Deep Graph Regularized Learning for Binary Classification

Minxiang Ye, Vladimir Stankovic, Lina Stankovic, Gene Cheung*

Electronic and Electrical Engineering Department, University of Strathclyde, Glasgow, UK
*Electrical Engineering and Computer Science Department, York University, Toronto, CA

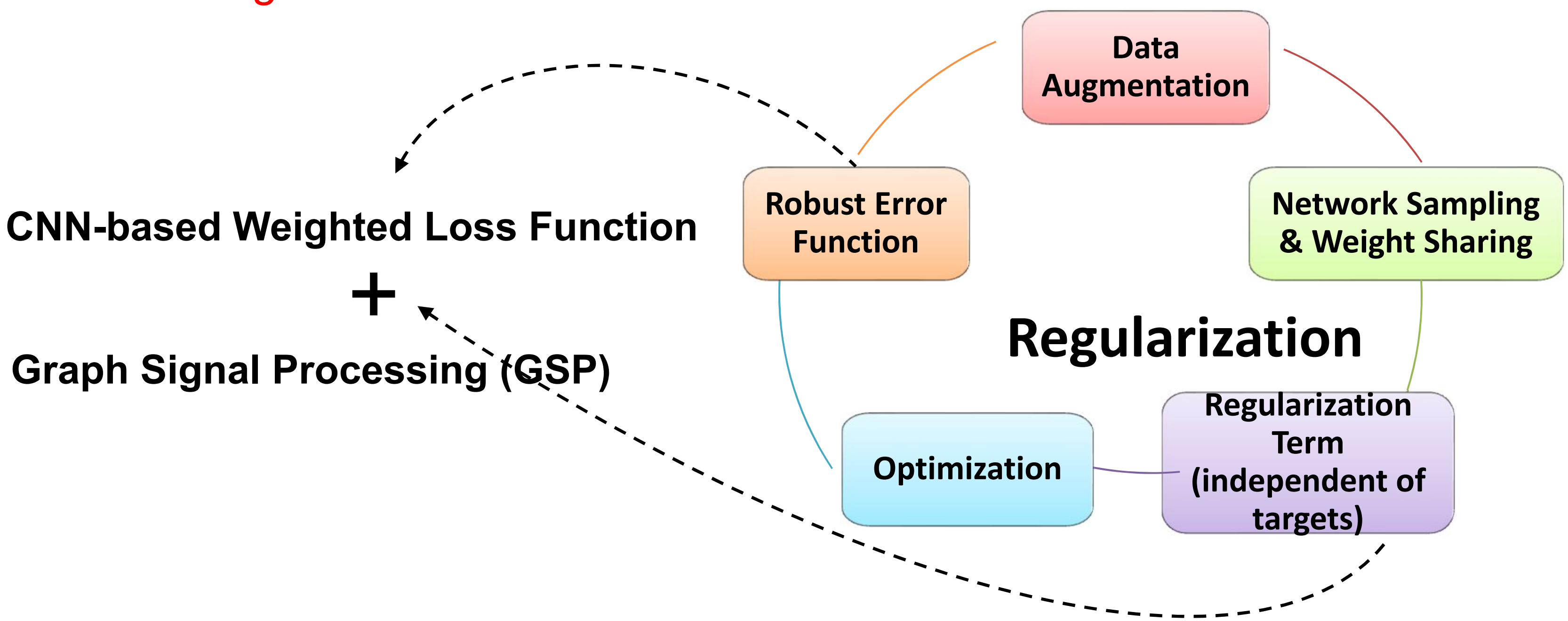
IEEE ICASSP-2019, Brighton, UK, May 2019

https://sensible.eee.strath.ac.uk

Motivation

- Collecting and labeling data is often impractical, expensive or time-consuming
- Deep neural networks tend to overfit, given limited labeled data for training

➢ Can we mitigate the overfit effects of insufficient data for the classification task?



Problem Statement

$\boldsymbol{X} = \{x_1, x_2, \dots, x_N\}$ Observations/samples

$\boldsymbol{Y} = \{y_1, y_2, \dots, y_M, \dots, y_N\}, \in \{-1, 1\}$ Class labels

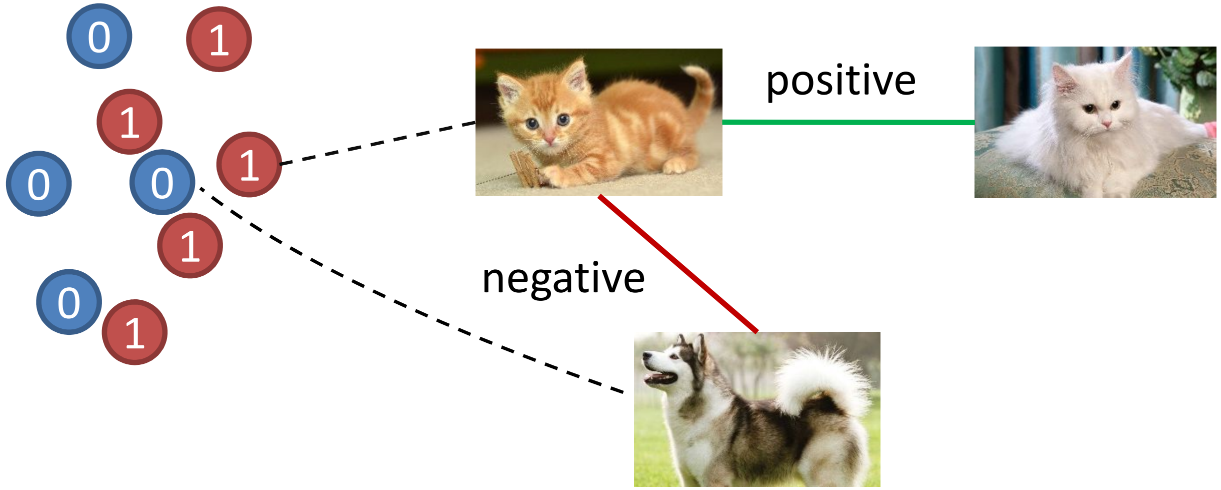
$\dot{\boldsymbol{Y}} = \{y_1, y_2, \dots, y_M\},$ $M \ll N$ Known labels (training set)

Classifier learning problem: Given a training set $\{x, y\}_{1,\dots,M}$ learn a function $\mathcal{F}(x)$ that maps input sample x to a label y

The main idea: Step 1

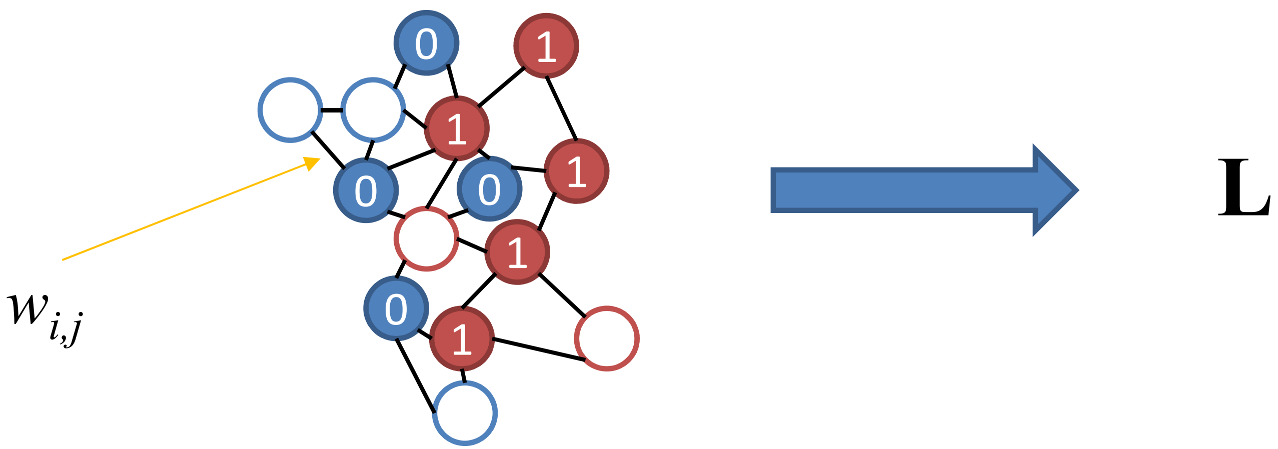
1. Use Convolutional Neural Network (CNN) to learn deep features

Example: Use CNN to extract features that promote small distance between Cat samples, and large distance between Cat and Dog samples



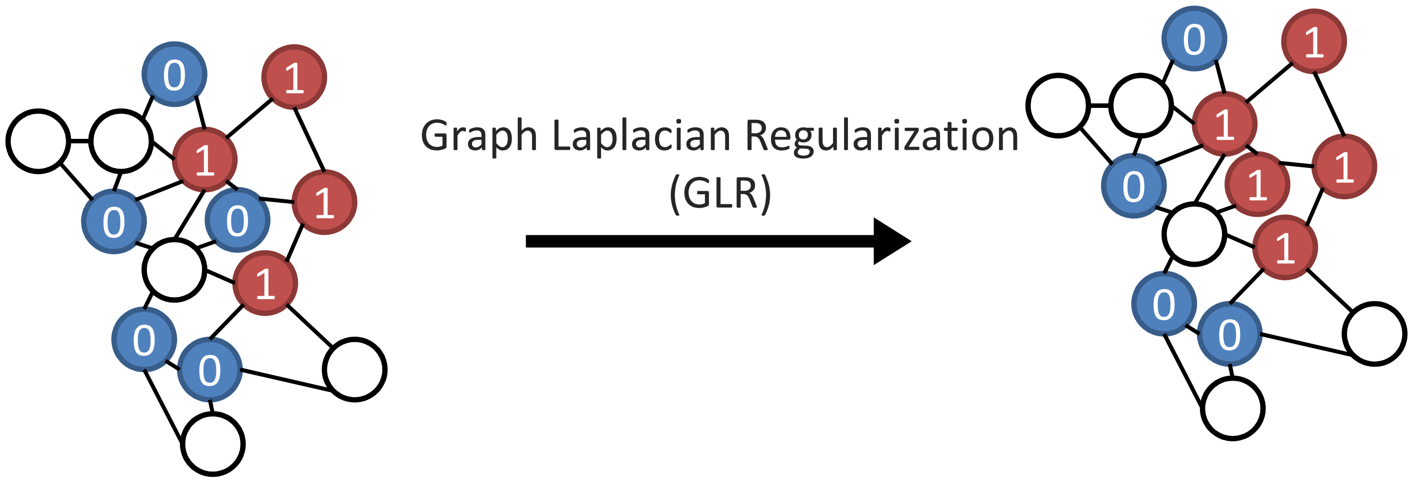
The main idea: Step 2

1. Use Convolutional Neural Network (CNN) to learn features
2. Use the learned *“deep”* features *to learn* the graph structure



The main idea: Step 3

1. Use Convolutional Neural Network (CNN) to learn features
2. Use the learned *"deep"* features *to learn* the graph
3. The graph is used to perform **graph Laplacian regularization**



Input Graph (After Graph Construction)

Smoothed Graph

Main steps

1. Use Convolutional Neural Network (CNN) to learn features
2. Use the learned *"deep"* features to learn the graph
3. The graph is used to perform graph Laplacian regularization
4. Update the CNN to improve feature learning via a weighted loss function that reflects the quality of learned underlying graph, **promoting** connections between the nodes with the same labels, and **penalizing** the connection of nodes with the opposite labels.

$$Loss = \sum_{a,p,n}^{M} \left[\alpha - \|F(x_a) - F(x_n)\|_2^2 \cdot \pi_{a,n} + \left\|F(x_a) - F\left(x_p\right)\right\|_2^2 \cdot \pi_{a,p}\right]_{relu}$$

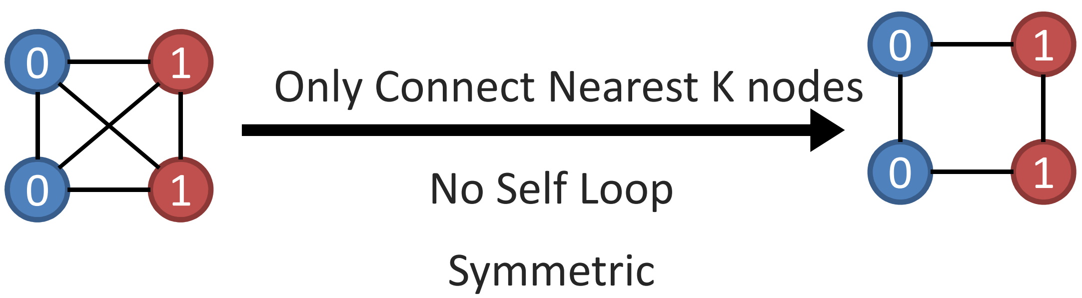
First step: Building a Graph

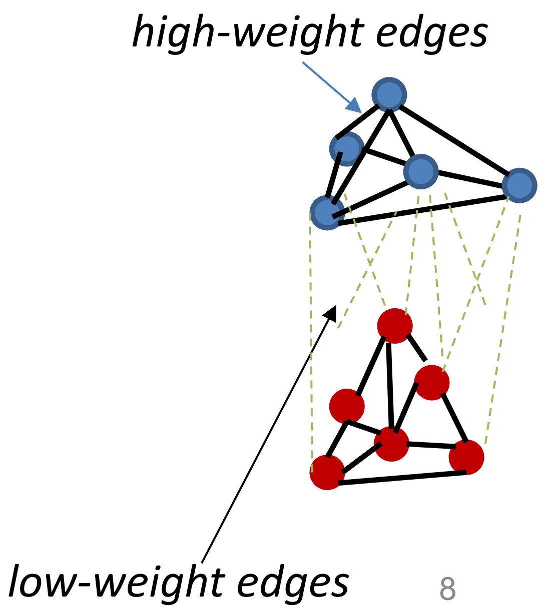
Construct a graph $\mathfrak{G} = (\boldsymbol{\mathcal{V}}, \mathbf{E}, \mathbf{W})$
Graph signal (**class labels**): $\mathbf{Y}$ with y_i corresponding to a Vertex i

How do we construct the graph (E, W) that truly reflects the signal statistics?

- Nodes with the same labels connected, and those with opposite labels disconnected
- Outliers penalized
- Sparse and connected graph

=> let's use a k-NN graph





Edge Loss Function

- **Idea**: Learn the model (i.e., the underlying graph) based on CNN 'deep features' as input, and then use a LOSS function that reflects how good the model is.
- Optimise the CNN weights (**C**) using *the new loss and update the features*
- Underlying graph defined by edges **E** (γ - KNN graph degree) & weights **W**
- How to define *the loss function*?

$$Loss_{\boldsymbol{E}} = \sum_{a,p,n}^{M} \Big[\overset{\text{margin}}{\alpha_{\boldsymbol{E}}} - \big\| \overbrace{\mathcal{F}(x_a) - \mathcal{F}(x_n)}^{\text{distance between nodes in the opposite class}} \big\|_2^2 + \big\| \overbrace{\mathcal{F}(x_a) - \mathcal{F}(x_p)}^{\text{distance between nodes in the same class}} \big\|_2^2 \Big]_{ReLu} \qquad (1)$$

$$y_a \neq y_n, y_a = y_p$$

Deep features at Vertex x_a

- *The loss function promotes small/large Euclidean distance between the nodes with the same/opposite labels, while keeping minimum a margin*

$$e_{i,j} = 1, if\ x_i \text{ in } \gamma\text{-neighbourhood of Node } i, \qquad (2)$$
$$e_{i,j} = 0, \text{ otherwise}$$

Controls sparsity to achieve a KNN-graph

Weight Loss Function

- Iterate between Eq.(1), (2) and backpropagation via ADAM optimiser
  => Optimised degree and edges γ and **E** + CNN weights **C**
- We still need to set the weights for our KNN graph

graph weights $w_{i,j} = exp(-\|\mathcal{F}_r(x_i) - \mathcal{F}_r(x_j)\|_2^2/(2\sigma^2))$ (3)

$$Loss_{\boldsymbol{W}} = \sum_{a,p,n}^{M} \left[\alpha_{\boldsymbol{W}} - \|\mathcal{F}_r(x_a) - \mathcal{F}_r(x_n)\|_2^2 \pi_{a,n} + \|\mathcal{F}_r(x_a) - \mathcal{F}_r(x_p)\|_2^2 \pi_{a,p}\right]_+ \quad (4)$$

$$\boldsymbol{\Pi} = \{\pi_{i,j}\} = \{\delta y_i * \delta y_j\}$$

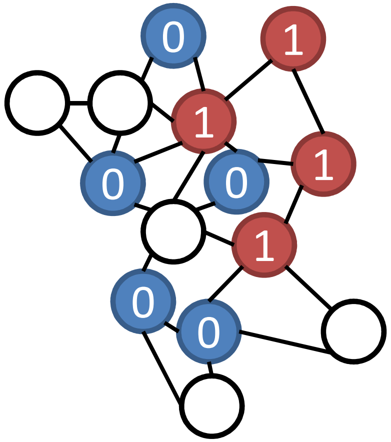
Amount of *attention* given to nodes with the same and opposite labels

$$\delta y_i = \begin{cases} 1, & if\ |\ddot{y}_i - \dot{y}_i| > \epsilon \\ 0, & if\ |\ddot{y}_i - \dot{y}_i| \leq \epsilon \end{cases}$$

$$\ddot{\boldsymbol{Y}} = \arg min_{\boldsymbol{U}}\{(\boldsymbol{U} - \dot{\boldsymbol{Y}}) + \mu \boldsymbol{U}\boldsymbol{L}\boldsymbol{U}^{\boldsymbol{T}}\} \quad (5)$$

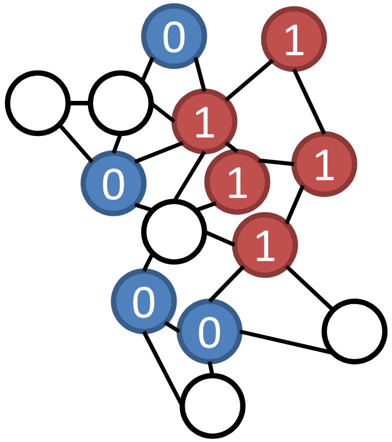
- Graph Laplacian regularization step that attempts to find the smoothest graph signal, $\ddot{\boldsymbol{Y}}$, for a given graph, that is close to the observed set of labels $\dot{\boldsymbol{Y}}$
- $Loss_{\boldsymbol{W}}$ fed back to the CNN for regularisation
- By calculating iteratively Eq. (3), (5), (4), batch-by-batch, and feeding back the loss to CNN to update $\mathcal{F}_r$, the loss of graph edge weight is minimised based on the edges with high attention value, while learning the best regularized deep metric function

Graph Regularisation



Graph Laplacian Regularisation (GLR)

①



Input Graph (After Graph Construction)

Smoothed Graph

$\boldsymbol{Y} = \{$ 1,1,1,1,1,-1,-1,-1,-1, 0,0,0,0,0 $\}$ $\widehat{\boldsymbol{Y}} = \{$ 0.8,0.9,0.7,0.9, -0.3,-0.8,-0.9,-0.2,-0.8, ?,?,?,?,? $\}$

For each edge, compute weighting factor π:

?: discard

δy_1=|1-0.8|=0.2< ϵ $\rightarrow$ δy_1=1

High Confident Node 1

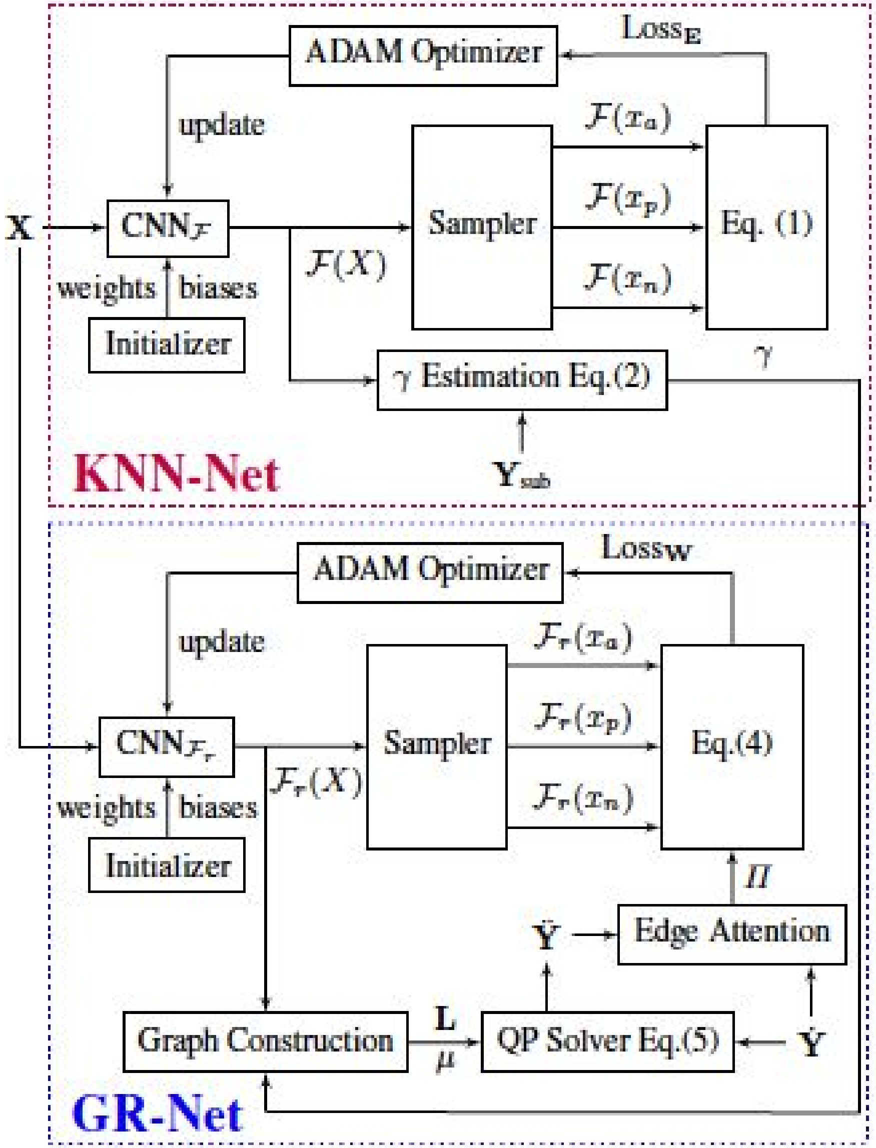
δy_5=|1- -0.3|=1.3> ϵ $\rightarrow$ δy_5=0

Low Confident Node 5

$\epsilon = 0.6$

Edge Weight Factor for Node 1&5: $\pi_{1,5}$=$\delta y_1 * \delta y_5$=1*0=0

Classification for insufficient data



$$Loss_{E} = \sum_{a,p,n}^{M} \left[\alpha_{E} - \|F(x_a) - F(x_n)\|_2^2 + \|F(x_a) - F(x_p)\|_2^2\right]_+ \quad y_a = y_n, y_a \neq y_p \qquad (1)$$

$e_{i,j} = 1, if\ xi\ in$ *γ-neighbourhood of Node* i,
$e_{i,j} = 0$, *otherwise* (2)

$$w_{i,j} = exp(-\|\mathcal{F}_r(x_i) - \mathcal{F}_r(x_j)\|_2^2/(2\sigma^2)) \qquad (3)$$

$$Loss_{W} = \sum_{a,p,n}^{M} \left[\alpha_{W} - \|\mathcal{F}_r(x_a) - \mathcal{F}_r(x_n)\|_2^2 \pi_{a,n} + \|\mathcal{F}_r(x_a) - \mathcal{F}_r(x_p)\|_2^2 \pi_{a,p}\right]_+ \quad \boldsymbol{\Pi} = \{\pi_{i,j}\} = \{\delta y_i * \delta y_j\}$$

(4)

$$\ddot{\boldsymbol{Y}} = \arg min_{\boldsymbol{U}} \{(\boldsymbol{U} - \dot{\boldsymbol{Y}}) + \mu \boldsymbol{U} \boldsymbol{L} \boldsymbol{U}^{\boldsymbol{T}}\} \qquad (5)$$

$$\delta y_i = \begin{cases} 1, & if\ |\ddot{y}_i - \dot{y}_i| > \epsilon \\ 0, & if\ |\ddot{y}_i - \dot{y}_i| \leq \epsilon \end{cases} \qquad (6)$$

Evaluation: Datasets

- Datasets from *Knowledge Extraction based on Evolutionary Learning* dataset (KEEL) (http://www.keel.es) KEEL-dataset Data set repository
  - **Phoneme**: classification of nasal (class 0) and oral sounds (class 1), with 5404 instances (frames) described by 5 phonemes of digitized speech (challenging)
  - **Spambase**: determining whether an email is spam (class 0) or not (class 1), with 4597 email messages summarized by 57 particular words or characters

- Support Vector Machines with radial basis function kernel (SVM-RBF)
- GSP-based classifier
- A classic CNN-based classifier
- Dynamic-graph CNN (DynGraph-CNN)[1]
- KNN-based deep metric classifier (DML-KNN)[2]

[1] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ICLR 2017*, vol. abs/1801.07829.
[2] E. Hoffer, N. Ailon, "Deep metric learning using triplet network," in *Intl. Workshop on Similarity-Based Pattern Recognition*. Springer, 2015.

Results

Classification error rate [%] for *Phoneme Dataset*

Training [%]	10	15	20	25	30
SVM-RBF	20.81	20.32	19.78	19.45	19.05
GSP	23.09	22.92	22.72	22.34	22.17
CNN	20.69	20.22	19.51	19.12	18.91
DynGraph-CNN	22.12	20.20	19.39	19.21	18.40
DML-KNN	20.37	20.37	19.31	19.18	18.12
Proposed	**19.86**	**19.37**	**18.93**	**18.78**	**17.89**

Classification error rate [%] for *Spambase Dataset*

Training [%]	10	15	20	25	30
SVM-RBF	10.04	9.30	9.00	8.61	8.41
GSP	20.22	20.10	19.68	19.13	18.72
CNN	9.72	9.18	8.75	8.65	8.26
DynGraph-CNN	11.84	10.71	9.52	9.38	9.09
DML-KNN	9.20	8.26	7.97	7.73	7.44
Proposed	**9.08**	**8.18**	**7.64**	**7.52**	**7.38**

Conclusion

- Integrating graph Laplacian regularization into a deep neural network to combat problem of insufficient training data
- Linking target independent regularization term and robust error function via semi-supervised graph learning
- Proven regularization effects compared with state-of-the-art approaches

In Preparation

- Numerical stability analysis for graph construction
- Tackle noisy training labels
- Evaluation on more types of data