padding



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



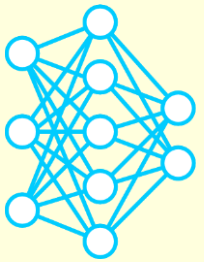
CONCLUSION: filtering performed by the GCNConv layer is analogous to Conv1D / Conv2D with 'same' padding and kernel of size = 3

NOTE: later on in this class, we will discuss yet another flavor of Graph Convolutional layer / transformation, the Chebyshev Convolution, which may be regarded as an analog of Conv1D / Conv2D with kernel of size > 3.

Prototype example #1 (cont.): training code

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

Spektral: *https://graphneural.network*



Header
- tensorflow
- spektral
- graph_net

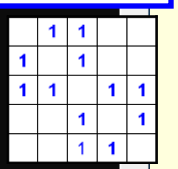
Get data
- modules
- adj matrix
- samples
  x_train
- labels
  y_train

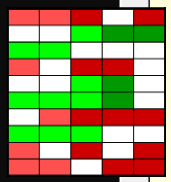
Define a model:
- Functional API
- GCNConv vs Dense
- loss and optimizer

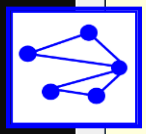
Run the model
- fit

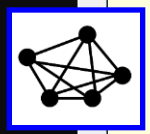
```
denisovga@biowulf:/usr/local/apps/DLBio/class6/bin
#!/usr/bin/env python
import tensorflow as tf, spektral, numpy as np, scipy.stats as stats
graph_net, n_g, n_train, bsize, epochs, prob=1, 5, 700, 50, 100, 0.7
np.random.seed(1); tf.random.set_seed(1)

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 ⇒ GCN ; 0 ⇒ MLP }

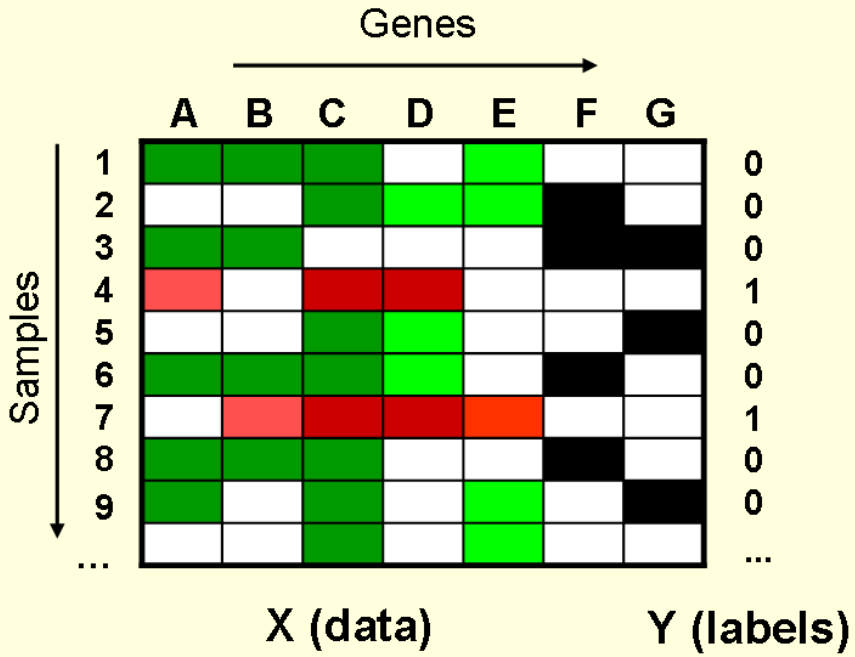








Prototype/toy example #2: is there a benefit in using GCN over MLP?

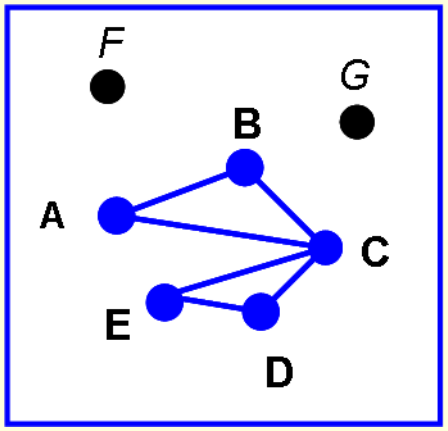


Two additional hyperparameters:

Imbalance ratio, IR (≥ 1)
= # green samples / # red samples

Number of singletons, n_s (≥ 2)
= # of genes that are not associated / co-expressed with other genes

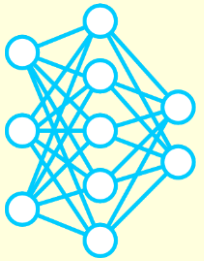
Undirected graph with singleton genes



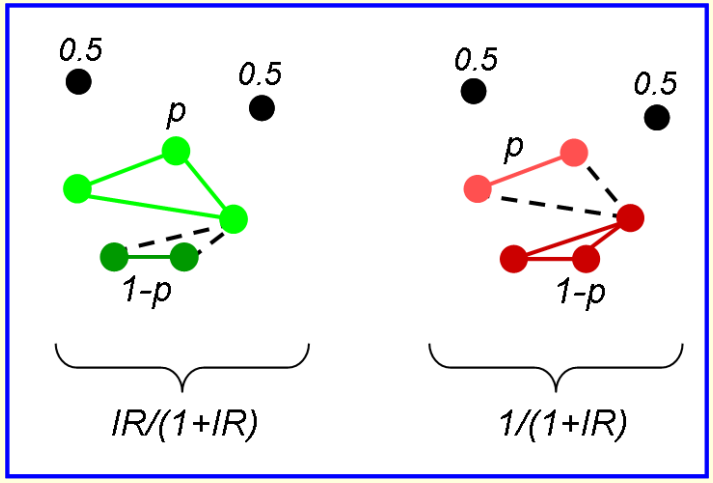
	A	B	C	D	E	F	G
A		1	1				
B	1		1				
C	1	1		1	1		
D			1		1		
E			1	1			
F							
G							

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

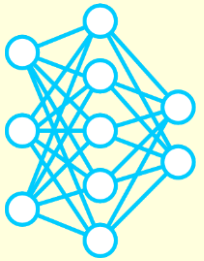
CONCLUSIONS:

- the presence of singtletons and the data imbalance are two dominant reasons for classification error

- IR=1: 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 balanced data

- IR=10: since MLP possesses more adjustable parameters than GCN, it provides more flexibility in handling the challenge of data imbalance, and therefore can outperform the GCN on imbalanced data

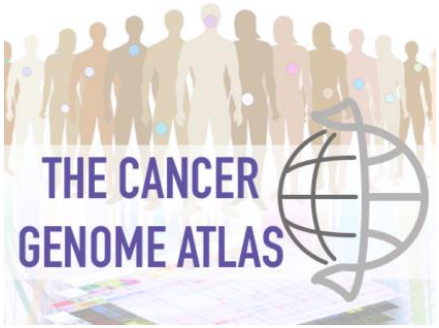
How to run the prototype/toy models on Biowulf?



```
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)

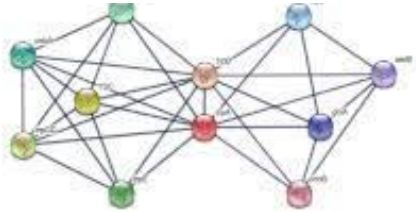


NIH program led by the NCI and NHGRI

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**

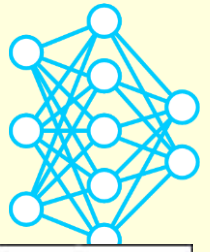
Balance training data; optionally, compute custom adjacency matrices

Use ~80% of ground truth data

Predict sample type using ~20% of data

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 layer
- ChebConv layer

Get data
- GCE, GCES
- PPI, PPIS
- data balancing
- SMOTE variants

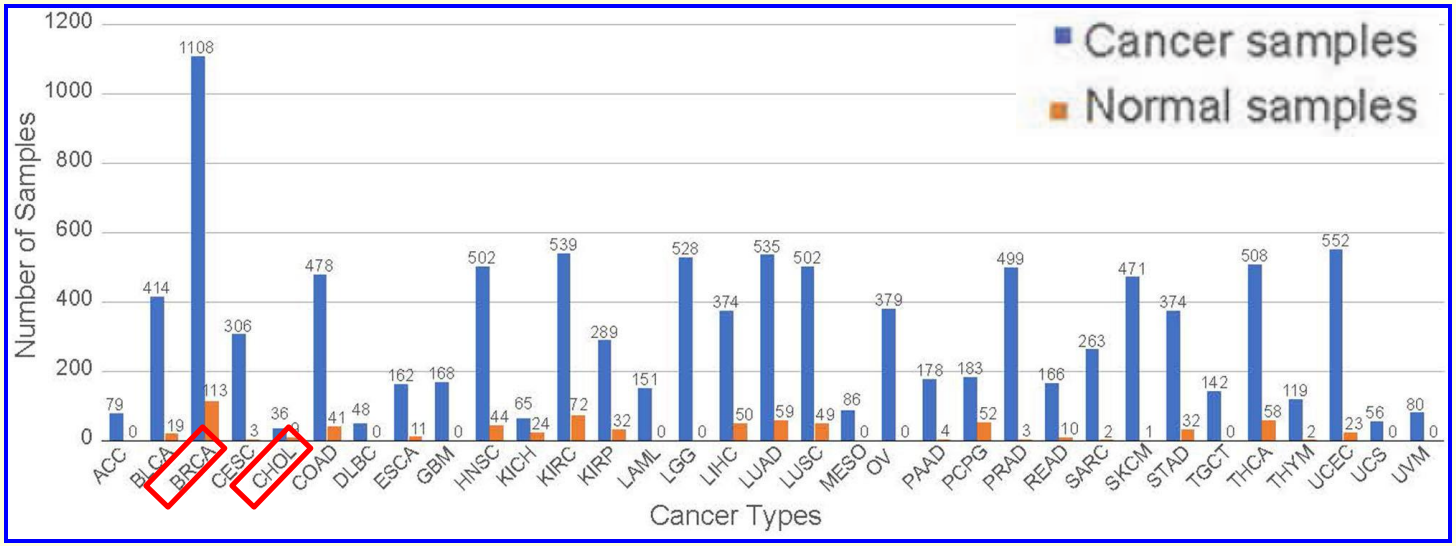
Run the models
- classification error

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, GCES
gene expr. Spearman correlation coefficients
+ threshold=0.6
Size: 3866 x 3866

PPI, PPIS
STRING association confidence scores
+ threshold=0.6
Size: 4444 x 4444

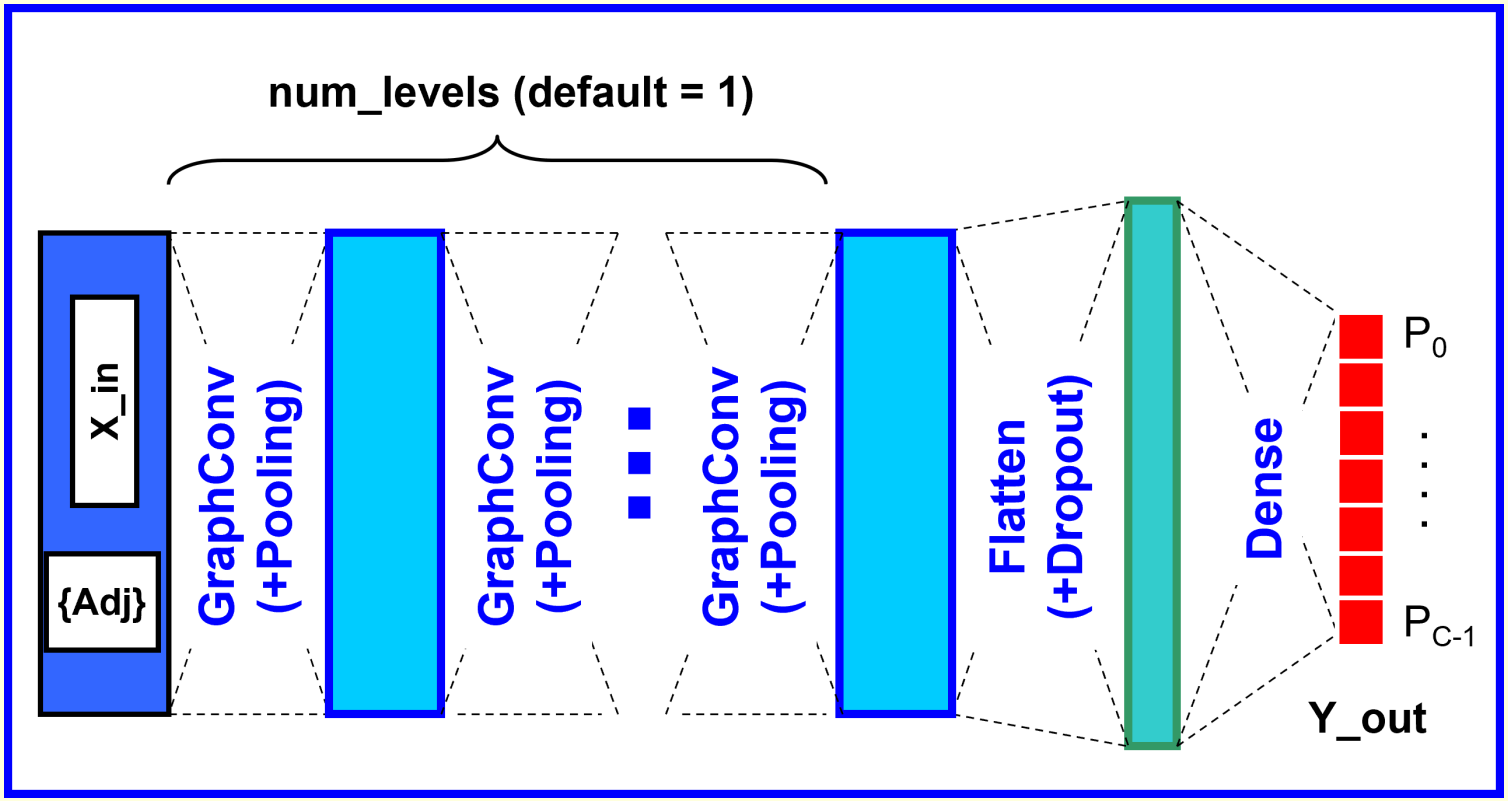
GCE: gene co-expression

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:

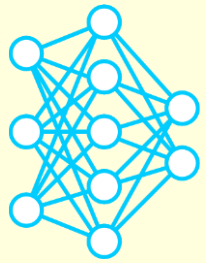
- classifies each sample as one of **C=34 types**
- supports GCNConv, **ChebConv** (=default) and Dense as the 1st layer in the network model
- allows for **balancing** of the number of training samples across different classes
- **optionally,** allows for **Pooling,** with two supported types of layers: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

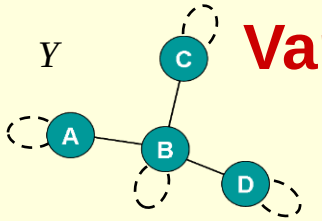
Vanilla Graph Convolution: the GCNConv layer

degree matrix, normalized adjacency matrix

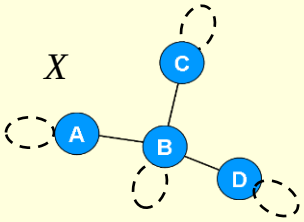
Kipf, T.N., Welling, M.: arXiv preprint (2016)

Spektral: https://graphneural.network





GCNConv filtering ⇑



$$Y_i = \frac{1}{\tilde{d}_i} \sum_{\text{one-hop neighbors, incl. self}} X_j \cdot w_{ij} + b_i$$

$\tilde{d}_i$ = degree of i-th node, incl. self-loop

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

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

GCNConv in the matrix form
$W = N \times N$ matrix of adjustable weights,
where $N = dim(X)$, and b= bias

	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

$D = diag\{\Sigma_j a_{ij}\}$
Degree matrix

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

I

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

$\tilde{A} = A + I$

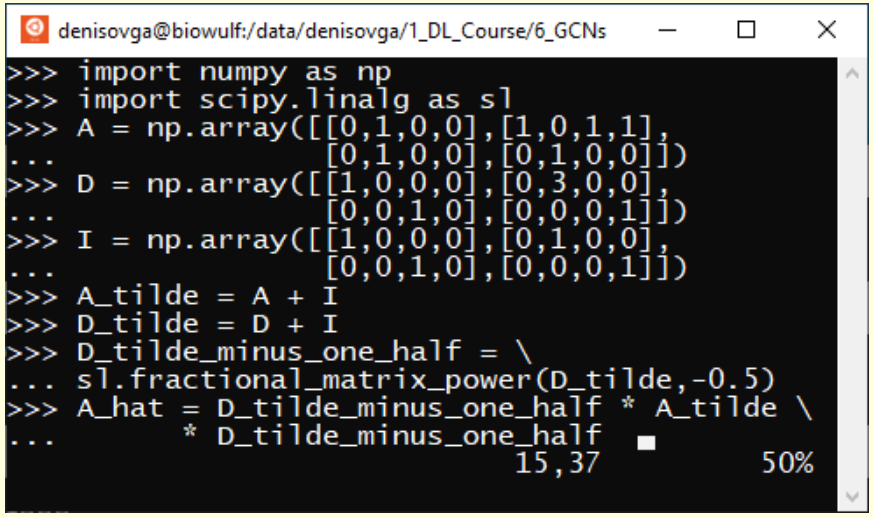
	A	B	C	D
A	2	0	0	0
B	0	4	0	0
C	0	0	2	0
D	0	0	0	2

$\tilde{D} = D + I$

Including self-loops

	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

$\hat{A}$



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

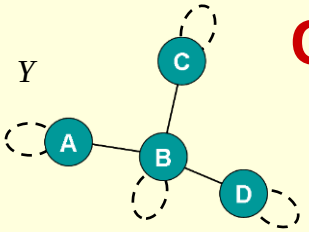
CONCLUSION:

GCNConv is a local transformation, involving current node and its one-hop neighbors

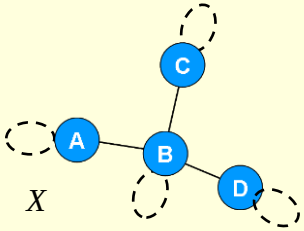
Chebyshev Convolution: the ChebConv layer

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

M.Defferard et al, arXiv preprint (2016)

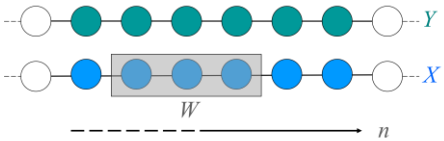


ChebConv filtering ⇑



1D Convolution

Regular Approach:



$$\Rightarrow (W * X)[n] = Y[n] = \sum_{i=-\infty}^{\infty} W[i] \cdot X[n-i]$$

Spectral Approach: Convolution theorem: $W * X = F^{-1}\{F\{W\} \cdot F\{X\}\}$; $F\{.\}$ = Fourier transform

Fast Fourier transform: $X[n] = \sum_{k=0}^{\infty} \left(a_k \cos\frac{2\pi k n}{N} + b_k \sin\frac{2\pi k n}{N} \right)$

$sin(\cdot)$ and $cos(\cdot)$ are the eigenfunctions of the 1D Laplacian $L = -d^2/dn^2$

all eigenvalues > 0, except one = 0

Graph Convolution:

X = N-vector of all node features

Graph Fourier transform:

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

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

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

u_1	u_2	u_3	u_4
-0.5	0.6	-0.5	-0.3
-0.5	0	0	0.9
-0.5	-0.7	-0.3	-0.3
-0.5	0.1	0.8	-0.3

Eigenvectors of L: $U = [u_1, u_2, u_3, u_4]$

```
denisovga@biowulf:/data/denisovga/1_DL_Cours...
>>> L = A_tilde - D_tilde
>>> 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
```

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

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

k – degree, $T^{(0)}(x) = 1$, $T^{(1)}(x) = x$,

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

$\tilde{L} = (2/\lambda_{max}) \cdot L - I$ - normalized Graph Laplacian; W = N x N weights matrix

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

CONCLUSIONS:

- **unlike GCNConv, ChebConv takes a spectral approach, which is nonlocal by nature**
- **ChebConv employs a K-hop neighborhood of a node to update its value**

Classification error: using models trained on the original / imbalanced data

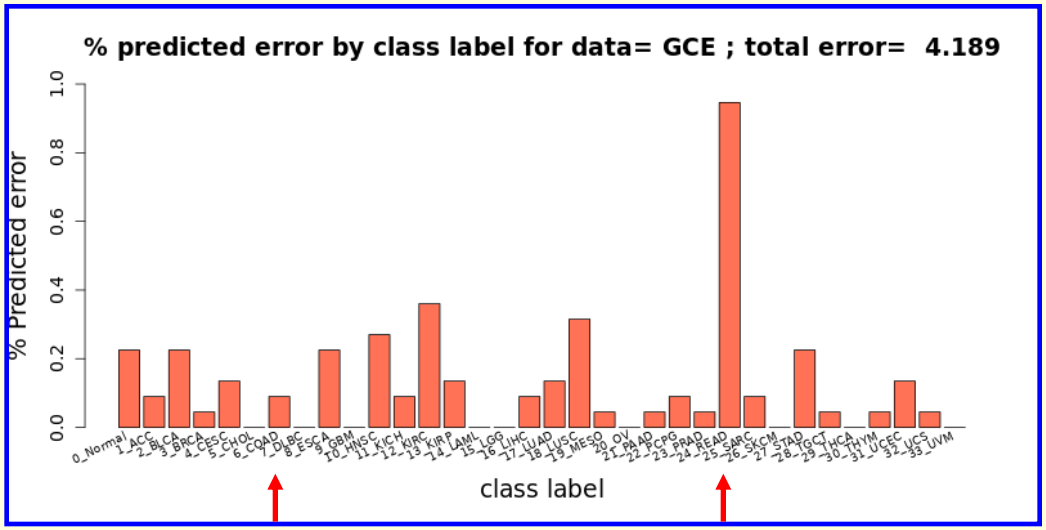
R.Ramirez et al., Frontiers in Physics (2020)

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

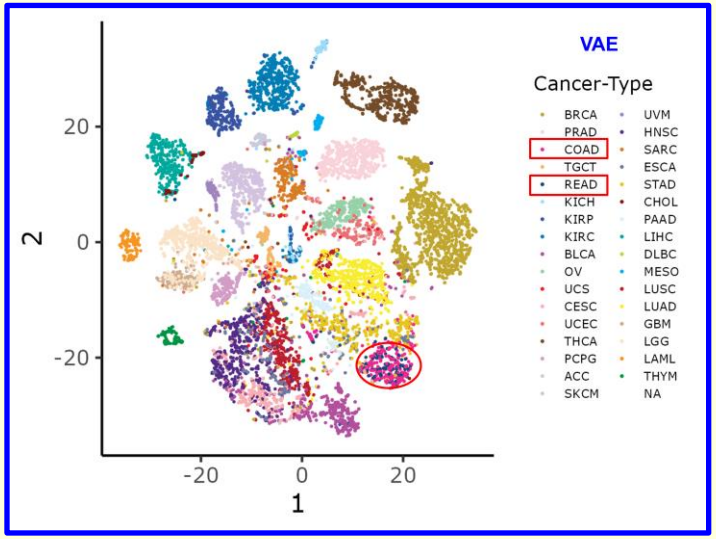
Data type	GCNNet	DenseNet	ChebNet, K=1	ChebNet, K=1 (orig. publication)
GCE	6.4%	4.32%	4.19%	5.77±0.146 %
GCES	9.28%	5.7%	4.5%	5.76±0.251%
PPI	6.44%	5.06%	5.02%	11.02±0.883%
PPIS	7.79%	5.09%	4.37%	5.39±0.107%

CONCLUSIONS:

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



6_COAD Colon adenocarcinoma
24_READ Rectum adenocarcinoma



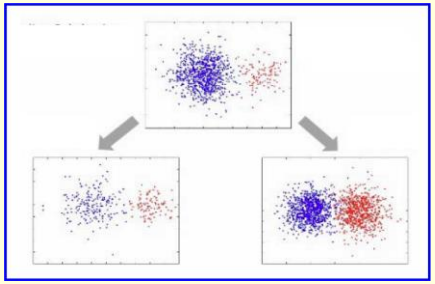
tSNE results from the class #3:
COAD and READ are not well separable

Classification error: using preprocessed/balanced training data and ChebNet with Chebyshev polynomials of higher degree

SMOTE: N.V.Chawla, J. Artif. Intel. Res. 16 (2002) 321–357

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

Imbalanced data



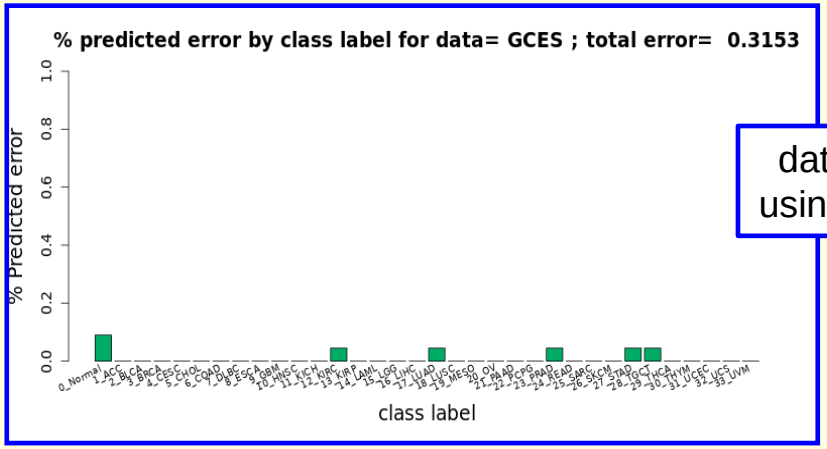
Undersampling Oversampling

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

0 errors

1 error

- **naïve balancing**: duplicating randomly selected minority samples (easy to implement, but training is unstable)
- **SMOTE variants**: currently, total = 92
- the **MWMOTE** and **LLE_SMOTE** work particularly well for the **TCGA data**
- while other good choices may exist, most of the SMOTE variants do not help



data balanced using MWMOTE

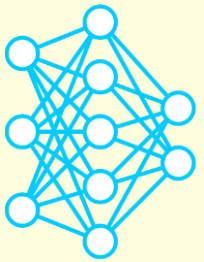
CONCLUSION:

The error in multi-class prediction from the TCGA data can be reduced dramatically by

- balancing the # of training samples across multiple classes using MWMOTE or LLE_SMOTE

- using a ChebNet model that employs Chebyshev polynomials of higher degree (e.g. K ≥ 20)

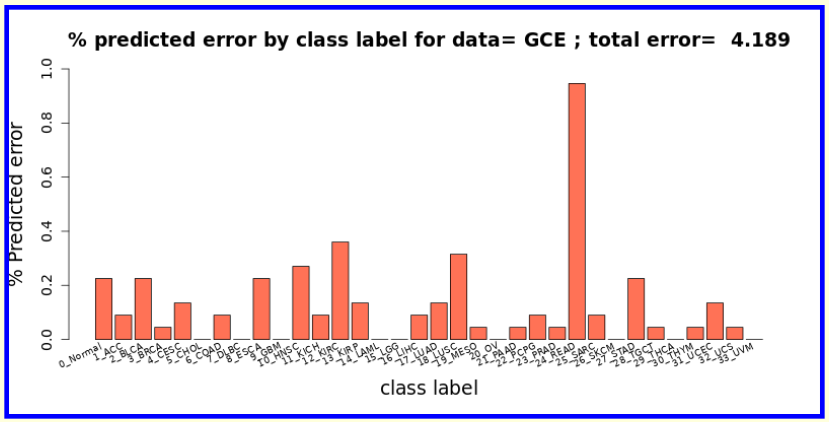
How to run the GCN_Cancer app on Biowulf

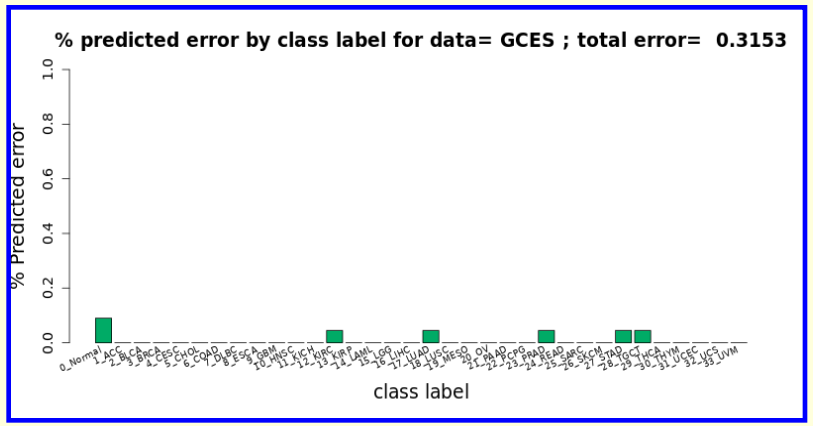


https://hpc.nih.gov/apps/GCN_Cancer.html

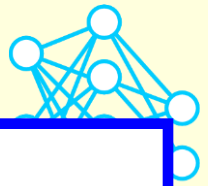
denisovga@biowulf:/data/denisovga/1_DL_Course/6_GCNs

```
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
```



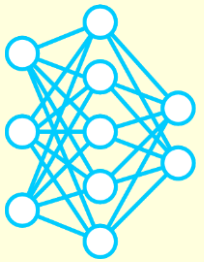


Summary



1) Intro using prototype/toy 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
   - graph provides additional knowledge about the data and imposes constraints on a model, which may results in more accurate class predictions as compared to the MLP

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 with MWMOTE or LLE_SMOTE dramatically reduces the classification error
   - the ChebConv layer that employs Chebyshev polynomials of higher degree allows further reduction in the classification error



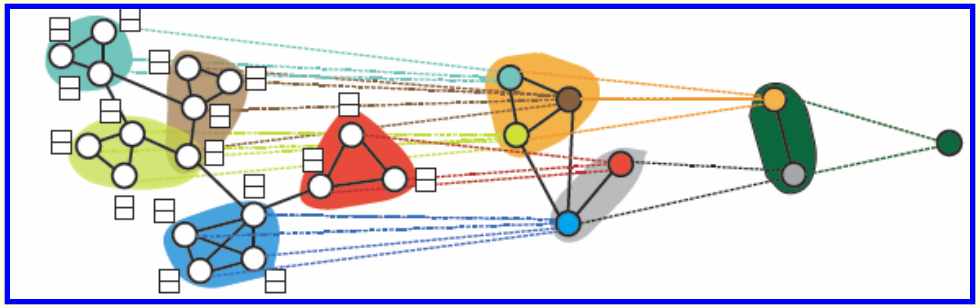
BACKUP SLIDES

Pooling layers: DiffPool and MinCutPool

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

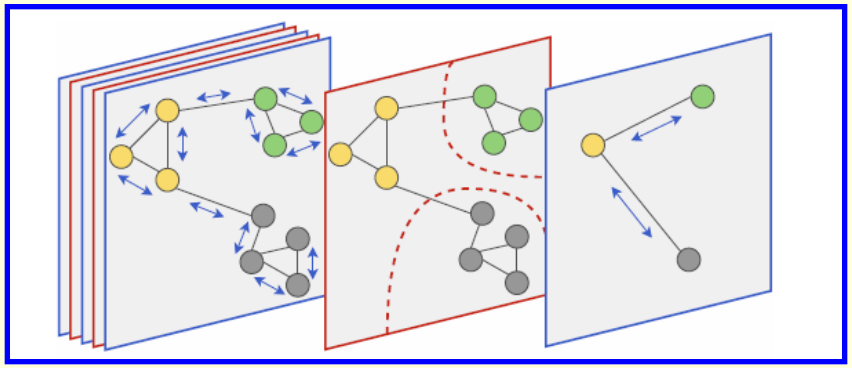
MinCutPool: F.M.Blanchi et al, arXiv:1907.00481 (2020)

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)

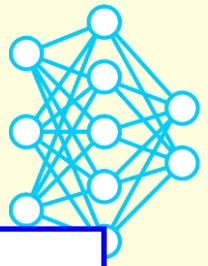


- **partition** nodes into a specified number C of clusters by removing the minimum # of links:

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

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

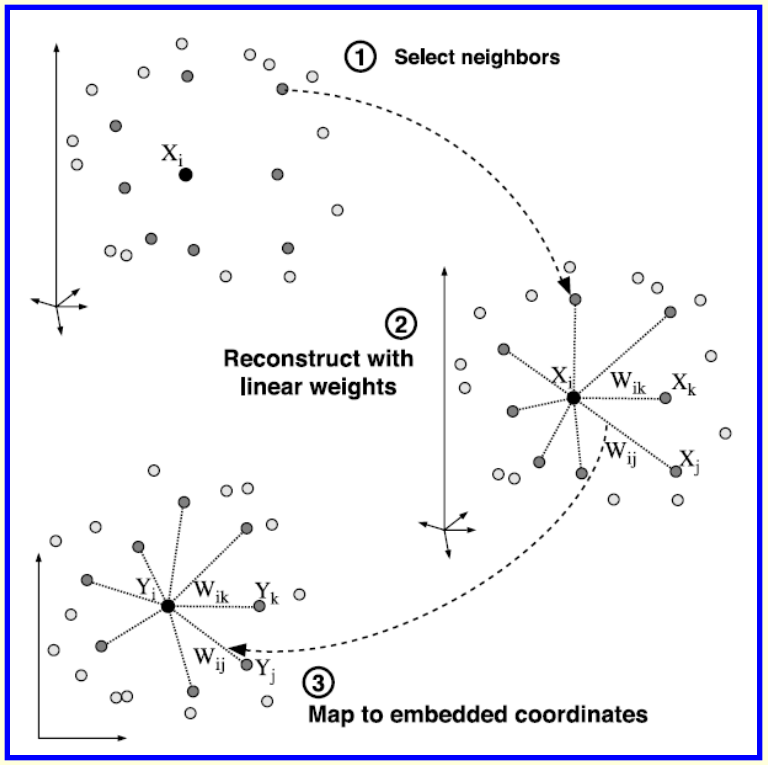
LLE SMOTE: J.Wang et al., ICSP 2006

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 = 92)

LLE SMOTE: Locally Linear Embedding SMOTE



MWMOTE: Majority Weighted Minority Oversampling Technique