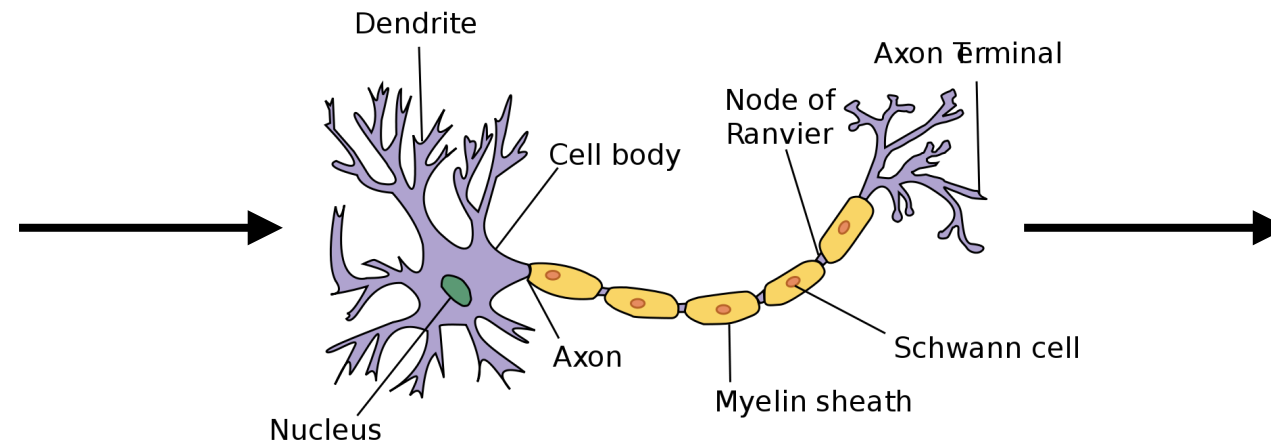


PY410 / 505  
Computational Physics 1

**Salvatore Rappoccio**

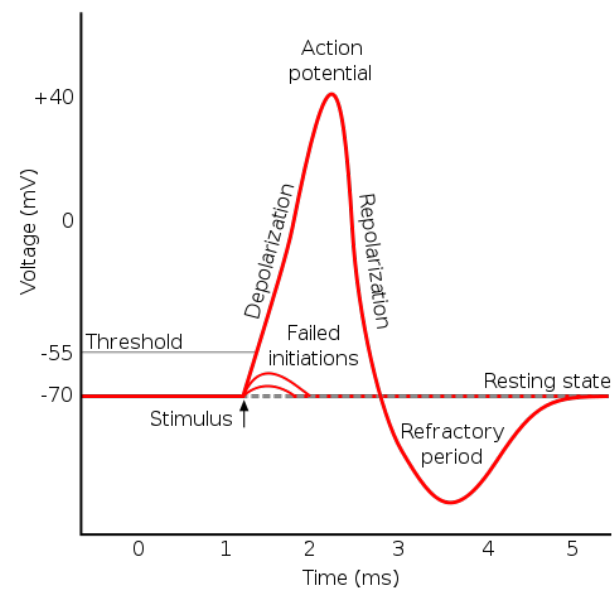
# Real Neural Networks

Chemical concentration input

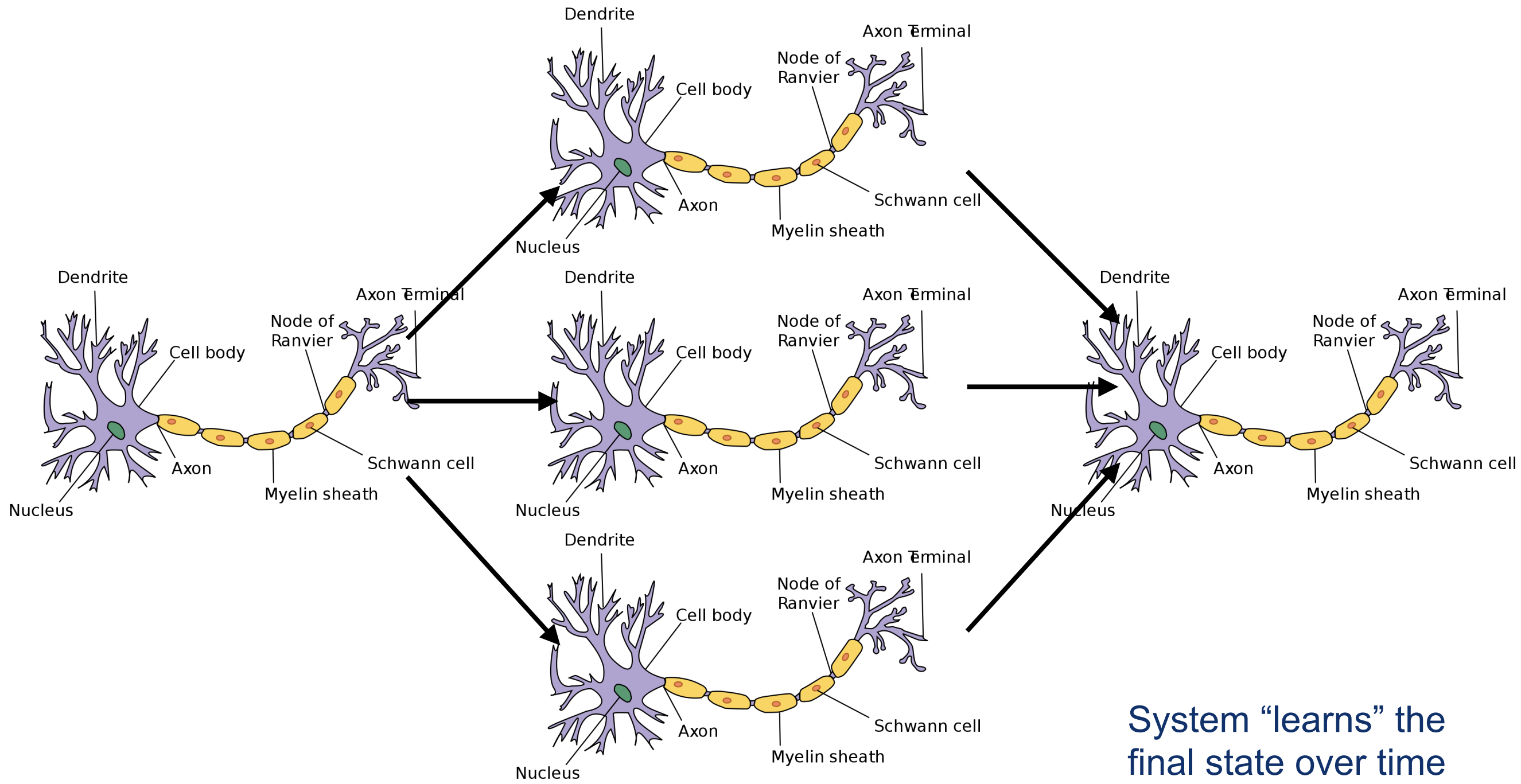


Chemical concentration output

## Activation



# Real Neural Networks

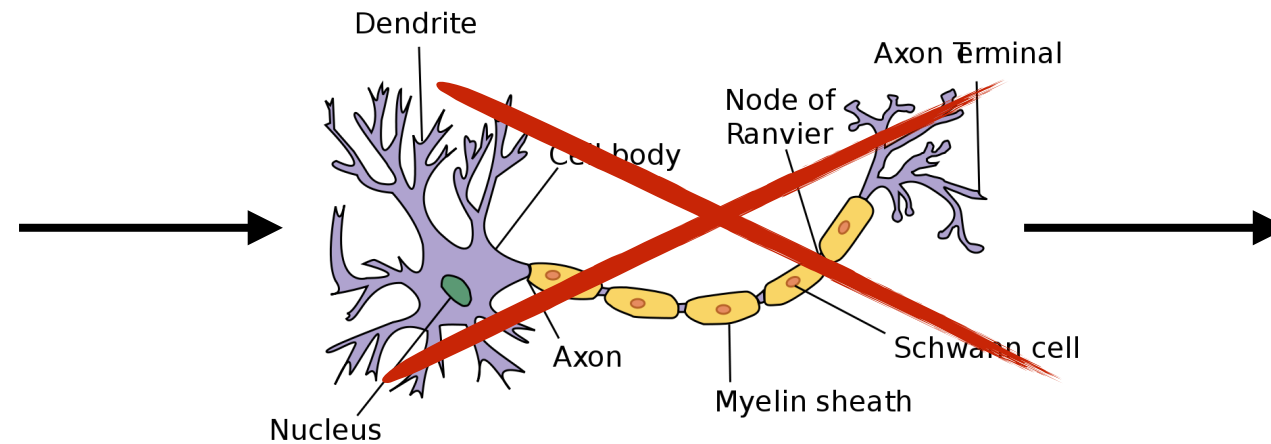


Neurons affect other neurons, and the "state" is saved

System "learns" the final state over time

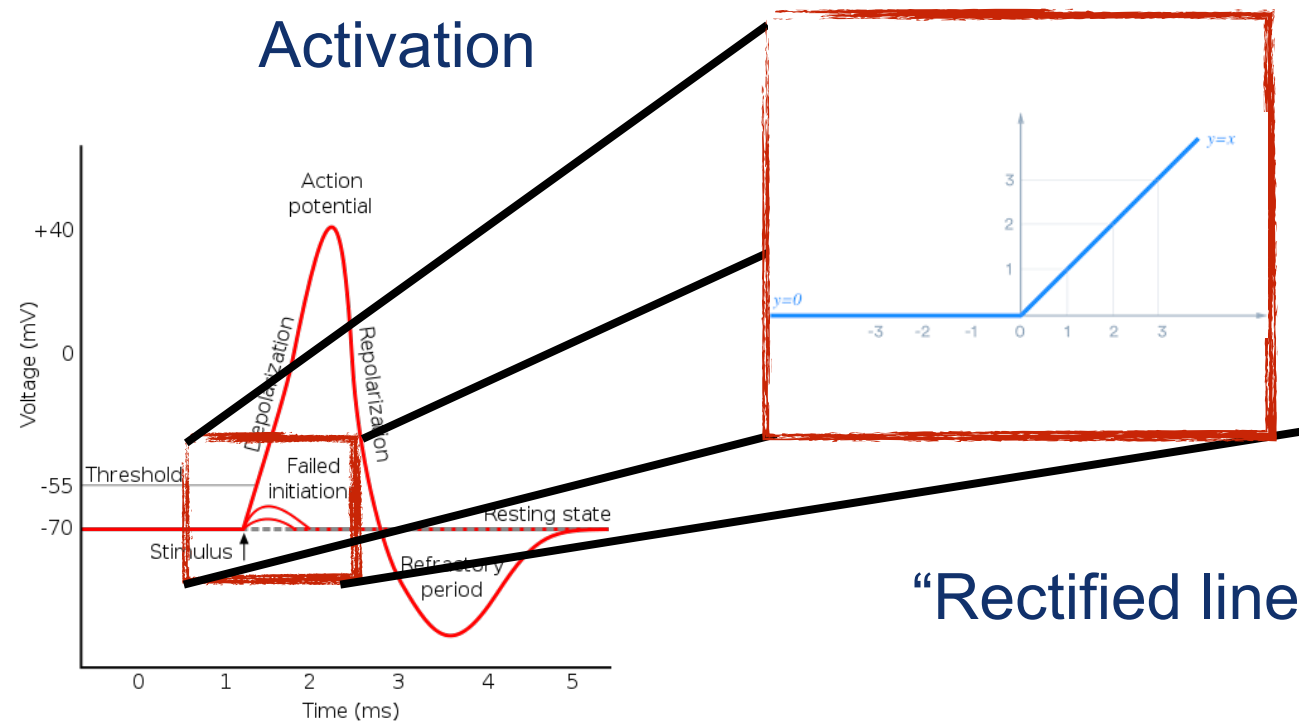
# Artificial Neural Networks

**Number**  
~~Chemical concentration~~  
input



**Number**  
~~Chemical concentration~~  
output

## Activation

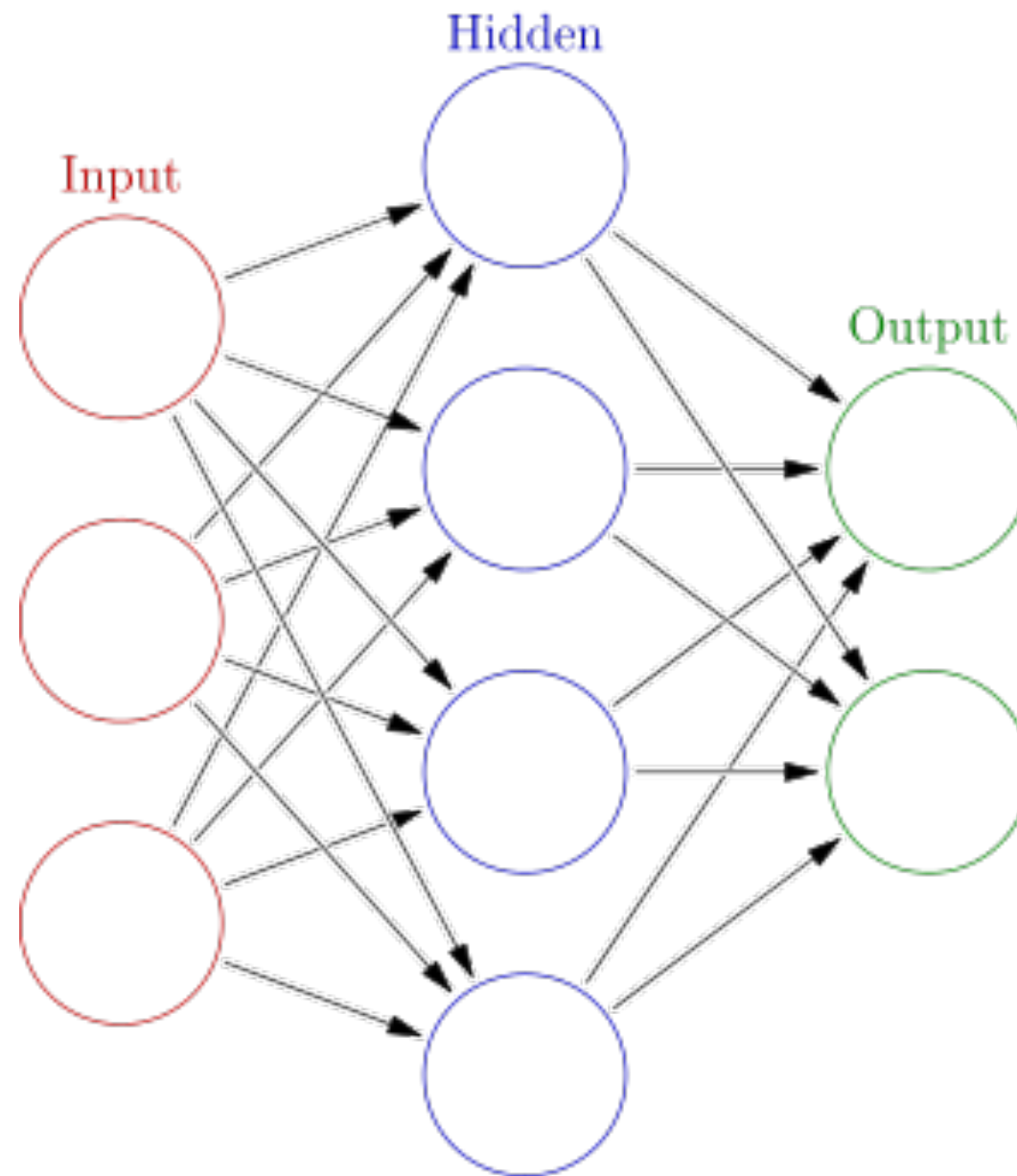


“Rectified linear unit”



# Neural Networks

- The neural network is a group of neurons working together in such a persistent state:

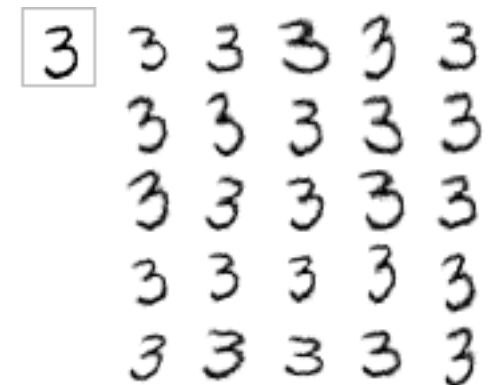


# Neural Networks

- Training:
  - Feed a sample where you know the answer to the network
  - Tell the network the answer
- Discrimination:
  - Network takes inputs from UNKNOWN source
  - Outputs weights relative to known outcomes

Example to recognize handwriting:

“These are all the number 3”:



“Is this a 3”?

3



Outputs “yes”

5

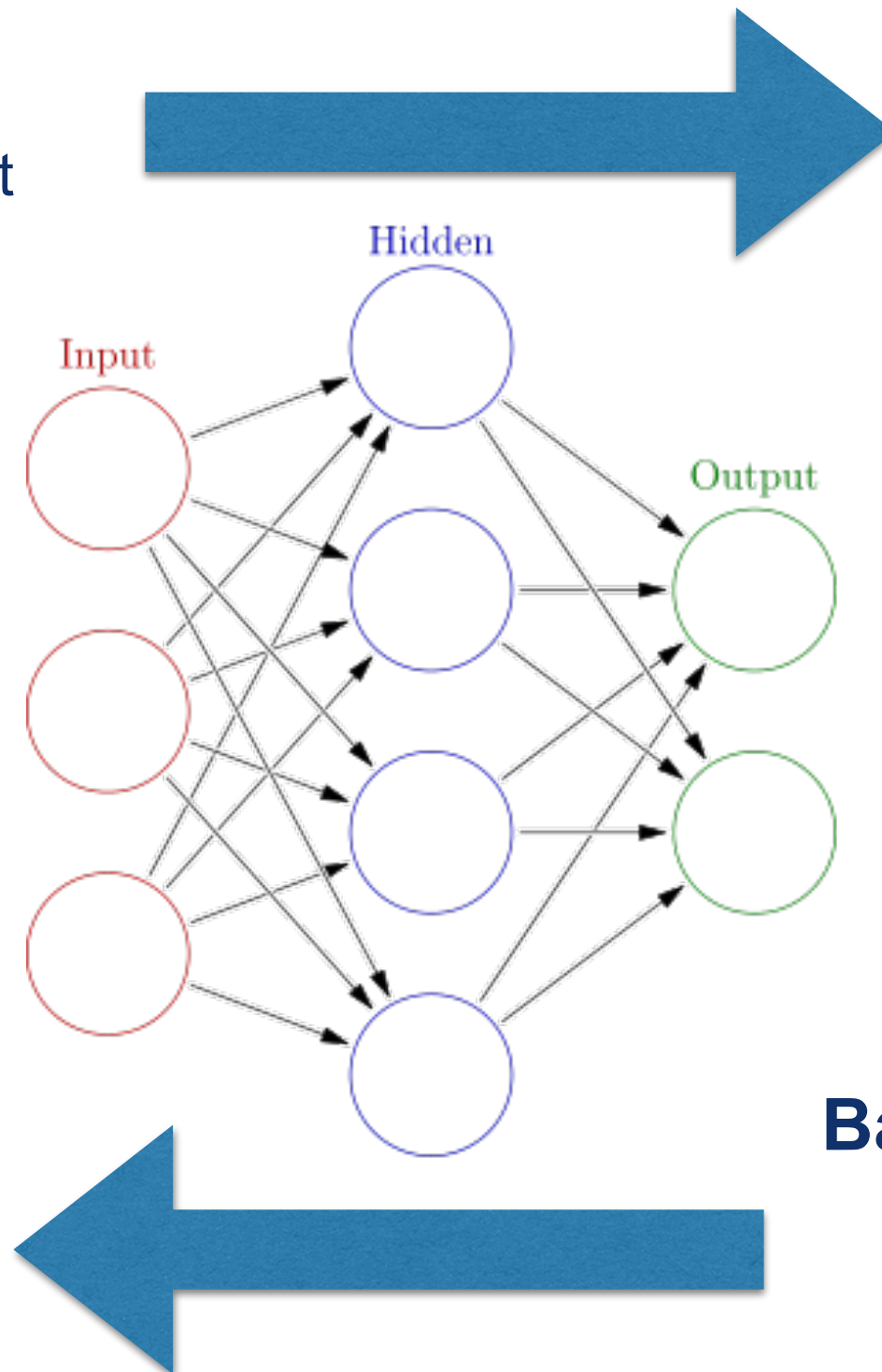


Outputs “no”

# Neural Networks

## Feed forward:

Weight inputs, get output



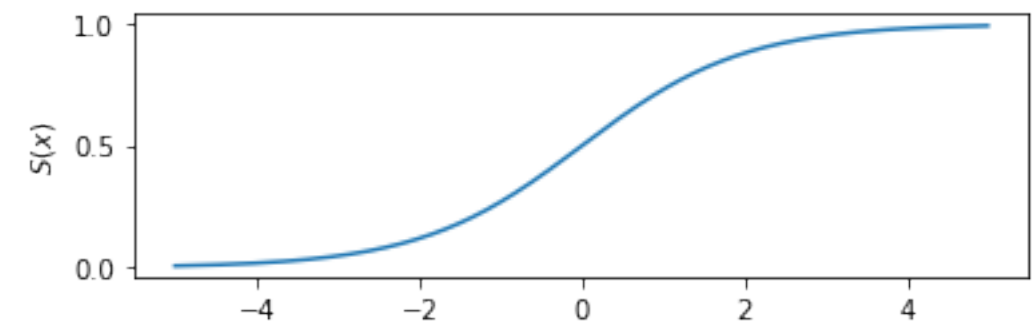
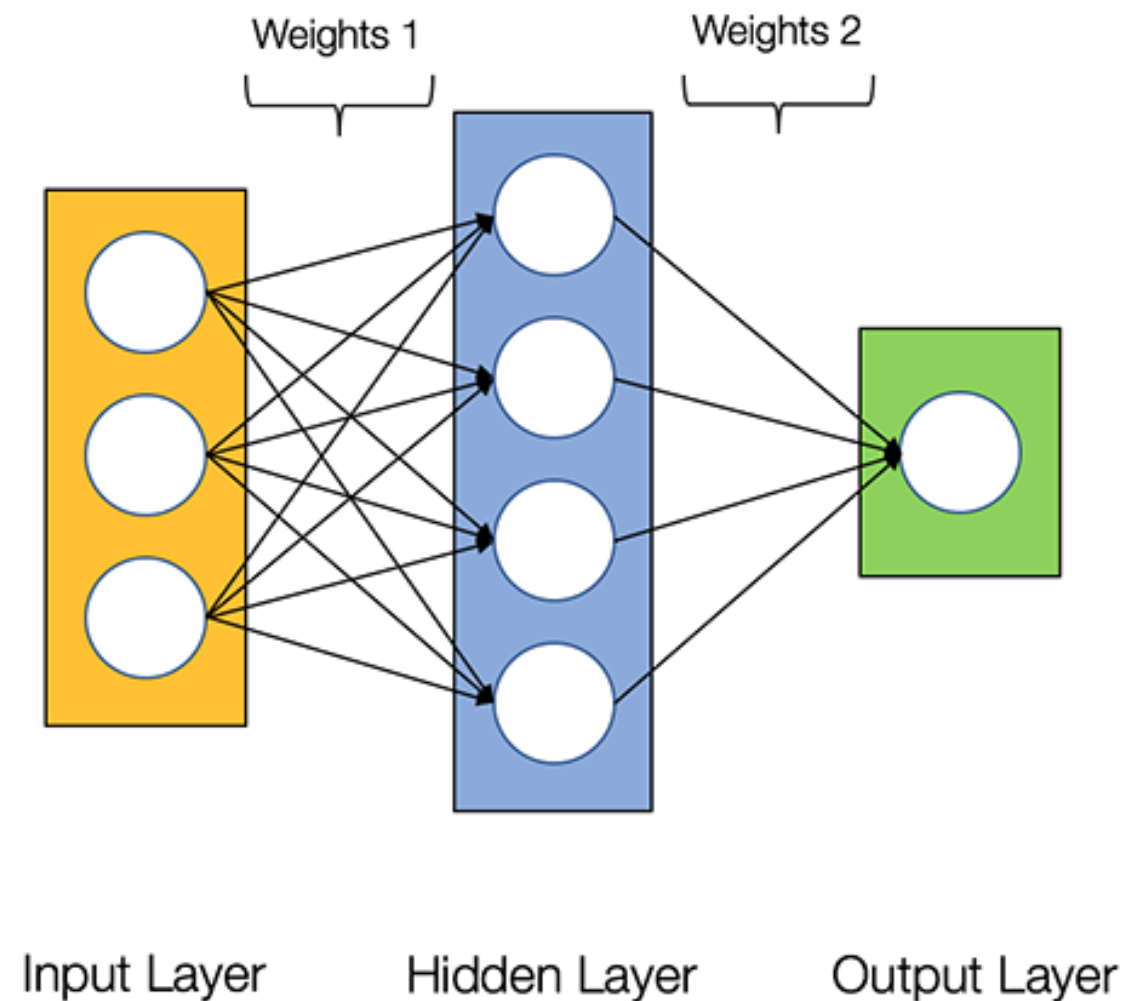
## Back-propagate:

Minimize difference between target output and current weights

# Neural Networks

- Consider a 2-layer network
- A bit more formally:
  - Input vector  $\vec{x}$
  - Output vector  $\vec{y}$
  - Weights and biases between layers  $W$  and  $b$
  - Activation function  $\sigma$

Usually rectified linear unit (ReLU), but we will use sigmoid because it is differentiable.



Adapted from

<https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6>

# Neural Networks

- For each training step:
  - Feed forward  $\vec{x} \rightarrow \hat{y}$ :
    - $\hat{y} = \sigma(W_2\sigma(W_1\vec{x} + b_1) + b_2)$
  - Compute loss function (least squares):  
$$L(y, \hat{y}) = \sum_i (y - \hat{y})^2$$
  - Compute gradient of loss function wrt weights  
$$\frac{\partial L(y, \hat{y})}{\partial W}$$
  - Back propagate :
    - Optimize using gradient descent (like BFGS!)

Caveats about optimization in multiple dimensions hold here!

Adapted from

<https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6>

# Deep Learning and Neural Networks

- Next step: instead of teaching, let the neural network learn on its own
  - Example 1: Convolutional Neural Network
    - Extract features using convolution, pass features to neural network
    - Fixed-size inputs: Suitable for images
  - Example 2: Recurrent Neural Network
    - Instead of “feed forward” like last time, has possible recurrent links
    - Any size inputs: Suitable for text / speech / handwriting recognition
    - Can also be Recursive (don’t get confused!)

[https://en.wikipedia.org/wiki/Deep\\_learning](https://en.wikipedia.org/wiki/Deep_learning)

[https://en.wikipedia.org/wiki/Recurrent\\_neural\\_network](https://en.wikipedia.org/wiki/Recurrent_neural_network)

[https://en.wikipedia.org/wiki/Recursive\\_neural\\_network](https://en.wikipedia.org/wiki/Recursive_neural_network)

[https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)

<https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/>

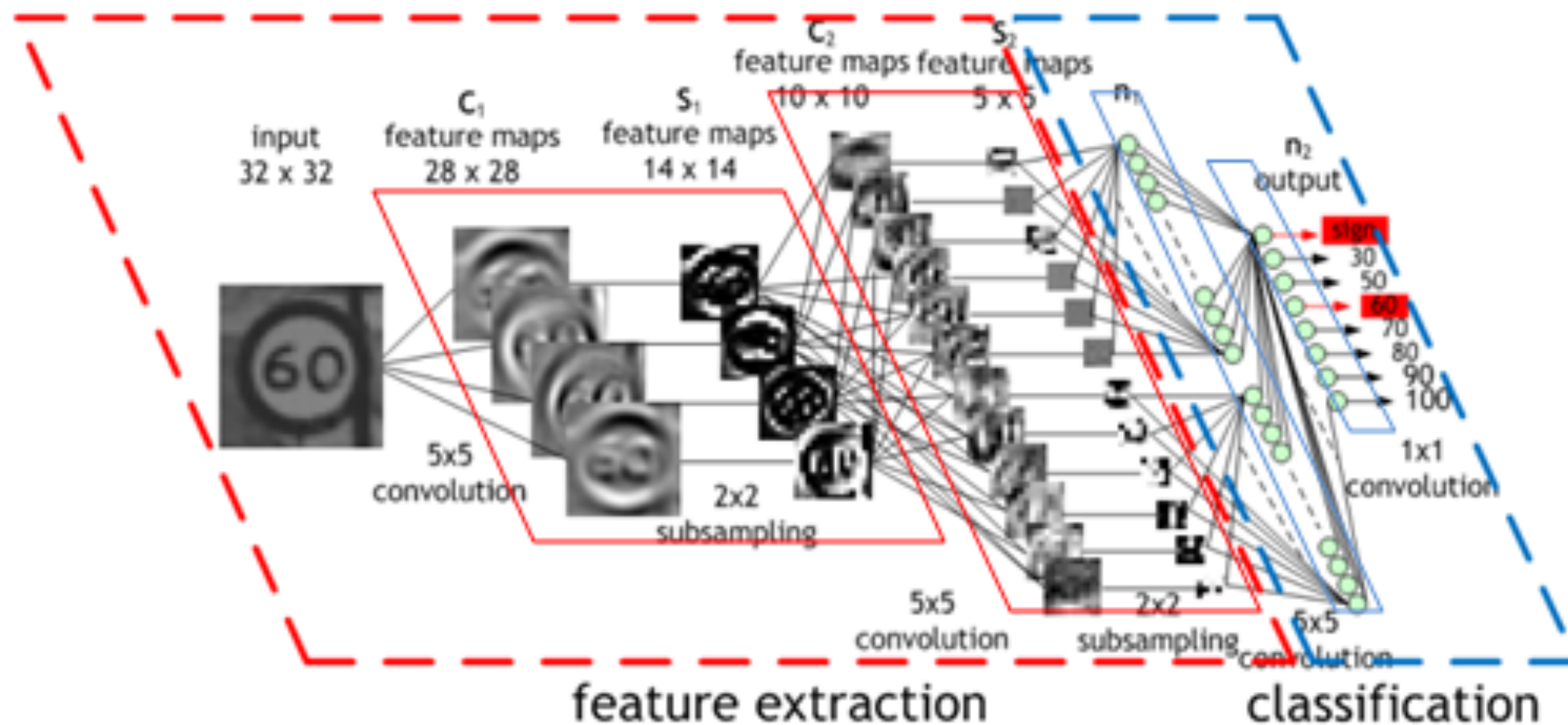
# Deep Learning and Neural Networks

- Options to train:
- Supervised
  - Give inputs, tell it what the output is
- Unsupervised
  - Give inputs, tell it to optimize a function



# Convolutional Neural Networks

- Preprocessing to perform feature extraction
  - Convolution or otherwise
- Classification
  - Standard neural network

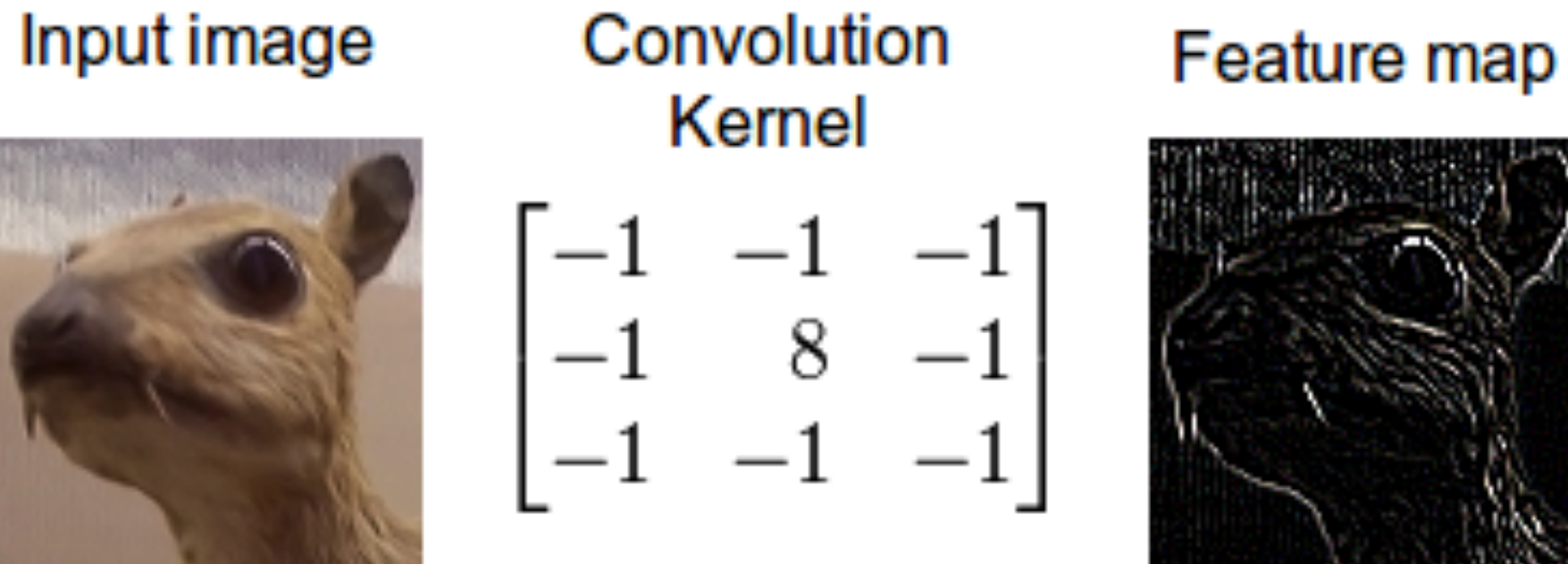




# Convolutional Neural Networks

- Does “preprocessing” of the image by convoluting with kernels:

2-d Fourier transform!



# Convolutional Neural Networks

- Animation of convolution:

- Kernel:

1	0	1
0	1	0
1	0	1

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

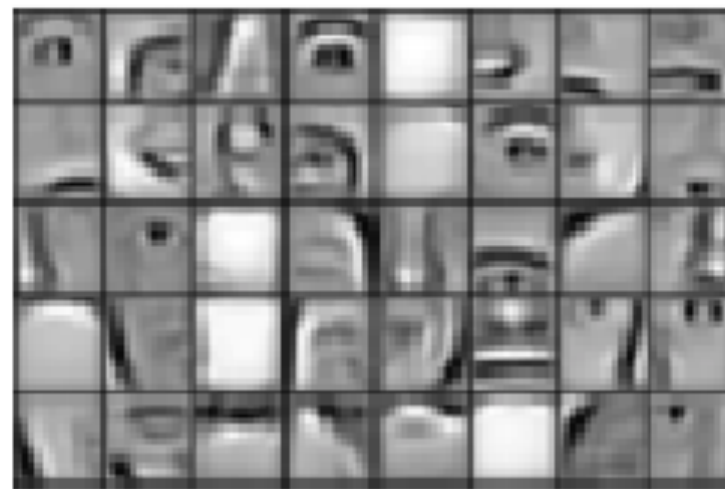
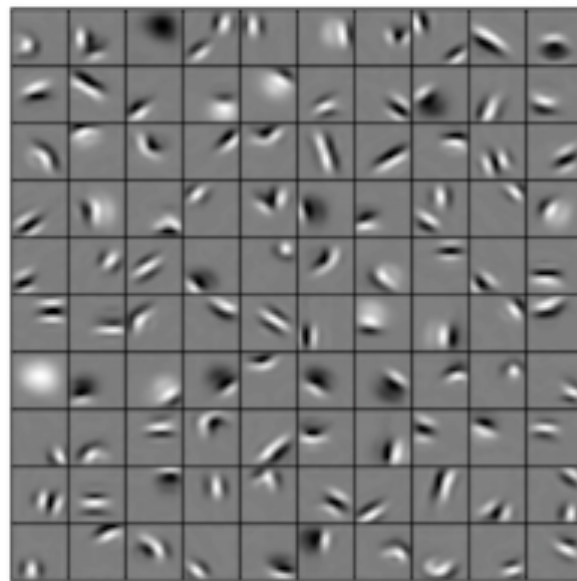
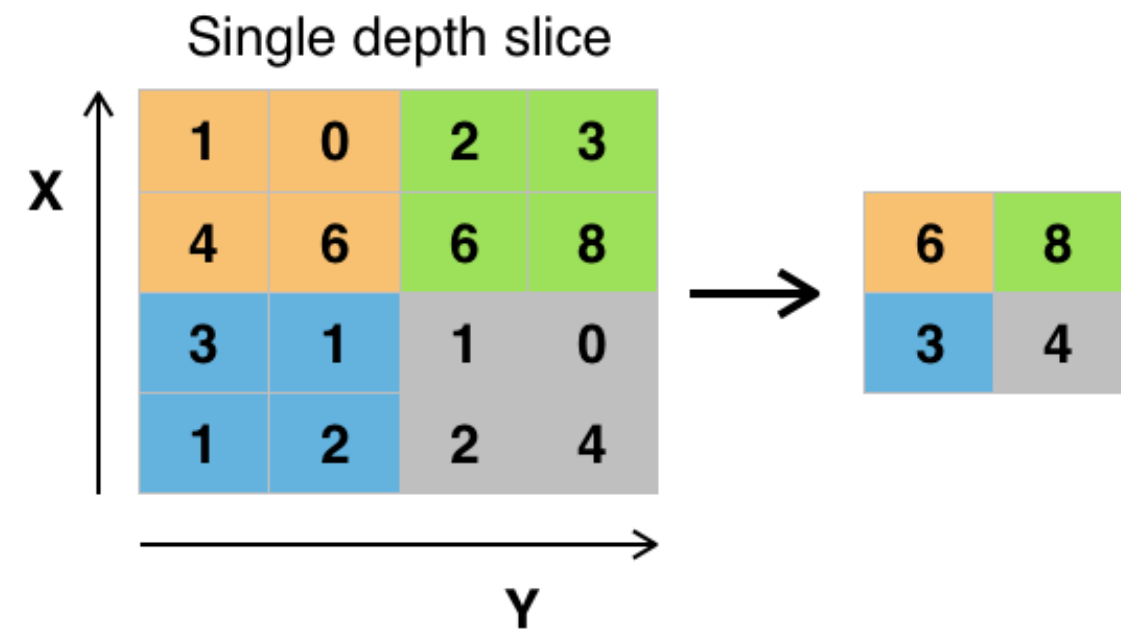
Image

4		

Convolved  
Feature

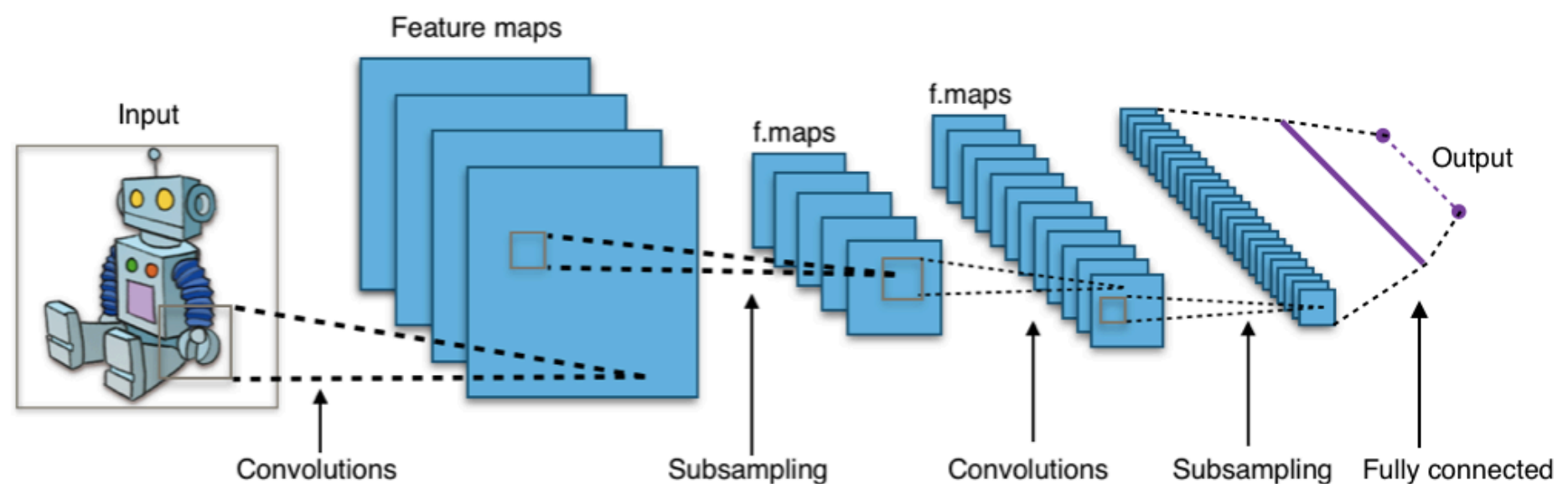
# Convolutional Neural Networks

- Pooling layer : used to obtain very big features
  - Downsampled to extract gross features
  - Is it “pointy”?
  - Is it “round”?
  - Does it have a face?



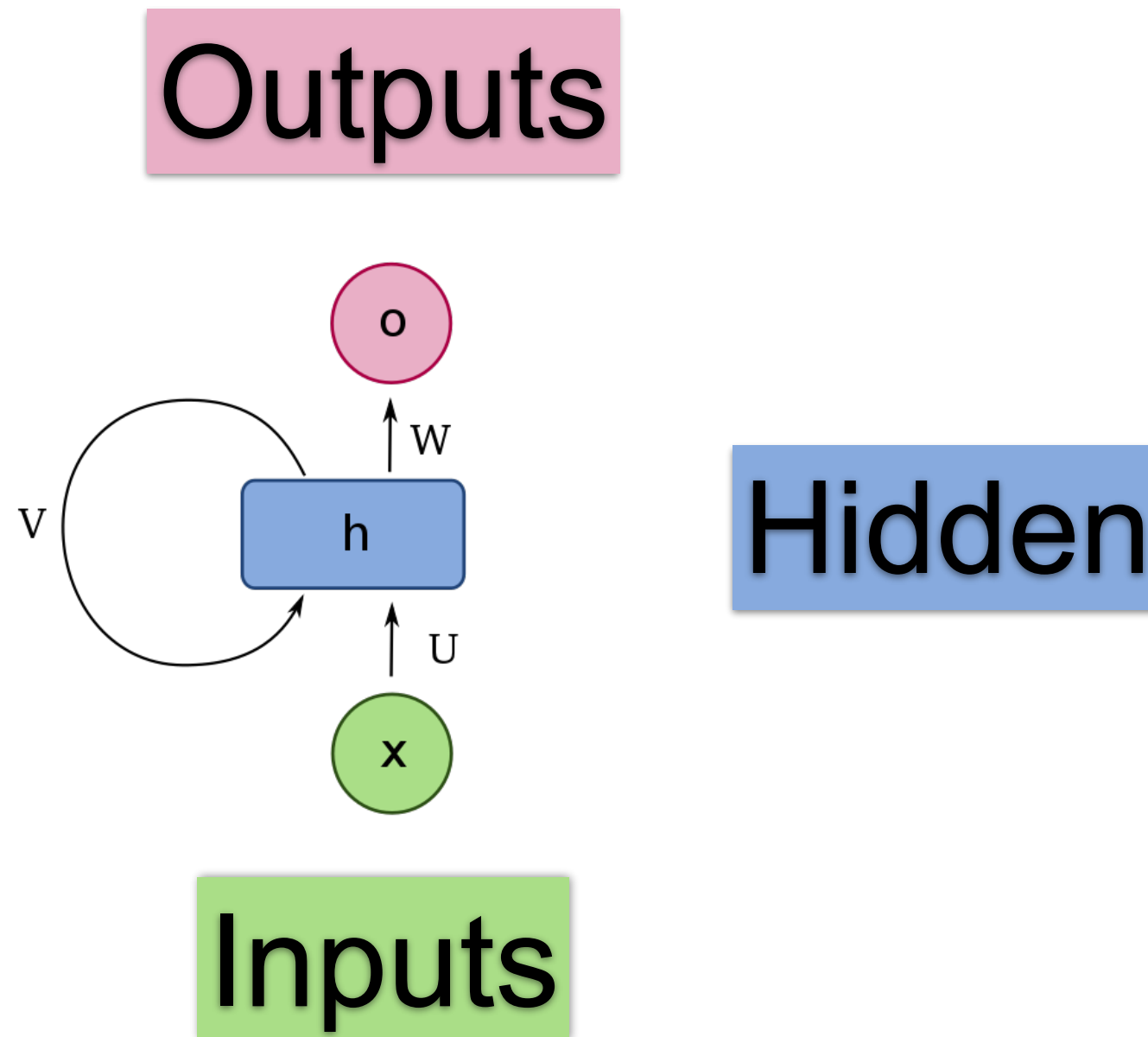
# Convolutional Neural Networks

- Putting it together:
  - Convolutional layer
    - Discern features
  - Pooling layer
    - Extract global features
  - Activation function (“Rectified Linear Units”)
    - Standard NN
  - Fully connected layer
    - Standard NN
  - Loss layer
    - How well did it do?



# Recurrent Neural Networks

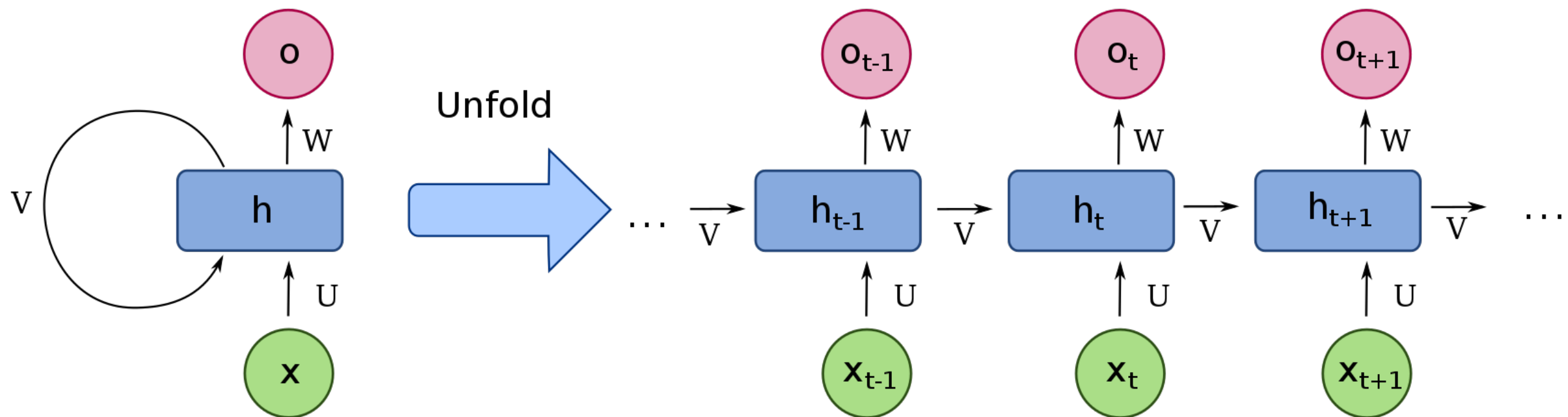
Dynamic directed graph to process **sequences** of inputs



# Recurrent Neural Networks

Dynamic directed graph to process **sequences** of inputs

“Dynamic” in the sense that it changes over time

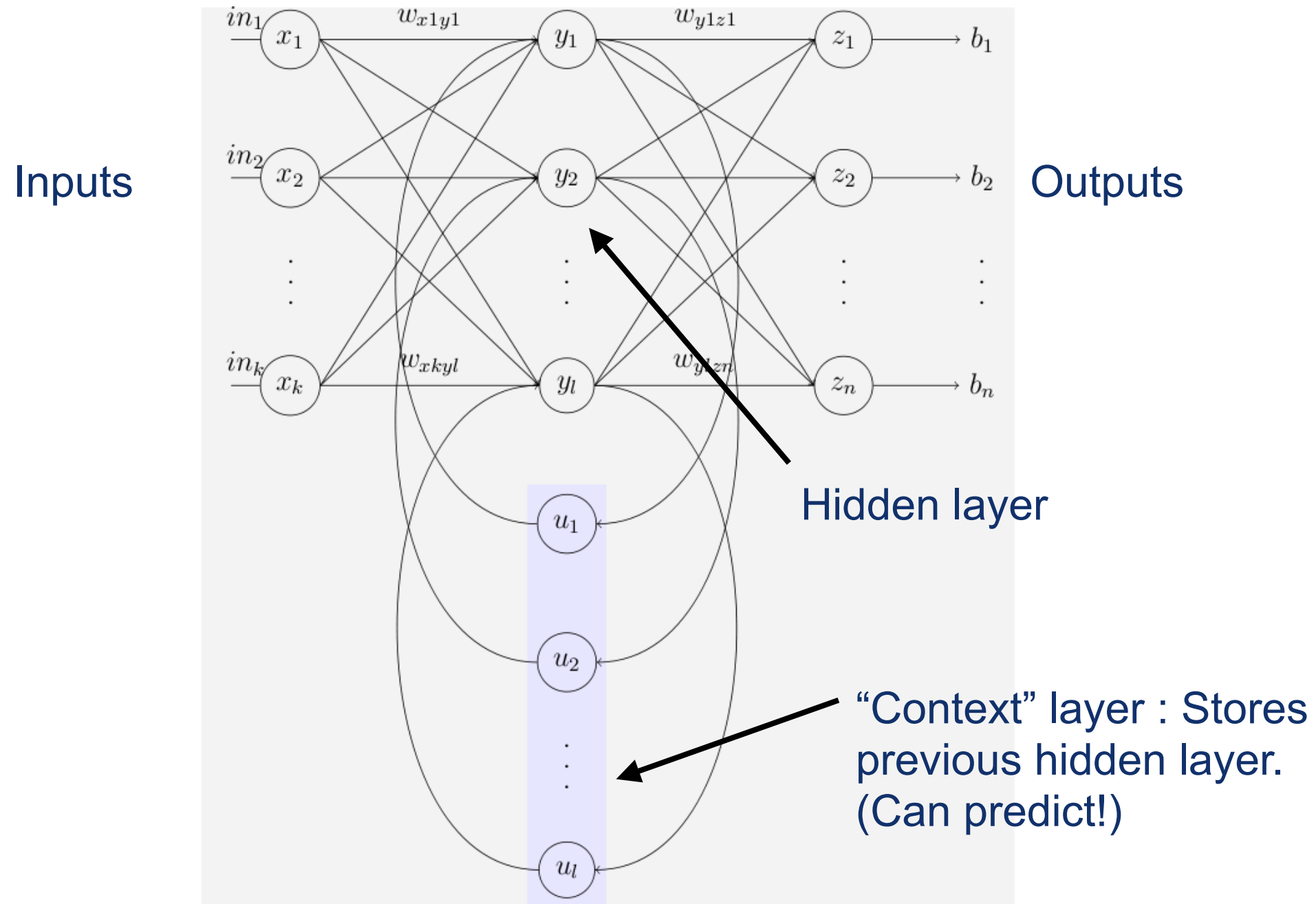


# Recurrent Neural Networks

- “Recurrent” : Can “save state” of the inputs
  - (CNNs do not)
- Has use in predicting the next value in a sequence (speech, text recognition)
- Can also see how similar two sequences are

# Recurrent Neural Networks

- Example: Elman network







# Highly Lorentz-Boosted Jets

$p^\mu = (E, \vec{p})$

$p_1^\mu = (E_1, \vec{p}_1)$

$p_2^\mu = (E_2, \vec{p}_2)$

$\theta_{12}$

$$p^\mu p_\mu = (p_1 + p_2)^\mu (p_1 + p_2)_\mu$$
$$m^2 = (E_1 + E_2)^2 - (\vec{p}_1 + \vec{p}_2) \cdot (\vec{p}_1 + \vec{p}_2)$$
$$m^2 \approx 2E_1 E_2 (1 - \cos(\theta_{12}))$$

Assume  $E_1 = E_2 = E/2$

$$m^2 \approx E^2 (1 - \cos(\theta_{12}))$$

$$\cos(\theta_{12}) \approx 1 - \frac{m^2}{E^2} \approx 1 - \frac{1}{\gamma^2}$$

$$\theta_{12} \approx \frac{2m}{E} = \frac{2}{\gamma}$$

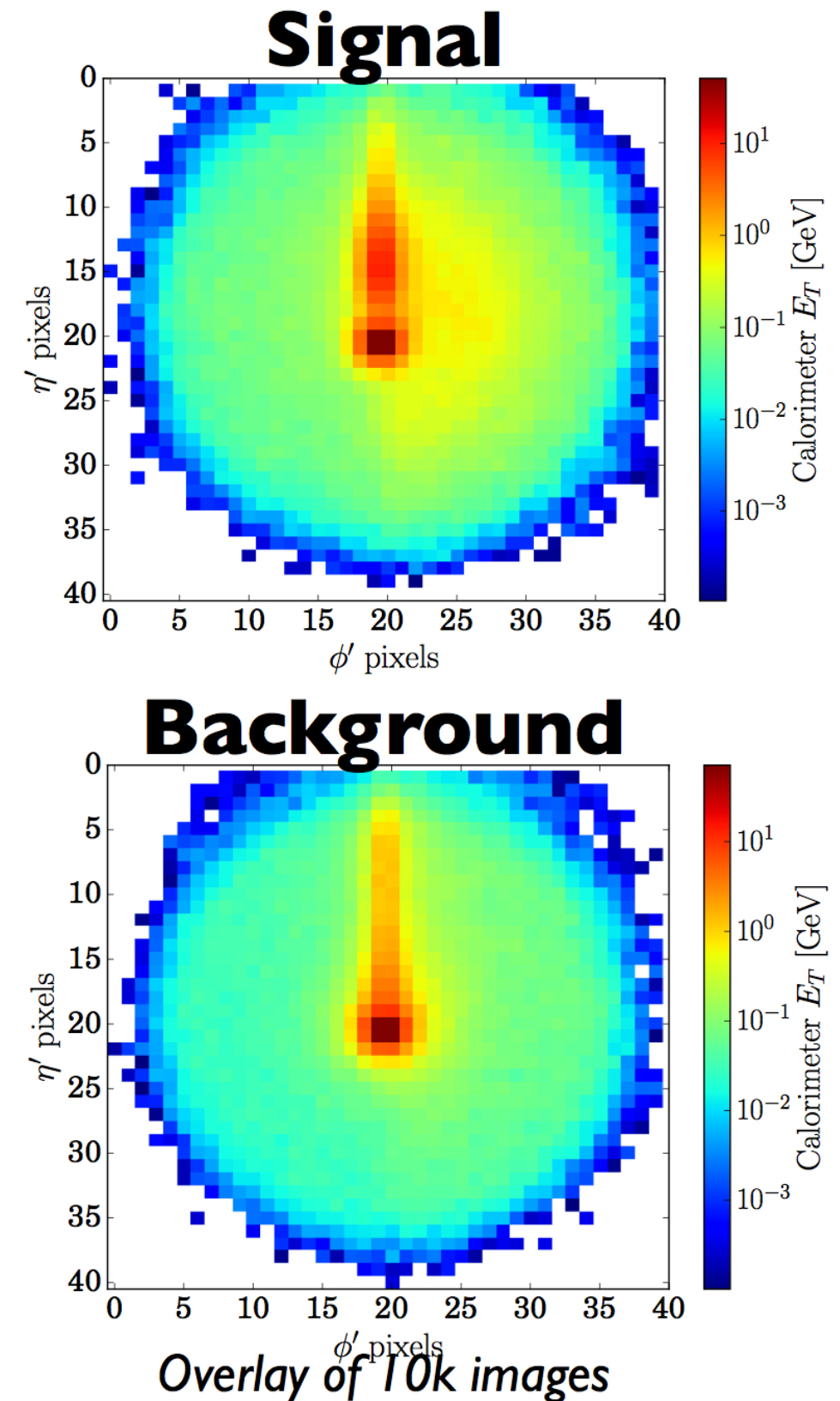
# Highly Lorentz-Boosted Jets

- Plot energy of particles as a 2d projection
- Take average, put at center
- Take “next blob”, put at top center
- Look at sum of many images
- Use image processing techniques!

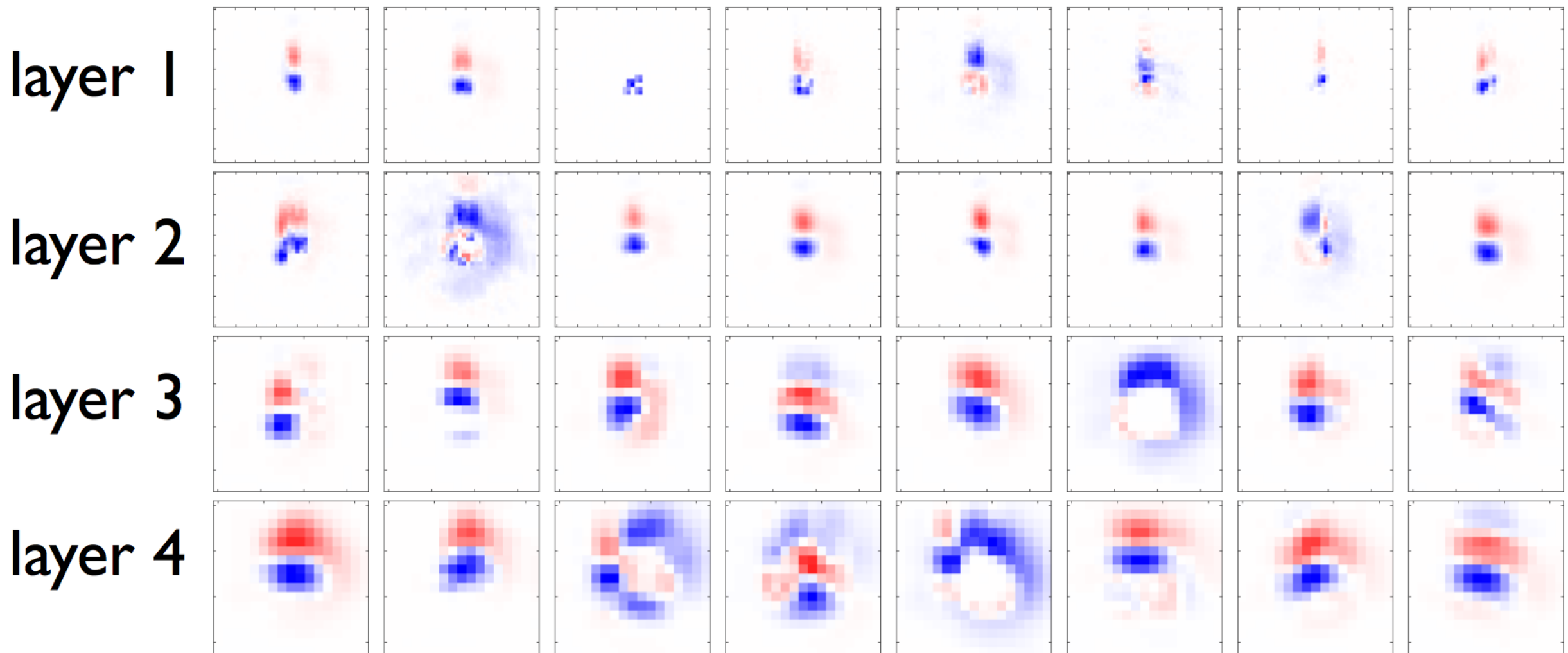
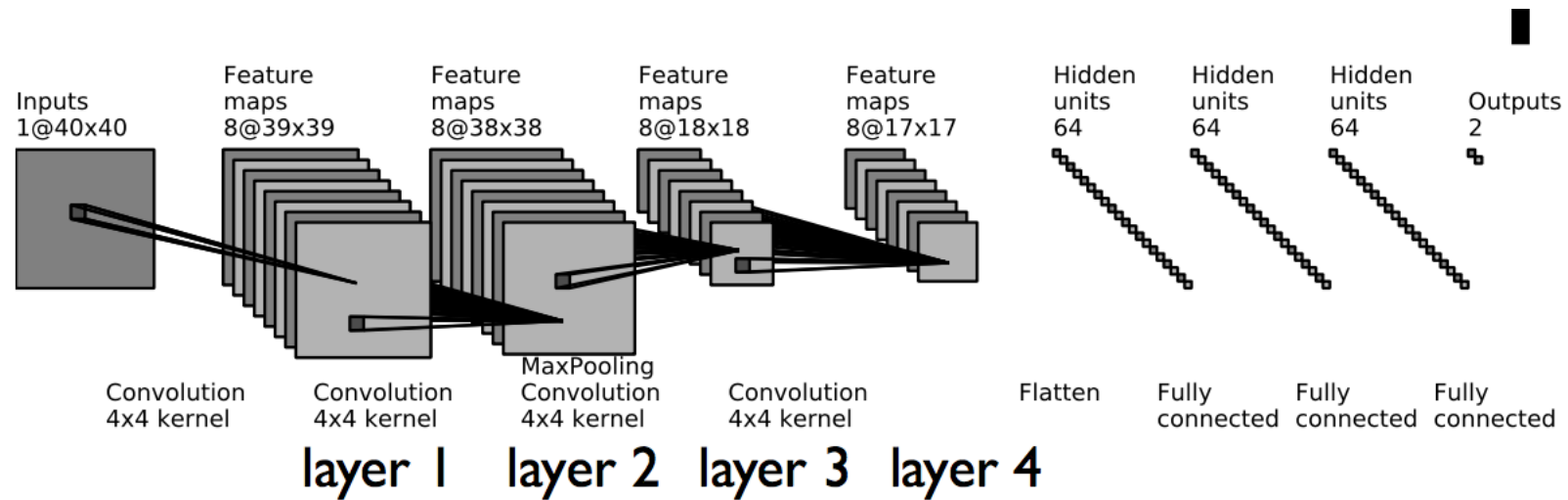
arXiv:1407.5675

JHEP 1607 069

JHEP 05 (2017) 006



# Highly Lorentz-Boosted Jets



# Software

- Lots of Deep Network software out there
- Popular ones:
  - Theano
  - TensorFlow
  - Scikit-learn
  - etc
- Theano and tensorflow are both installed in vidia, so we can work through an example there:
  - <https://www.tensorflow.org/tutorials/>

# The Inverse Problem

- The inverse problem is, to first order, “simply” inverting functions

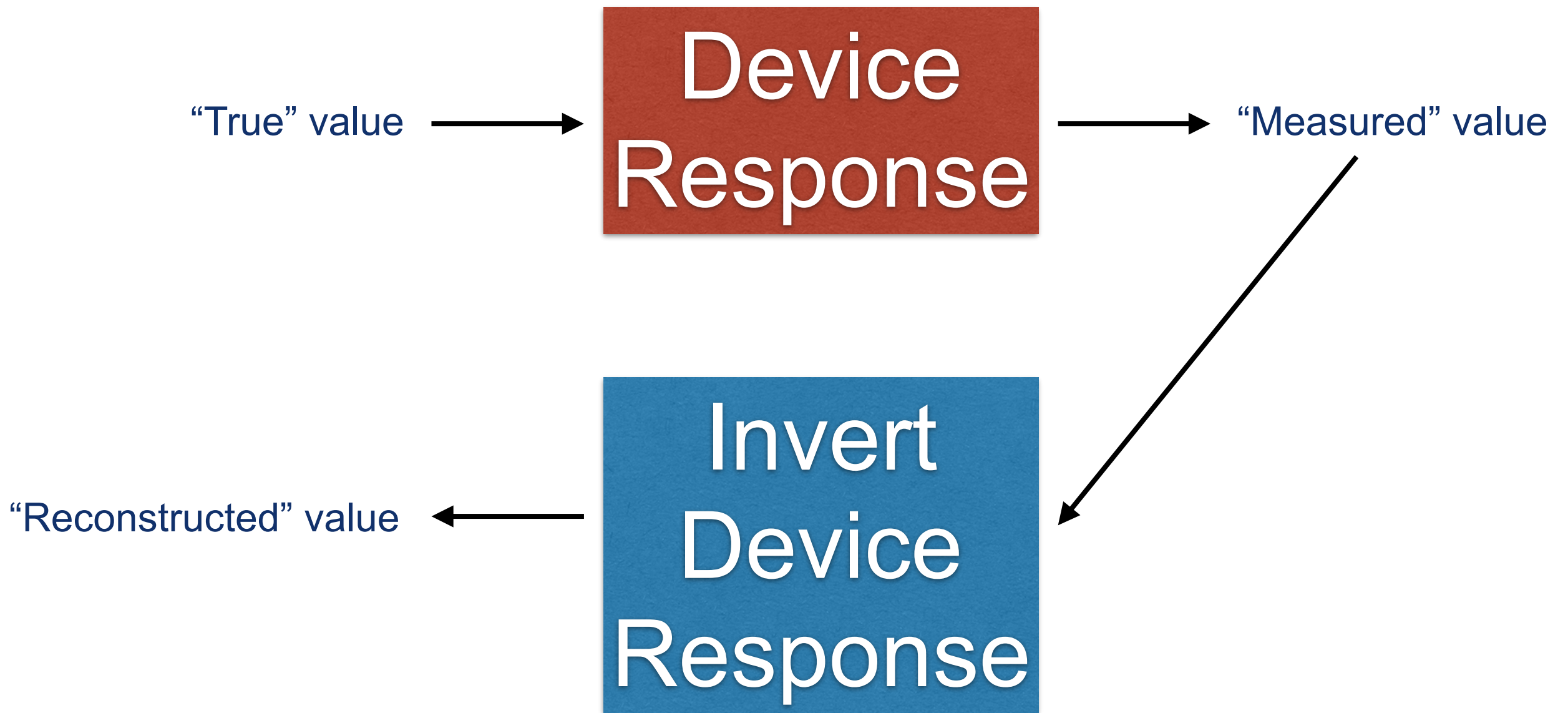
- Linear functions: trivial:  $f(x) = mx + b$ 
  - But! If  $m = 0$ , ?????  $f^{-1}(x) = \frac{1}{m}x - \frac{b}{m}$

- Everything else: not even guaranteed to exist!
  - Can be many-to-one mapping
  - Can be non-invertible
- Becomes even trickier in multiple dimensions!

# The Inverse Problem

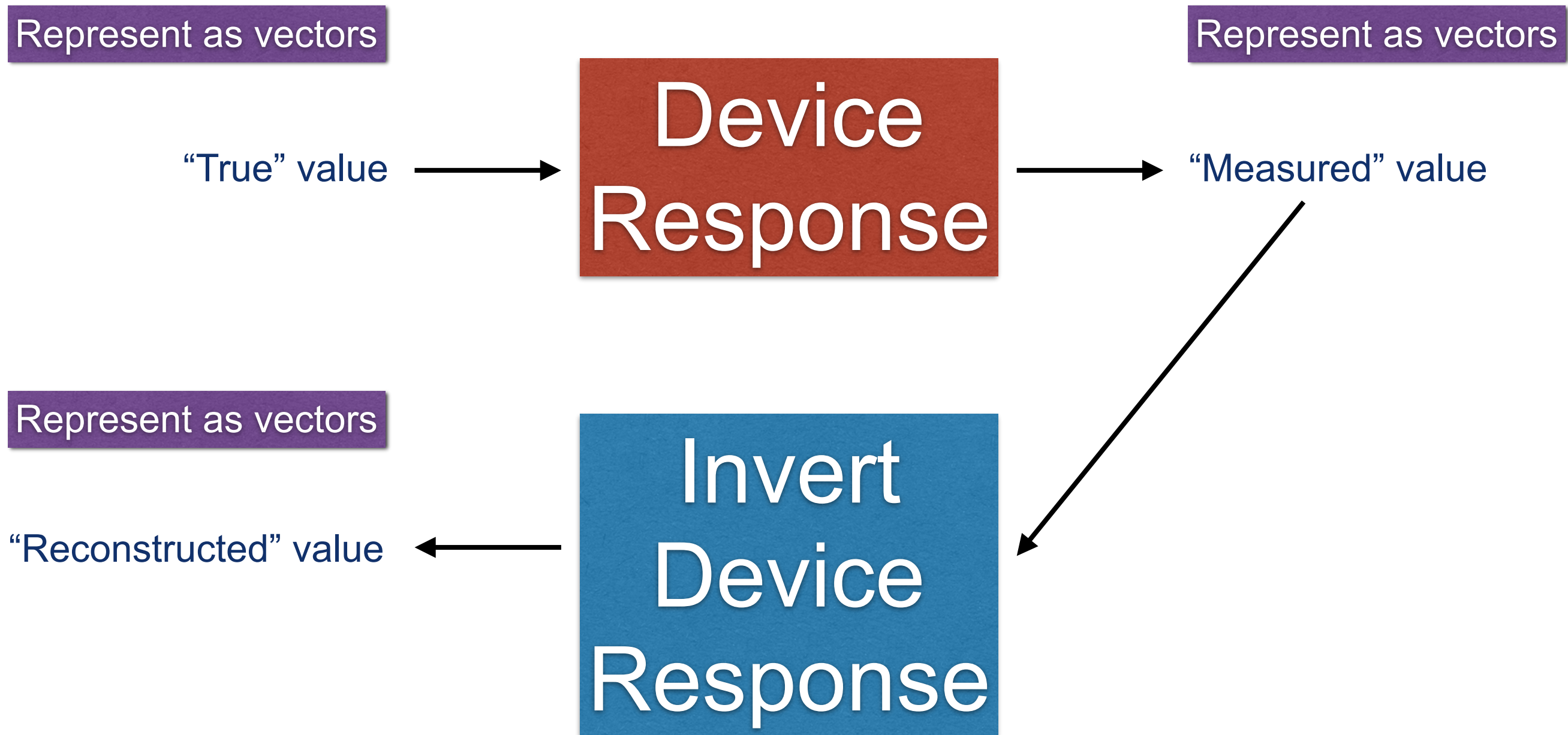


# The Inverse Problem

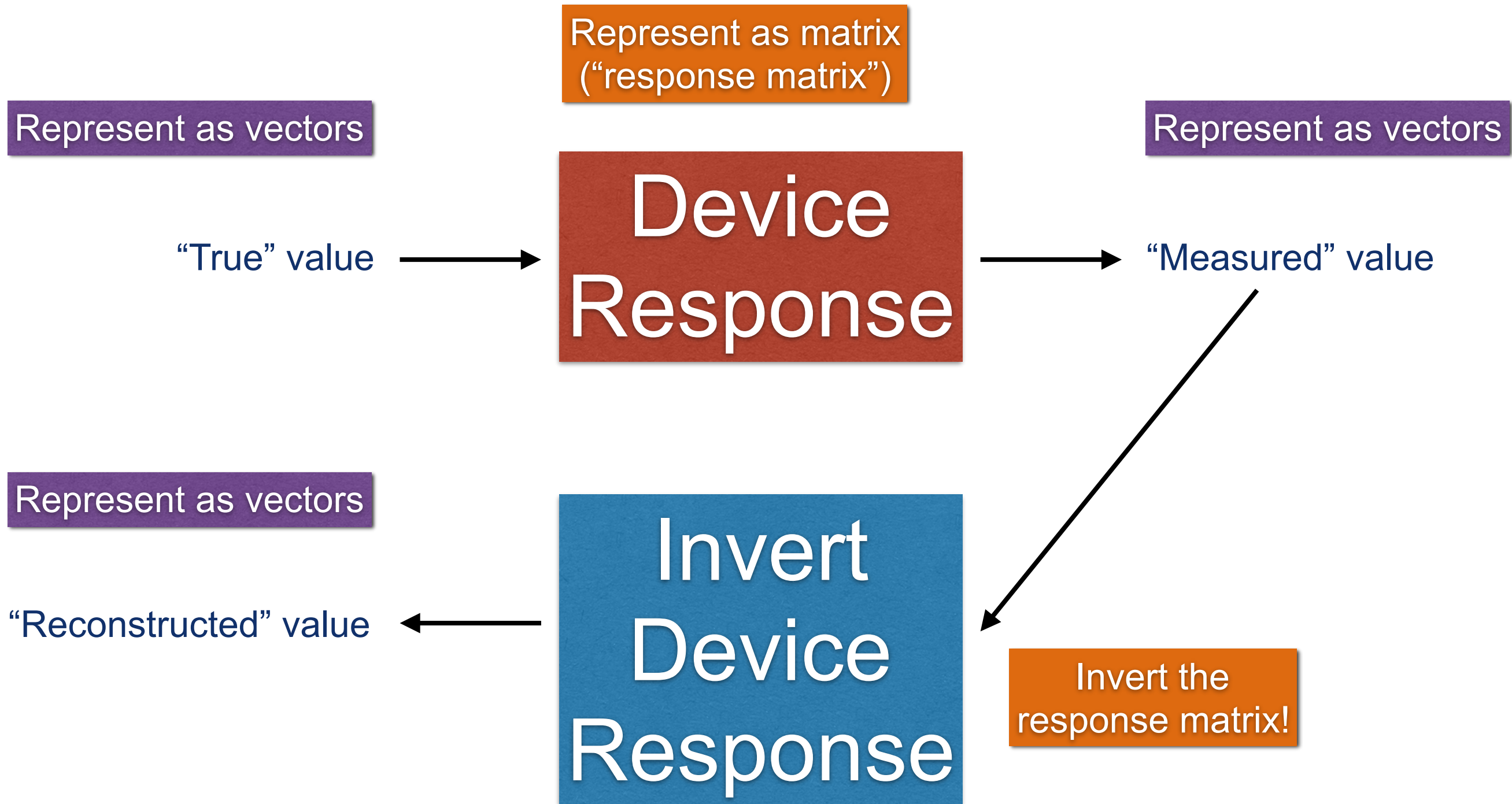




# The Inverse Problem

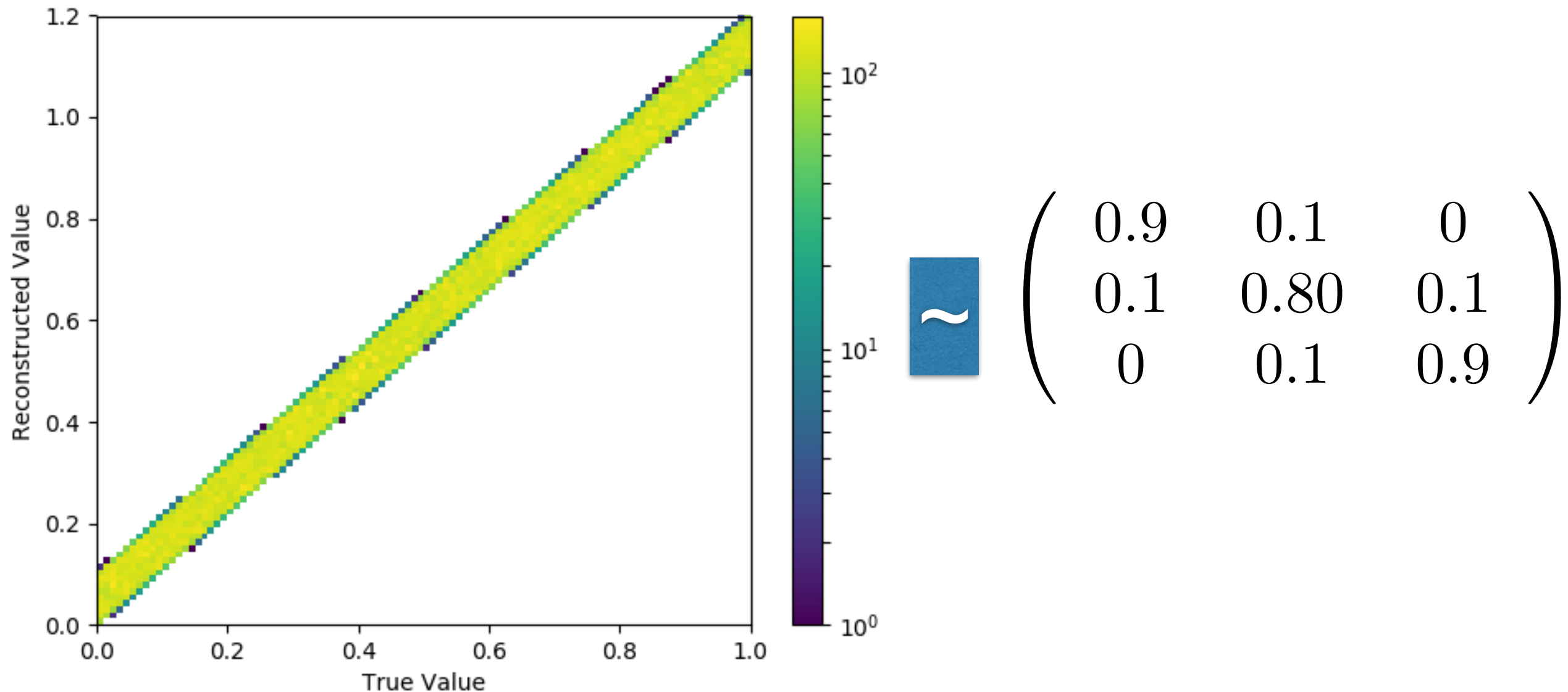


# The Inverse Problem



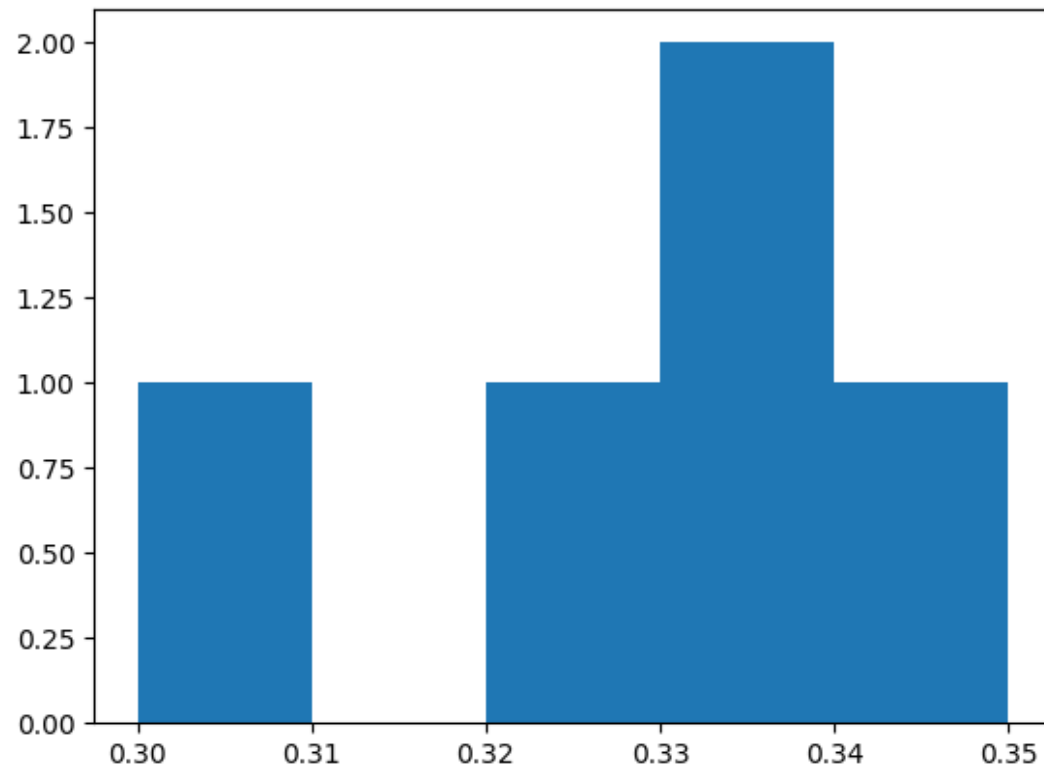
# The Inverse Problem

- But! Need one of these for EVERY possible input (0, 0.1, 0.1... 1.0)
- Represent as a matrix:



# The Inverse Problem

- Represent our histogram as a vector



$$\begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \\ 0 \\ 1 \\ 2 \\ 1 \\ \dots \\ 0 \end{pmatrix}$$

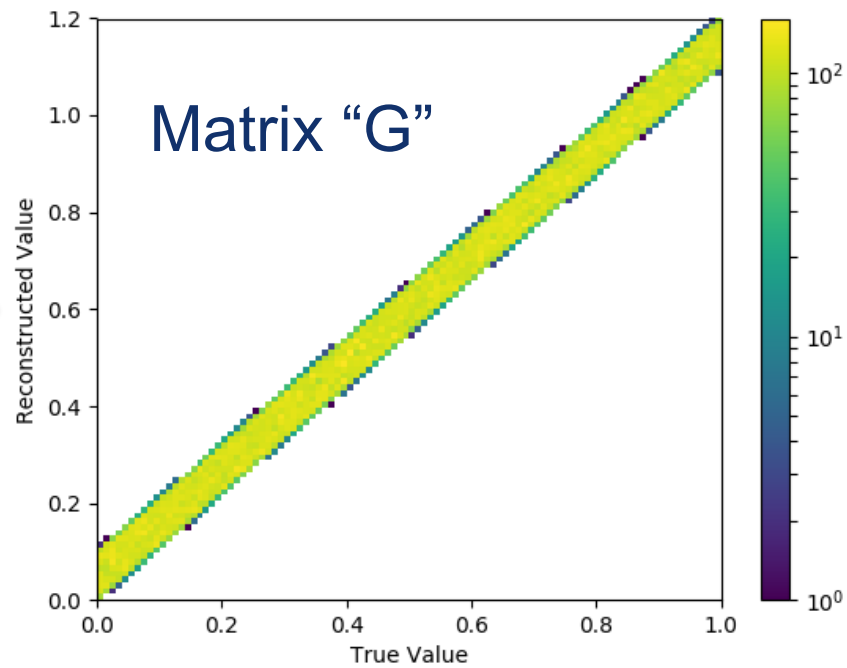
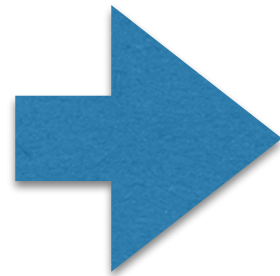
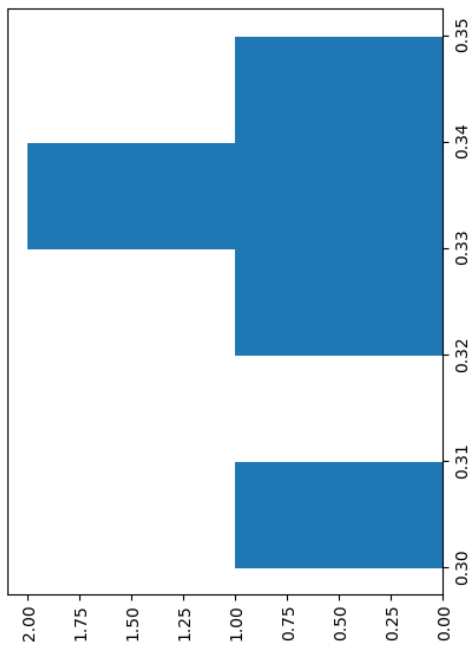
Index=0: 0.0

Index=30: 0.3

# The Inverse Problem

$$d = Gm$$

Vector "d"



Invert G to get vector "m"

# The Inverse Problem

- You're aware of the first case: Linear inverses (i.e. inverting matrices)
- There are generalizations
- Reminder:
  - To solve  $\vec{y} = A\vec{x} + \vec{b}$  :
  - Invert the matrix:
$$\vec{x} = A^{-1} (\vec{y} - \vec{b})$$
  - Gauss-Jordan elimination, other techniques we did last semester

# The Inverse Problem

- What if the matrix is not invertible?
  - Can still get information, but not perfectly determined
  - Often sufficient to have partial information
- Examine the over-constrained case:
- Suppose we have a matrix  $G$  (the “observation matrix”) with data “ $d$ ” and true value “ $m$ ”.
- Want to minimize the difference between the prediction ( $Gm$ ) and the data:  
$$\phi = |d - Gm|^2$$
- Where  $|a|^2 = (a^T a)$
- Want to find the place where the difference is minimized!

# The Inverse Problem

- Difference minimized when

$$\nabla_m \phi = 0$$

- That is, the gradient wrt the “m” components is zero
- Using chain rules for matrix functions:

$$\nabla_m \phi = 2 (G^T G m - G^T d) = 0$$

- Needs to be satisfied for all of the components, so require each to vanish:

$$G^T G m = G^T d$$

- So we solve for m:

$$m = (G^T G)^{-1} G^T d$$

Least squares distance  
from last semester!

[https://en.wikipedia.org/wiki/Inverse\\_problem](https://en.wikipedia.org/wiki/Inverse_problem)

<https://atmos.washington.edu/~dennis/MatrixCalculus.pdf>



# The Inverse Problem

- So the inverse problem is similar to least squares
- Don't get the "full inverse" but do get the closest to it
- For intuition, remember our "design matrix" from last semester:

$$A_{ij} = \frac{Y_j(x_i)}{\sigma_i}$$

$$\mathbf{A} = \begin{bmatrix} Y_1(x_1)/\sigma_1 & Y_2(x_1)/\sigma_1 & \dots \\ Y_1(x_2)/\sigma_2 & Y_2(x_2)/\sigma_2 & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

$$\vec{a} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \vec{b}$$

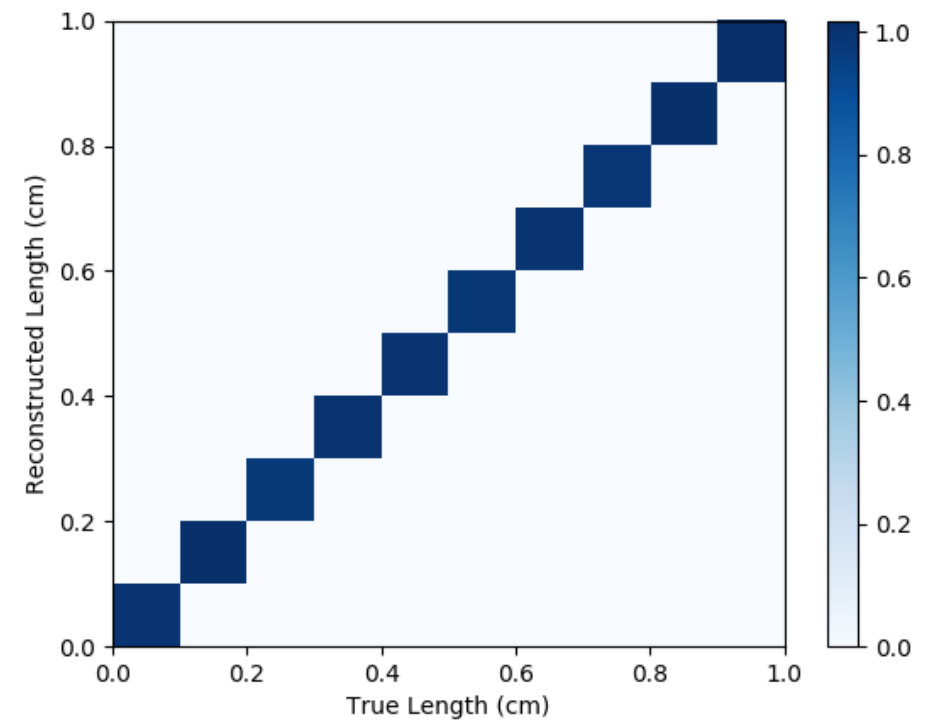
- Then for polynomial fits  $A_{ij} = a_j x_i^j / \sigma_i$

# The Inverse Problem

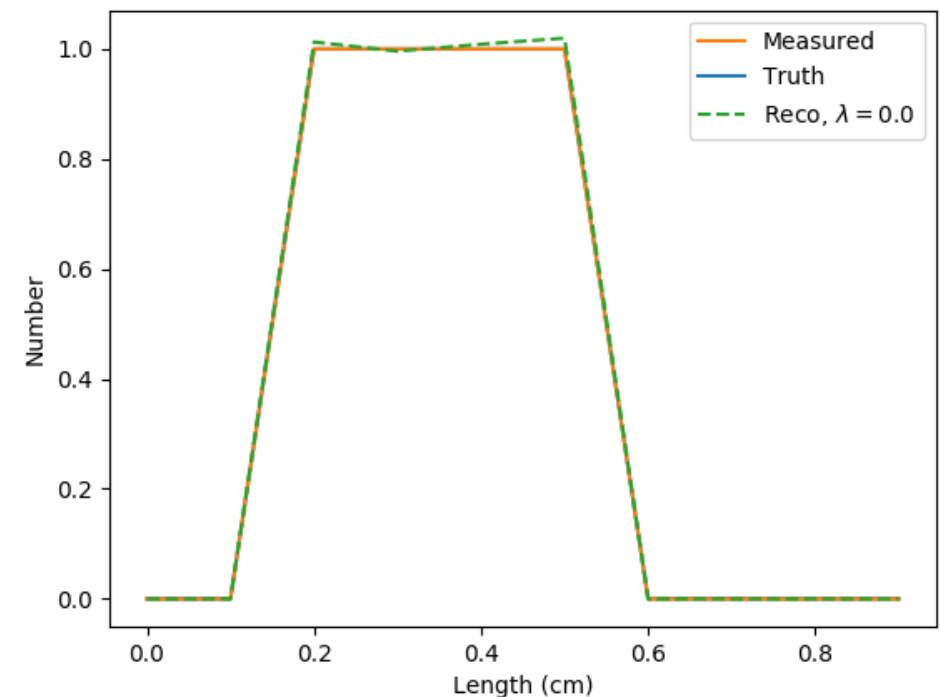
- Try to invert the response of some experimental device or detector to get “true” values

- Example 1:

- Device to measure length is perfect.
  - Response created from 100000 “pseudo-experiments”:



- You measure 4 objects, and obtain:
  - 0.2 cm, 0.3 cm, 0.4 cm, 0.5 cm
  - Measured vs true histogram:

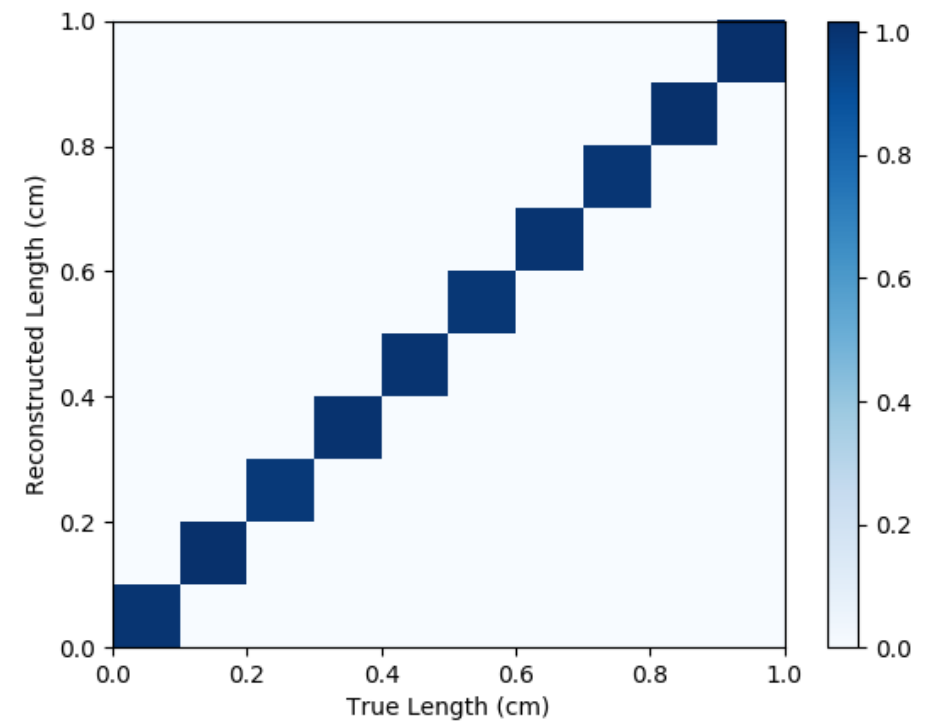


# The Inverse Problem

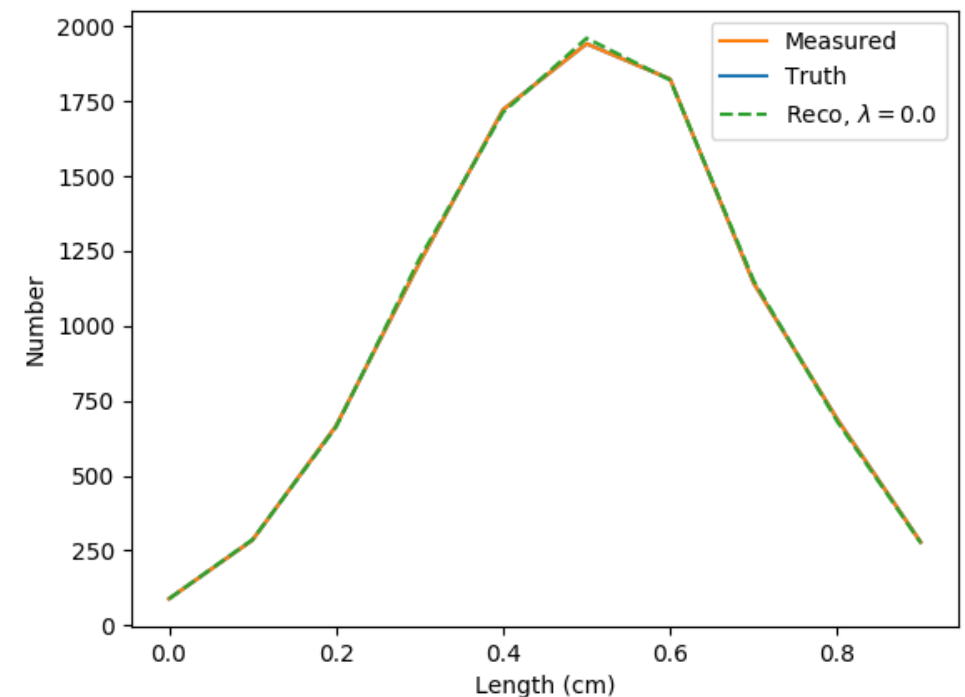
- Try to invert the response of some experimental device or detector to get “true” values

## – Example 2:

- Device to measure length is perfect.
  - Response created from 100000 “pseudo-experiments”:



- You measure 10000 objects, and obtain:
  - Gaussian with width = 0.2, mean = 0.5
  - Measured histogram:

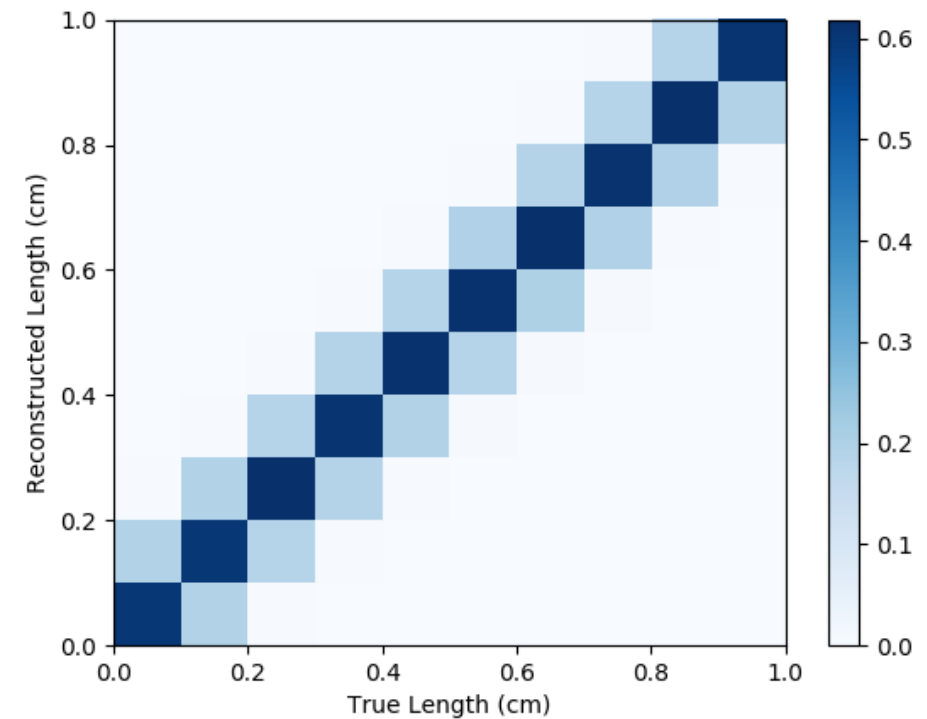


# The Inverse Problem

- Try to invert the response of some experimental device or detector to get “true” values

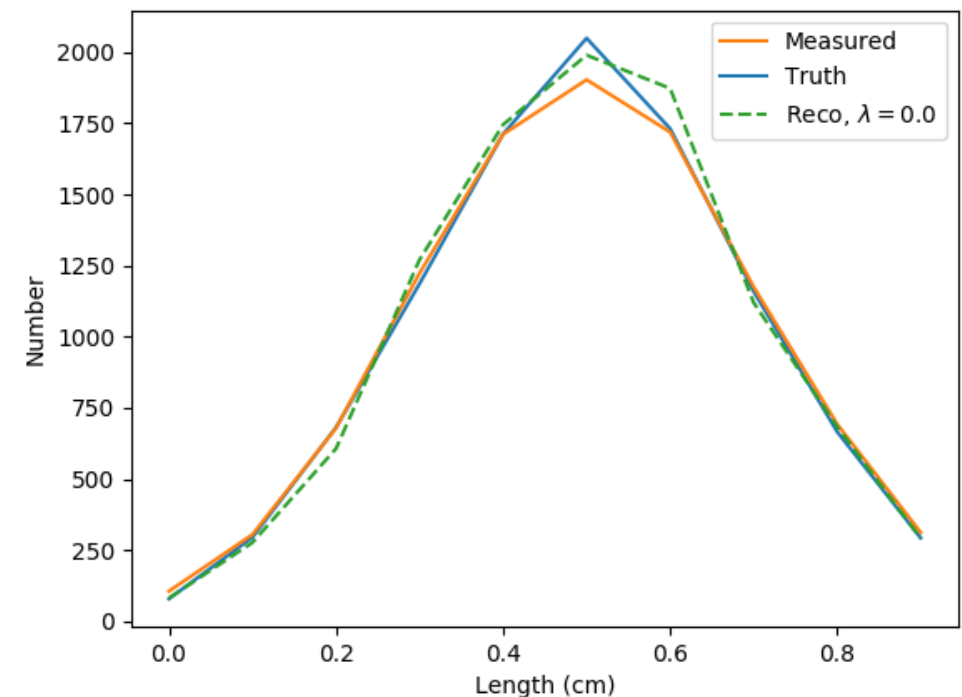
- Example 3:

- Device to measure length has resolution of 0.05 cm
  - Response created from 100000 “pseudo-experiments”:



- You measure 10000 objects, and obtain:

- Gaussian with width = 0.2, mean = 0.5
- Measured histogram:

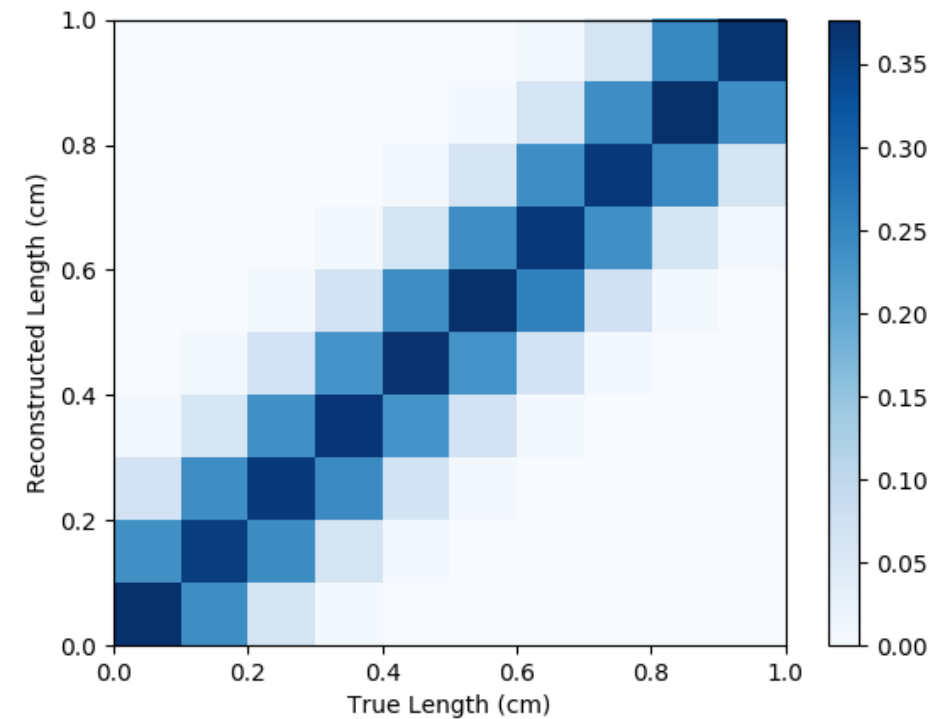


# The Inverse Problem

- Try to invert the response of some experimental device or detector to get “true” values

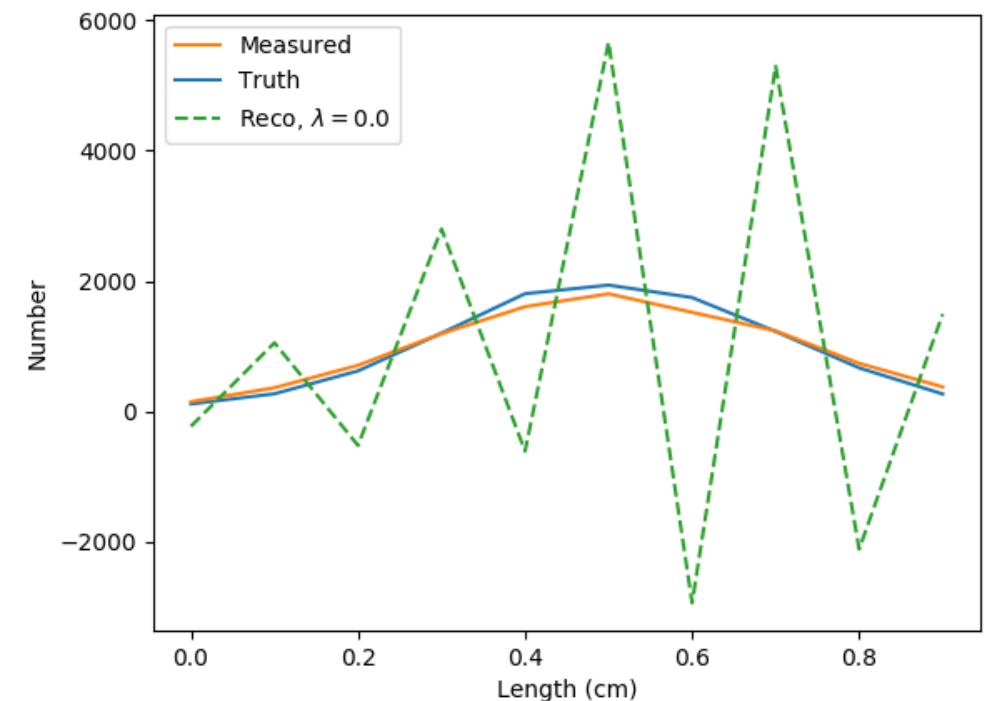
## – Example 4:

- Device to measure length has resolution of 0.1 cm
  - Response created from 100000 “pseudo-experiments”:



- You measure 10000 objects, and obtain:

- Gaussian with width = 0.2, mean = 0.5
- Measured histogram:



# The Inverse Problem

- What is going on???
- Statistical uncertainties bin-to-bin are greatly amplified by inversion
  - Need to damp those out... “regularization” (from Tikhonov)

[https://en.wikipedia.org/wiki/Tikhonov\\_regularization](https://en.wikipedia.org/wiki/Tikhonov_regularization)

# The Inverse Problem

- How to formally do this?
- Ordinary formula for the inversion is:

$$m = (G^T G)^{-1} G^T d$$

- Introduce a penalty term (here, a Lagrange multiplier) to solve:

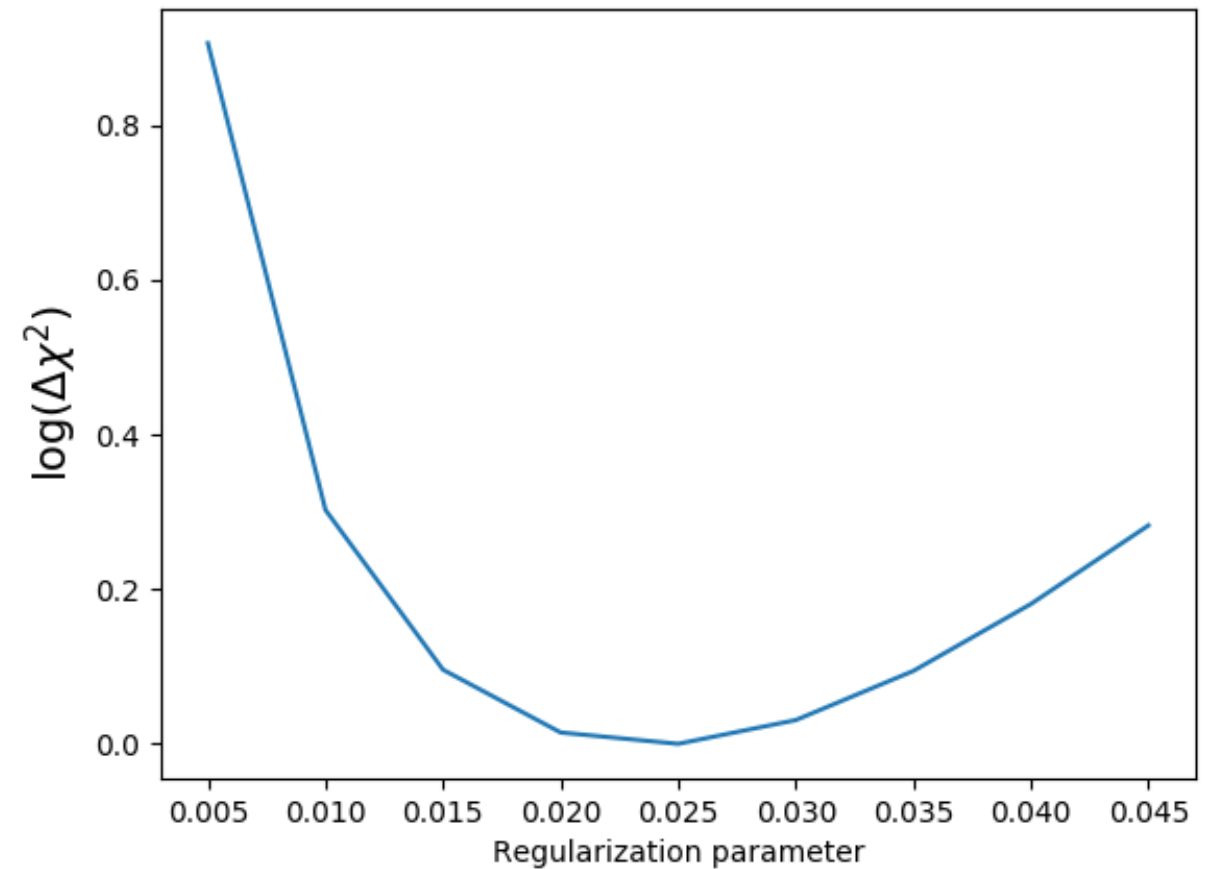
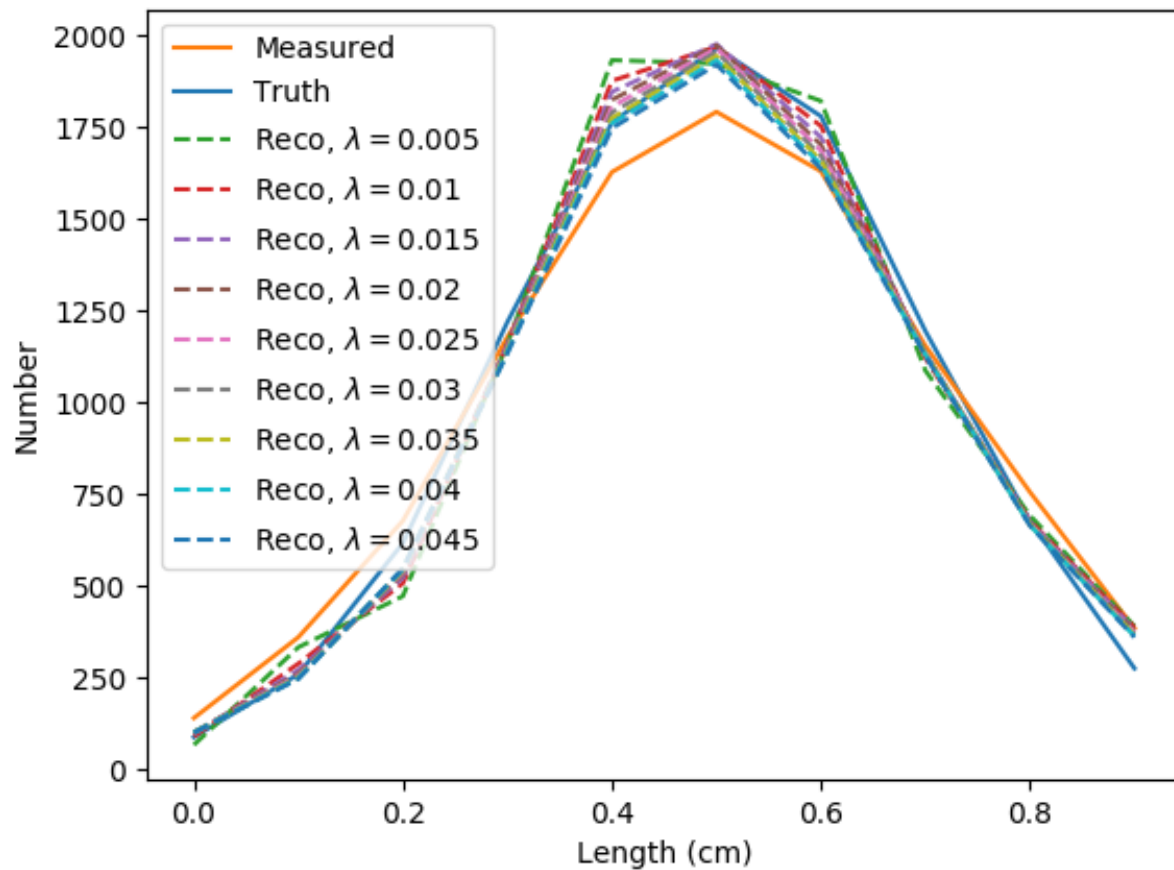
$$m = (G^T G + \lambda I)^{-1} G^T d$$

- Can then vary as a function of lambda to pick when the regularization is complete

[https://www.researchgate.net/publication/274138835\\_NumPy\\_SciPy\\_Recipes\\_for\\_Data\\_Science-Regularized\\_Least\\_Squares\\_Optimization](https://www.researchgate.net/publication/274138835_NumPy_SciPy_Recipes_for_Data_Science-Regularized_Least_Squares_Optimization)

# The Inverse Problem

- Example 4: vary regularization parameter for our example:

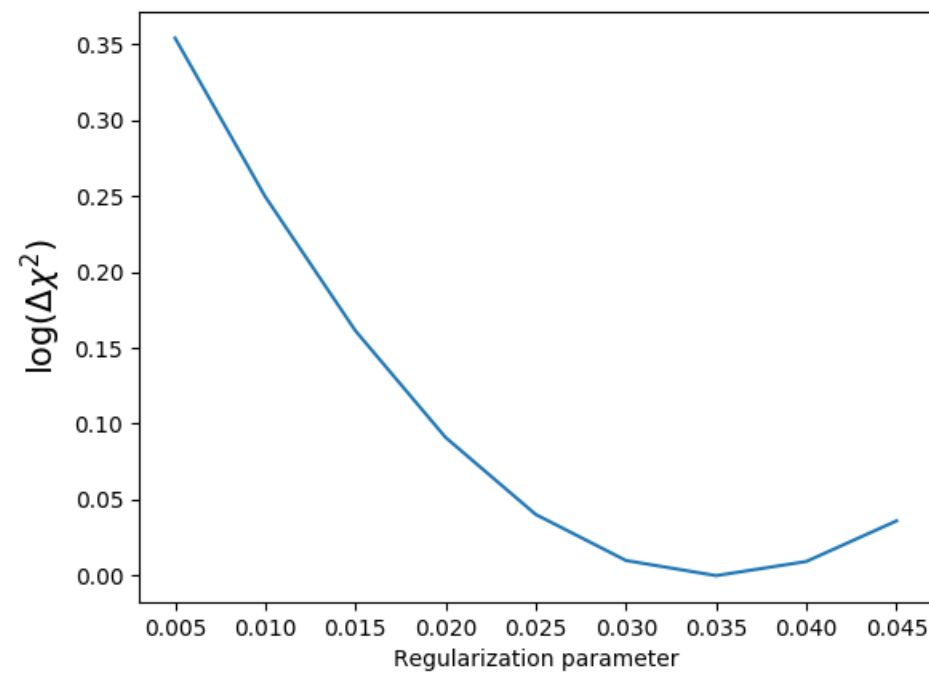
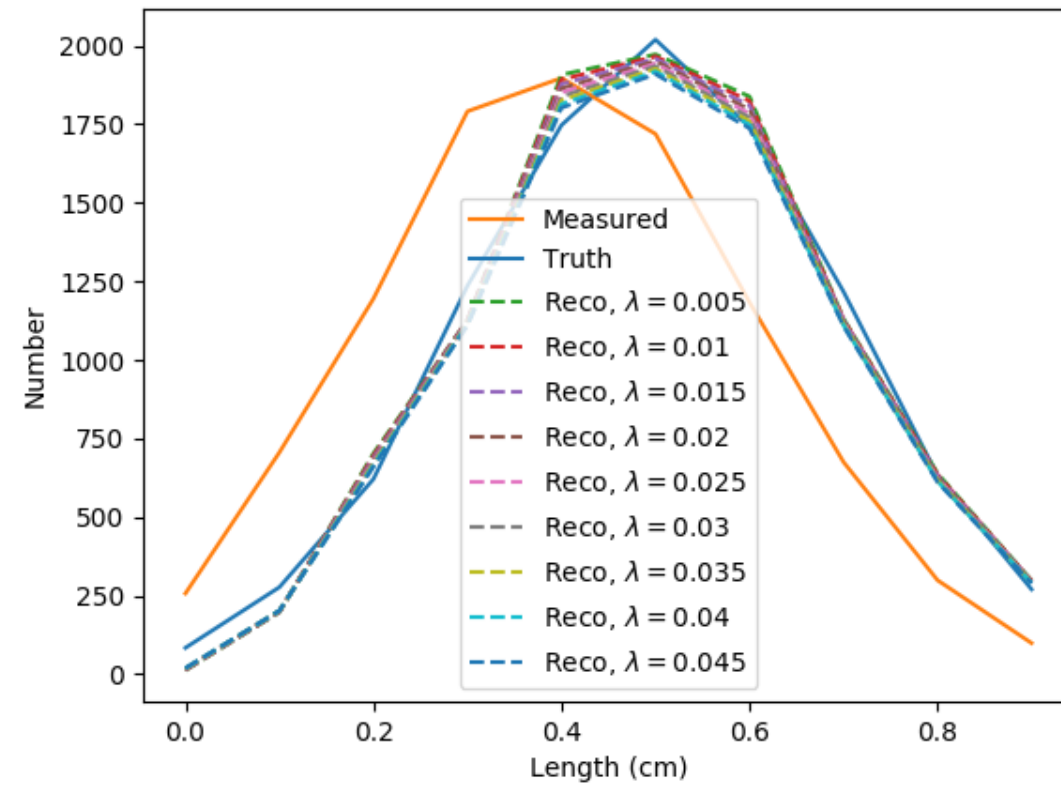
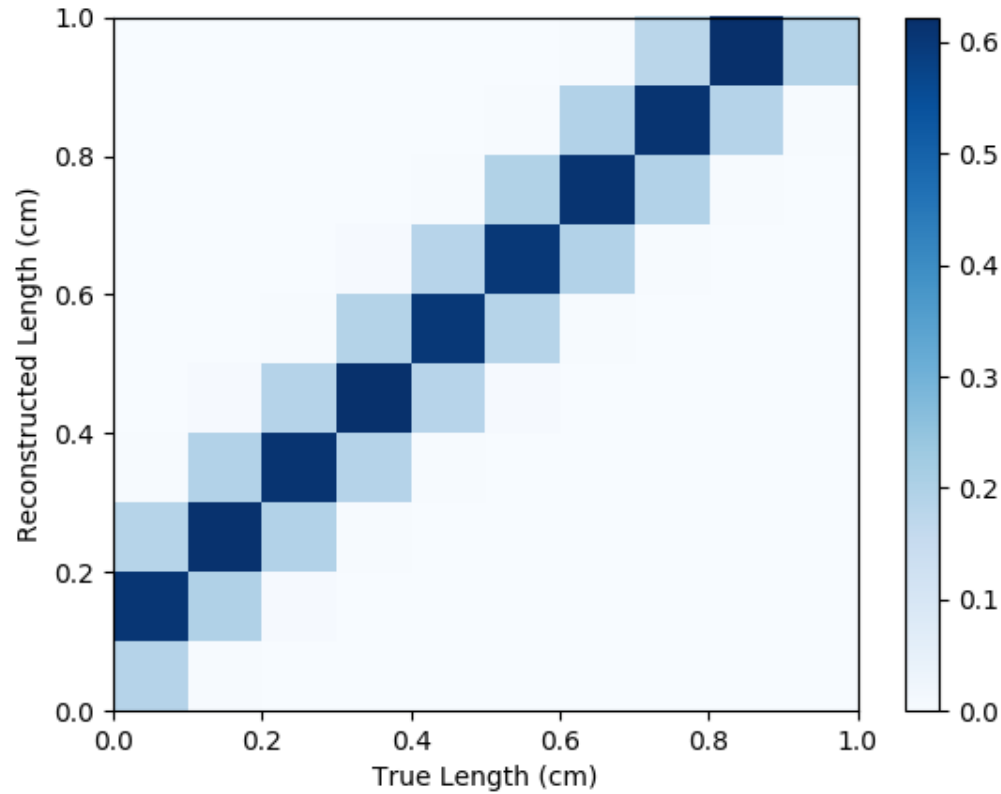


↑  
Pick ~0.025



# The Inverse Problem

- Example 5: Can also have biases



# The Inverse Problem

- Sometimes people decide to put the biases in a separate matrix so it is a “diagonal matrix”
- In practice it doesn't matter.

# Clustering

- Goal: Given  $N$  points in space, associate to  $k$  partitions
- Many applications:
  - Data classification
  - Galaxy clustering
  - Jet clustering
  - Sociology
  - Social networks



(a) Original points.



(b) Two clusters.



(c) Four clusters.



(d) Six clusters.

# Clustering

- Goal: Given  $N$  points in space, associate to  $k$  partitions
- Many applications:
  - Data classification
  - Galaxy clustering
  - Jet clustering
  - Sociology
  - Social networks

Criminal

Athlete

Princess

Nerd



Basket case

# Clustering

- Goal: Given  $N$  points in space, associate to  $k$  partitions
- Many applications:
  - Data classification
  - Galaxy clustering
  - Jet clustering
  - Sociology
  - Social networks

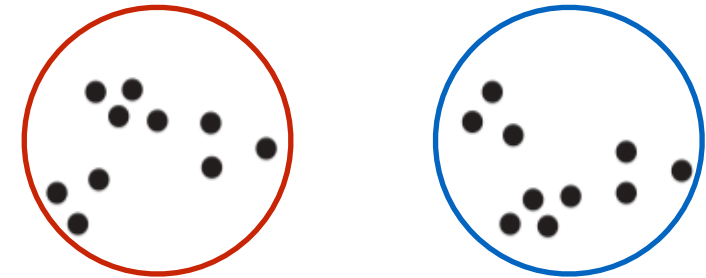




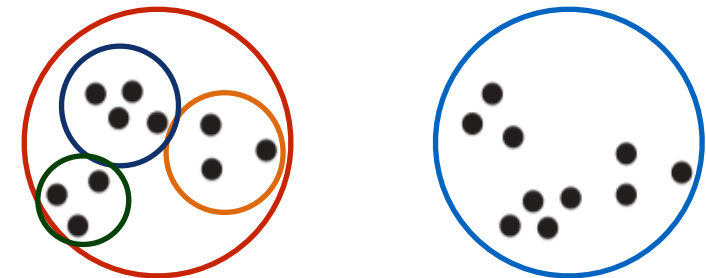
# Clustering

- Uses:
  - Summarize / compress “spatial” information
  - Find nearest neighbors
- Types of clustering algorithms:
  - Exclusive:
    - Partitional: divide into  $k$  exclusive categories
    - Hierarchical: can have sub-clusters
  - Non-exclusive:
    - Add same element to more than one cluster
  - Fuzzy:
    - Weight elements according to cluster
- Each can be either complete or partial

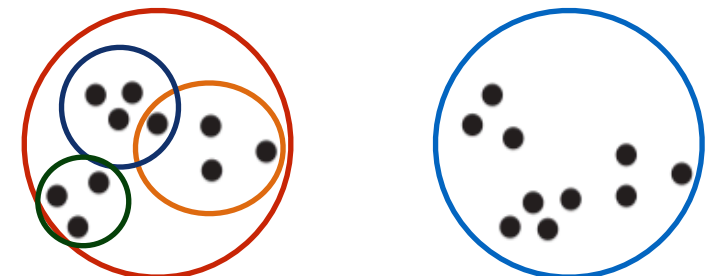
Partitional



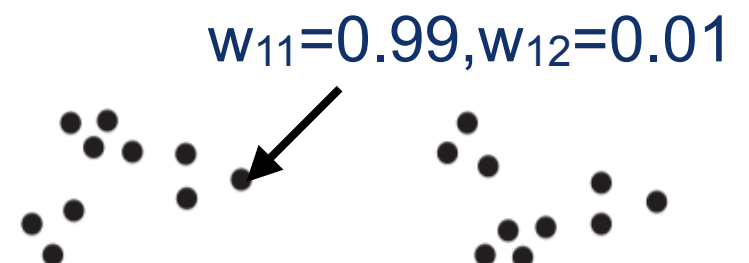
Hierarchical



Non-exclusive



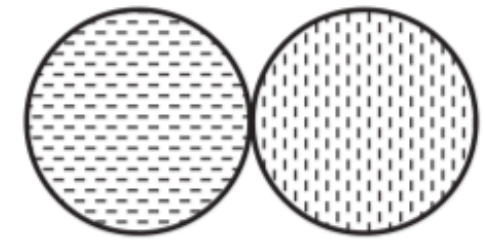
Fuzzy



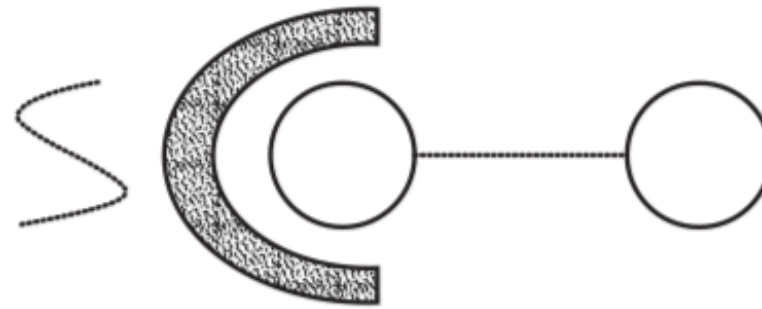
# Clustering



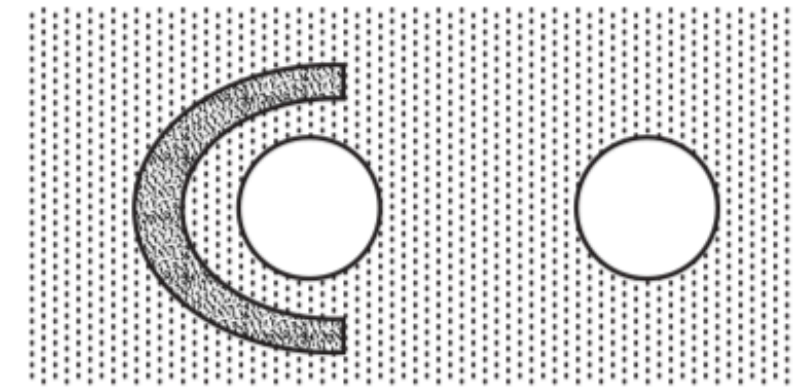
(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.



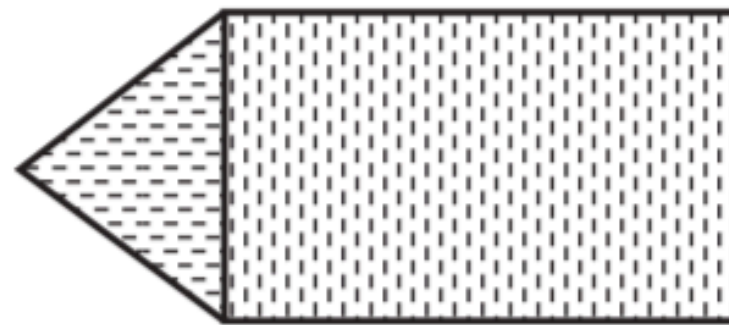
(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.



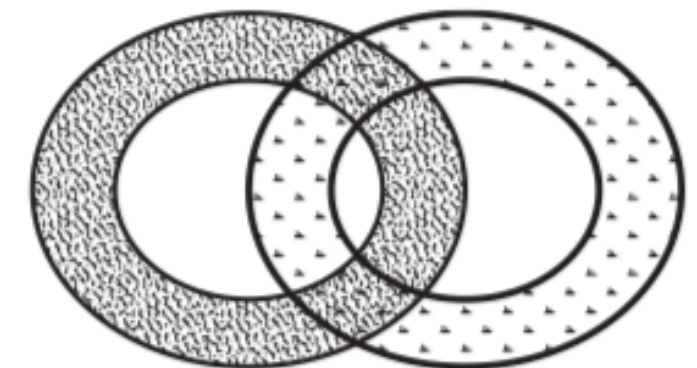
(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.



(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.



(e) Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)



- Types of clusters:

# Clustering

- Popular categories :
  - K-means
    - Partition into  $k$  clusters using mean central values (usually exclusively)
  - Agglomerative hierarchical clustering
    - Pair individual elements into clusters given some distance metric
  - Density based scan
    - Considers low-density regions to be noise, not exclusive clustering



# Clustering

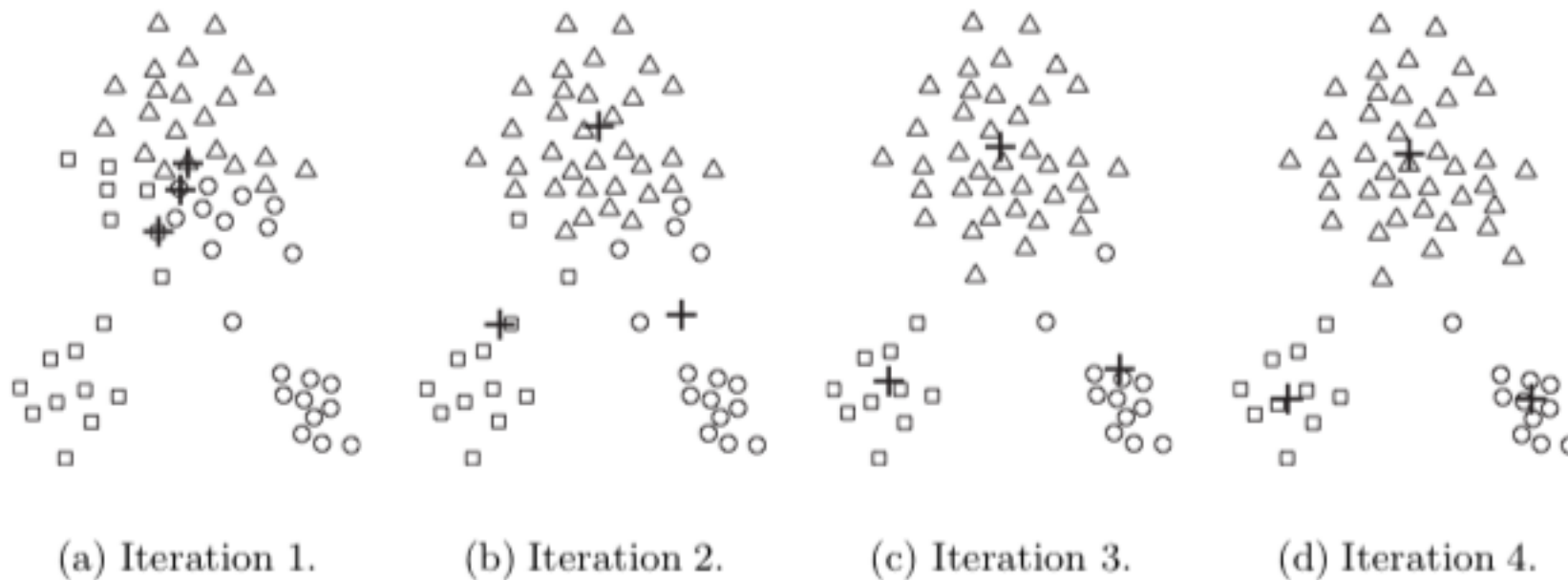
- K-means algorithm:

---

**Algorithm 8.1** Basic K-means algorithm.

---

- 1: Select  $K$  points as initial centroids.
  - 2: **repeat**
  - 3:   Form  $K$  clusters by assigning each point to its closest centroid.
  - 4:   Recompute the centroid of each cluster.
  - 5: **until** Centroids do not change.
- 



# Clustering

- K-means algorithm:
  - Formally, computes RMS:

Number of Centroids

Number of Data points

$$RMS^2 = \sum_{i=1}^k \sum_{j=1}^n |\vec{c}_i - \vec{x}_j|^2$$

Centroid positions

Data positions

The diagram illustrates the formula for the Root Mean Square (RMS) in the context of K-means clustering. The formula is  $RMS^2 = \sum_{i=1}^k \sum_{j=1}^n |\vec{c}_i - \vec{x}_j|^2$ . Annotations include: 'Number of Centroids' pointing to the variable  $k$  in the upper limit of the first summation; 'Number of Data points' pointing to the variable  $n$  in the upper limit of the second summation; 'Centroid positions' pointing to the vector  $\vec{c}_i$ ; and 'Data positions' pointing to the vector  $\vec{x}_j$ .

- Minimize the RMS by adjusting the centroids
- Note: Other distance metrics can be used, but the principle is the same (minimize the metric)

# Clustering

- Computational complexity is ~linear in product of:
  - Number of points
  - Number of “dimensions” (or attributes)
  - Number of clusters
  - Number of iterations to converge
- Shortcomings of k-means:
  - As in all minimization routines, danger of local minima
    - Need heuristic methods to avoid them
  - Can result in empty clusters if initialized poorly
  - Outliers have disproportionate impact
- Can try to split and merge centroids to mitigate these!

# Clustering

- Alternative to k-means: Bisecting k-means:

---

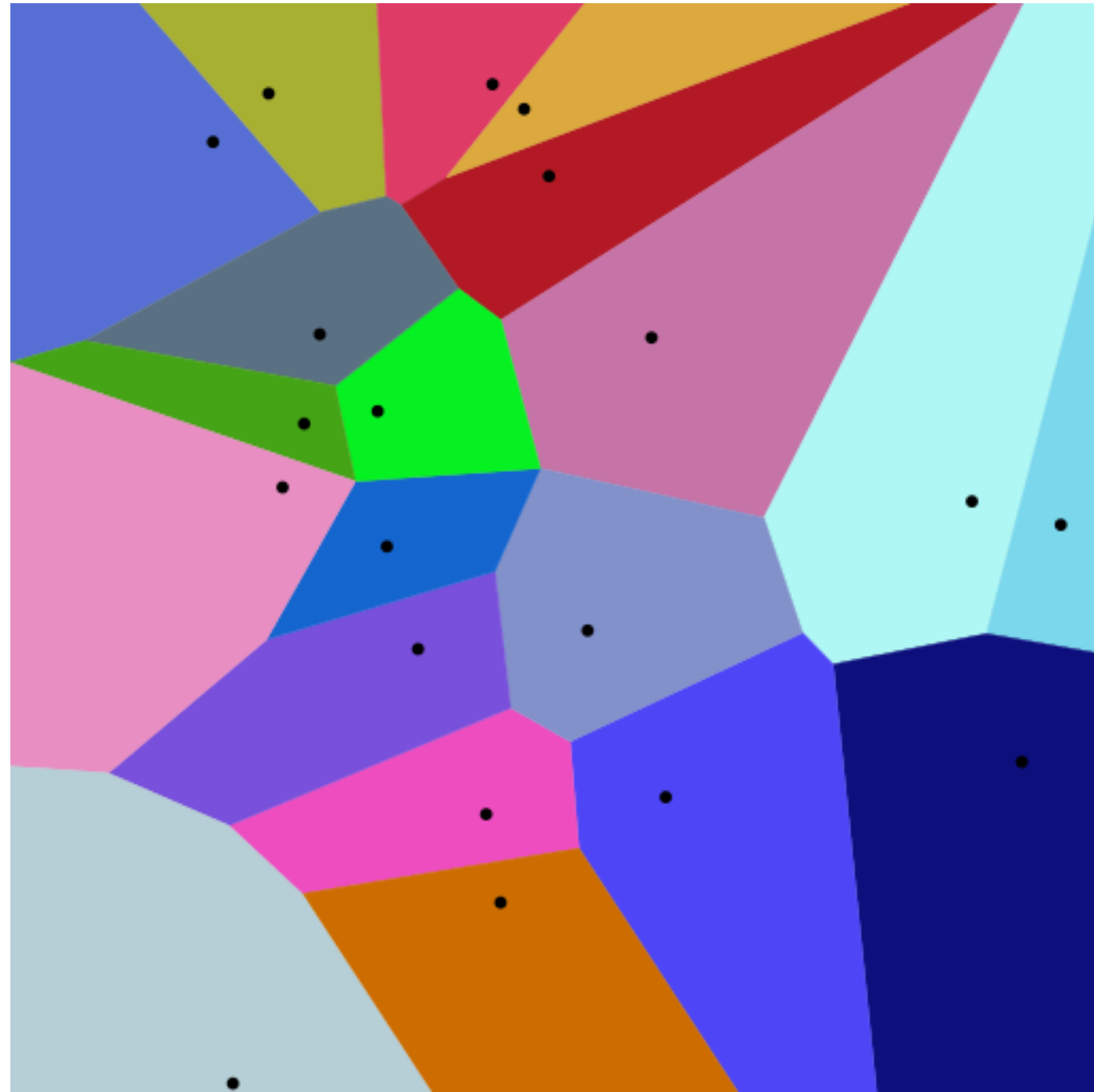
**Algorithm 8.2** Bisecting K-means algorithm.

---

- 1: Initialize the list of clusters to contain the cluster consisting of all points.
  - 2: **repeat**
  - 3:   Remove a cluster from the list of clusters.
  - 4:   {Perform several “trial” bisections of the chosen cluster.}
  - 5:   **for**  $i = 1$  to *number of trials* **do**
  - 6:     Bisect the selected cluster using basic K-means.
  - 7:   **end for**
  - 8:   Select the two clusters from the bisection with the lowest total SSE.
  - 9:   Add these two clusters to the list of clusters.
  - 10: **until** Until the list of clusters contains  $K$  clusters.
-

# Clustering

- Result is a Voronoi diagram
- Each point is closer to all points in its cell than other cells
- Also referred to as “catchment areas”
  - From river basins, water tables, etc... where does the water pool when it rains?

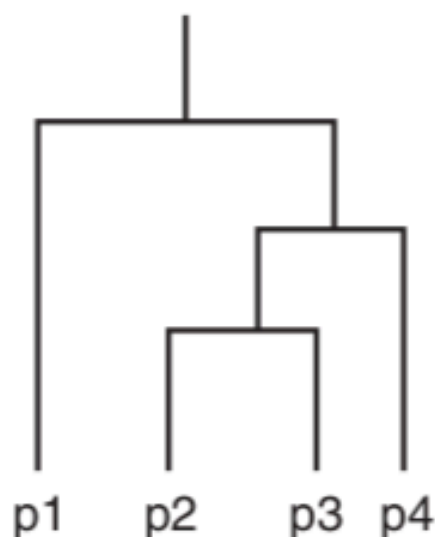


[https://en.wikipedia.org/wiki/Voronoi\\_diagram](https://en.wikipedia.org/wiki/Voronoi_diagram)

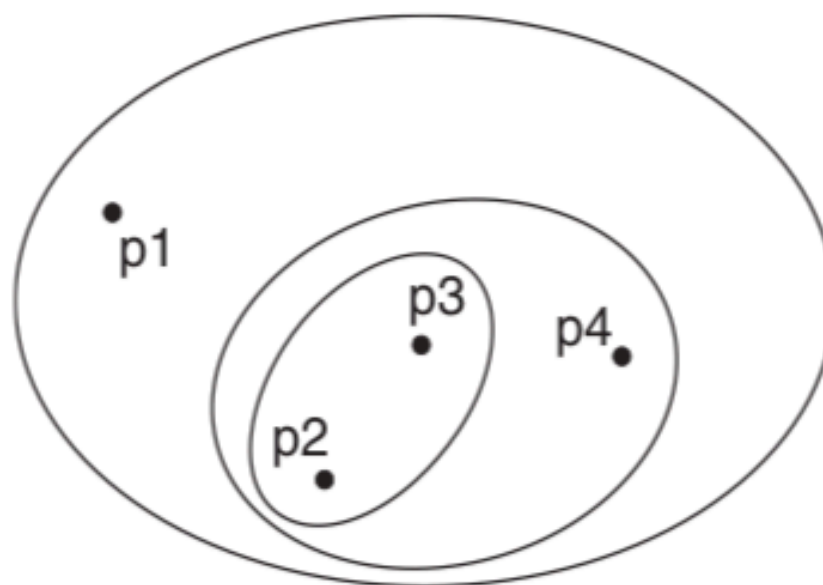
# Clustering

- Hierarchical clustering:
  - Agglomerative (“bottom up”):
    - Start with individual constituents, merge until criteria met
  - Divisive (“top down”):
    - Start with conglomerate, split until criteria met (or you get to individual constituents)
  - Represent by a tree (“dendrogram”) or Venn diagram (“nested cluster diagram”):

Basically the same,  
but in reverse



(a) Dendrogram.



(b) Nested cluster diagram.

# Clustering

---

**Algorithm 8.3** Basic agglomerative hierarchical clustering algorithm.

---

- 1: Compute the proximity matrix, if necessary.  $O(n^2)$ , done once
  - 2: **repeat**
  - 3: Merge the closest two clusters.  $O(1)$ , done  $n$  times
  - 4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.  $O(n^2)$ , done  $n$  times
  - 5: **until** Only one cluster remains.
- 

$n^3$  algorithm?

# Clustering

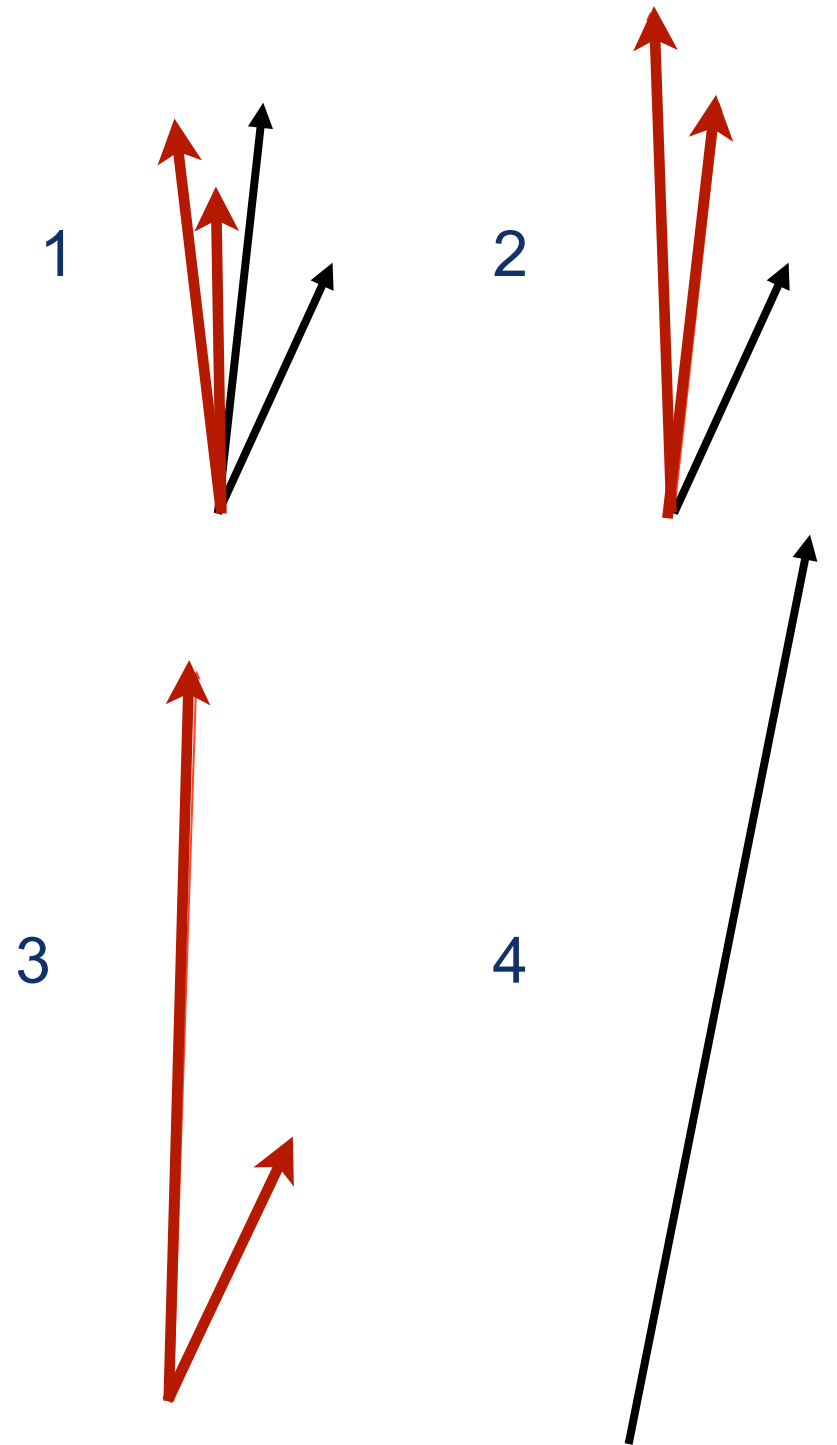
- Faster clustering algorithms (i.e. fastjet):
  - Can precompute nearest neighbors in the metric, and then only look at those instead of all (nearest is nearest is nearest!)

Calculate nearest neighbors while inputs are left:

    Compute distances to neighbors, and self

    If distance to neighbor is smallest, merge + iterate

    Else if distance to self is smallest, stop



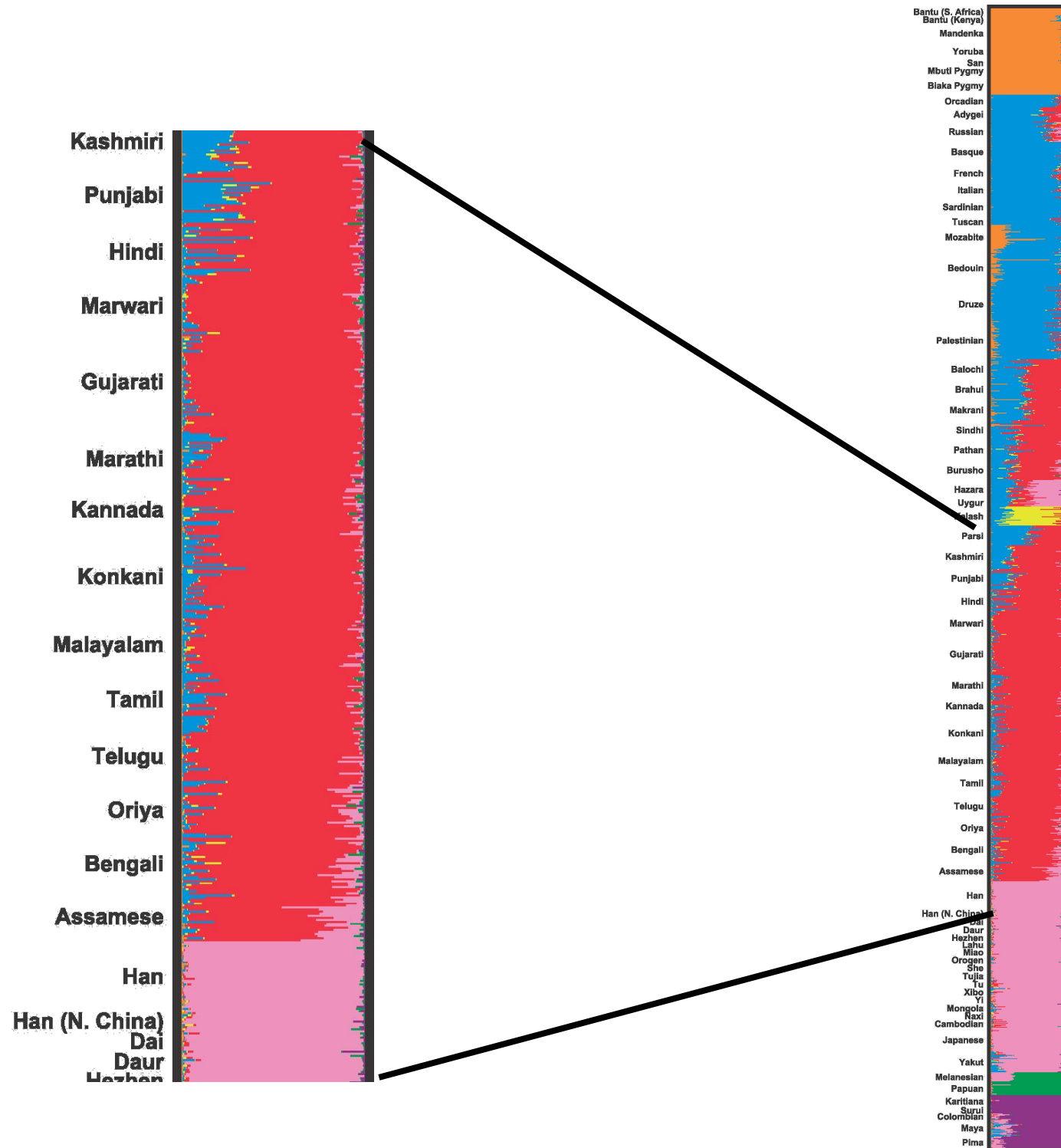


# Clustering

- This reduces to  $O(n^2)$  complexity
- Can reduce further!
  - To find nearest neighbors, can use Voronoi diagrams (like we had before)
  - This reduces complexity to  $O(n \ln(n))$ !

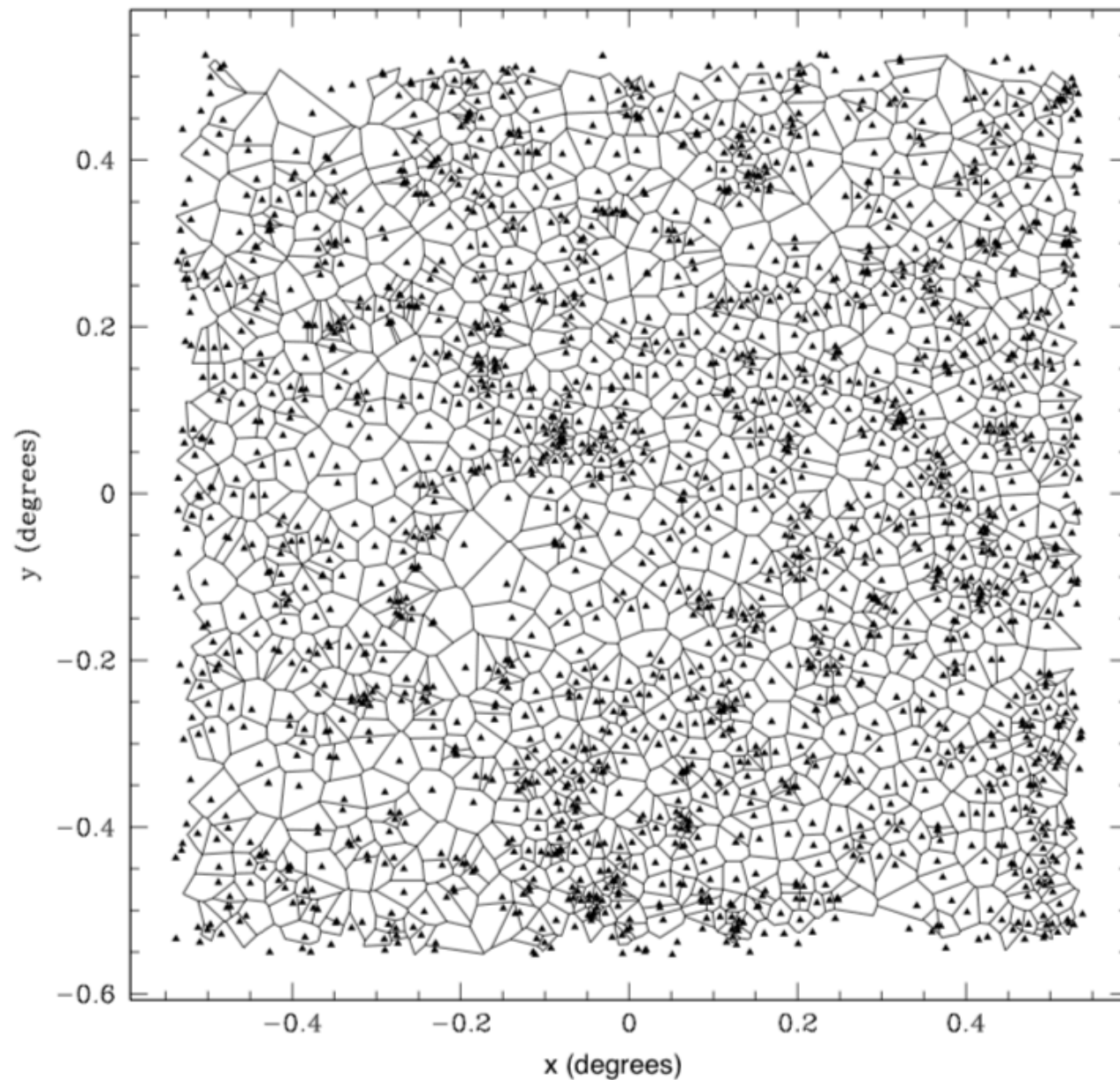
# Clustering

- Example: Genetic clustering
  - Including those “who are your ancestors” DNA kits!



# Clustering

- Example: Finding galaxy clusters using Voronoi tessellations
  - <https://www.aanda.org/articles/aa/pdf/2001/12/aa10522.pdf>



**Fig. 1.** Voronoi tessellation of a galaxy field

# Clustering

- Example: Finding hadronic jets (The Anti-kT Jet Clustering Algorithm)
- <http://inspirehep.net/record/779080>

- Hierarchical clustering with various metrics

