CPSC 340: Machine Learning and Data Mining

Neural Networks

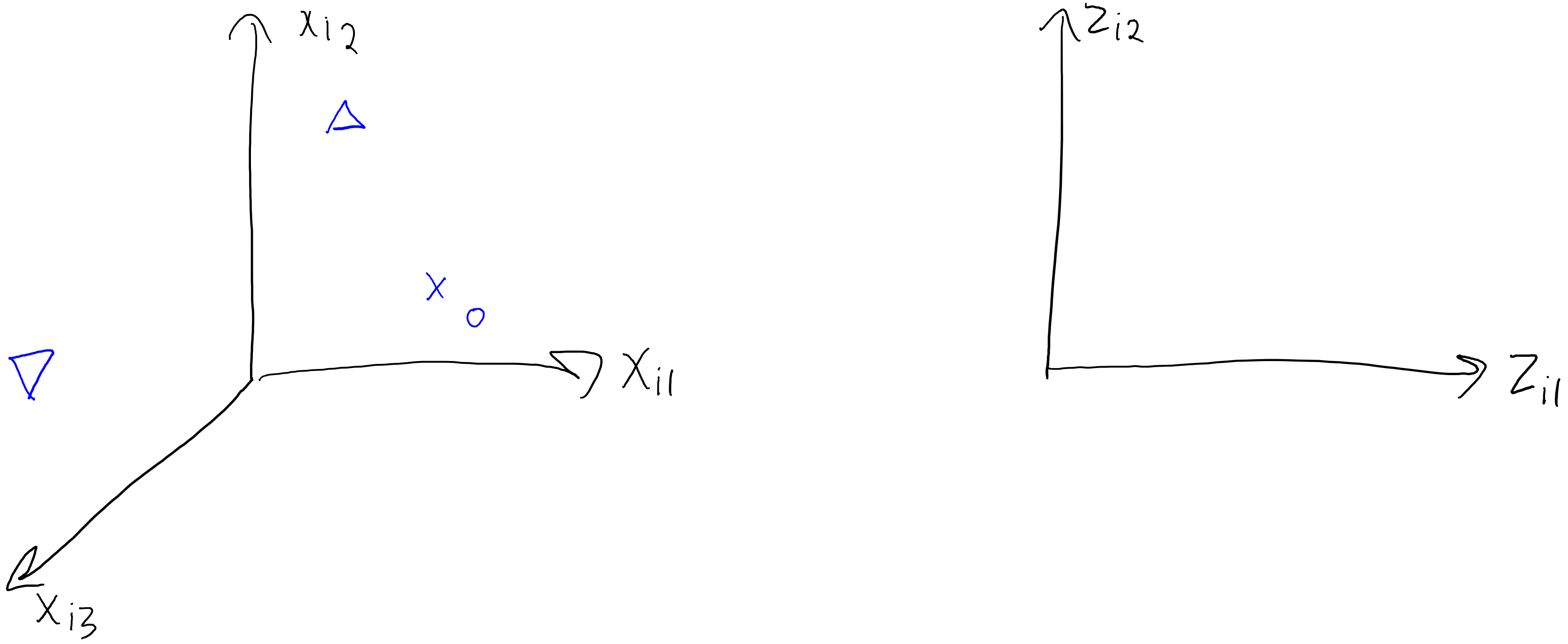
Fall 2015

Admin

- Assignment 2 marks updated.
- Remaining midterms can be picked up after class.
- Assignment 4 due Friday.

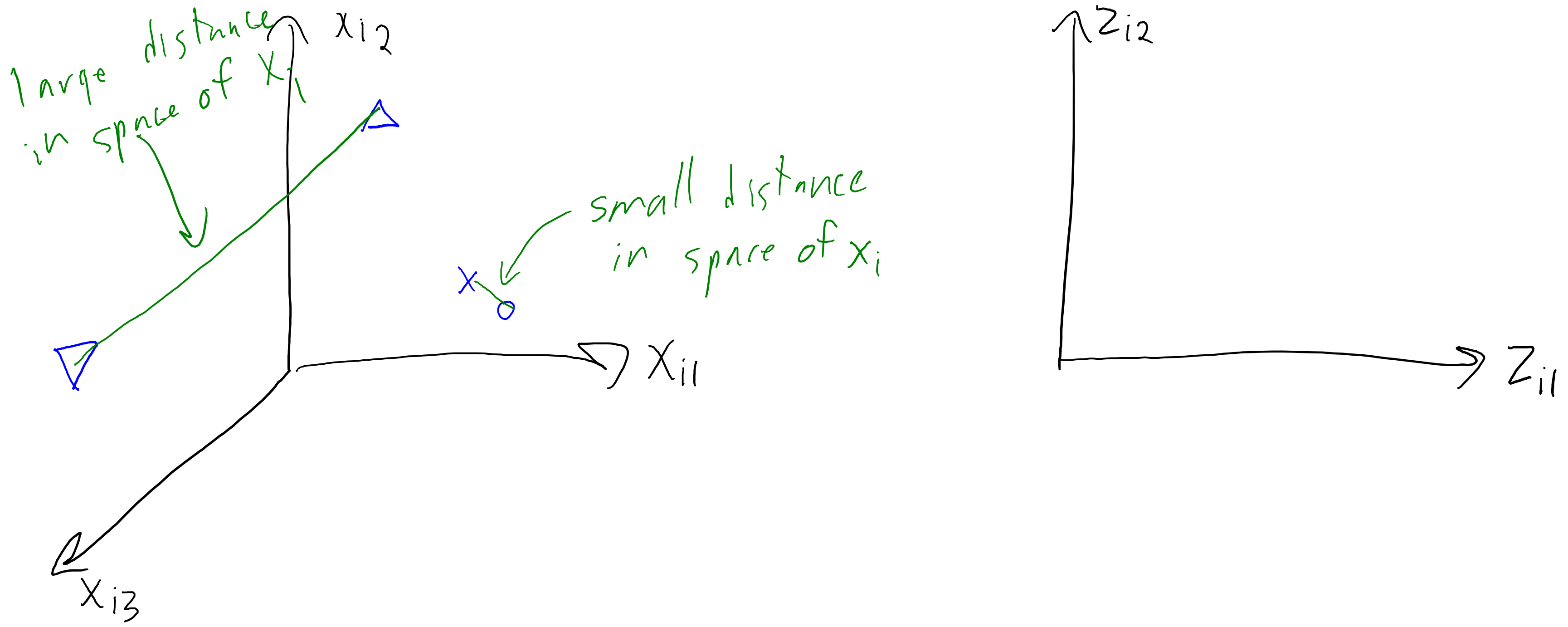
Last Time: Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Non-parametric dimensionality reduction and visualization methods.
  - Main idea: make distances between z_i close to distances between x_i.



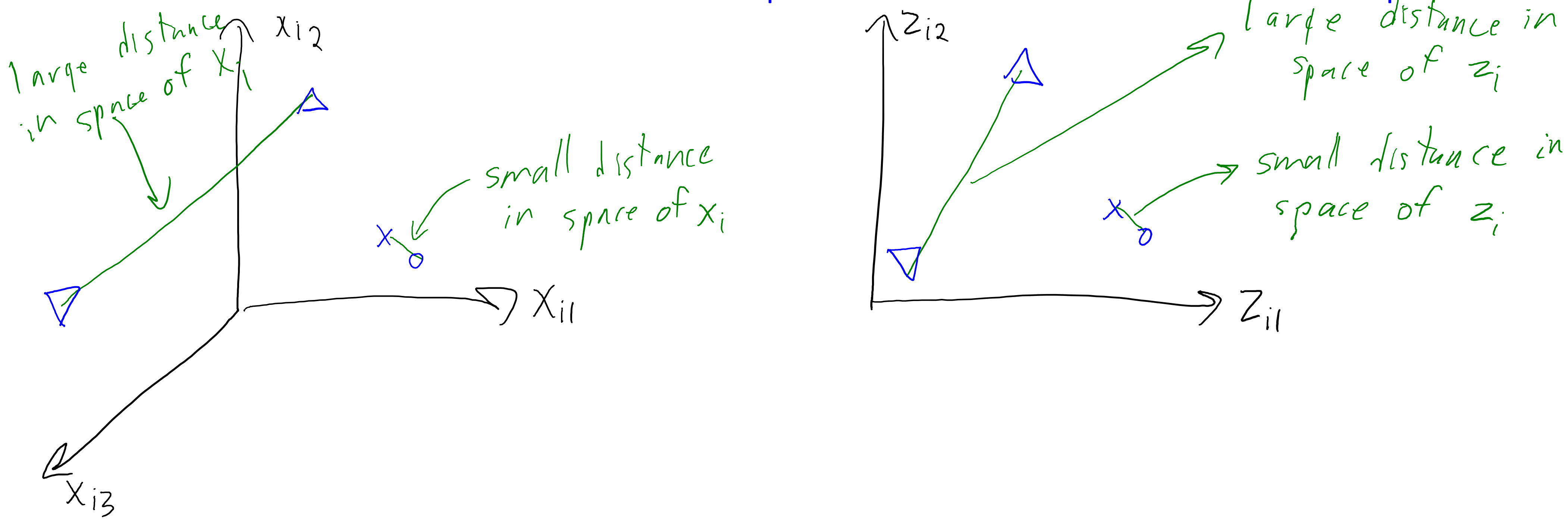
Last Time: Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Non-parametric dimensionality reduction and visualization methods.
  - Main idea: make distances between z_i close to distances between x_i.



Last Time: Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Non-parametric dimensionality reduction and visualization methods.
  - Main idea: make distances between z_i close to distances between x_i.



Last Time: Multi-Dimensional Scaling

- General MDS formulation:

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} g(d_1(x_i, x_j), d_2(z_i, z_j))$$

  - d_1: distance in high-dimensional space of 'x_i'.
  - d_2: distance in low-dimensional space of 'z_i'.
  - g: penalizes differences in 'd_1' and 'd_2'.
- Solution is computed using gradient descent.
  - To compute derivative, we need multivariate chain rule.

Univariate Chain Rule

- The univariate chain rule:

  If $f : \mathbb{R} \mapsto \mathbb{R}$ and $g : \mathbb{R} \mapsto \mathbb{R}$ then $\frac{d}{dx}[f(g(x))] = f'(g(x))g'(x)$.

- Example:

$$\frac{d}{dx}\left[\log(\underbrace{1+\exp(-x)}_{g(x)})\right] = f'(1+\exp(-x))[-\exp(-x)] = -\frac{\exp(-x)}{1+\exp(-x)}$$

$$= -\frac{1}{1+\exp(x)}$$

multiply by $\frac{\exp(x)}{\exp(x)}$

$f(z) = \log(z)$

$f'(z) = \frac{1}{z}$

$g(x) = 1 + \exp(-x)$

$g'(x) = -\exp(-x)$

Multivariate Chain Rule

- The multivariate chain rule:

  If $f: \mathbb{R} \to \mathbb{R}$ and $g: \underline{\mathbb{R}^d} \mapsto \mathbb{R}$ then $\nabla f(g(x)) = \underbrace{f'(g(x))}_{1 \times 1} \underbrace{\nabla g(x)}_{d \times 1}$

- Example:

$$\nabla_w \frac{1}{2}(y - w^T x)^2$$

$$g(w) = \sum_{i=1}^{d} w_i x_i$$

$$f(z) = \frac{1}{2}(y - z)^2$$

$$f'(z) = -(y - z)$$

$$\nabla g(w) = x$$

$$= f'(w^T x)\, x$$

$$= -(y - w^T x)\, x$$

Multivariate Chain Rule for MDS

- General MDS formulation:

$$\underset{Z \in \mathbb{R}^{n \times k}}{\text{argmin}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} g(d_1(x_i, x_j), d_2(z_i, z_j))$$

- Using multivariate chain rule we have:

$$\nabla_{z_i} g(d_1(x_i, x_j), d_2(z_i, z_j)) = g'(d_1(x_i, x_j), d_2(z_i, z_j)) \nabla_{z_i} d_2(z_i, z_j)$$

- Example: If $d_1(x_i, x_j) = \|x_i - x_j\|$ and $d_2(z_i, z_j) = \|z_i - z_j\|$ and $g(d_1, d_2) = \frac{1}{2}(d_1 - d_2)^2$

$$\nabla_{z_i} g(d_1(x_i, x_j), d_2(z_i, z_j)) = \underbrace{-(d_1(x_i, x_j) - d_2(z_i, z_j))}_{g'(d_1, d_2)} \underbrace{\left[- \frac{(z_i - z_j)}{2\|z_i - z_j\|} \right]}_{\nabla_{z_i} d_2(z_i, z_j)}$$

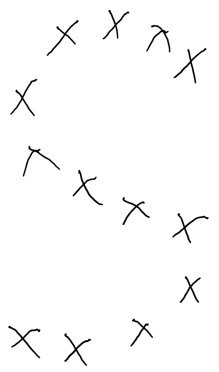
Assuming $z_i \neq z_j$

(move distances closer)

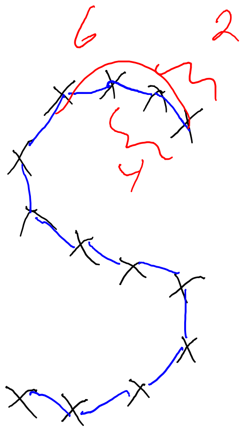
(how distance changes in z-space)

ISOMAP

- ISOMAP performs dimensionality reduction for data on a manifold:



make graph
from "neighbours"

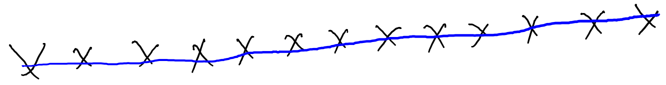


compute distances
through graph

(assume each neighbour has distance "2")

$$D_1 = \begin{bmatrix} 0 & 2 & 4 & 6 & \cdots \\ 2 & 0 & 2 & 4 & \cdots \\ 4 & 2 & 0 & 2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

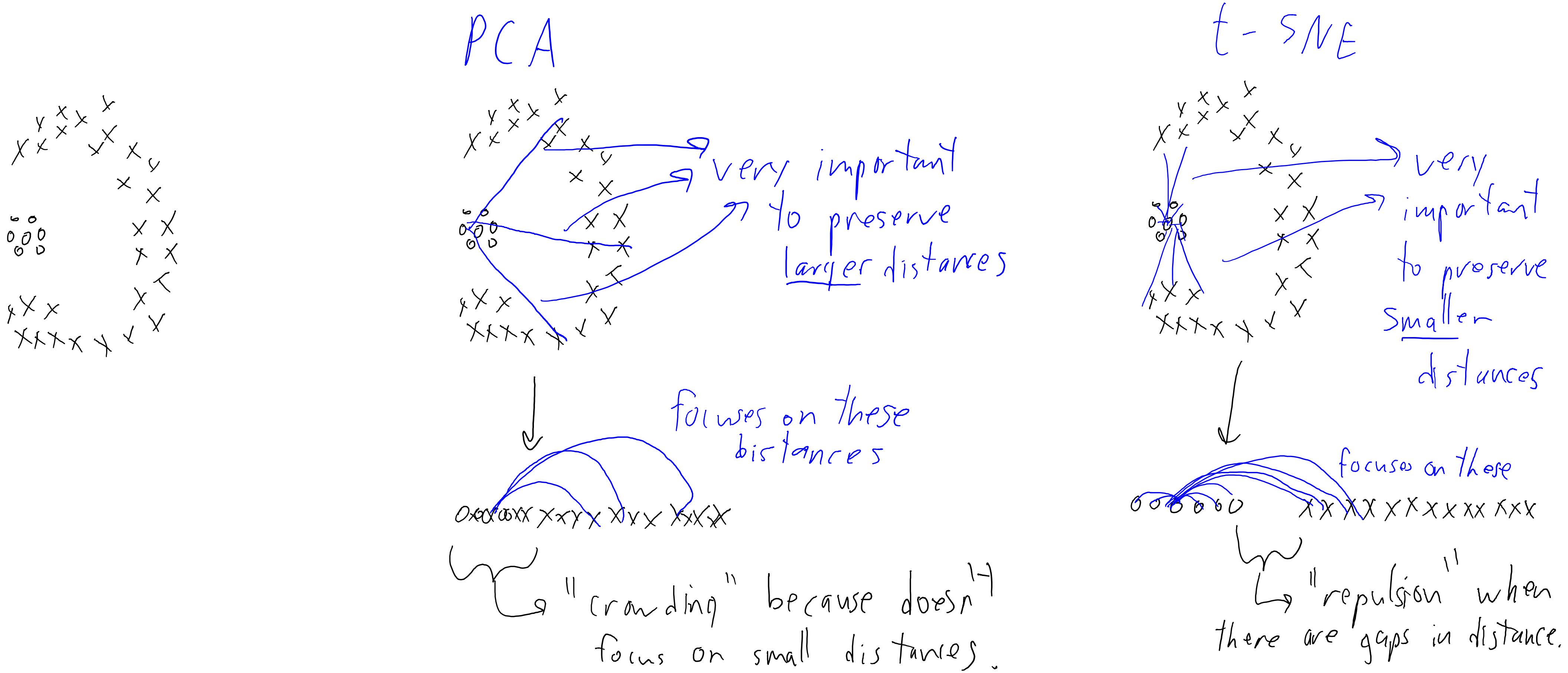
Run MDS with these distances.



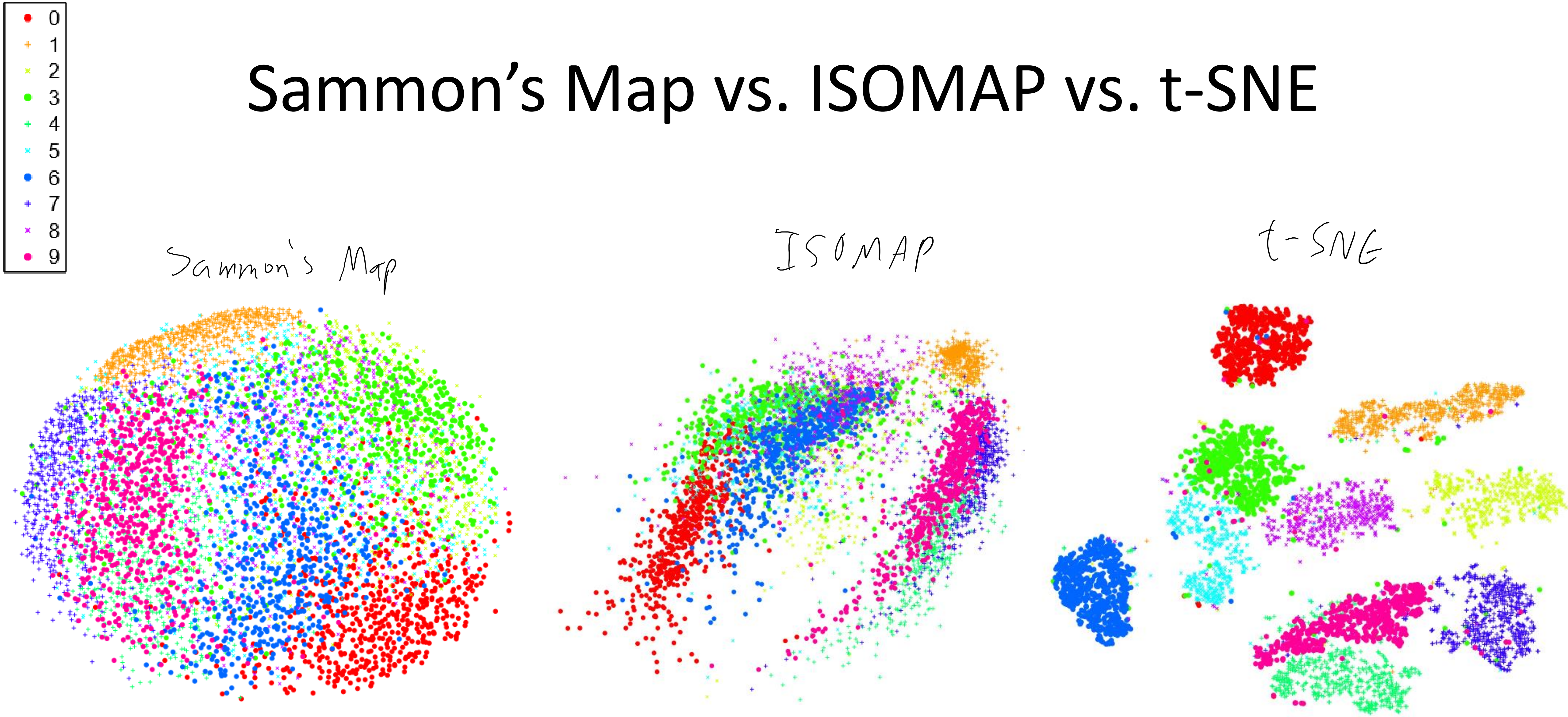
ISOMAP z_i values in 2D or 1D.

t-Distributed Stochastic Neighbour Embedding

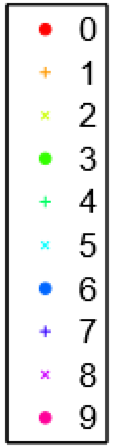
- One key idea in t-SNE:
  - Focus on small distances by allowing large variance in large distances.



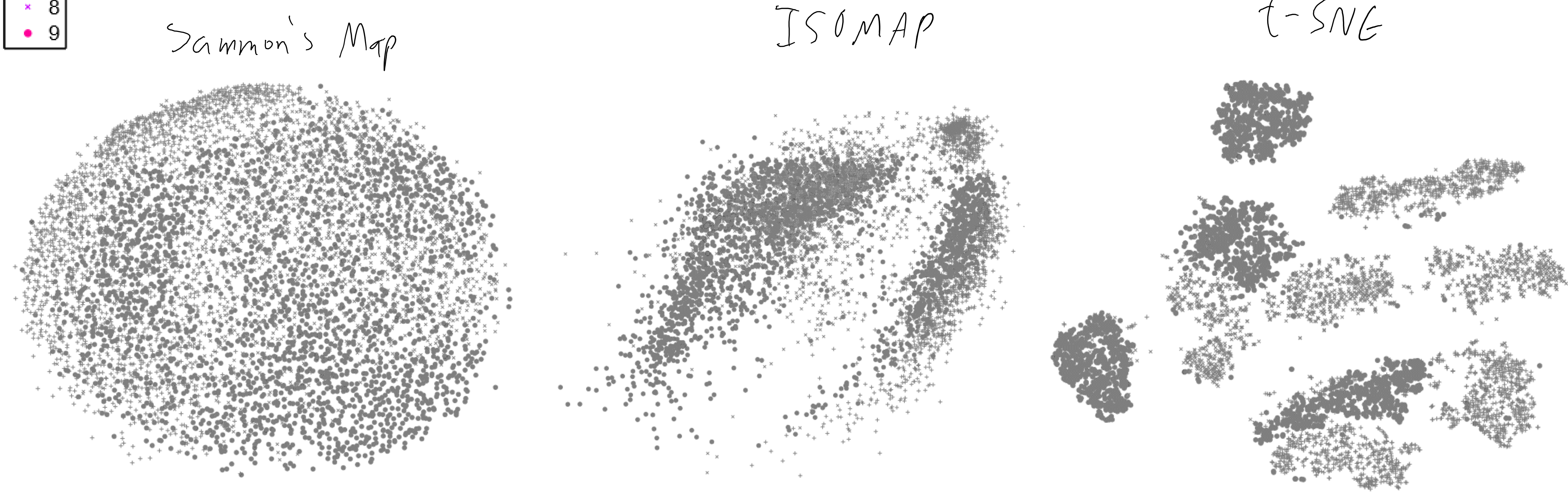
Sammon's Map vs. ISOMAP vs. t-SNE



http://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

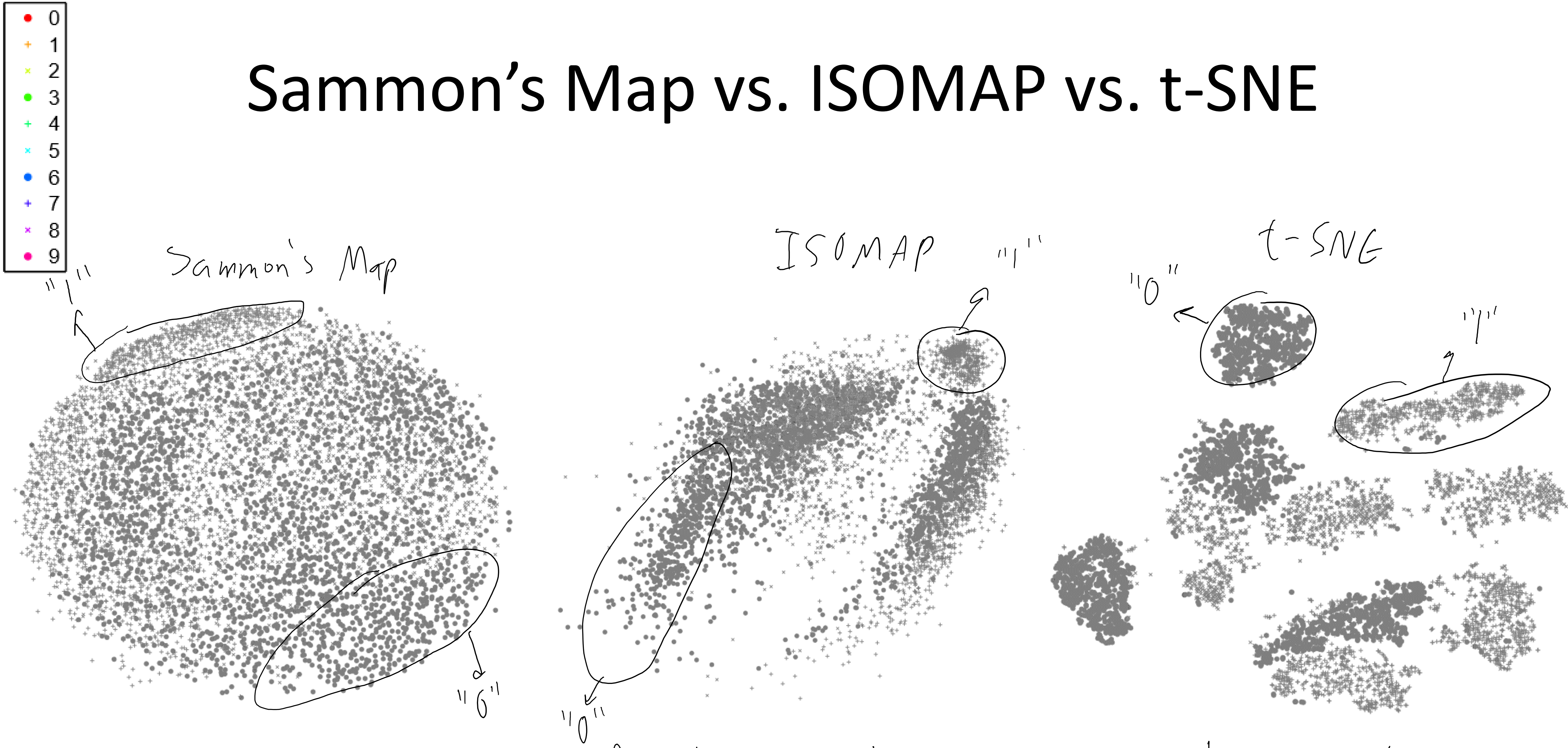


Sammon's Map vs. ISOMAP vs. t-SNE



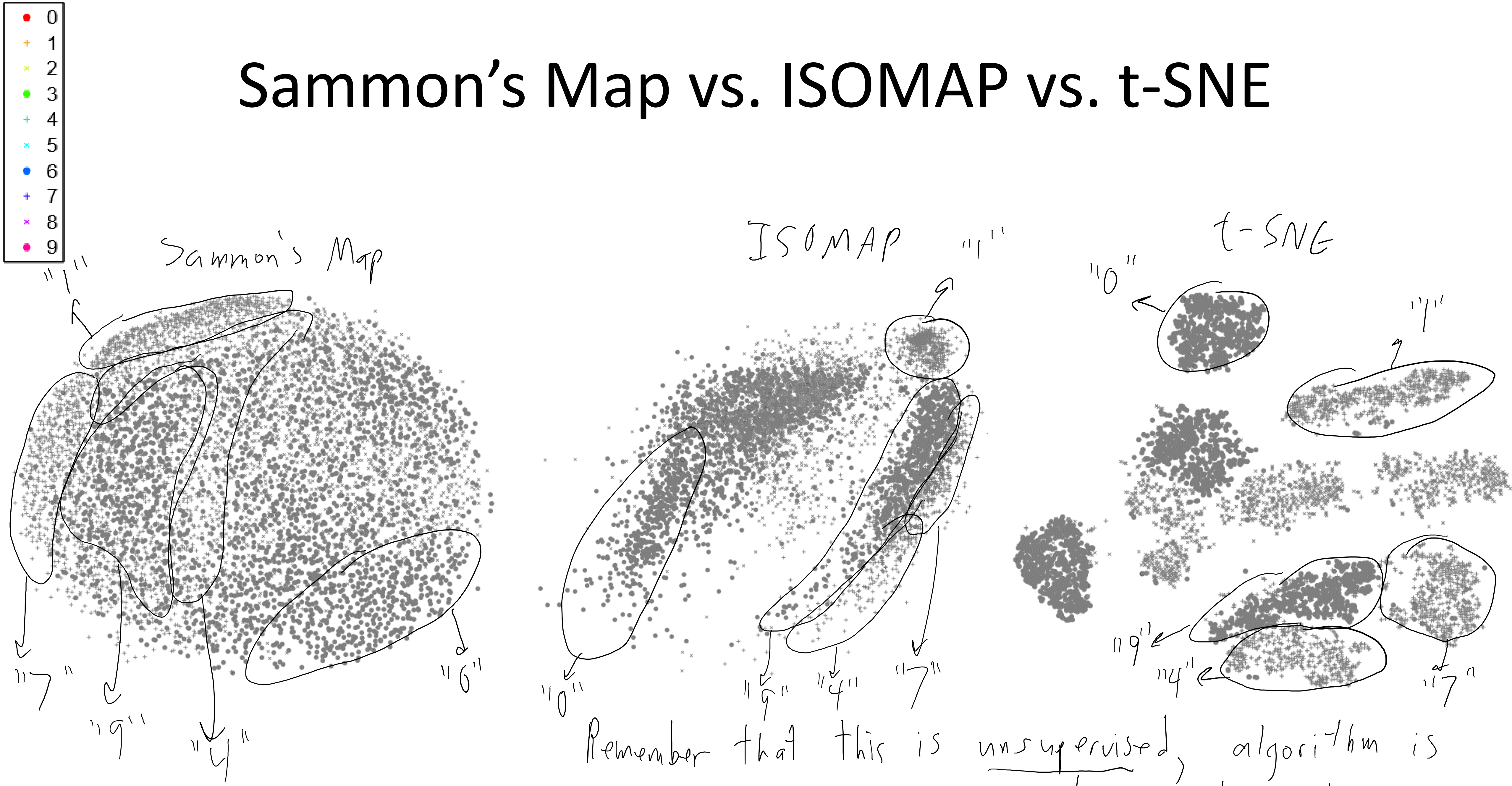
Remember that this is unsupervised, algorithm is not given the colours.

Sammon's Map vs. ISOMAP vs. t-SNE



Remember that this is unsupervised, algorithm is not given the colours.

Sammon's Map vs. ISOMAP vs. t-SNE

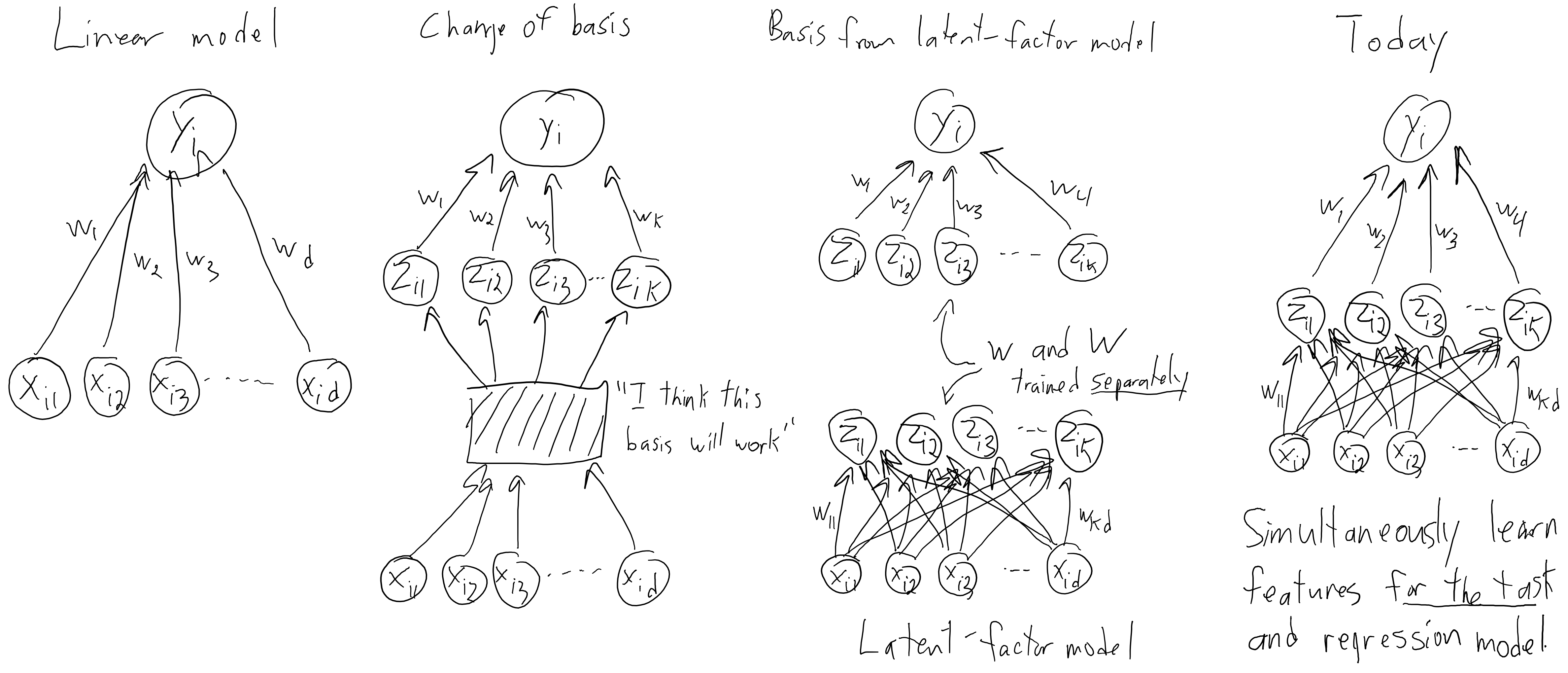


Remember that this is unsupervised, algorithm is not given the colours.

Supervised Learning Roadmap

- Supervised Learning Parts 1 and 2:
  - Assumed that we are given the features x_i.
  - Could also use basis functions or kernels.
- Unsupervised Learning Part 2:
  - We considered learning a representation z_i based on features x_i.
  - Can also be used for supervised: use z_i as features.
- Supervised Learning Part 3:
  - Learn features z_i that are good for supervised learning.

Supervised Learning Roadmap



Linear-Linear Model

- Natural choice: linear regression with linear basis:

Representation: $z_i = Wx_i$ Prediction: $\hat{y}_i = w^T z_i$

Loss: $\frac{1}{2}(y_i - \hat{y}_i)^2$

- To train this model, we could solve:

$$\underset{w \in \mathbb{R}^k, W \in \mathbb{R}^{k \times d}}{\text{argmin}} \underbrace{\frac{1}{2}\sum_{i=1}^{n}(y_i - w^T z_i)^2}_{\text{linear regression with } z_i \text{ as features.}} = \frac{1}{2}\sum_{i=1}^{n}(y_i - w^T \underbrace{(Wx_i}_{\text{features come from latent-factor model}}))^2$$

- But this is just a linear model: $w^T(Wx_i) = (\underbrace{W^T w}_{\text{vector}})^T x_i = \underline{\tilde{w}^T x_i}$

this is just a usual linear regression model.

Introducing Non-Linearity

- To increase flexibility, something needs to be non-linear.
- Typical choice: transform z_i by non-linear function 'h'.

Representation: $z_i = h(Wx_i)$ Prediction: $\hat{y}_i = w^T z_i$

- Common choice for 'h': applying sigmoid function element-wise:

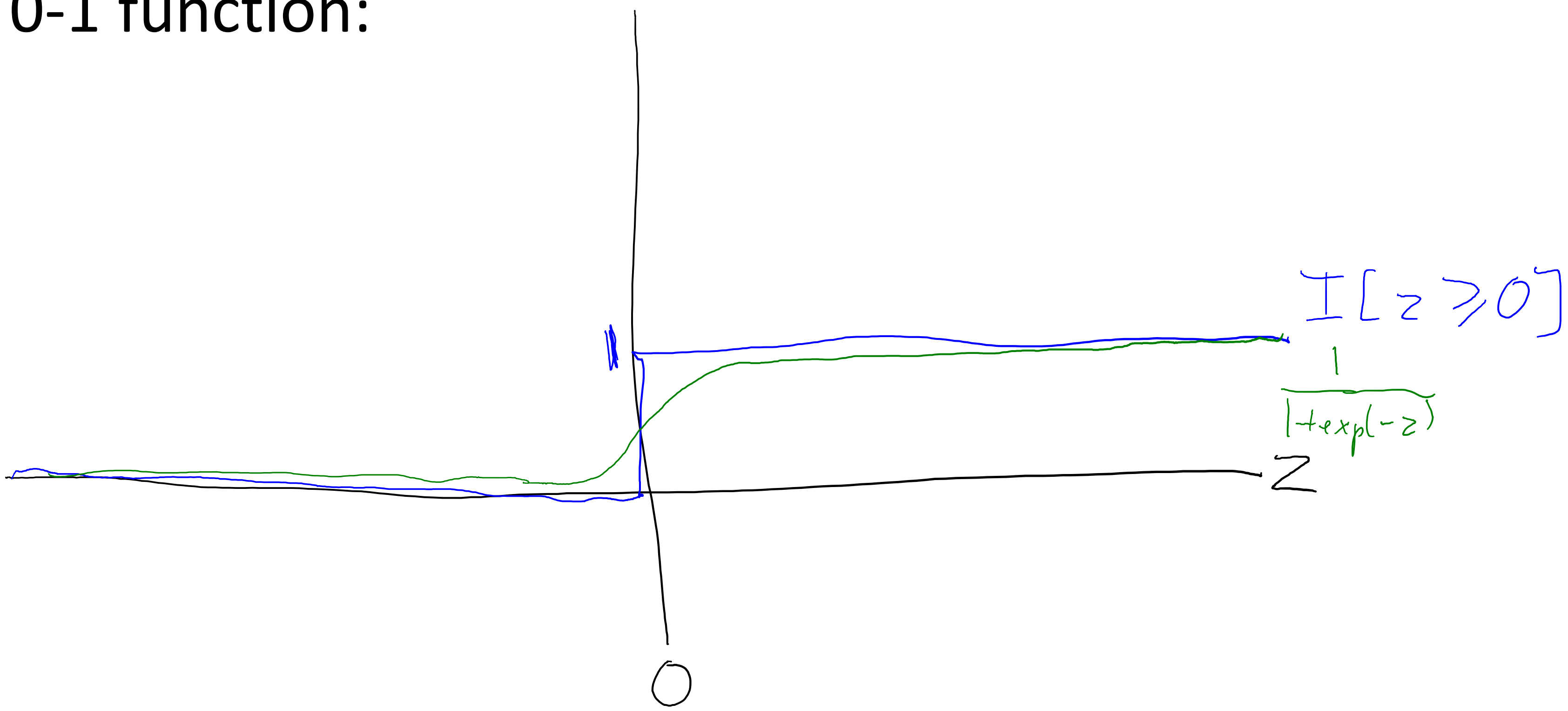
$$p(z) = \frac{1}{1+\exp(-z)} \qquad z_{ic} = \frac{1}{1+\exp(-w_c^T x_i)}$$

"sigmoid"

- This is called a 'multi-layer perceptron' or 'neural network'.

Why Sigmoid?

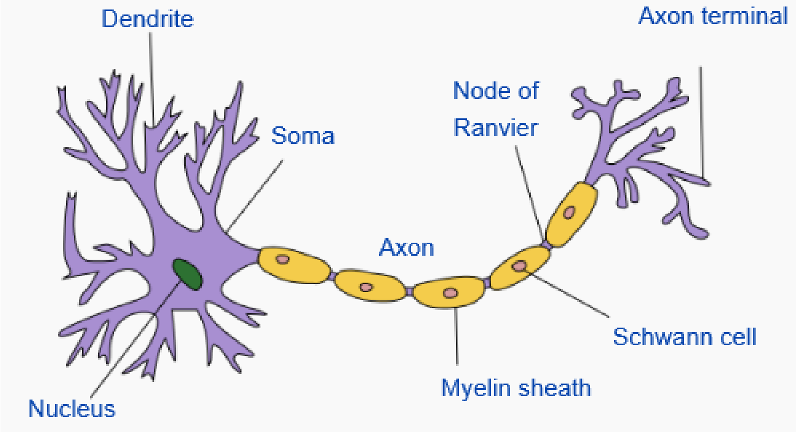
- Recall the 0-1 function:



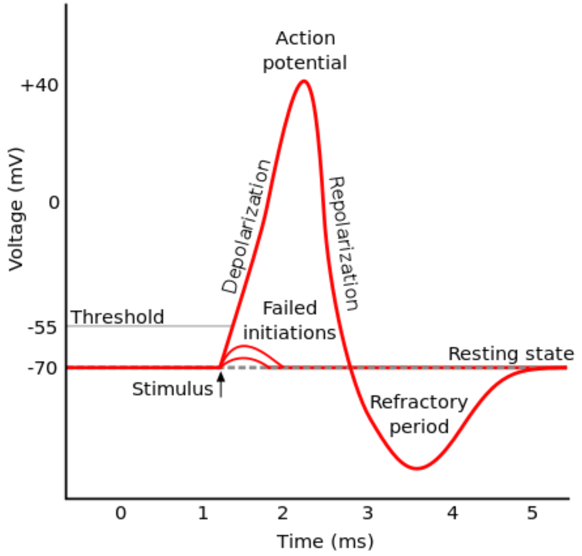
- Element-wise 0-1 function would give us a binary z_i.
  - Wx_i has a 'concept' encoded by each of its 2^k possible signs.
- Sigmoid function is a smooth approximation to 0-1 function.

Why ‘Neural Network’?

- Cartoon of ‘typical’ neuron:

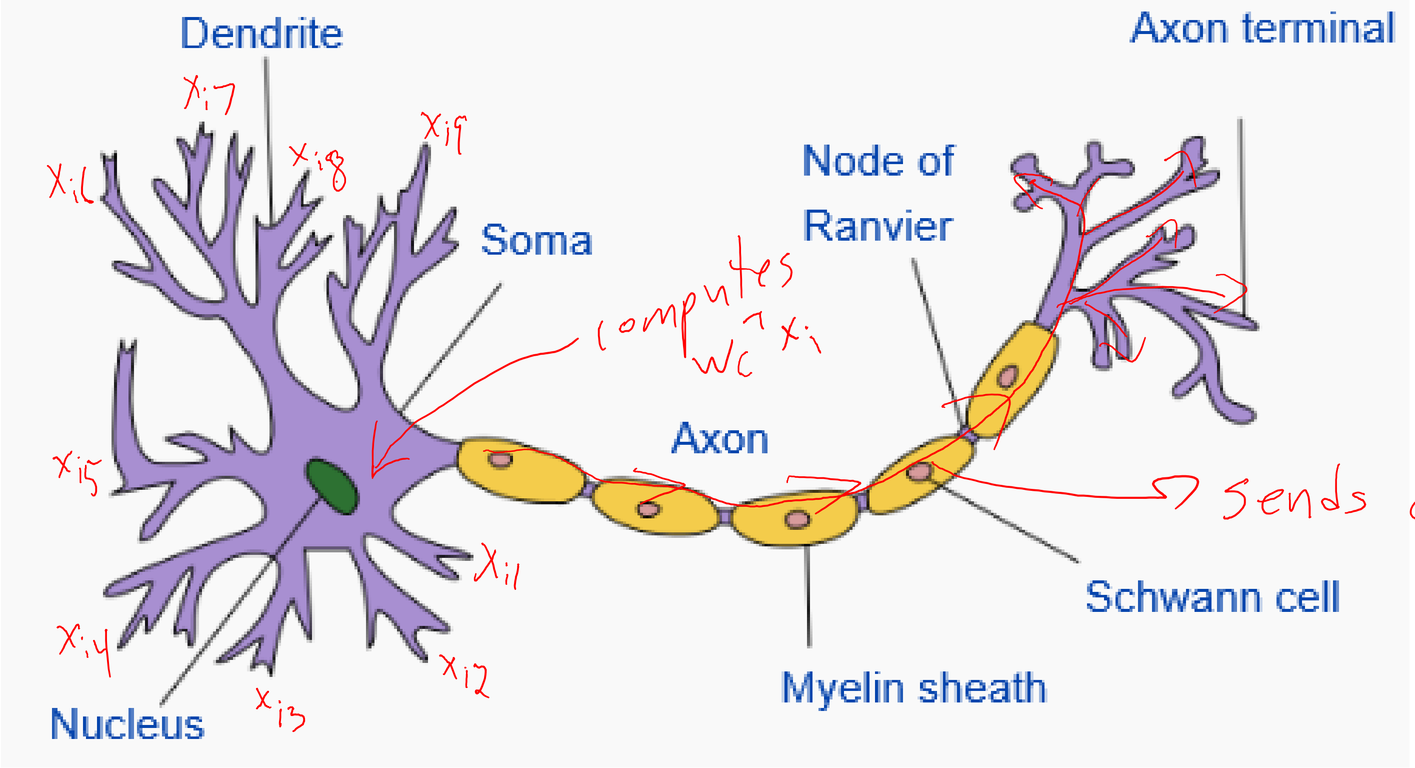


- Neuron has many dendrites, which take ‘input’.
- Neuron has a single axon, which sends ‘output’.
- With the right input to dendrites:
  - ‘Action potential’ along axon (like a binary signal):



https://en.wikipedia.org/wiki/Neuron
https://en.wikipedia.org/wiki/Action_potential

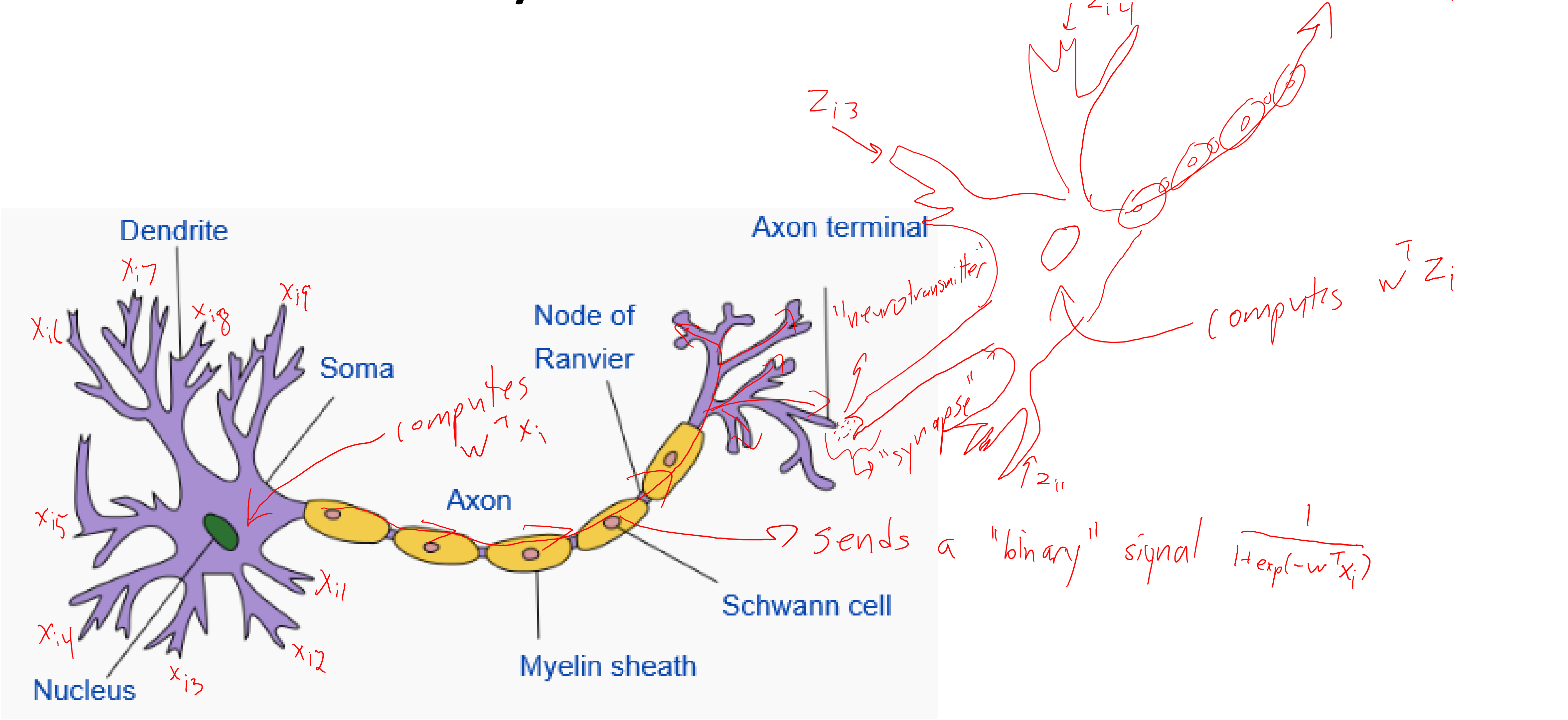
Why 'Neural Network'?



sends a "binary" signal $\frac{1}{1+\exp(-w_c^T x_i)}$

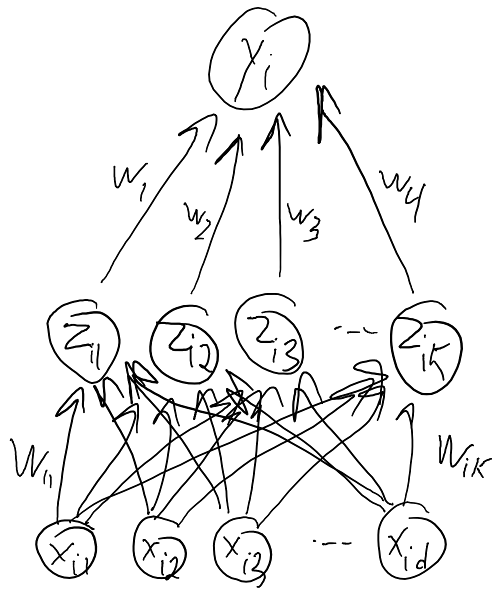
https://en.wikipedia.org/wiki/Neuron

Why ‘Neural Network’?



https://en.wikipedia.org/wiki/Neuron

‘Artificial’ Neural Nets vs. ‘Real’ Networks Nets



- Artificial neural network:
  - x_i is measuring of the world.
  - z_i is internal representation of world.
  - y_i as output of neuron for classification/regression.
- Real neural networks are more complicated:
  - Timing of action potentials seems to be important.
    - ‘Rate coding’: frequency of action potentials simulates continuous output.
  - Neural networks don’t reflect sparsity of action potentials.
  - How much computation is done inside neuron?
  - Brain is highly organized (e.g., substructures and cortical columns).
  - Connection structure changes.
  - Different types of neurotransmitters.

Artificial Neural Networks

- With squared loss, our objective function is:

$$\underset{w \in \mathbb{R}^k, W \in \mathbb{R}^{k \times d}}{\text{argmin}} \frac{1}{2} \sum_{i=1}^{n} (y_i - w^T h(Wx_i))^2$$

- Usual training procedure: stochastic gradient.
  – Compute gradient of random example ‘i’, update ‘w’ and ‘W’.
- Computing the gradient is known as “backpropagation”.
- Adding regularization to ‘w’ and/or ‘W’ is known as “weight decay”.

Backpropagation

- Consider the loss for a single example:

$$f(w,W) = \frac{1}{2}\Big(y_i - \sum_{j=1}^{k} \underbrace{w_j}_{\text{element 'j' of } w}\, h(\overbrace{W_j}^{\text{row 'j' of } W}\, x)\Big)^2$$

- Derivatives with respect to 'w_j':

$$\frac{\partial f}{\partial w_j} f(w,W) = \underbrace{-\Big(y - \sum_{j=1}^{k} w_j h(W_j x)\Big)}_{\text{from squared loss}}\ \underbrace{h(W_j x)}_{\text{derivative of } \sum_{j=1}^{k} w_j h(W_j x)}$$

- The gradient with respect to 'W_{ij}'

$$\frac{\partial f}{\partial W_{ij}} f(w,W) = \underbrace{-\Big(y - \sum_{j=1}^{k} w_j h(W_j x)\Big)}_{\text{from squared loss (same as before)}}\ \underbrace{w_j}_{\text{from } w_j h(W_j^T x)}\ \underbrace{h'(W_j x) x_i}_{\text{derivative of } h(W_j^T x)}$$

Backpropagation

- Notice repeated calculations in gradients:

$$\frac{\partial f}{\partial w_j} f(w,W) = -\left(y - \sum_{j=1}^{k} w_j h(W_j x)\right) h(W_j x)$$

$$\frac{\partial f}{\partial W_{ij}} f(w,W) = -\left(y - \sum_{j=1}^{k} w_j h(W_j x)\right) w_j h'(W_j x) x_i$$

Same value for $\frac{\partial f}{\partial w_j}$ with all j and $\frac{\partial f}{\partial W_{ij}}$ with all i and j.

Same value for $\frac{\partial f}{\partial W_{ij}}$ with all i.

"Forward" pass:

- Compute $\left(y - \sum_{j=1}^{k} w_j h(W_j x)\right)$

"Backward" pass:

- Compute all $\frac{\partial f}{\partial w_j}$
- Compute all $w_j h'(W_j x)$
- Compute all $\frac{\partial f}{\partial W_{ij}}$

Summary

- Multivariate chain rule computes gradient of compositions.
- Neural networks learn representation zi for supervised learning.
- Sigmoid function avoids degeneracy by introducing non-linearity.
- Biological motivation for binary representations.
- Backpropagation computes neural network gradient via chain rule.

- Next time:
  - Learning representations of complicated concepts with “deep” learning.