Semi-Supervised Classification with Graph Convolutional Networks (GCN)

Published as a conference paper at ICLR 2017
DOI : arXiv:1609.02907

Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

Presented by:

Jerry Liao, MS in Data Engineering

María Victoria Liendro, MS in Data Engineering

Introduction

Problem: classifying nodes in a graph, where labels are limited

Traditional approach: graph Laplacian regularization term in the loss function

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}\,, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2 = f(X)^\top \Delta f(X)\,.$$

Limitations: assumes that connected nodes are likely to share the same label. This restricts modeling capacity, as graph edges may not mean node similarity, but could contain complex relationships

Contributions

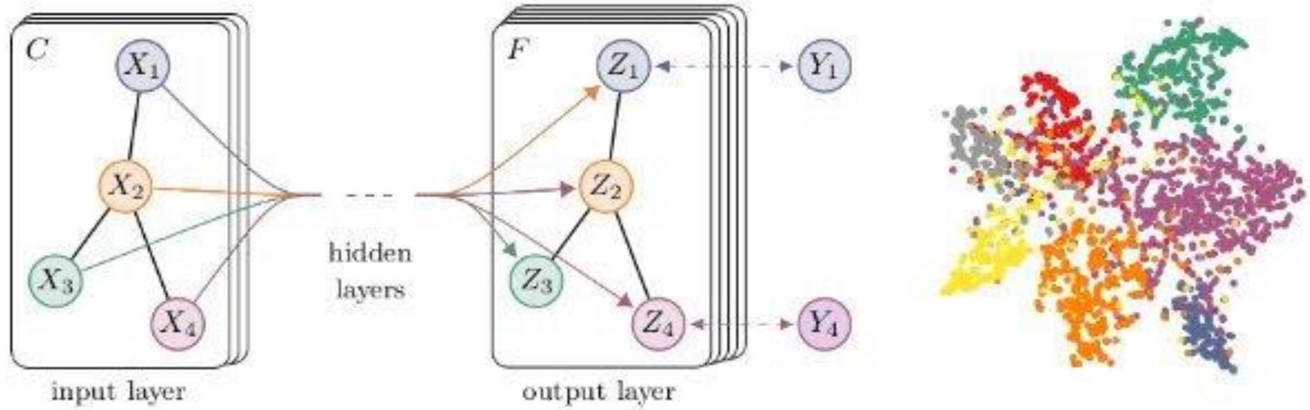
- Introduce a propagation rule for neural network models
- Comparison with traditional approaches and show that is faster and scalable

Model: Propagation rule

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

$\tilde{A} = A + I_N$: adjacency matrix of the graph with added self-connections
$\tilde{D}$: degree matrix: $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
H: matrix of features in the l-*th layer*. H(0) = X
W: layer-specific trainable weight matrix
σ: activation function, we will use Softmax for final result and ReLU for $H^{(1)}$

Model: Semi-supervised node classification



(a) Graph Convolutional Network

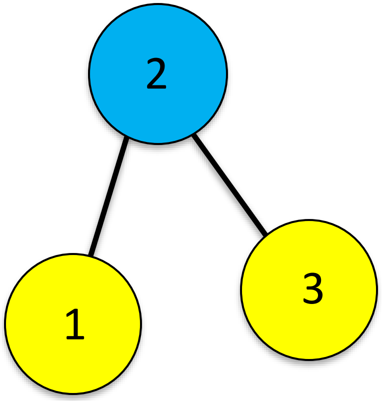
(b) Hidden layer activations

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right) W^{(1)}\right)$$

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

- Two-layer GCN on a graph with a symmetric adjacency matrix
- Calculated $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ in a pre-processing step
- Ws were trained using batch gradient descent using the full dataset for every training iteration
- W(0): input-to-hidden weight matrix for a hidden layer with H feature maps
- W(1): hidden-to-output weight matrix
- Evaluate the cross-entropy error over all labeled

Simple example



$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\tilde{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$\tilde{D} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

$$\tilde{D}^{-\frac{1}{2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\hat{A} = \begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{6}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{1}{3} & \frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{6}} & \frac{1}{2} \end{bmatrix}$$

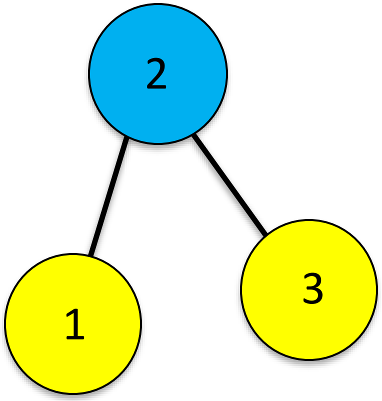
Assume:

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$W^{(0)} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$W^{(1)} = \begin{bmatrix} 0.8 & 0.2 \\ -0.3 & 0.7 \end{bmatrix}$$

Simple example: first layer



$$H^{(1)} = \mathrm{ReLU}(\hat{A}XW^{(0)})$$

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$W^{(0)} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

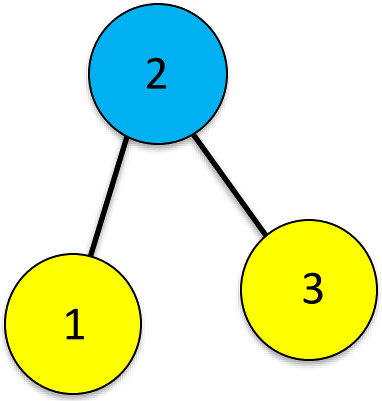
$$XW^{(0)} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H^{(1)} = \mathrm{ReLU}\left(\begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{6}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{1}{3} & \frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{6}} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} \right)$$

$$H^{(1)} = \begin{bmatrix} \mathrm{ReLU}(\frac{1}{2} - \frac{1}{\sqrt{6}}) & \mathrm{ReLU}(-\frac{1}{2} + \frac{1}{\sqrt{6}}) \\ \mathrm{ReLU}(\frac{2}{\sqrt{6}} - \frac{1}{3}) & \mathrm{ReLU}(-\frac{2}{\sqrt{6}} + \frac{1}{3}) \\ \mathrm{ReLU}(\frac{1}{2} - \frac{1}{\sqrt{6}}) & \mathrm{ReLU}(-\frac{1}{2} + \frac{1}{\sqrt{6}}) \end{bmatrix}$$

$$H^{(1)} = \begin{bmatrix} 0.0918 & 0 \\ 0.4832 & 0 \\ 0.0918 & 0 \end{bmatrix}$$

Simple example: Second layer



$$H^{(2)} = \text{softmax}(\hat{A}H^{(1)}W^{(1)})$$

$$H^{(1)} = \begin{bmatrix} 0.0918 & 0 \\ 0.4832 & 0 \\ 0.0918 & 0 \end{bmatrix} \qquad W^{(1)} = \begin{bmatrix} 0.8 & 0.2 \\ -0.3 & 0.7 \end{bmatrix}$$

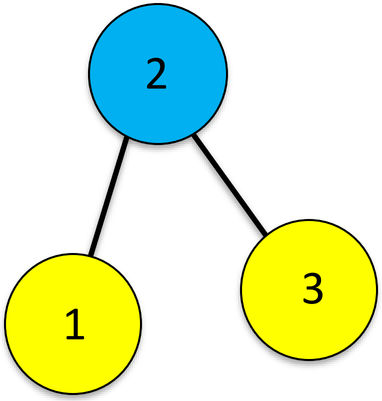
$$H^{(1)}W^{(1)} = \begin{bmatrix} (0.0918 \times 0.8) + (0 \times -0.3) & (0.0918 \times 0.2) + (0 \times 0.7) \\ (0.4832 \times 0.8) + (0 \times -0.3) & (0.4832 \times 0.2) + (0 \times 0.7) \\ (0.0918 \times 0.8) + (0 \times -0.3) & (0.0918 \times 0.2) + (0 \times 0.7) \end{bmatrix}$$

$$= \begin{bmatrix} 0.0734 & 0.0184 \\ 0.3865 & 0.0966 \\ 0.0734 & 0.0184 \end{bmatrix}$$

$$H^{(2)}_{\text{raw}} = \begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{6}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{1}{3} & \frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{6}} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 0.0734 & 0.0184 \\ 0.3865 & 0.0966 \\ 0.0734 & 0.0184 \end{bmatrix}$$

$$H^{(2)}_{\text{raw}} = \begin{bmatrix} 0.1945 & 0.0486 \\ 0.1888 & 0.0472 \\ 0.1945 & 0.0486 \end{bmatrix}$$

Simple example: Second layer



$$H^{(2)} = \text{softmax}(\hat{A}H^{(1)}W^{(1)}) \qquad p_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

$$H^{(2)}_{\text{raw}} = \begin{bmatrix} 0.1945 & 0.0486 \\ 0.1888 & 0.0472 \\ 0.1945 & 0.0486 \end{bmatrix}$$

$$e^{H^{(2)}_{\text{raw}}} = \begin{bmatrix} e^{0.1945} & e^{0.0486} \\ e^{0.1888} & e^{0.0472} \\ e^{0.1945} & e^{0.0486} \end{bmatrix}$$

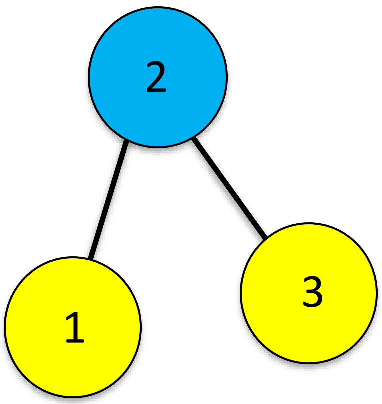
$$H^{(2)}_{1,1} = \frac{1.2148}{2.2646} \approx 0.5364, \quad H^{(2)}_{1,2} = \frac{1.0498}{2.2646} \approx 0.4636$$

$$H^{(2)}_{2,1} = \frac{1.2080}{2.2563} \approx 0.5353, \quad H^{(2)}_{2,2} = \frac{1.0483}{2.2563} \approx 0.4647$$

$$H^{(2)}_{3,1} = \frac{1.2148}{2.2646} \approx 0.5364, \quad H^{(2)}_{3,2} = \frac{1.0498}{2.2646} \approx 0.4636$$

$$H^{(2)} = \begin{bmatrix} 0.5364 & 0.4636 \\ 0.5353 & 0.4647 \\ 0.5364 & 0.4636 \end{bmatrix}$$

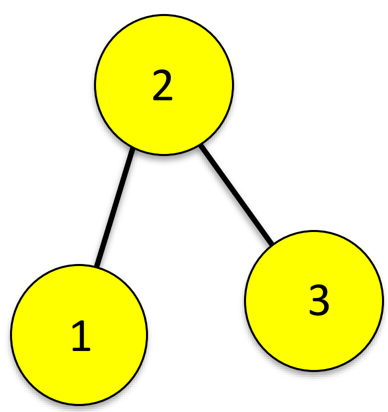
Simple example: Results



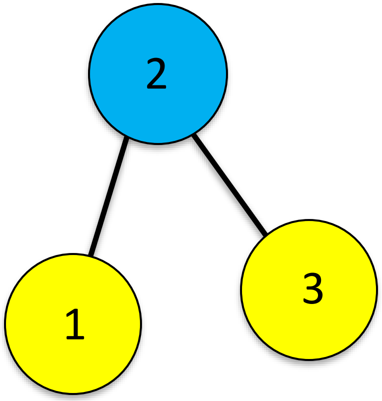
$$H^{(2)} = \begin{array}{cc} & \begin{array}{cc} C1 & \quad C2 \end{array} \\ \begin{array}{c} N1 \\ N2 \\ N3 \end{array} & \begin{bmatrix} 0.5364 & 0.4636 \\ 0.5353 & 0.4647 \\ 0.5364 & 0.4636 \end{bmatrix} \end{array}$$

In this model we consider 2 layer which means H2 is the results:

For Node1: classified to C1 ,with probability 0.5364
For Node2: classified to C1,,with probability 0.5353
For Node3: classified to C1,with probability 0.5364



Example: Loss function



$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

$$H^{(2)} = \begin{bmatrix} 0.5364 & 0.4636 \\ 0.5353 & 0.4647 \\ 0.5364 & 0.4636 \end{bmatrix}$$

$$\mathbf{Y}_{\text{true}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Y_{true} means that the true label of the node

$L_1 = -(1*\ln(0.5364)+0*\ln(0.4636)) \approx 0.6229$
$L_2 = -(0*\ln(0.5353)+1*\ln(0.4647)) \approx 0.7664$
$L_3 = -(1*\ln(0.5364)+0*\ln(0.4636)) \approx 0.6229$

$L_{total} = 0.6229 + 0.7664 + 0.6229 = 2.0122$

To minimize Loss, H2 should approach Y_{true}

Example: Update Weights

Learning rate

$$W^{(l)} \leftarrow W^{(l)} - \eta \cdot \frac{\partial L}{\partial W^{(l)}}$$

By back propagation rule:

$$\frac{\partial L}{\partial H^{(2)}_{i,c}} = H^{(2)}_{i,c} - Y_{i,c}$$

$$\frac{\partial L}{\partial H^{(1)}W^{(1)}} = \hat{A}^T \frac{\partial L}{\partial H^{(2)}}$$

Gradient of L with aspect to $H^{(1)}W^{(1)}$, denoted by $H^{(1)}W^{(1)}_{\text{grad}}$

$$\frac{\partial L}{\partial H^{(2)}} = \begin{bmatrix} 0.5364-1 & 0.4636-0 \\ 0.5353-0 & 0.4647-1 \\ 0.5364-1 & 0.4636-0 \end{bmatrix}$$

$$= \begin{bmatrix} -0.4636 & 0.4636 \\ 0.5353 & -0.5353 \\ -0.4636 & 0.4636 \end{bmatrix}$$

$$H^{(1)}W^{(1)}_{\text{grad}} = \begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{6}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{1}{3} & \frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{6}} & \frac{1}{2} \end{bmatrix}^T \begin{bmatrix} -0.4636 & 0.4636 \\ 0.5353 & -0.5353 \\ -0.4636 & 0.4636 \end{bmatrix}$$

$$H^{(1)}W^{(1)}_{\text{grad}} = \begin{bmatrix} -0.0694 & 0.0694 \\ 0.0721 & -0.0721 \\ -0.0694 & 0.0694 \end{bmatrix}$$

Example: Update Weights

$$W^{(l)} \leftarrow W^{(l)} - \eta \cdot \frac{\partial L}{\partial W^{(l)}}$$

Rewrite as

$$W^{(1)}_{\text{new}} = W^{(1)} - \eta \cdot W^{(1)}_{\text{grad}}$$

By back propagation rule:

$$\frac{\partial L}{\partial W^{(1)}} = H^{(1)^T} \cdot H^{(1)} W^{(1)}_{\text{grad}}$$

Gradient of L with aspect to W1 denoted by $W^{(1)}_{\text{grad}}$

$$H^{(1)} W^{(1)}_{\text{grad}} = \begin{bmatrix} -0.0694 & 0.0694 \\ 0.0721 & -0.0721 \\ -0.0694 & 0.0694 \end{bmatrix}$$

$$W^{(1)}_{\text{grad}} = \begin{bmatrix} 0.0918 & 0 \\ 0.4832 & 0 \\ 0.0918 & 0 \end{bmatrix}^T \begin{bmatrix} -0.0694 & 0.0694 \\ 0.0721 & -0.0721 \\ -0.0694 & 0.0694 \end{bmatrix}$$

$$= \begin{bmatrix} -0.0192 & 0.0192 \\ -0.0231 & 0.0231 \end{bmatrix}$$

Result:

$$W^{(1)}_{\text{new}} = \begin{bmatrix} 0.8 & 0.2 \\ -0.3 & 0.7 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0192 & 0.0192 \\ -0.0231 & 0.0231 \end{bmatrix}$$

$$W^{(1)}_{\text{new}} = \begin{bmatrix} 0.8019 & 0.1981 \\ -0.2977 & 0.6977 \end{bmatrix}$$

Repeat this until find the minimize!
Then do the same to $W^{(0)}$

Experiments

Set-Up:

- Validation set: 500 labeled data for hyperparameter optimization
- Test set: 1,000 labeled data
- 200 epochs
- Adam with learning rate of 0.01 and early stopping if validation loss does not decrease in 10 epochs

Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Results: Classification

- Mean accuracy of 100 runs with random initializations
- Time testing: Planetoid it used the implementation provided by the authors and used the same hardware as our model

Table 2: Summary of results in terms of classification accuracy (in percent).

Method	**Citeseer**	**Cora**	**Pubmed**	**NELL**
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Results: Propagation model

- Mean accuracy of 100 runs with random initializations

Table 3: Comparison of propagation models.

Description		Propagation model	Citeseer	Cora	Pubmed
Chebyshev filter (Eq. 5)	$K=3$	$\sum_{k=0}^{K} T_k(\tilde{L})X\Theta_k$	69.8	79.5	74.4
	$K=2$		69.6	81.2	73.8
1st-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X\Theta$	69.3	79.2	77.4
Renormalization trick (Eq. 8)		$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$	**70.3**	**81.5**	**79.0**
1st-order term only		$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta$	68.7	80.5	77.8
Multi-layer perceptron (Not graph structure)		$X\Theta$	46.5	55.1	71.4

Results: Training time

- Mean training time per epoch of 100 runs with random initializations

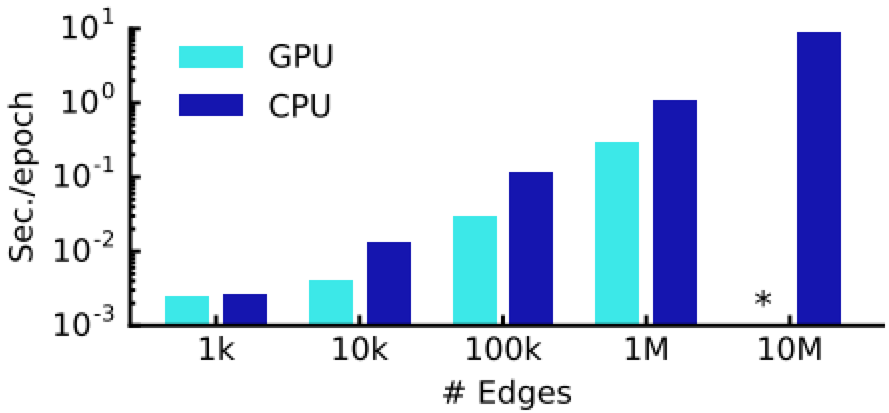


Figure 2: Wall-clock time per epoch for random graphs. (*) indicates out-of-memory error.

Discussion

- **Overcoming Previous Limitations:**
  - Graph-Laplacian regularization: assumes that edges encode similarity of nodes
  - Skip-gram: are based on a multi-step pipeline which is difficult to optimize
- **Feature propagation:** information from neighboring nodes in every layer improves classification performance, outperforming methods where only label information is aggregated
- **Propagation model:** improved efficiency and better predictive performance

Discussion: Limitations

• **Memory requirement**: it grows linearly with the size of the dataset. Large graphs that do not fit in GPU memory can still be train in CPU.

  o Proposition: mini-batch stochastic gradient descent.

• **Directed edges and edge features**: this model does not naturally support edge features and is limited to undirected graphs.

• **Limiting assumptions**: We assume locality and equal importance of self-connections vs edges to neighboring nodes.

  oProposition: introduce a trade-off parameter.

Thank you

Franck Fotso, MS in Electrical Engineering

Jerry Liao, MS in Data Engineering

María Victoria Liendro, MS in Data Engineering