GraphCMR and Graph convolutional network

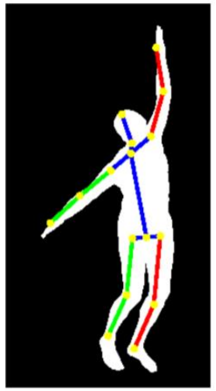
Convolutional Mesh Regression for
Single-Image Human Shape Reconstruction

GraphCMR: Related work

- optimization-based approaches (e.g. SMPLify)
  - Most reliable solution
  - Slow running time
  - Reliance on a good initialization, bad local minima

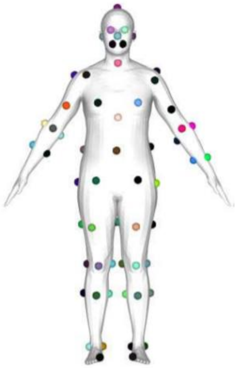
GraphCMR: Related work

- Learning-based approaches (regress pose and shape)
  - Input: raw image(HMR), semantic part segmentation, silhouettes and pose keypoints, surface landmarks
  - SMPL is not modeling hand pose or facial expressions
  - Constraint of parametric space
    - 3d rotation: challenging prediction target (periodicity, non-minimal representation, discontinuities)



Projected
silhouette & keypoints

Semantic part segmentation

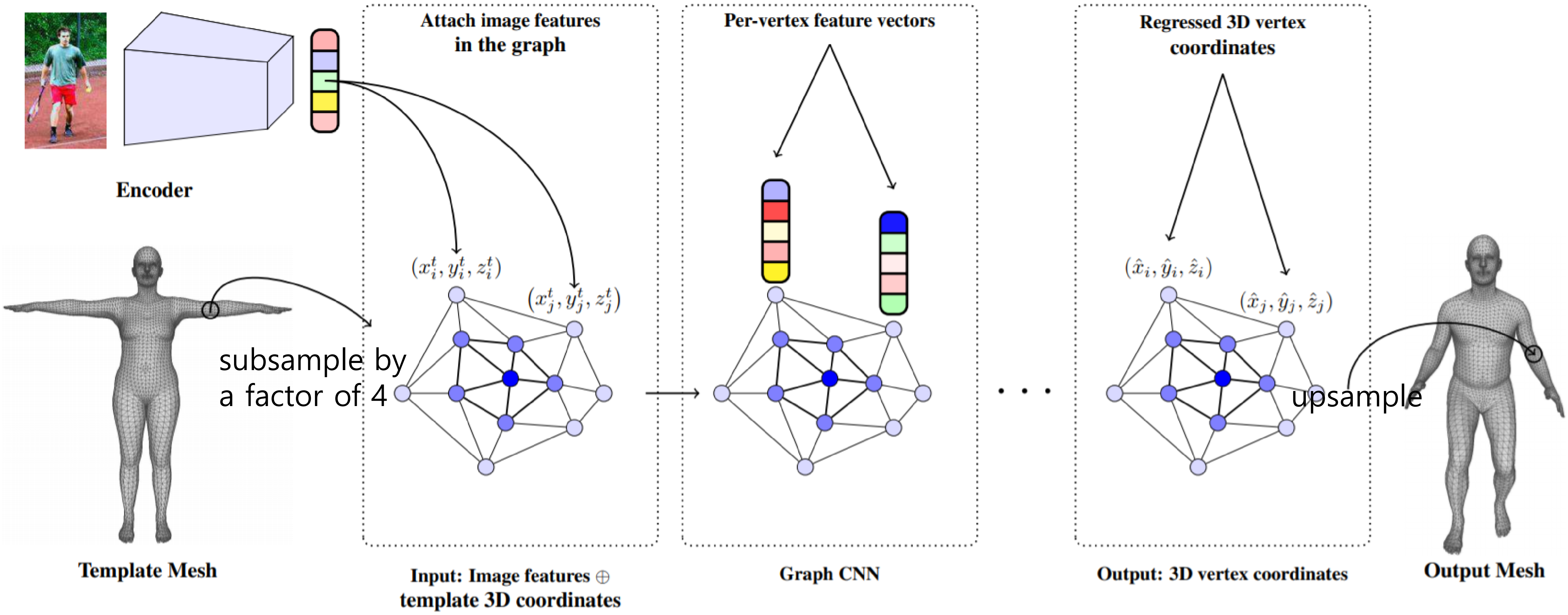


Surface landmarks

GraphCMR: Contributions

- Output: model parameters -> 3d location of mesh vertices
- Graph CNN (GCN)
- Outperform when using various input types
- State-of-the-art result

GraphCMR


Encoder
Attach image features
in the graph
(x_i^t, y_i^t, z_i^t)
(x_j^t, y_j^t, z_j^t)
Per-vertex feature vectors
Regressed 3D vertex
coordinates
$(\hat{x}_i, \hat{y}_i, \hat{z}_i)$
$(\hat{x}_j, \hat{y}_j, \hat{z}_j)$
subsample by
a factor of 4
upsample
Template Mesh
Input: Image features $\oplus$
template 3D coordinates
Graph CNN
Output: 3D vertex coordinates
Output Mesh

GraphCMR: Image-based CNN

- Resnet-50
- Ignore final FC layer, keeping 2048-D feature vector after average pooling

50-layer
7×7, 64, stride 2
3×3 max pool, stride 2
$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$
$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
average pool, 1000-d fc, softmax
3.8×10^9

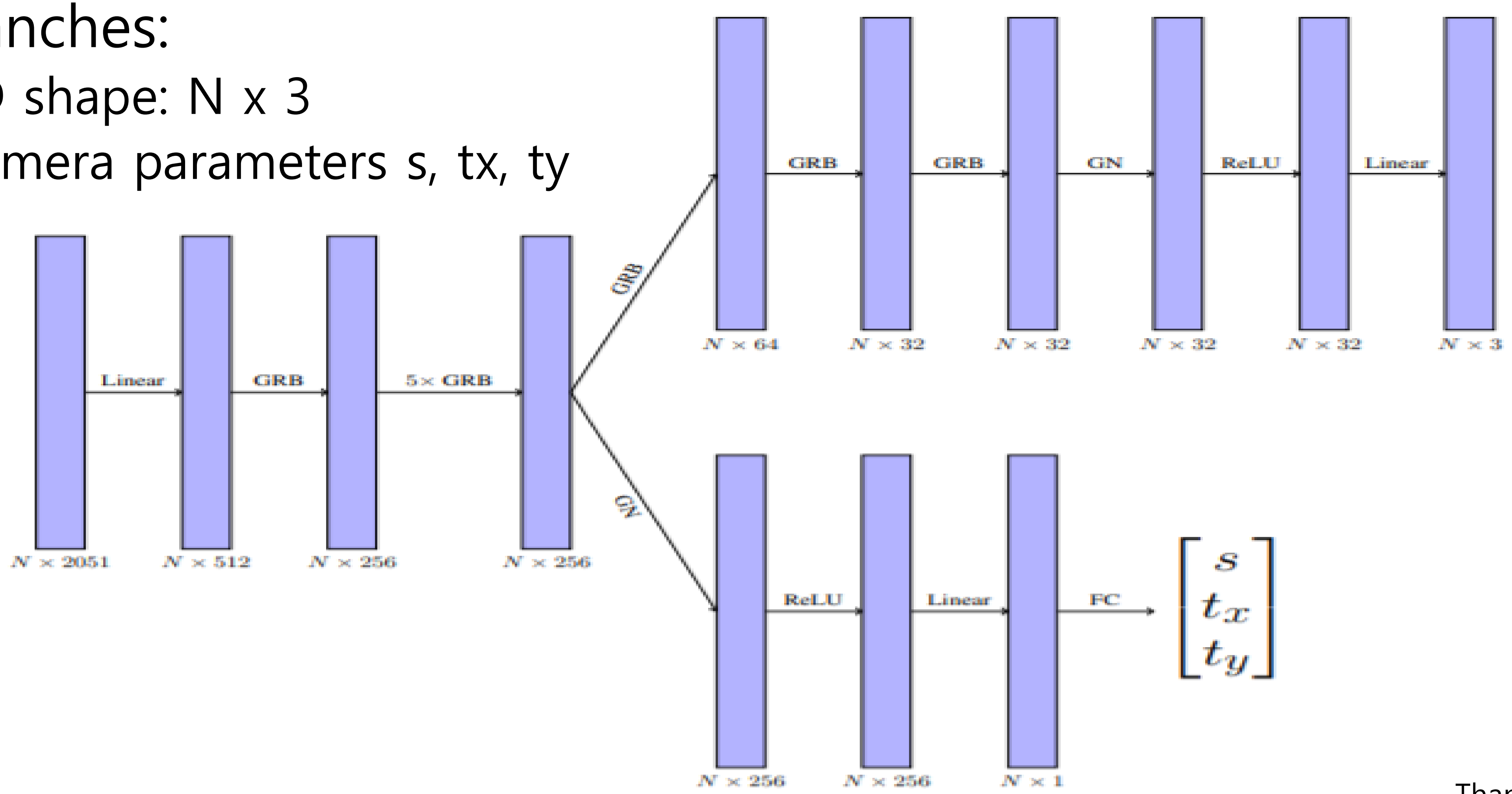
GraphCMR: Graph CNN

- $Y = \tilde{A}XW$
  - $X \in \mathbb{R}^{N \times k}$ is input feature matrix
  - $W \in \mathbb{R}^{k \times l}$ is weight matrix
  - $\tilde{A} \in \mathbb{R}^{N \times N}$ is the row normalized adjacency matrix of the graph

- performing per-vertex fully connected operations followed by a neighborhood averaging operation. -> smoothing

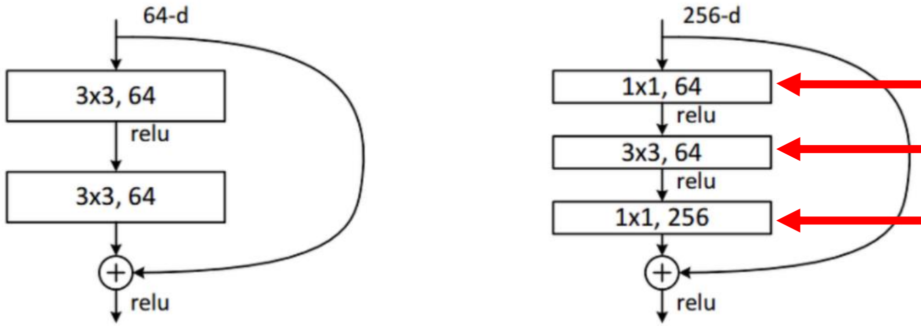
GraphCMR: Graph CNN

- 2 Branches:
  - 3D shape: N x 3
  - Camera parameters s, tx, ty

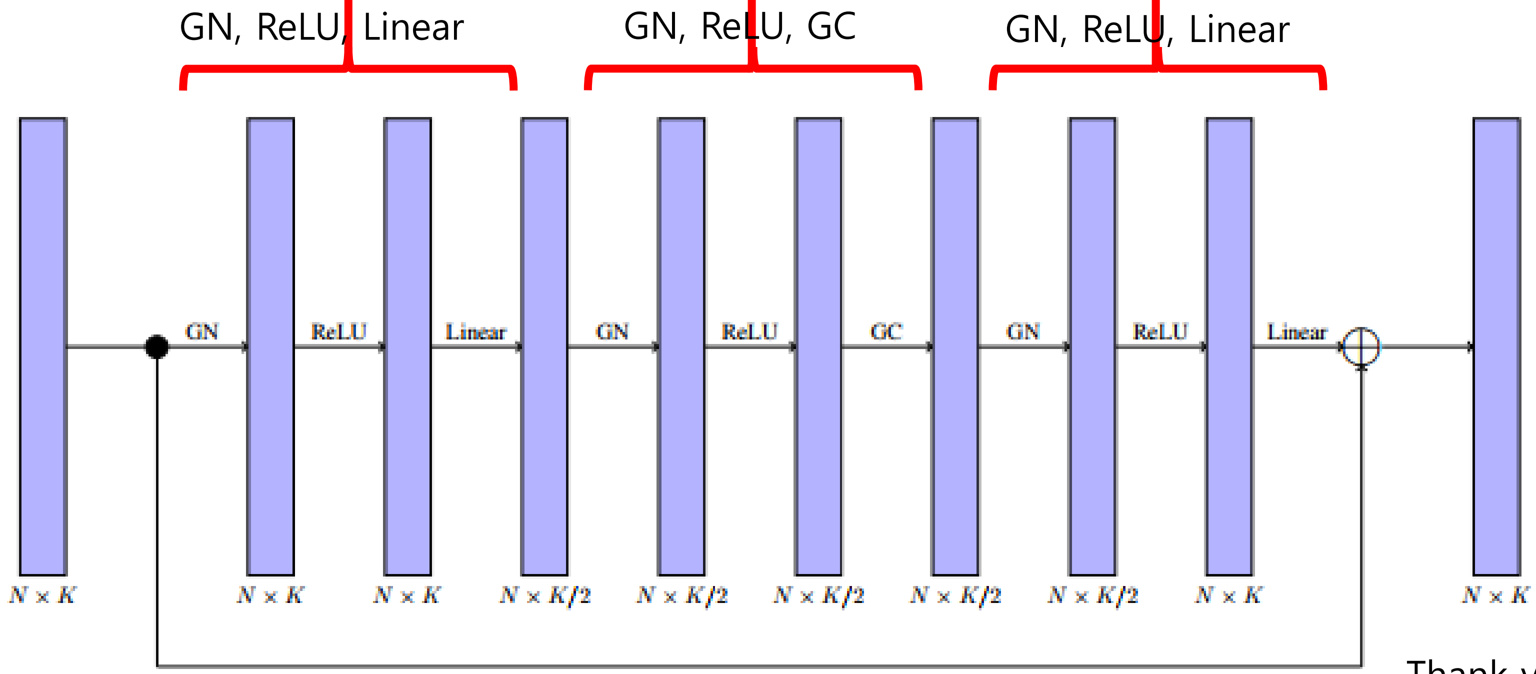


Thank you Thai!

GraphCMR: Graph CNN: Residual block



Bottleneck structure



Thank you Thai!

GraphCMR: Group Normalization

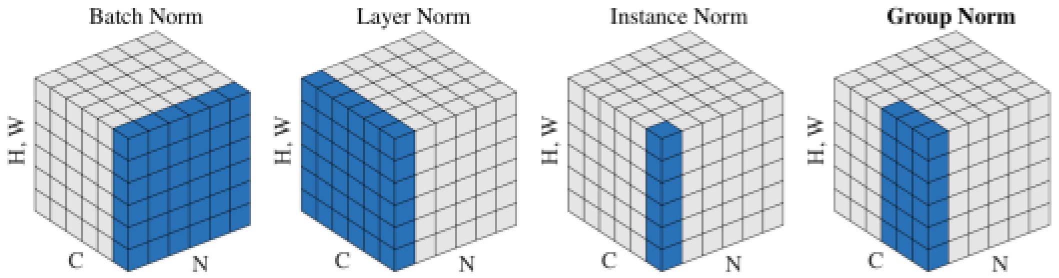


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

```
def GroupNorm(x, gamma, beta, G, eps=1e−5):
    # x: input features with shape [N,C,H,W]
    # gamma, beta: scale and offset, with shape [1,C,1,1]
    # G: number of groups for GN

    N, C, H, W = x.shape
    x = tf.reshape(x, [N, G, C // G, H, W])

    mean, var = tf.nn.moments(x, [2, 3, 4], keep_dims=True)
    x = (x − mean) / tf.sqrt(var + eps)

    x = tf.reshape(x, [N, C, H, W])

    return x ∗ gamma + beta
```

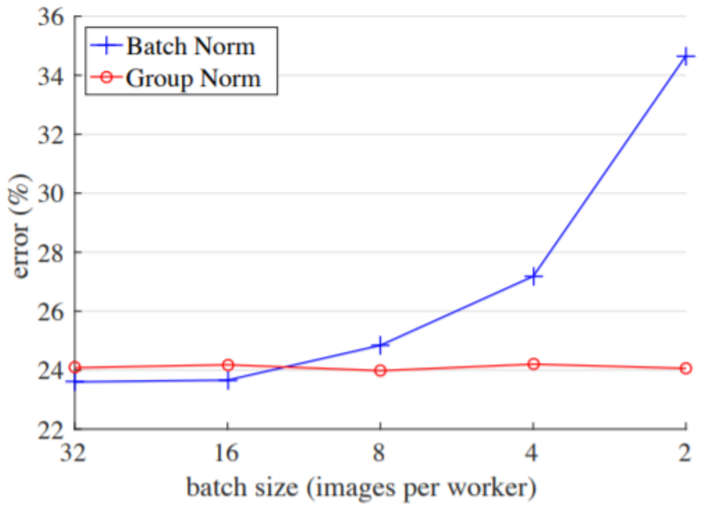


Figure 1. **ImageNet classification error *vs.* batch sizes**. This is a ResNet-50 model trained in the ImageNet training set using 8 workers (GPUs), evaluated in the validation set.

GraphCMR: Group Normalization

CLASS `torch.nn.GroupNorm(num_groups, num_channels, eps=1e-05, affine=True, device=None, dtype=None)` [SOURCE]

Applies Group Normalization over a mini-batch of inputs as described in the paper Group Normalization

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

The input channels are separated into `num_groups` groups, each containing `num_channels / num_groups` channels. The mean and standard-deviation are calculated separately over the each group. γ and β are learnable per-channel affine transform parameter vectors of size `num_channels` if `affine` is `True`. The standard-deviation is calculated via the biased estimator, equivalent to *torch.var(input, unbiased=False)*.

This layer uses statistics computed from input data in both training and evaluation modes.

Parameters

- **num_groups** (*int*) – number of groups to separate the channels into
- **num_channels** (*int*) – number of channels expected in input
- **eps** – a value added to the denominator for numerical stability. Default: 1e-5
- **affine** – a boolean value that when set to `True`, this module has learnable per-channel affine parameters initialized to ones (for weights) and zeros (for biases). Default: `True`.

Shape:

- Input: $(N, C, *)$ where $C = \text{num\_channels}$
- Output: $(N, C, *)$ (same shape as input)

GraphCMR: Training

- Per-vertex loss

$$\mathcal{L}_{shape} = \sum_{i=1}^{N} ||\hat{Y}_i - Y_i||_1.$$

- 2D-joints loss (employing same regressor that the SMPL model is using to recover joints from vertices)

$$\mathcal{L}_J = \sum_{i=1}^{M} ||\hat{X}_i - X_i||_1$$

GraphCMR: Empirical evaluation

- Datasets:
- Training: provide 3D groun d truth for training
  - Human3.6M [10]
  - UP-3D [18]
- Evaluation:
  - Human3.6M [10]
  - LSP dataset [13]

- Protocol:
  - Follow HMR [15]
  - P1:
    - trained on 5 subjects (S1, S5, S6, S7, S8)
    - Test on 2 (S9, S11)
  - P2:
    - Training: same with P1
    - Testing: S9, S11 but only use frontal camera (cam 3).
  - For comparison with previous methods.

Thank you Thai!

GraphCMR: result

Image Non-parametric Parametric

Figure 5: Examples of erroneous reconstructions. Typical failures can be attributed to challenging poses, severe self-occlusions, or interactions among multiple people.

Grap

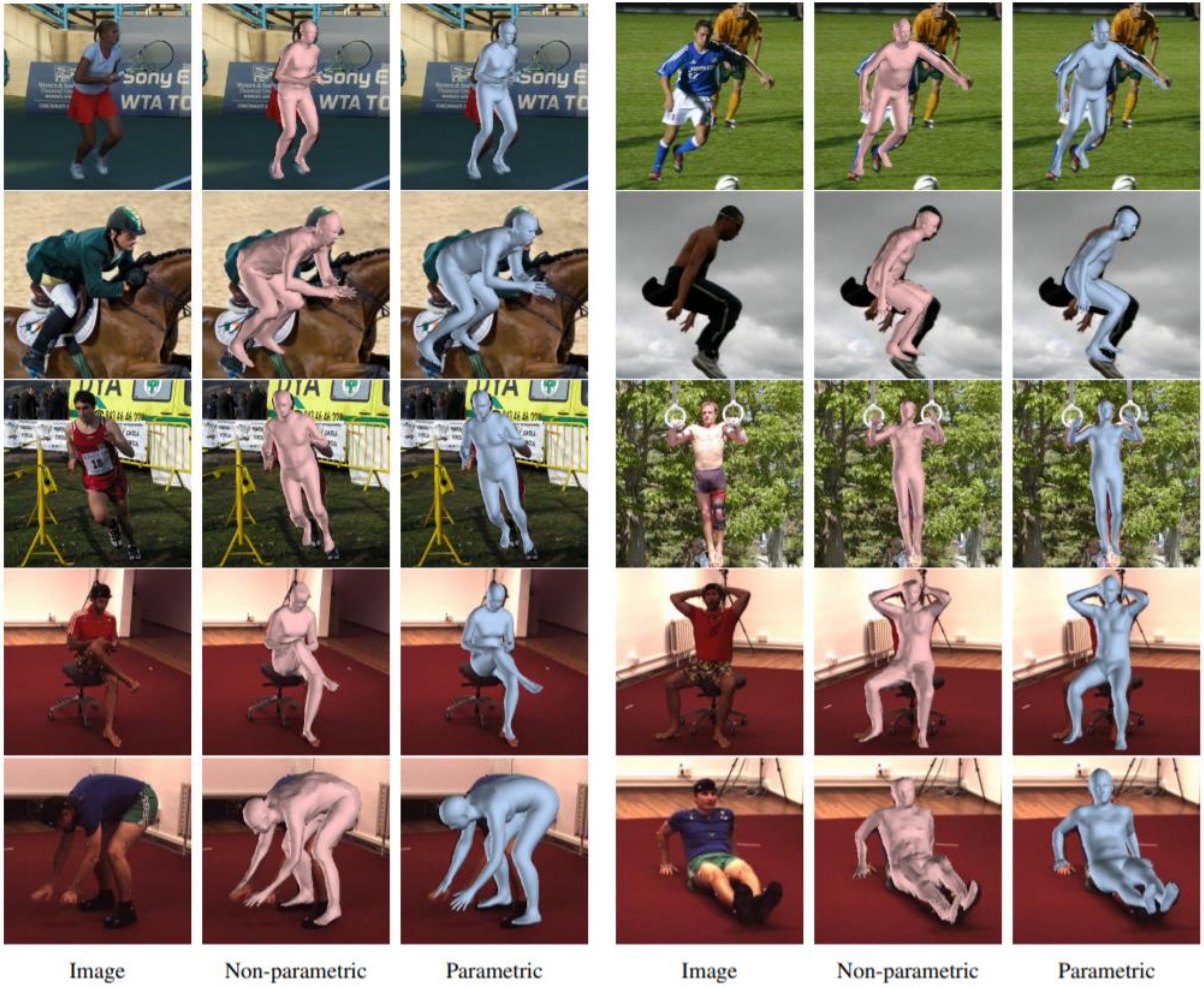
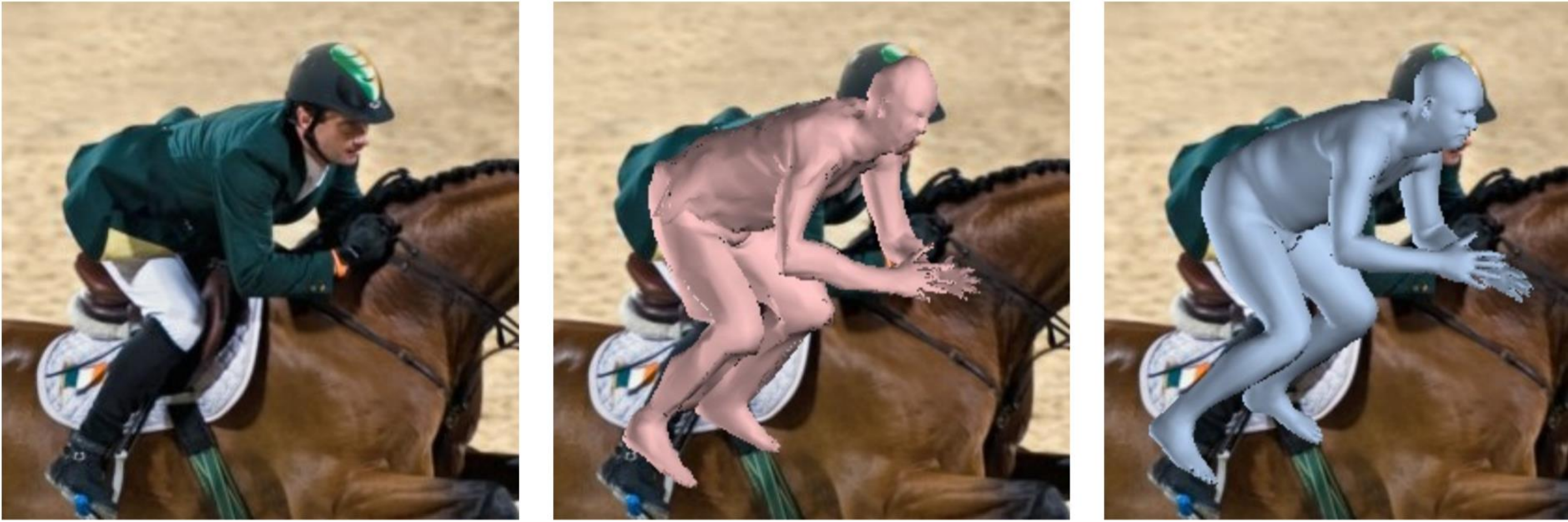


Image
Non-parametric
Parametric
Image
Non-parametric
Parametric

GraphCMR: result

GraphCMR: result



Graph convolutional network

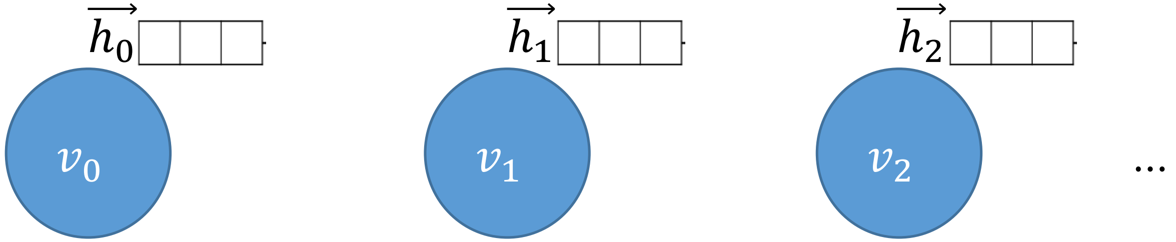
Semi-Supervised Classification with Graph Convolutional Networks

Wavelets on graphs via spectral graph theory

https://untitledtblog.tistory.com/152

Graph Convolutional Network: Basic

- Graph $G(V, A)$
- V: vertices
- A: adjacency matrix ($\mathrm{N} \times N$, N is the number of vertices)
- $\overrightarrow{h_i}$: embedded feature vector of i-th vertex
- H: feature matrix of vertices ($\mathrm{N} \times C$, C is the number of features)



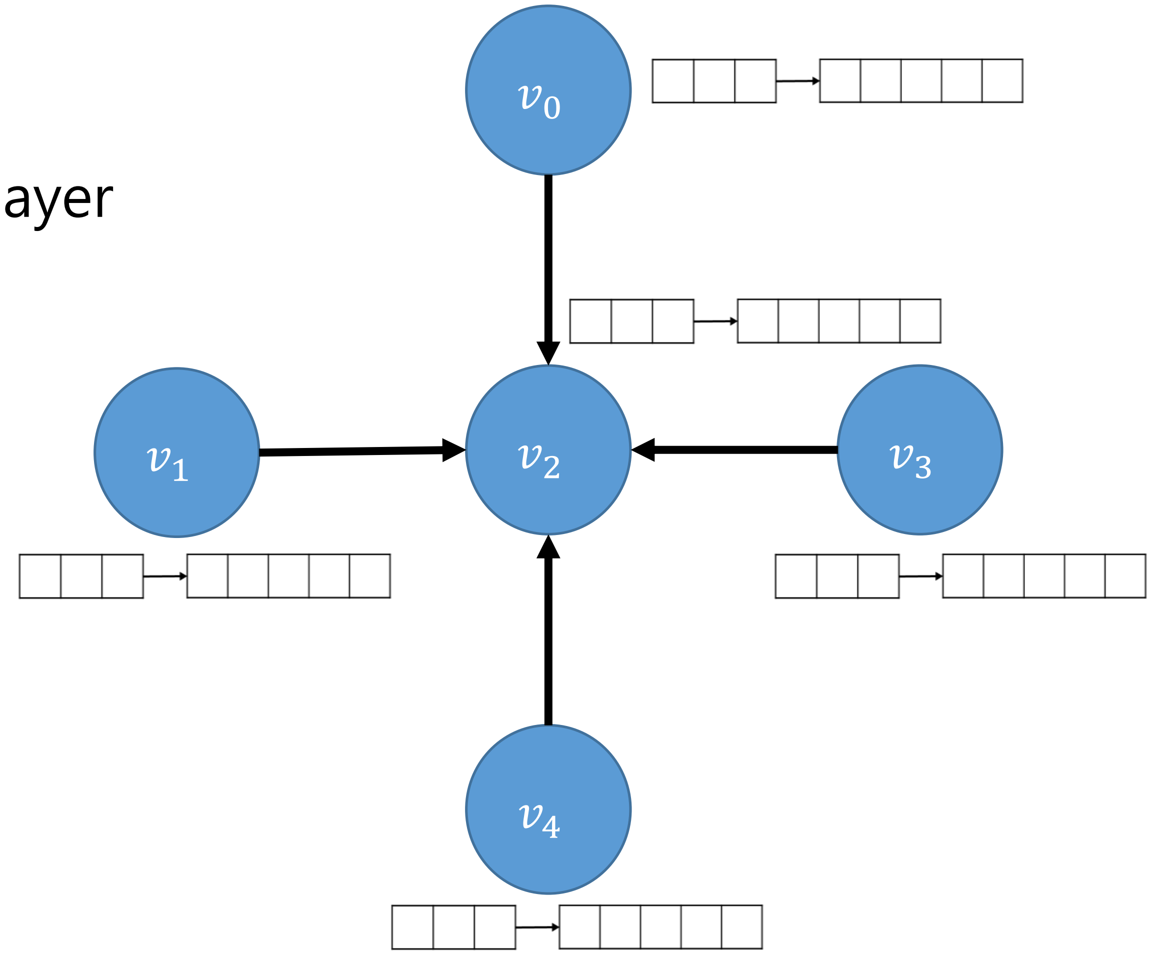
$$H = \begin{bmatrix} \overrightarrow{h_0} \\ \overrightarrow{h_1} \\ \overrightarrow{h_2} \\ \vdots \end{bmatrix}$$

Graph Convolutional Network: Basic

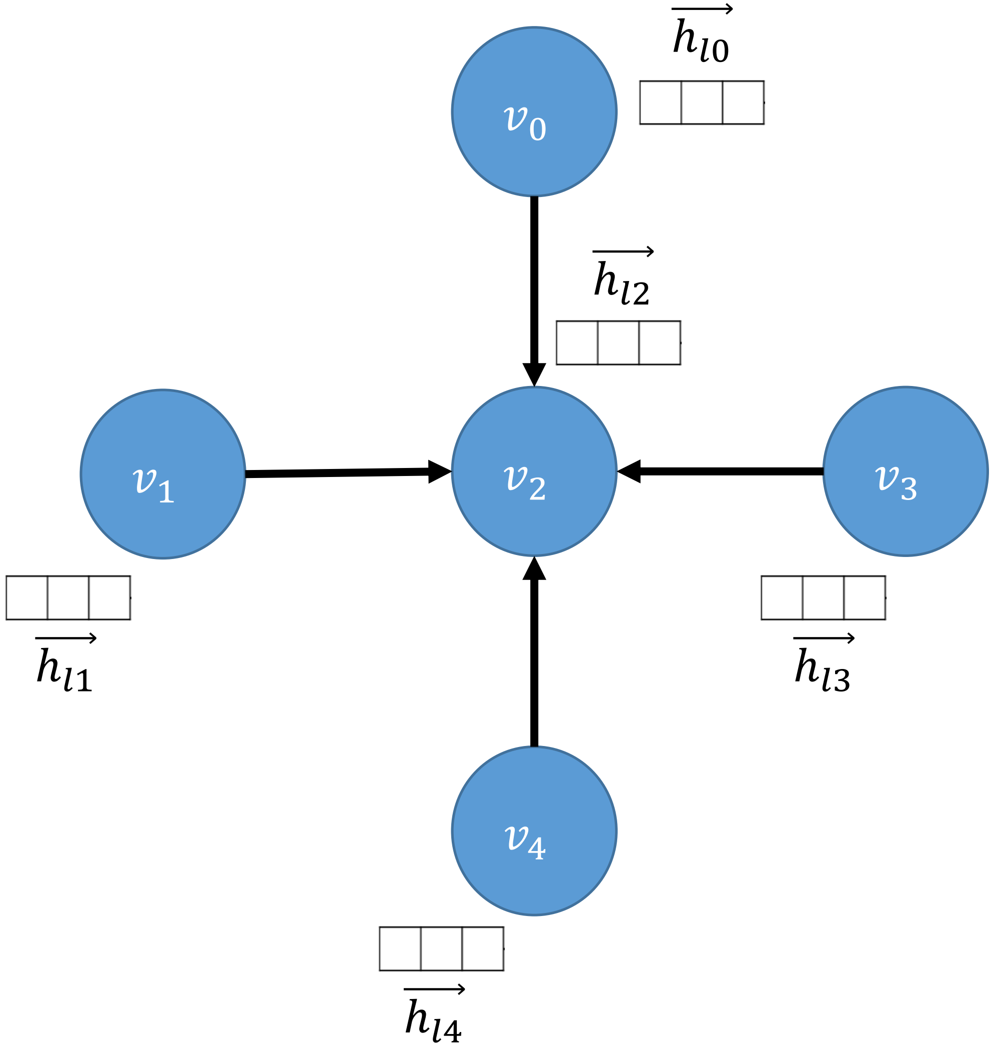
- $H_{l+1} = \sigma(AH_lW_l)$
  - A : adjacency matrix
  - H_l: latent node feature matrix in l-th layer
  - W_l: weight matrix in l-th layer

- $H_{l+1} = \sigma(\tilde{A}H_lW_l)$
  - $\tilde{A} = A + I$

- $H_{l+1} = \sigma(\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}H_lW_l)$
  - $\widetilde{D}$ is the degree matrix of $\tilde{A}$



Graph Convolutional Network: Basic



Propagation rules

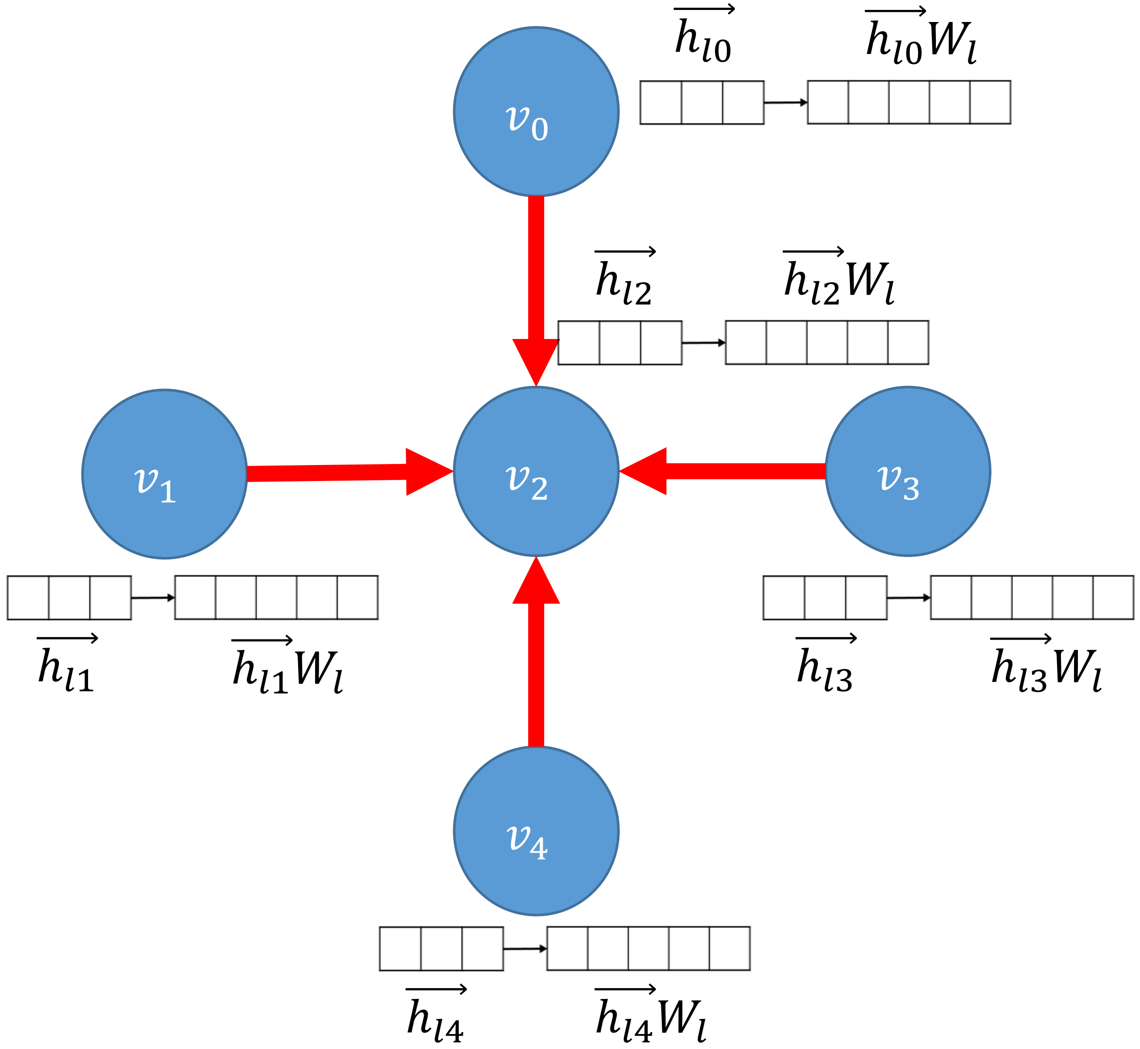
$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

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

$$\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}, \ \widetilde{D} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$H_l = \begin{bmatrix} \overrightarrow{h_{l0}} \\ \overrightarrow{h_{l1}} \\ \overrightarrow{h_{l2}} \\ \overrightarrow{h_{l3}} \\ \overrightarrow{h_{l4}} \end{bmatrix} \quad W_l$$

Graph Convolutional Network: Basic



Propagation rules

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

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

$$\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}, \quad \widetilde{D} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$H_l W_l = \begin{bmatrix} \overrightarrow{h_{l0}W_l} \\ \overrightarrow{h_{l1}}W_l \\ \overrightarrow{h_{l2}}W_l \\ \overrightarrow{h_{l3}}W_l \\ \overrightarrow{h_{l4}}W_l \end{bmatrix}$$

Spectral Graph theory

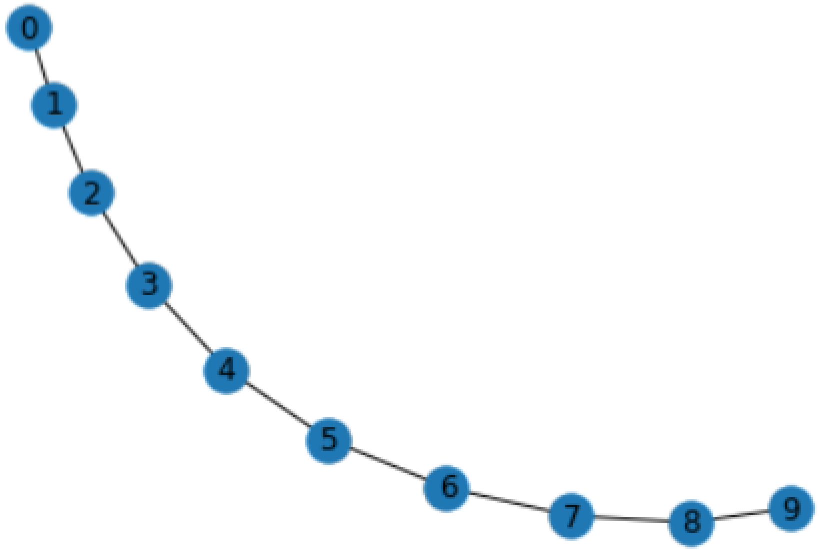
- Weighted undirected graph $G = \{E, V, w\}$
  - Set of vertices V
  - Set of edges E
  - Weighted function w: $\mathrm{E} \rightarrow \mathbb{R}^{+}$
  - Adjacency matrix A
  - Entry in adjacency matrix $a_{m,n} = \begin{cases} w(e_{m,n}),\ if\ e_{m,n} \in E \\ 0,\ otherwise \end{cases}$
  - Degree matrix D
  - Laplacian $\mathcal{L} = D - A$

Spectral Graph theory

- The complex exponentials $e^{i\omega x}$ defining the Fourier transform are eigenfunctions of the Laplacian operator (2nd differential)

- $\mathcal{L}X = X\Lambda$
- $\mathcal{L}\chi_l = \lambda_l \chi_l$ for $l = 0, \ldots, N-1$
  - λ_l: l-th eigenvalue
  - χ_l: l-th eigenvector

Spectral Graph theory



graph

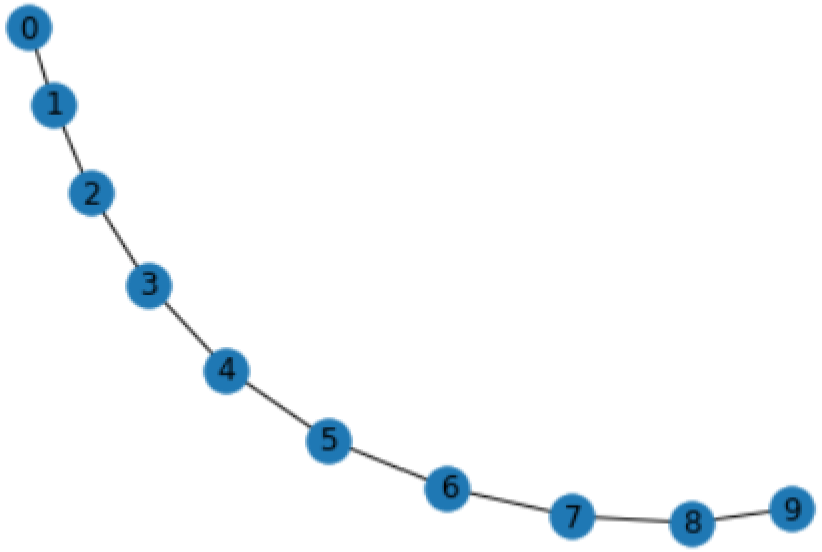
```
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
```

Adjacency matrix

```
[[ 1. -1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [-1.  2. -1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. -1.  2. -1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. -1.  2. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. -1.  2. -1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -1.  2. -1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -1.  2. -1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -1.  2. -1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. -1.  2. -1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. -1.  1.]]
```

Laplacian matrix

Spectral Graph theory

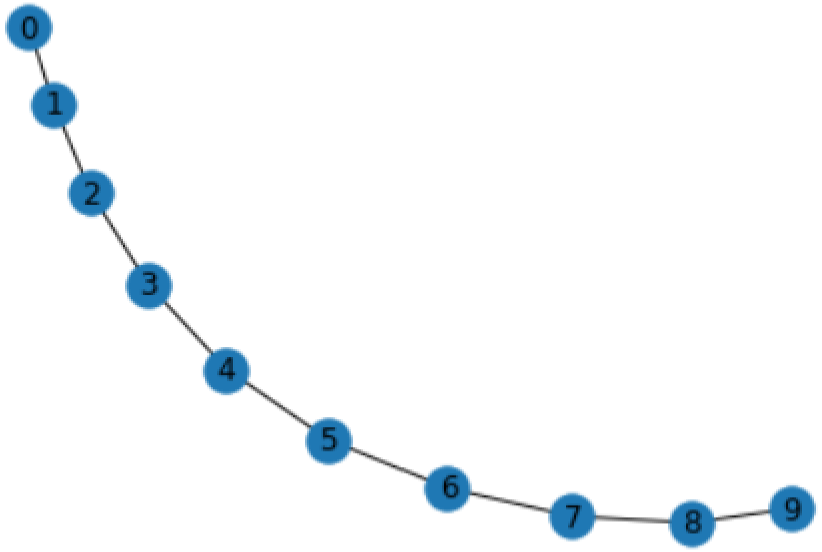


graph

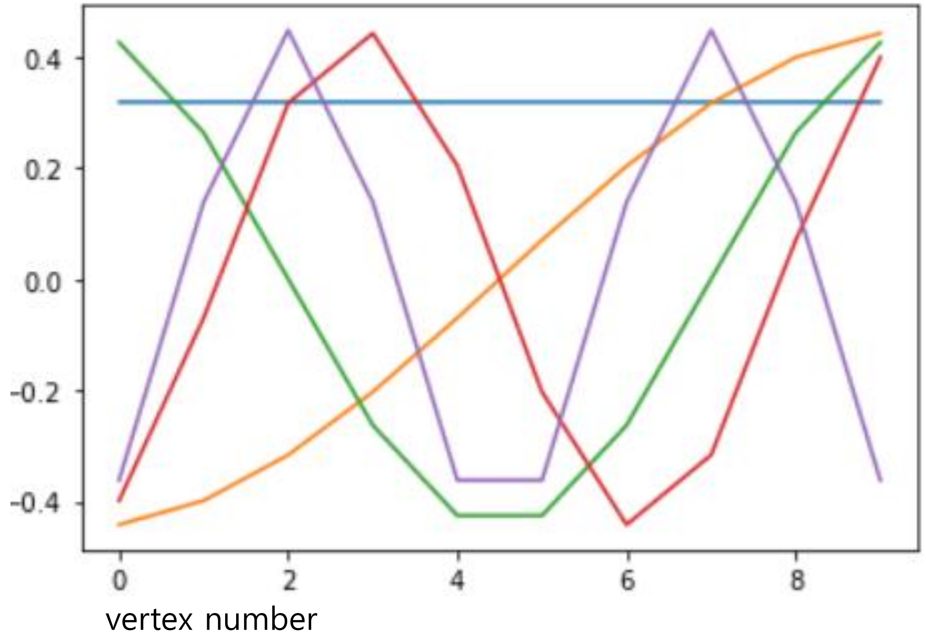
```
[-0.     0.098  0.382  0.824  1.382  2.     2.618  3.176  3.618  3.902]

[[ 0.316 -0.442  0.425 -0.398 -0.362 -0.316 -0.263 -0.203  0.138 -0.07 ]
 [ 0.316 -0.398  0.263 -0.07   0.138  0.316  0.425  0.442 -0.362  0.203]
 [ 0.316 -0.316 -0.     0.316  0.447  0.316 -0.    -0.316  0.447 -0.316]
 [ 0.316 -0.203 -0.263  0.442  0.138 -0.316 -0.425 -0.07  -0.362  0.398]
 [ 0.316 -0.07  -0.425  0.203 -0.362 -0.316  0.263  0.398  0.138 -0.442]
 [ 0.316  0.07  -0.425 -0.203 -0.362  0.316  0.263 -0.398  0.138  0.442]
 [ 0.316  0.203 -0.263 -0.442  0.138  0.316 -0.425  0.07  -0.362 -0.398]
 [ 0.316  0.316  0.    -0.316  0.447 -0.316  0.     0.316  0.447  0.316]
 [ 0.316  0.398  0.263  0.07   0.138 -0.316  0.425 -0.442 -0.362 -0.203]
 [ 0.316  0.442  0.425  0.398 -0.362  0.316 -0.263  0.203  0.138  0.07 ]]
```

Eigenvalue and eigenvector of
Laplacian matrix

Spectral Graph theory

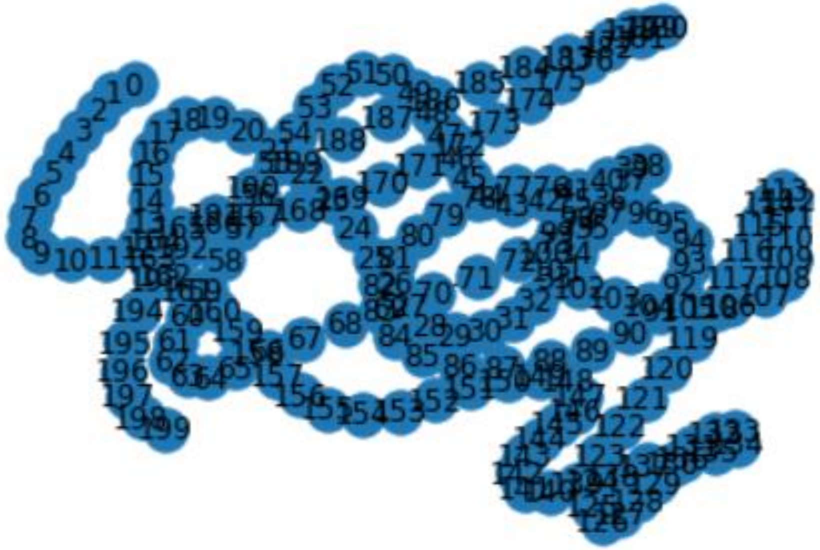


graph

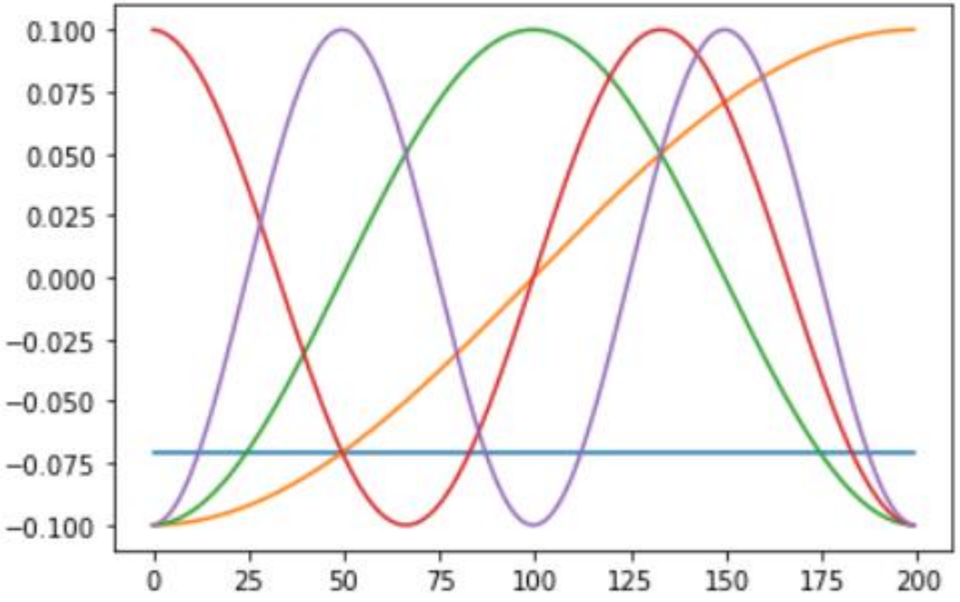


Eigenvector of Laplacian matrix

Spectral Graph theory



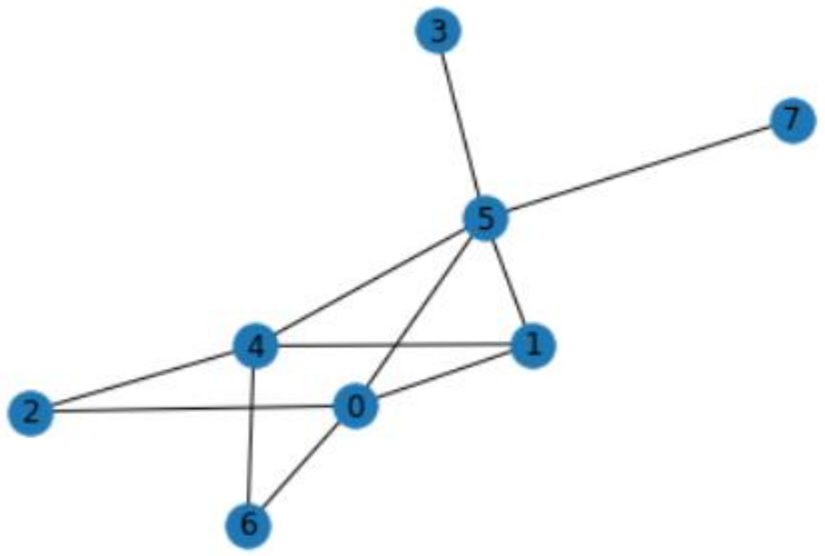
graph



vertex number

Eigenvector of Laplacian matrix

Spectral Graph theory



graph

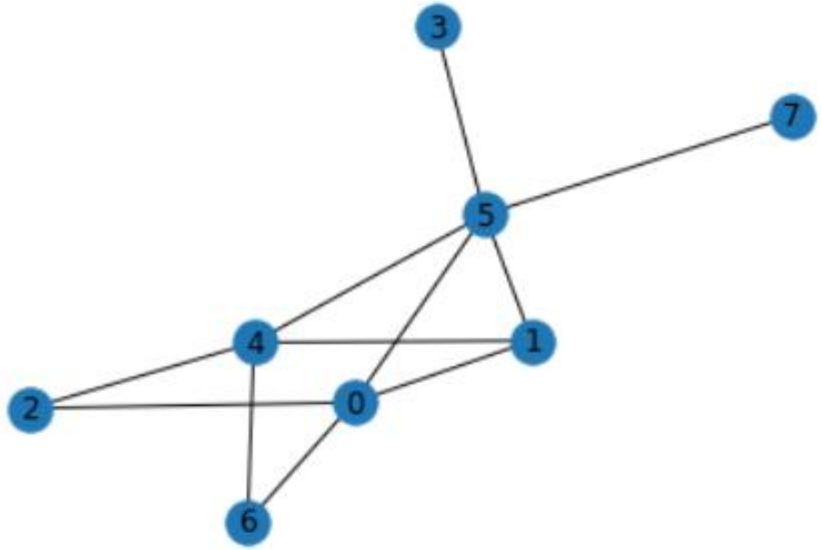
```
[[0 1 1 0 0 1 1 0]
 [1 0 0 0 1 1 0 0]
 [1 0 0 0 1 0 0 0]
 [0 0 0 0 0 1 0 0]
 [0 1 1 0 0 1 1 0]
 [1 1 0 1 1 0 0 1]
 [1 0 0 0 1 0 0 0]
 [0 0 0 0 0 1 0 0]]
```

Adjacency matrix

```
[[ 4 -1 -1  0  0 -1 -1  0]
 [-1  3  0  0 -1 -1  0  0]
 [-1  0  2  0 -1  0  0  0]
 [ 0  0  0  1  0 -1  0  0]
 [ 0 -1 -1  0  4 -1 -1  0]
 [-1 -1  0 -1 -1  5  0 -1]
 [-1  0  0  0 -1  0  2  0]
 [ 0  0  0  0  0 -1  0  1]]
```

Laplacian matrix

Spectral Graph theory

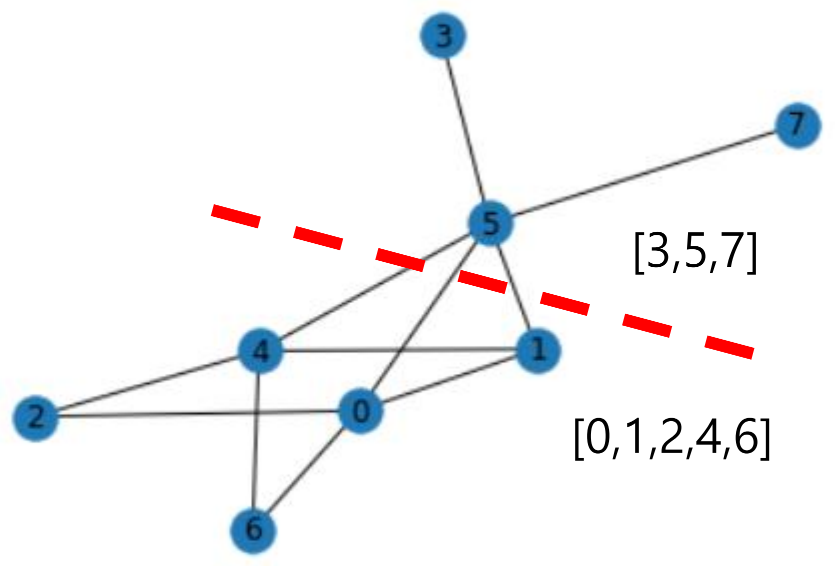


graph

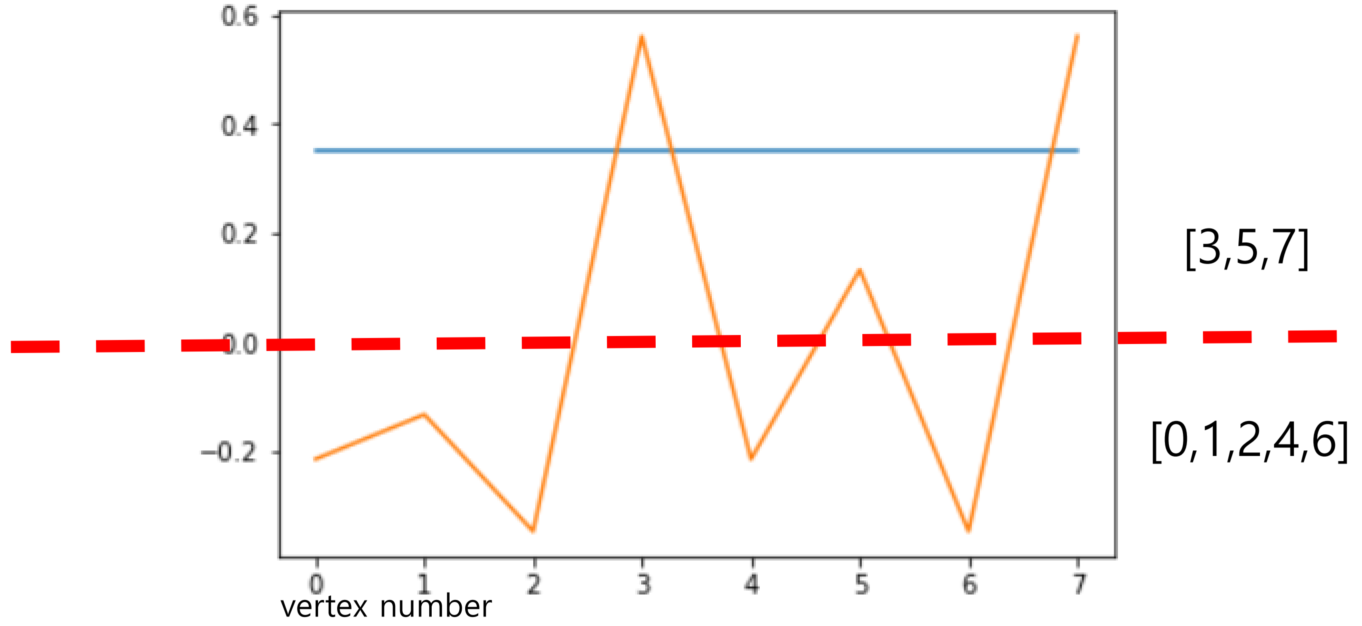
```
[0.    0.764 1.    2.    2.438 4.    5.236 6.562]

[[ 0.354 -0.215 -0.     0.     0.09   0.707  0.347 -0.447]
 [ 0.354 -0.133  0.    -0.     0.734  0.    -0.562  0.055]
 [ 0.354 -0.347 -0.     0.707 -0.412 -0.    -0.215  0.196]
 [ 0.354  0.562 -0.707 -0.    -0.161 -0.    -0.133 -0.126]
 [ 0.354 -0.215 -0.     0.     0.09  -0.707  0.347 -0.447]
 [ 0.354  0.133  0.    -0.     0.231 -0.     0.562  0.699]
 [ 0.354 -0.347 -0.    -0.707 -0.412  0.    -0.215  0.196]
 [ 0.354  0.562  0.707 -0.    -0.161 -0.    -0.133 -0.126]]
```

Eigenvalue and eigenvector of
Laplacian matrix

Spectral Graph theory



graph



Eigenvector of Laplacian matrix

Spectral Graph theory

- Fourier transform
  - $\hat{f}(\omega) = \int_{-\infty}^{\infty} (e^{2\pi i x \omega})^* f(x)\, dx = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \omega}\, dx$

- Graph Fourier transform
  - For $f\colon V \rightarrow \mathbb{R}$

$$\hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{n=1}^{N} \chi_\ell^*(n) f(n)$$

Spectral Graph theory

- Inverse Fourier transform

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\, e^{2\pi i x \xi}\, d\xi,$$

- Inverse Graph Fourier transform

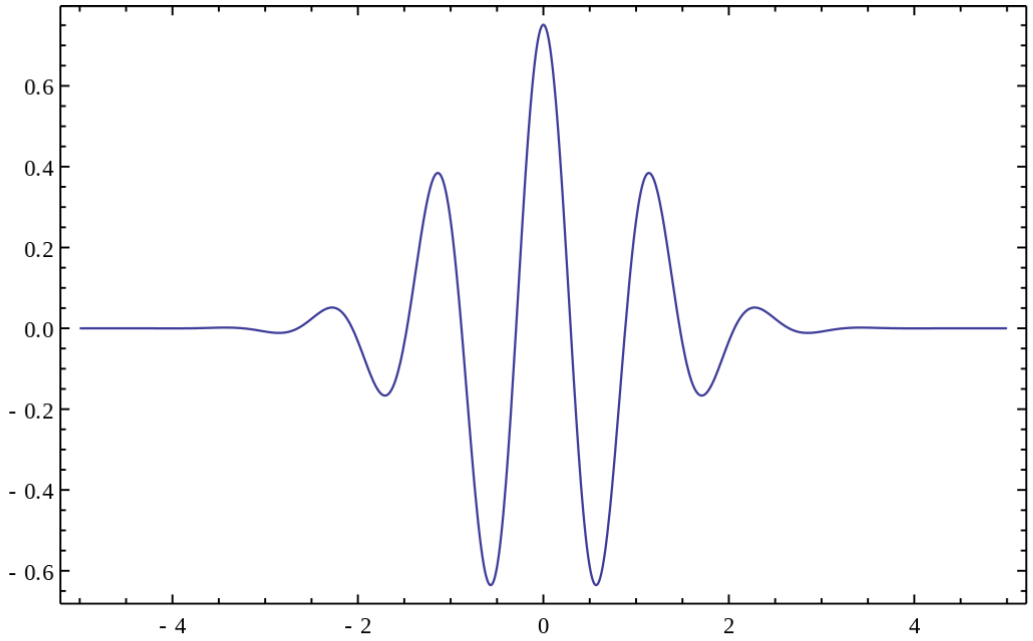
$$f(n) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(n)$$

Classical Wavelet Transform

$$\psi_{s,a}(x) = \frac{1}{s}\psi\left(\frac{x-a}{s}\right)$$

$$\int_0^\infty \frac{|\hat{\psi}(\omega)|^2}{\omega} d\omega = C_\psi < \infty$$

$$\hat{\psi}(0) = \int \psi(x) dx = 0.$$



$$W_f(s,a) = \int_{-\infty}^{\infty} \frac{1}{s}\psi^*\left(\frac{x-a}{s}\right) f(x) dx$$

Classical Wavelet Transform

- For a given function $\psi(x)$ defined on the vertices of a weighted graph, it is not obvious how to define $\psi(sx)$, as if x is a vertex of the graph there is no interpretation of sx for a real scalar s.

Classical Wavelet Transform

$$\psi_{s,a}(x) = \frac{1}{s}\psi\left(\frac{x-a}{s}\right)$$

$$W_f(s,a) = \int_{-\infty}^{\infty} \frac{1}{s}\psi^*\left(\frac{x-a}{s}\right) f(x)dx$$

- Let $\psi_s(x) = \frac{1}{s}\psi(\frac{x}{s})$ to use cross-correlation theorem
- $W_f(s,a) = W_{s,f}(a) = \int_{-\infty}^{\infty} {\psi_s}^*(x-a)f(x)\,dx = (\psi_s \bigstar f)(a)$
- $\widehat{W_{s,f}}(\omega) = \widehat{\psi_s}^*(\omega)\hat{f}(\omega) = \hat{\psi}^*(s\omega)\hat{f}(\omega)$

Classical Wavelet Transform

Cross-correlation theorem [edit]

Main article: Cross-correlation

In an analogous manner, it can be shown that if $h(x)$ is the cross-correlation of $f(x)$ and $g(x)$:

$$h(x) = (f \star g)(x) = \int_{-\infty}^{\infty} \overline{f(y)} g(x+y)\, dy$$

then the Fourier transform of $h(x)$ is:

$$\hat{h}(\xi) = \overline{\hat{f}(\xi)} \cdot \hat{g}(\xi).$$

Classical Wavelet Transform

- $W_{s,f}(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega x}\hat{\psi}^*(s\omega)\hat{f}(\omega)d\omega$
  - Note that x was previously a

Spectral Graph Wavelet Transform

- The transform will be determined by the choice of a kernel function $g : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ which is analogous to Fourier domain wavelet $\hat{\psi}^*$
- This kernel g should behave as a band-pass filter, i.e. it satisfies $g(0) = 0$ and $\lim_{x \rightarrow \infty} g(x) = 0$
- $W_{s,f}(x) = \sum_{l=0}^{N-1} g(s\lambda_l)\hat{f}(l)\chi_l(x)$

Chebyshev polynomial

- https://en.wikipedia.org/wiki/Chebyshev_polynomials

$$
\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
T_5(x) &= 16x^5 - 20x^3 + 5x \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1 \\
T_7(x) &= 64x^7 - 112x^5 + 56x^3 - 7x \\
T_8(x) &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\
T_9(x) &= 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\
T_{10}(x) &= 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 \\
T_{11}(x) &= 1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x
\end{aligned}
$$

Polynomial Approximation and Fast SGWT

- https://en.wikipedia.org/wiki/Minimax_approximation_algorithm
- G. M. Phillips, Interpolation and Approximation by Polynomials, CMS Books in Mathematics, Springer-Verlag, 2003.

Graph Convolution Network

- $W_{s,f}(x) = \sum_{l=0}^{N-1} g(s\lambda_l)\hat{f}(l)\chi_l(x)$
- --
- Let $s = 1$
- $W_f(x) = \sum_{l=0}^{N-1} g(\lambda_l)\hat{f}(l)\chi_l(x)$
- Use normalized Laplacian $D^{-\frac{1}{2}}LD^{-\frac{1}{2}} = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = X\Lambda X^T$
- View $f: V \rightarrow \mathbb{R}$ as vector $f \in \mathbb{R}^N$
- $W_f = g_f = Xg(\Lambda)X^T f$

Graph Convolution Network

- View $f: V \to \mathbb{R}$ as vector $f \in \mathbb{R}^N$
- $W_f = g_f = Xg(\Lambda)X^T f$
- $g(\Lambda) \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})$
  - θ is a vector of Chebyshev coefficients
  - $\tilde{\Lambda} = \frac{2}{\lambda_{\max}} \Lambda - I_N$
- $g_f \approx \sum_{k=0}^{K} \theta_k X T_k(\tilde{\Lambda}) X^T f = \sum_{k=0}^{K} \theta_k T_k(X \tilde{\Lambda} X^T) f = \sum_{k=0}^{K} \theta_k T_k(\tilde{L}) f$
  - $\tilde{L} = \frac{2}{\lambda_{\max}} L - I_N$

Graph Convolution Network

- Let $K = 1$ as we can still recover a rich class of convolutional filter functions by stacking multiple such layers

- Let $\lambda_{max} \approx 2$ as we can expect that neural network parameters will adapt to this change in scale during training

- $g_f \approx \theta_0 T_0(L - I_N)f + \theta_1 T_1(L - I_N)f = \theta_0 f + \theta_1 (L - I_N)f = \theta_0 f - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} f$

Graph Convolution Network

- Let $\theta_0 = -\theta_1$, it can be beneficial to constrain the number of parameters further to address overfitting and to minimize the number of operations

- $g_f \approx \theta\left(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)f$

$$I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$$

$$\tilde{A} = A + I_N \text{ and } \tilde{D}_{ii} = \textstyle\sum_j \tilde{A}_{ij}$$

Graph Convolution Network

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

$$\tilde{A} = A + I_N \text{ and } \tilde{D}_{ii} = \textstyle\sum_j \tilde{A}_{ij}$$

- $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ has eigenvalues in the range [0, 2]. Repeated application of this operator can lead to numerical instabilities and exploding/vanishing gradients.

- $g_f \approx \theta\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}\right) f$

Graph Convolution Network

We can generalize this definition to a signal $X \in \mathbb{R}^{N \times C}$ with C input channels (i.e. a C-dimensional feature vector for every node) and F filters or feature maps as follows:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \,, \tag{8}$$

where $\Theta \in \mathbb{R}^{C \times F}$ is now a matrix of filter parameters and $Z \in \mathbb{R}^{N \times F}$ is the convolved signal matrix. This filtering operation has complexity $\mathcal{O}(|\mathcal{E}|FC)$, as $\tilde{A}X$ can be efficiently implemented as a product of a sparse matrix with a dense matrix.