Machine Learning 2

DS 4420 - Spring 2020

Midterm topics

Byron C Wallace

Machine Learning 2

DS 4420 - Spring 2020

PROVIDES AN OVERVIEW BUT NOT EXHAUSTIVE!!!

Midterm topics

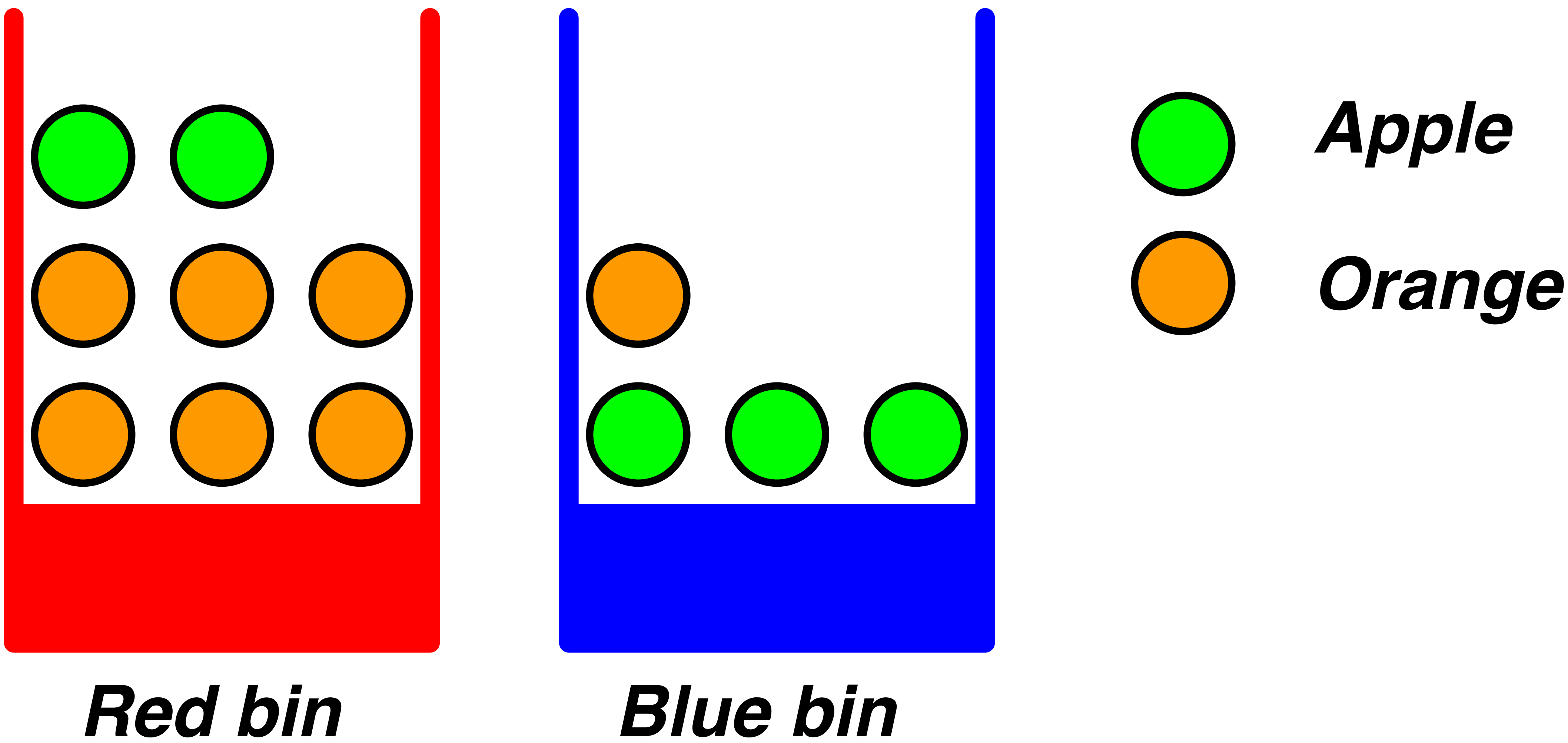
Byron C Wallace

What have we covered?

Logistics, overview
Math Review
MLE, MAP, and graphical models
Neural networks / backprop
Clustering I
Clustering II → Mixture models and EM
Topic modeling I
Topic modeling II
Dimensionality reduction I
Dimensionality reduction II
Auto-encoders/"Self-supervision"; Learning to embed
Structured prediction I
Structured prediction II

The fundamentals

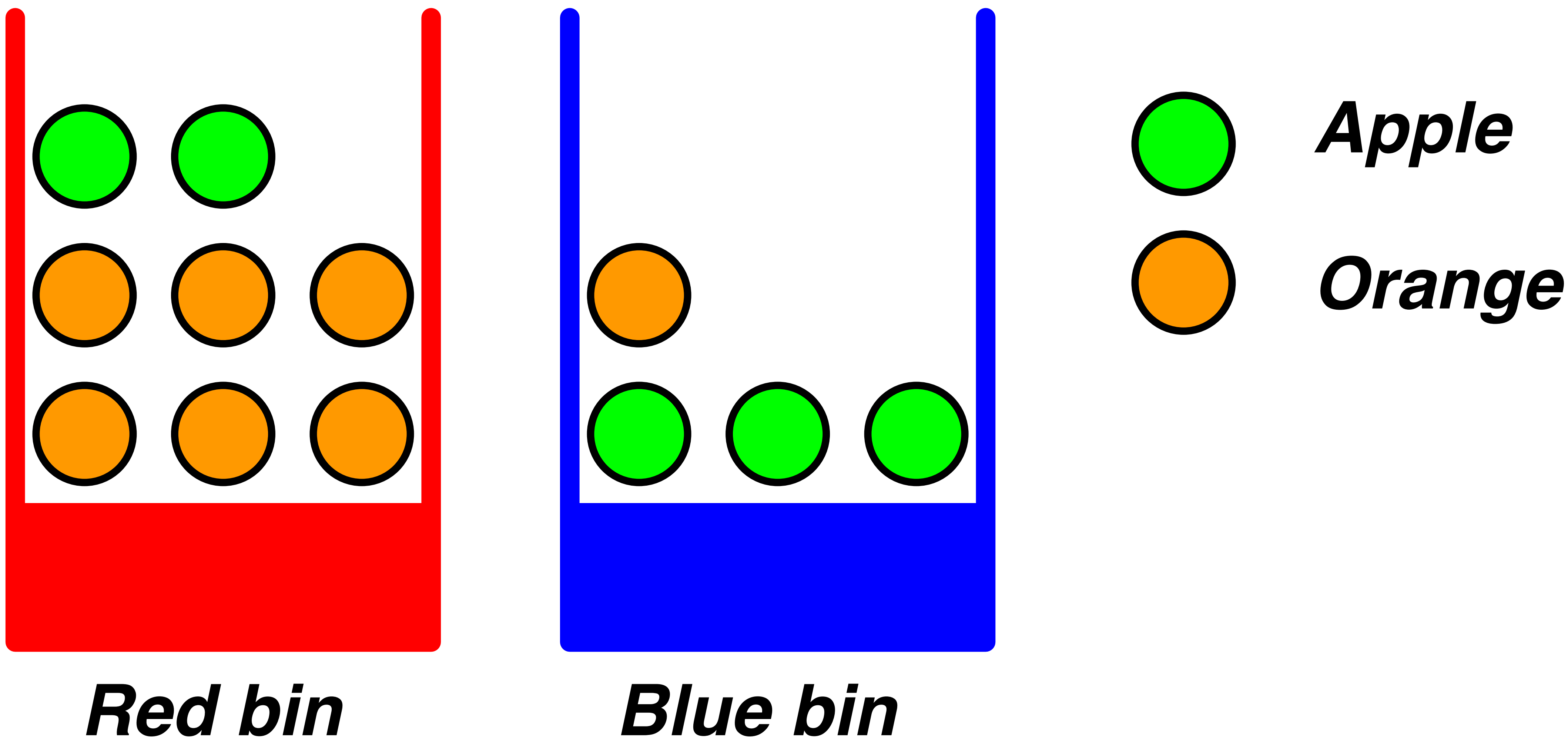
Dependent Events



Conditional Probability

P(fruit = apple | bin = red) = 2 / 8

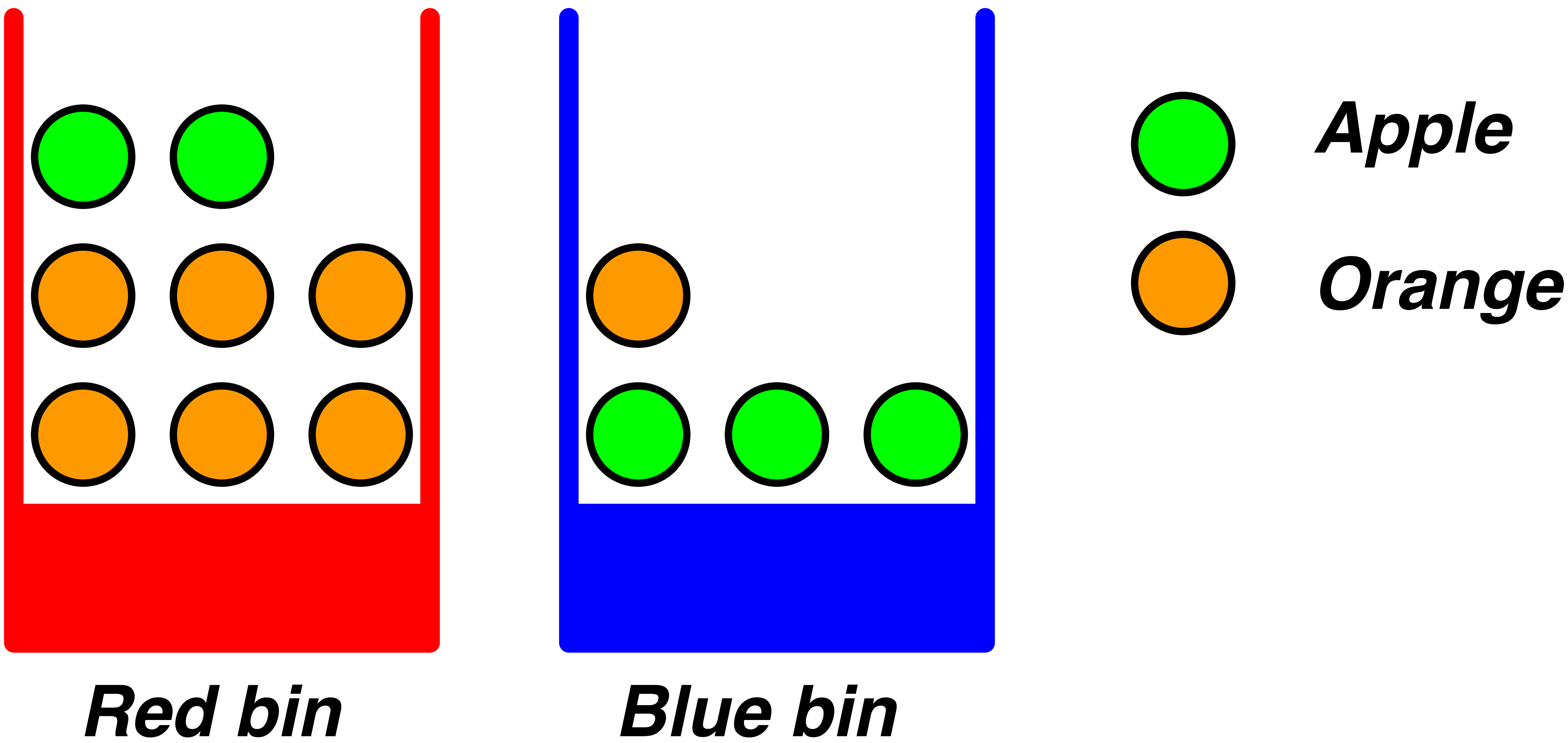
Dependent Events



Joint Probability

P(fruit = apple , bin = blue) = ?

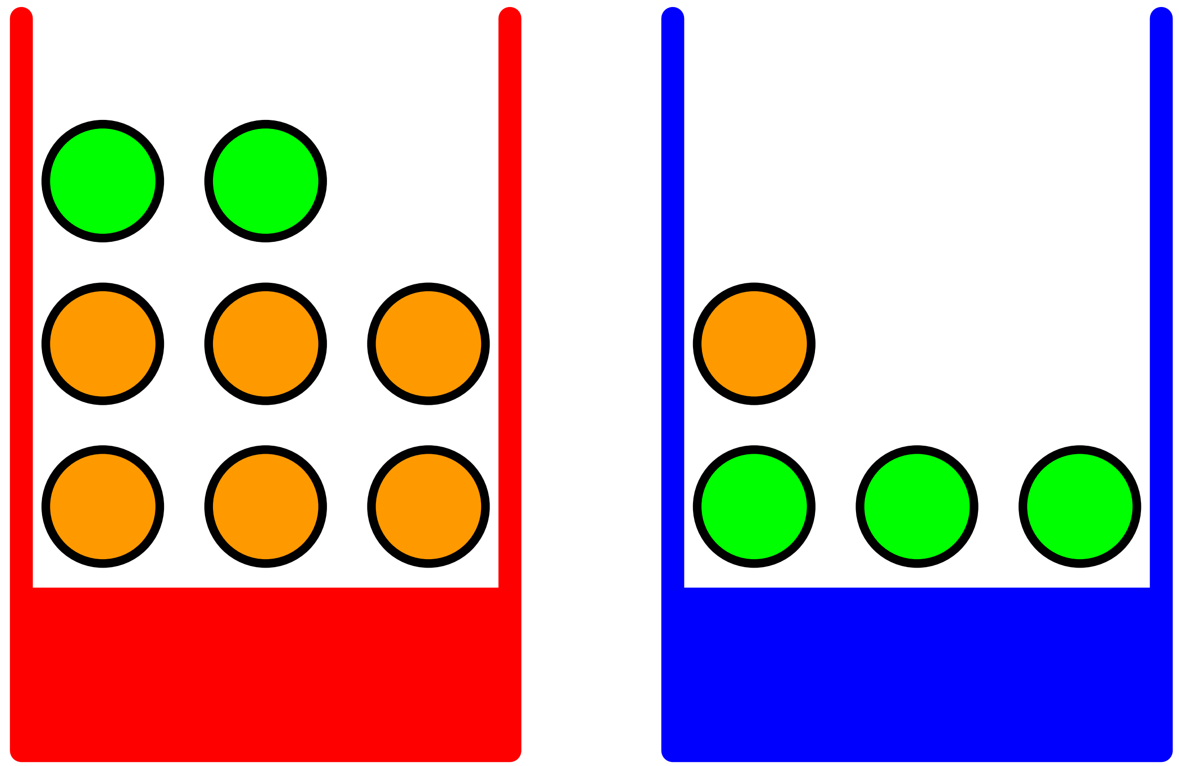
Dependent Events



Joint Probability

P(fruit = apple , bin = blue) = 3 / 12

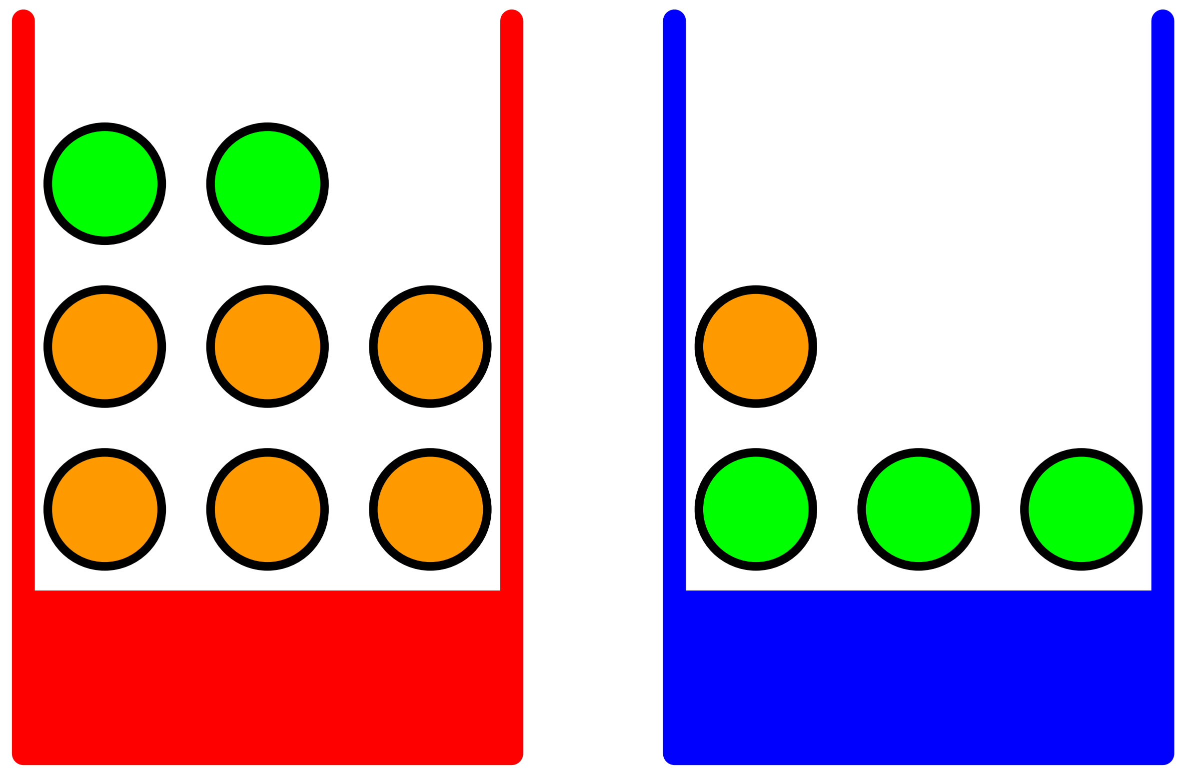
Two rules of Probability



1. Sum Rule (Marginal Probabilities)

P(fruit = apple) = ?

Two rules of Probability



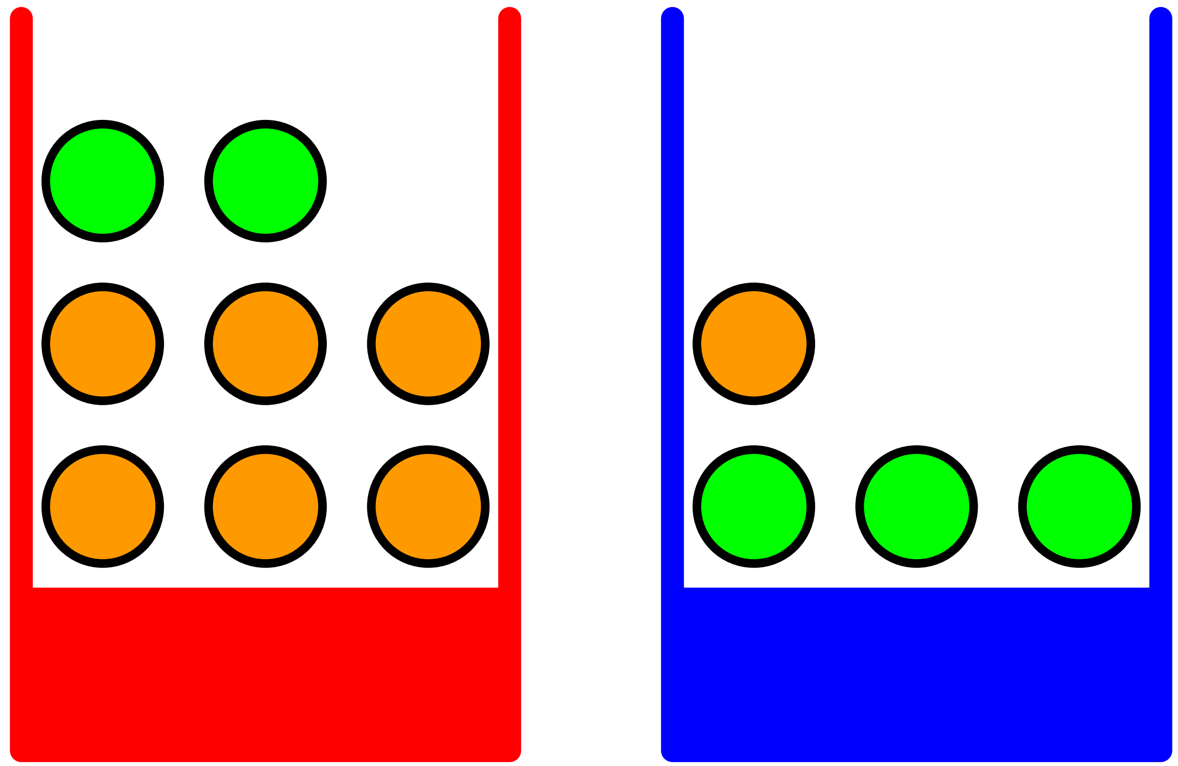
1. Sum Rule (Marginal Probabilities)

P(fruit = apple) = P(fruit = apple , bin = blue)

+ P(fruit = apple , bin = red)

= ?

Two rules of Probability



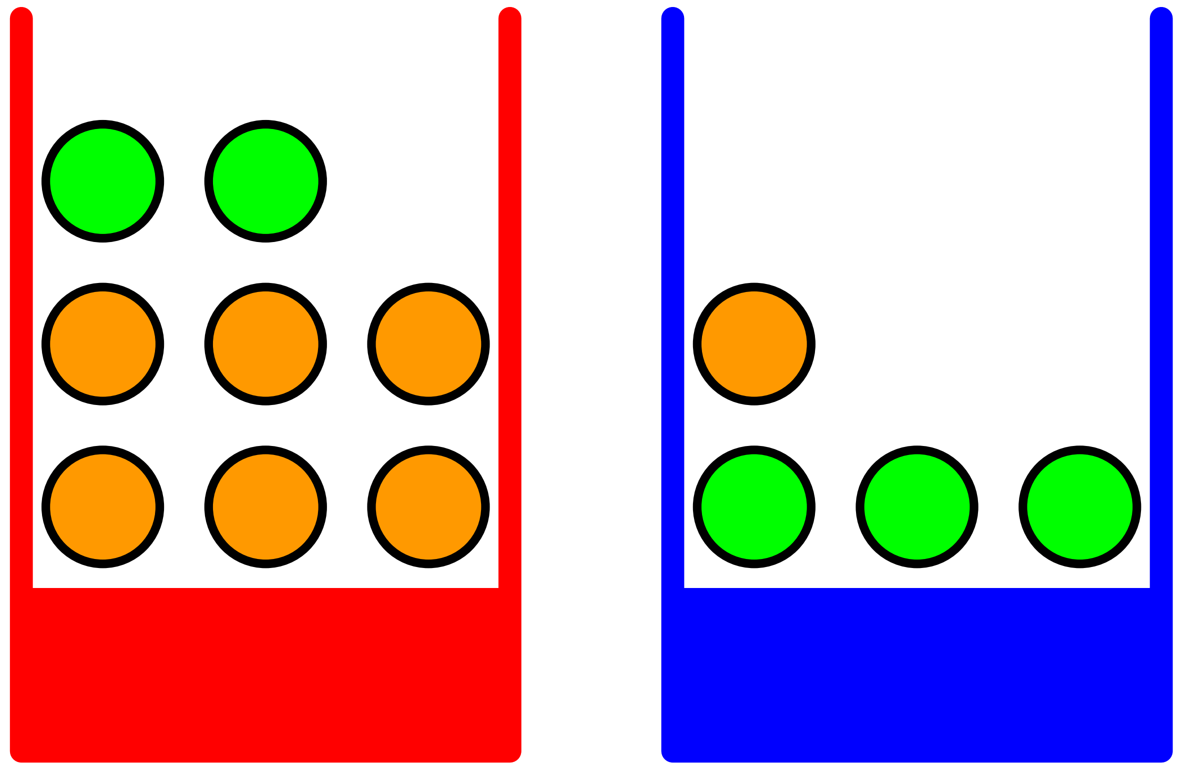
1. Sum Rule (Marginal Probabilities)

P(fruit = apple) = P(fruit = apple , bin = blue)

+ P(fruit = apple , bin = red)

= 3 / 12 + 2 / 12 = **5 / 12**

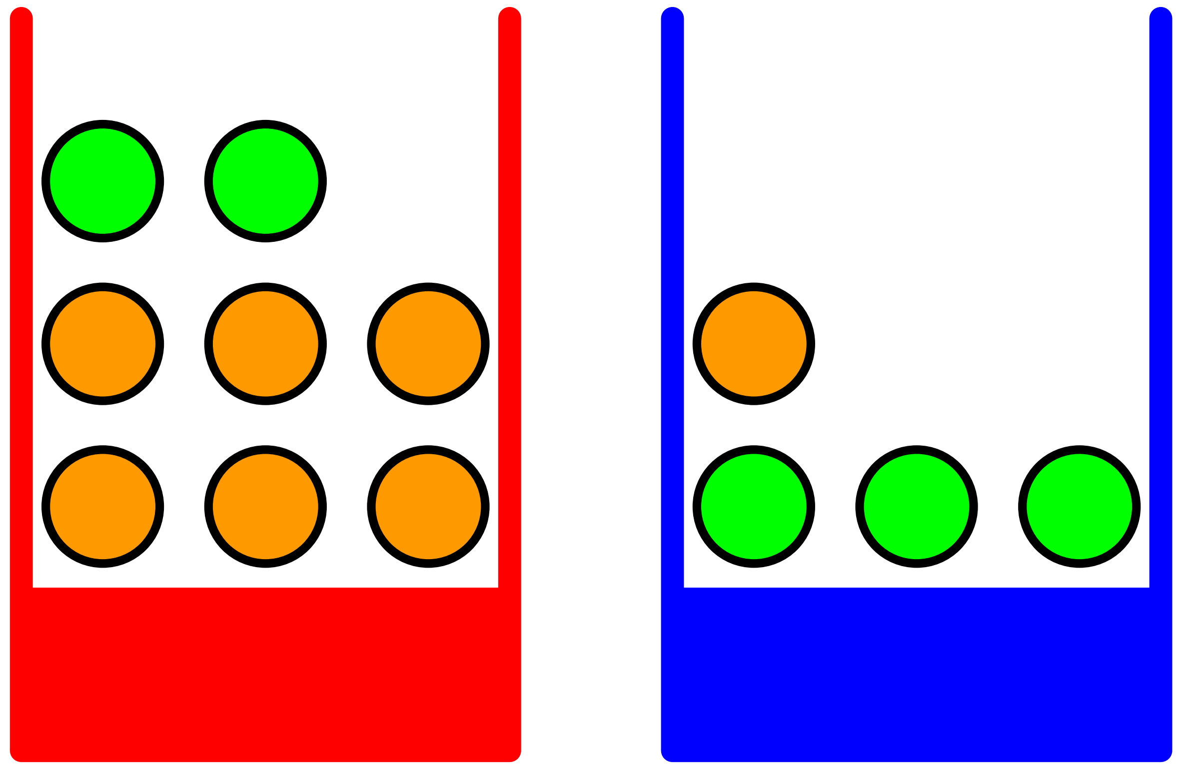
Two rules of Probability



2. Product Rule

P(fruit = apple , bin = red) = ?

Two rules of Probability



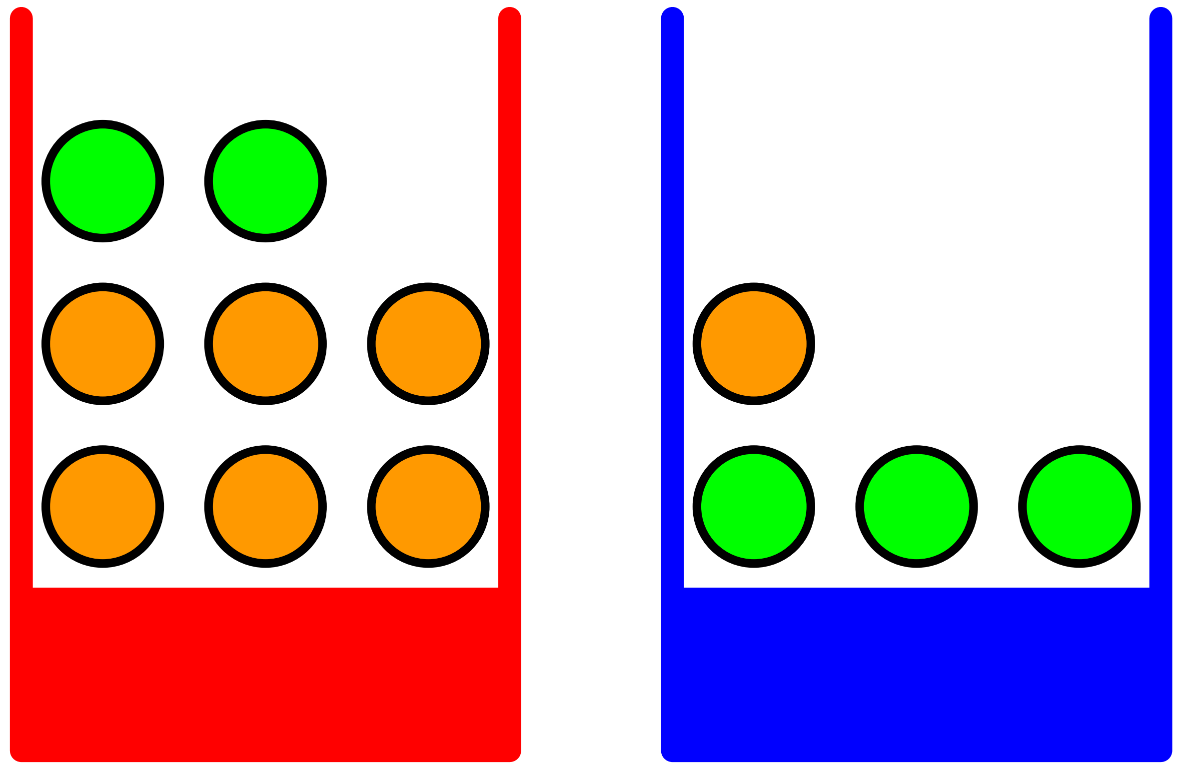
2. Product Rule

P(fruit = apple , bin = red) =

P(fruit = apple | bin = red) p(bin = red)

= ?

Two rules of Probability



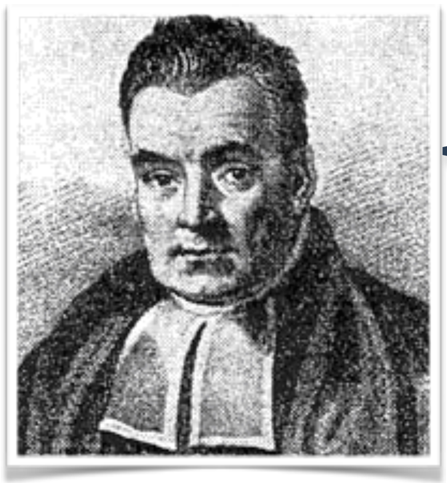
2. Product Rule

P(fruit = apple , bin = red) =

P(fruit = apple | bin = red) p(bin = red)

= 2 / 8 * 8 / 12 = **2 / 12**

Bayes' Rule



$$\underbrace{p(\mathbf{x} \mid \mathbf{y})}_{\text{Posterior}} = \underbrace{p(\mathbf{y} \mid \mathbf{x})}_{\text{Likelihood}} \underbrace{p(\mathbf{x})}_{\text{Prior}} / p(\mathbf{y})$$

Calc: Univariate Functions

$$y = f(x),\ x, y \in \mathbb{R}$$

Difference Quotient

$$\frac{\delta y}{\delta x} := \frac{f(x + \delta x) - f(x)}{\delta x}$$

Sum Rule

$$(f(x) + g(x))' = f'(x) + g'(x)$$

Product Rule

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$$

Chain Rule

$$\big(g(f(x))\big)' = (g \circ f)'(x) = g'(f(x))f'(x)$$

More Dims —> Gradients

Group the gradients into a vector (the *gradient*)

$$\nabla_{\boldsymbol{x}} f = \mathrm{grad} f = \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} & \frac{\partial f(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{1 \times n}$$

Example

$$f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3$$

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 x_2 + x_2^3$$

$$\frac{\partial f(x_1, x_2)}{\partial x_2} = x_1^2 + 3x_1 x_2^2$$

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f(x_1, x_2)}{\partial x_1} & \frac{\partial f(x_1, x_2)}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 x_2 + x_2^3 & x_1^2 + 3x_1 x_2^2 \end{bmatrix} \in \mathbb{R}^{1\times 2}$$

Rules still hold!

Sum rule: $$\frac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x}) + g(\boldsymbol{x})\big) = \frac{\partial f}{\partial \boldsymbol{x}} + \frac{\partial g}{\partial \boldsymbol{x}}$$

Product rule: $$\frac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x})g(\boldsymbol{x})\big) = \frac{\partial f}{\partial \boldsymbol{x}}g(\boldsymbol{x}) + f(\boldsymbol{x})\frac{\partial g}{\partial \boldsymbol{x}}$$

Chain rule: $$\frac{\partial}{\partial \boldsymbol{x}}(g \circ f)(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}\big(g(f(\boldsymbol{x}))\big) = \frac{\partial g}{\partial f}\frac{\partial f}{\partial \boldsymbol{x}}$$

… but be mindful of dims!

MLE Framework

Observe some data $X = x_1, ..., x_n \qquad x_i \in R^d$

We assume this is a random draw (sample) from some parameterized distribution P_θ

Goal: find θ

In MLE we pick

$$\theta_{\mathrm{MLE}} = \mathrm{argmax}_\theta P(X|\theta)$$

$$P(X|\theta) = \prod_i P(x_i|\theta)$$

Maximum Likelihood Estimation

Likelihood of ***N*** independent events:

$$p_\theta(x_1, \dots, x_N) = \prod_{n=1}^{N} p_\theta(x_n) \qquad p_\theta(x_n) = \prod_{k=1}^{K} \theta_k^{x_{n,k}}$$

Maximum likelihood estimation

$$\begin{aligned}\theta^* &= \operatorname*{argmax}_\theta \; p_\theta(x_1, \dots, x_N) \\ &= \operatorname*{argmax}_\theta \; \log p_\theta(x_1, \dots, x_N) \\ &= \operatorname*{argmax}_\theta \sum_{k=1}^{K} N_k \log \theta_k \qquad N_k = \sum_{n=1}^{N} x_{n,k}\end{aligned}$$

(known as cross-entropy loss in neural net libraries)

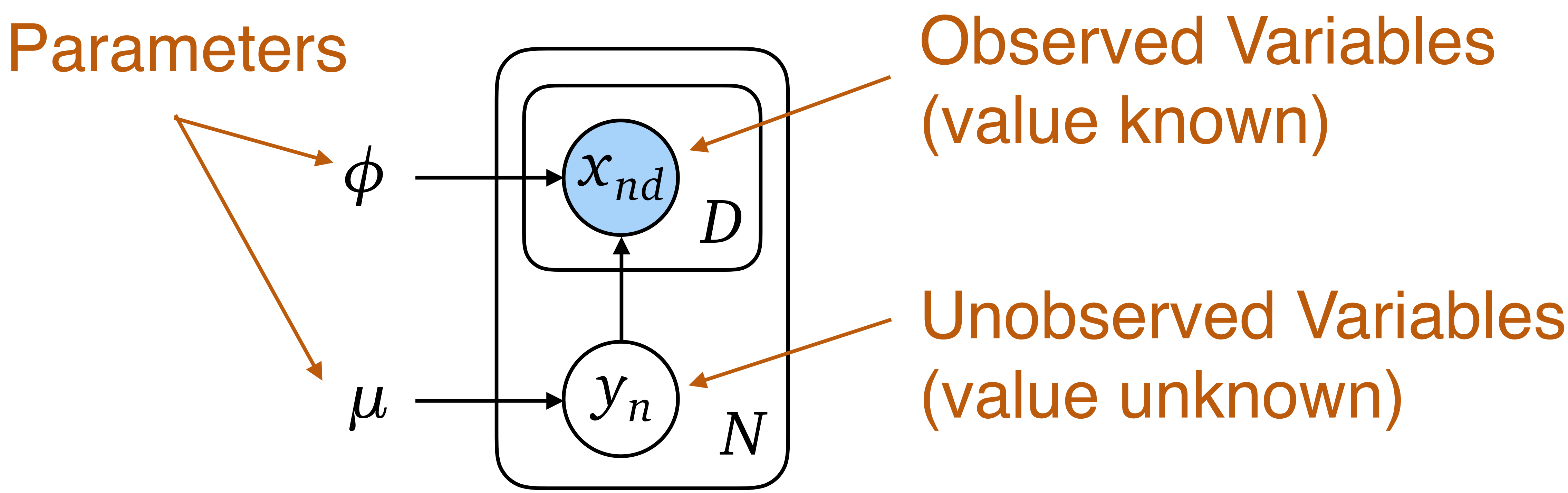
Problems with MLE?

- Provides a *point estimate*; no notion of uncertainty around parameters
- Does not naturally incorporate prior beliefs (maybe a pro, if you're a frequentist?)

Graphical Model: Naive Bayes

$$y_n \sim \text{Bernoulli}(\mu) \qquad n = 1, \ldots, N$$

$$\boldsymbol{x}_{nd} \mid y_n = k \sim \text{Bernoulli}(\phi_{kd}) \qquad k = 0, 1 \quad d = 1, \ldots, D$$

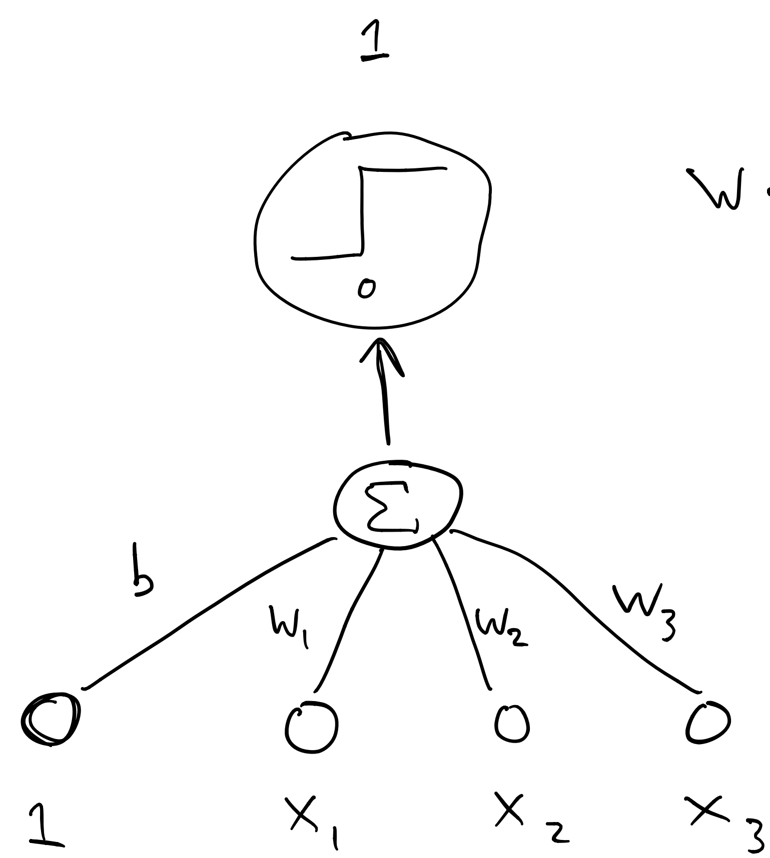


$$p(x, y \mid \mu, \phi) = \prod_{n=1}^{N} p(y_n \mid \mu) \prod_{d=1}^{D} p(x_{nd} \mid y_n, \phi)$$

Neural nets/backprop

Perceptron

$$\hat{y} = \begin{cases} 1 & \text{if } \quad W \cdot x > 0 \\ -1 & \text{otherwise} \end{cases}$$

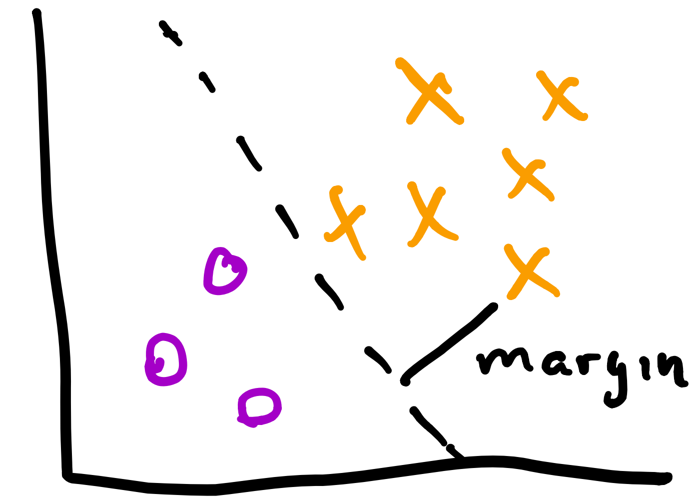


$$W \cdot x = 1 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

Problems with 0/1 loss

- If we're wrong by .0001 it is "as bad" as being wrong by .9999

- Because it is discrete, optimization is hard if the instances are not linearly separable



Smooth loss

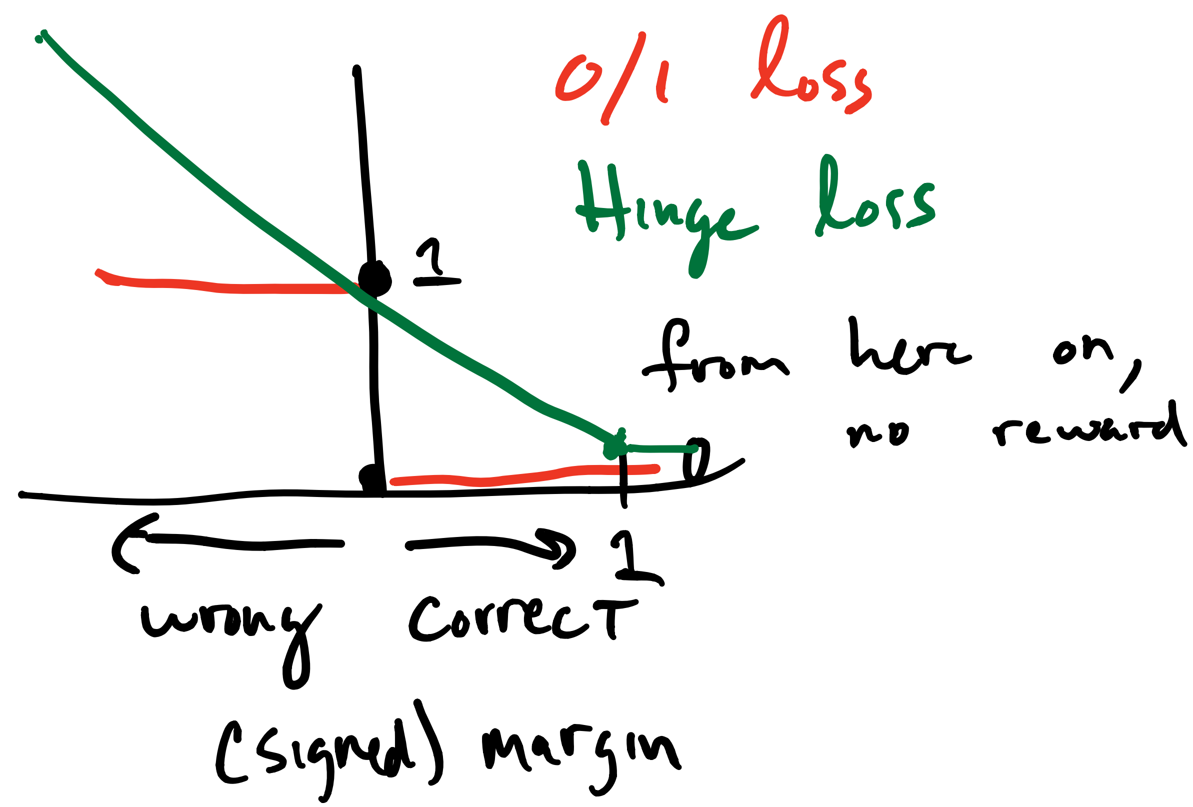
Idea: Introduce a "smooth" loss function to make optimization easier

Example: Hinge loss

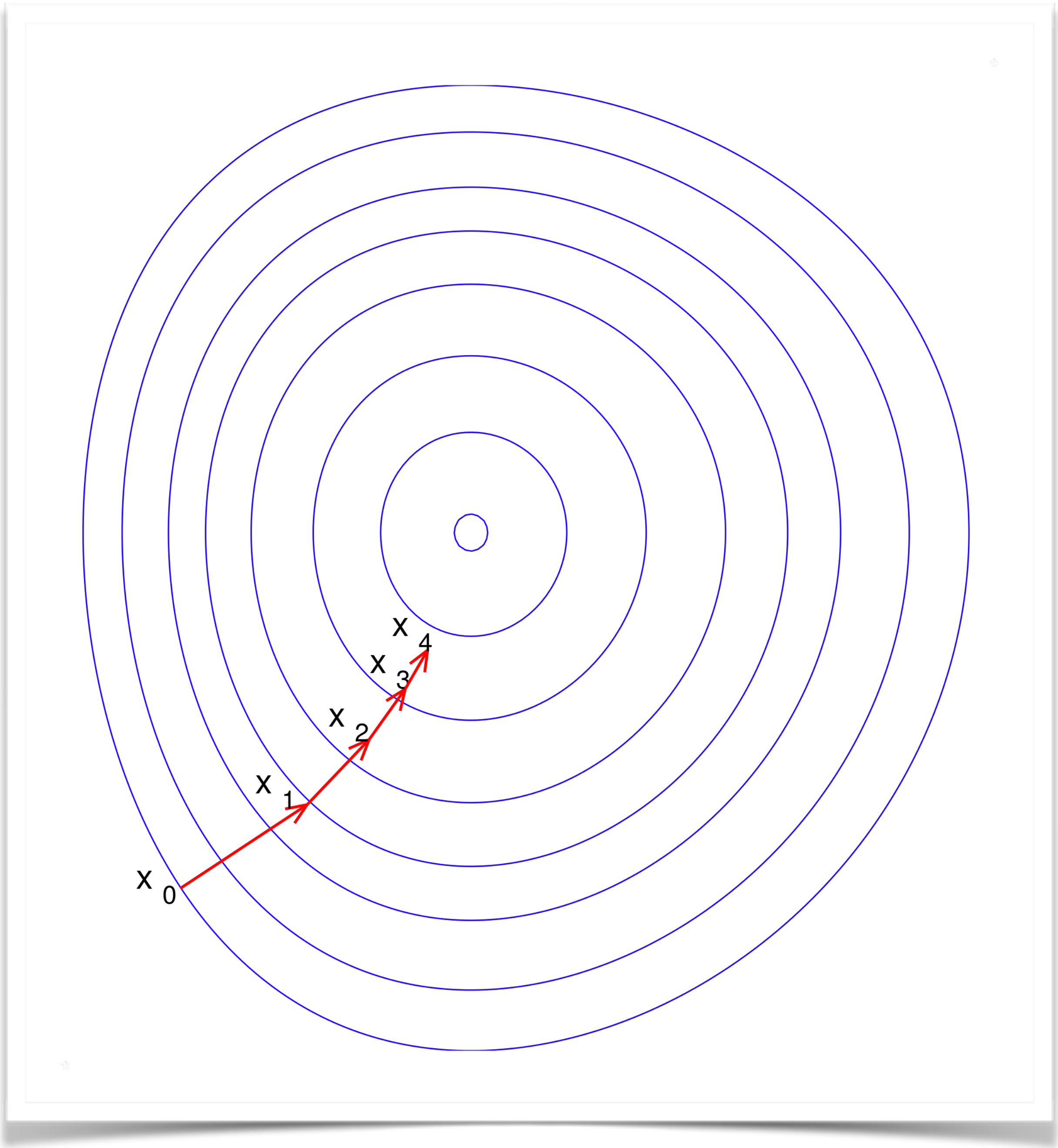
$$\mathcal{L}_{Hinge}(y,z) = \max\{0, 1 - y\cdot z\}$$

$y \in \{1,-1\}$

$z = w\cdot x_i$ ("raw" output)



Gradient descent



By Gradient_descent.png: The original uploader was Olegalexandrov at English Wikipedia.derivative work: Zerodamage - This file was derived from: Gradient descent.png:, Public Domain, https://commons.wikimedia.org/w/index.php?curid=20569355

Algorithm 21 GradientDescent($\mathcal{F}$, K, η_1, ...)

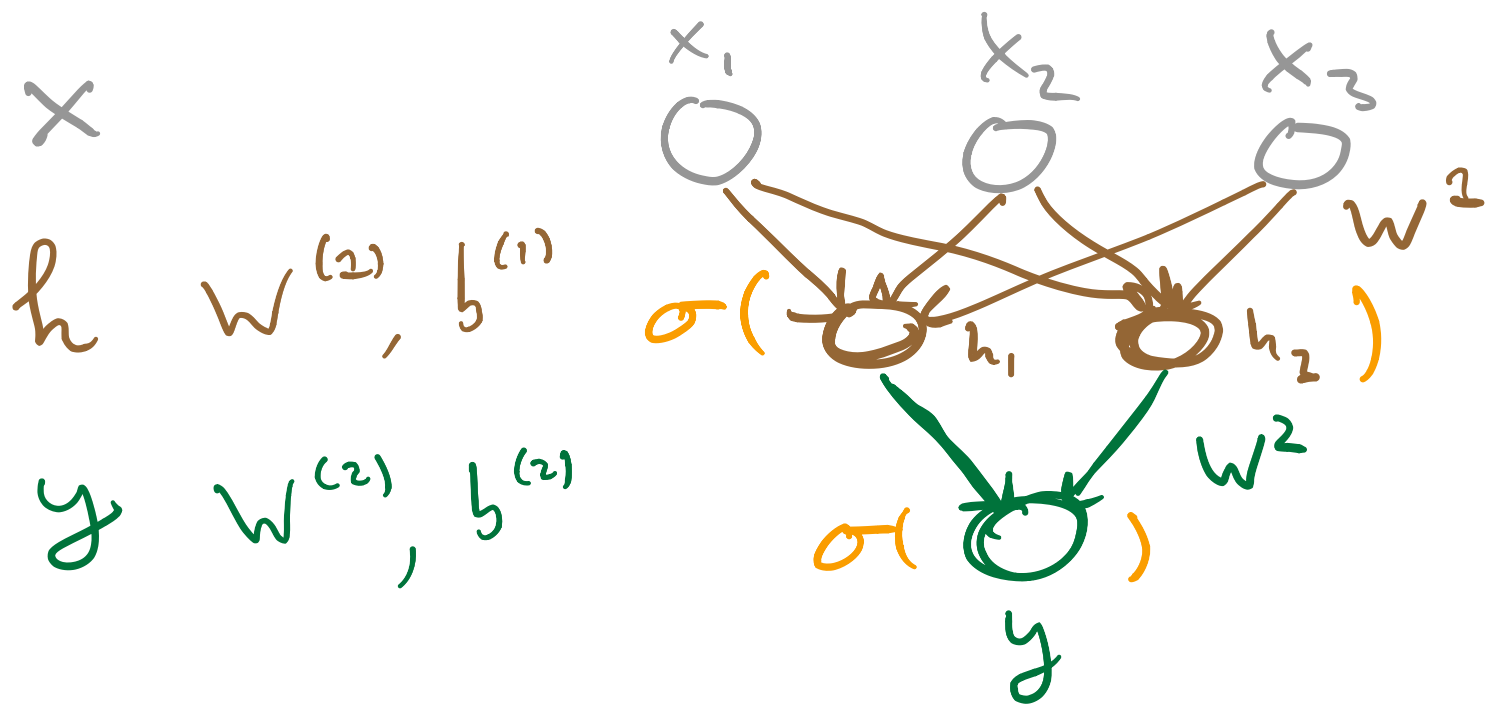
$z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing
for $k = 1 \ldots K$ **do**
    $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location
    $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient
end for
return $z^{(K)}$

Alg from CIML [Daume]

Neural networks

Idea: Basically stack together a bunch of linear models.

This introduces *hidden units* which are neither observations (*x*) nor outputs (*y*)



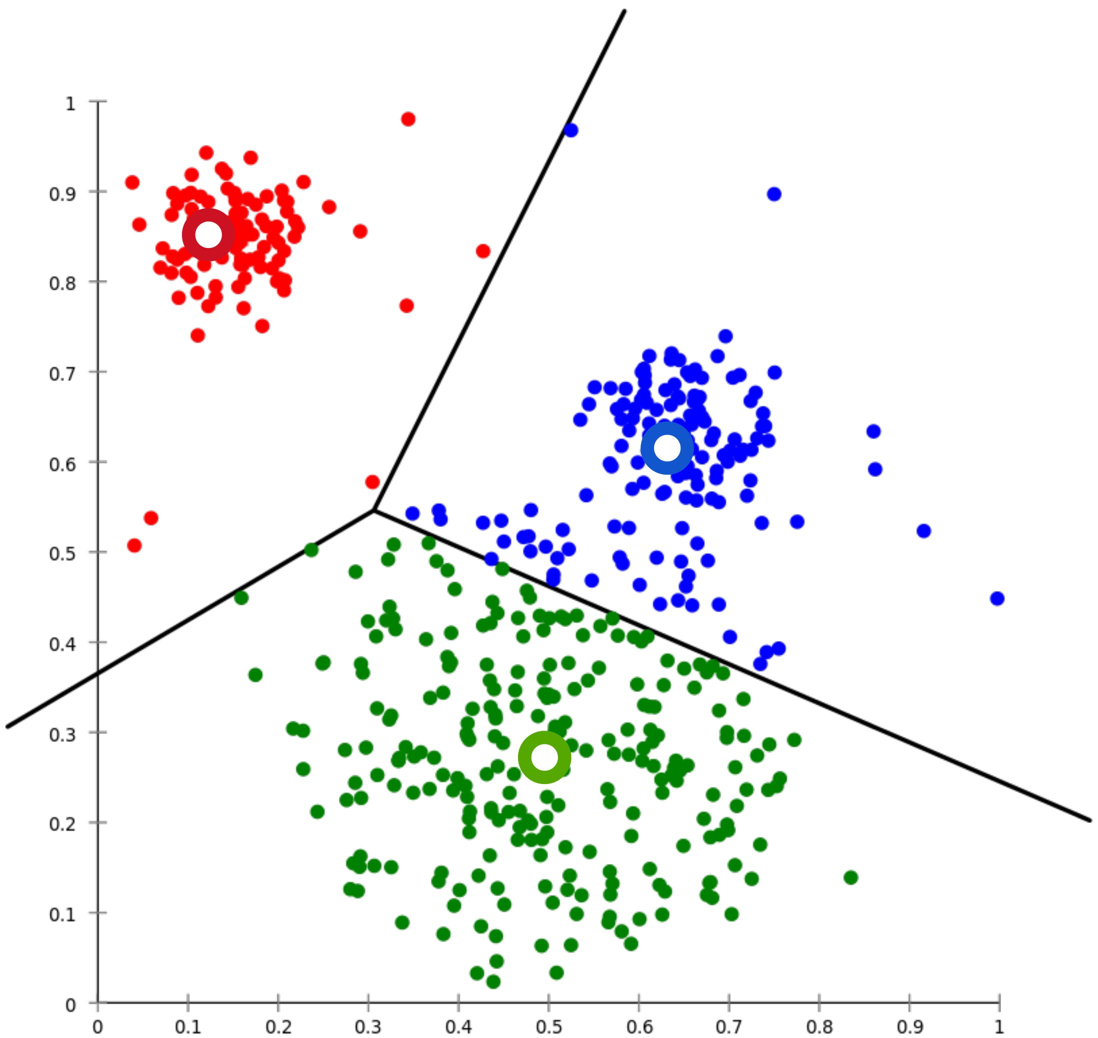
The challenge: How do we update weights associated with each node in this *multi-layer* regime?

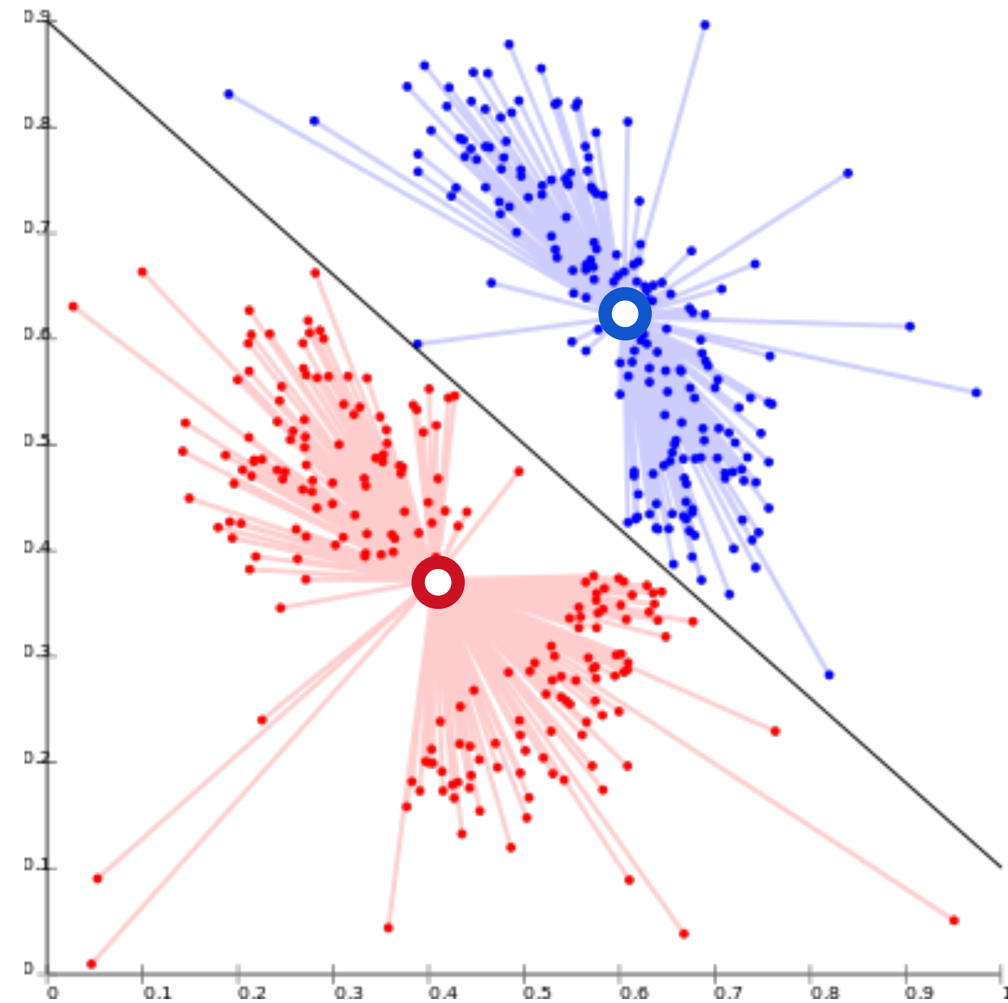
$$\text{back-propagation} = \text{gradient descent} + \text{chain rule}$$

Clustering —> EM

Four Types of Clustering

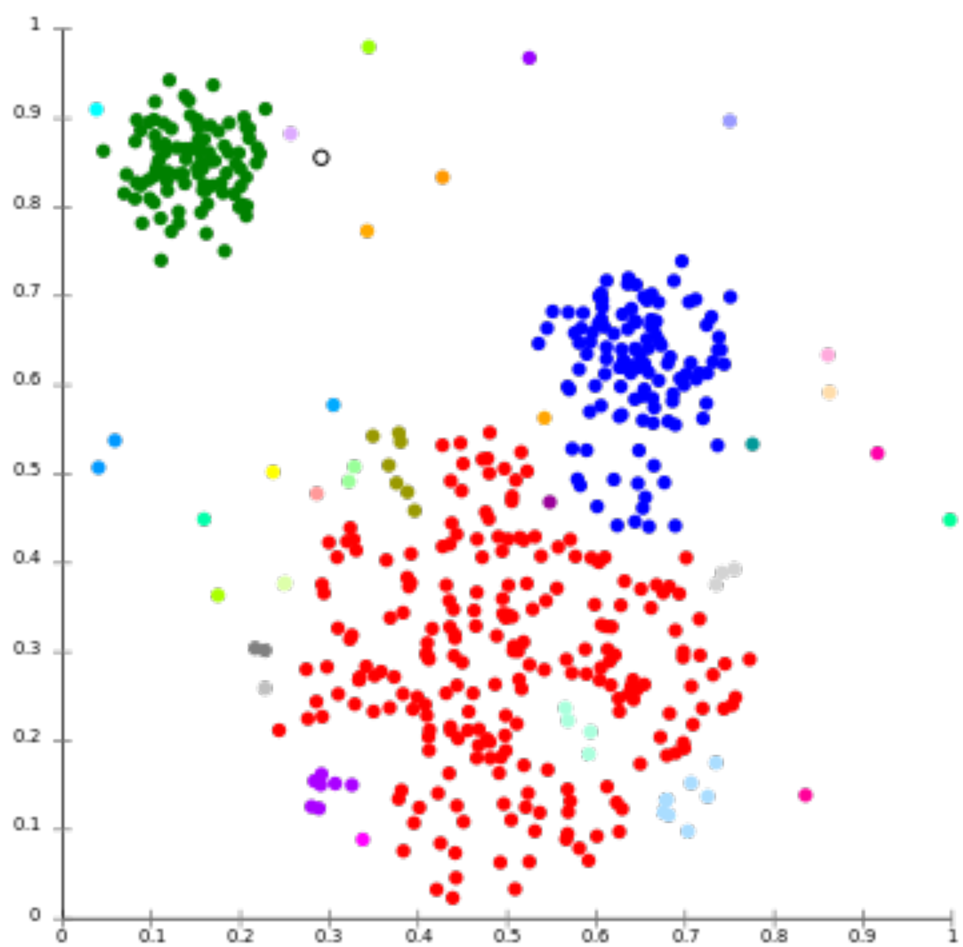
1. *Centroid-based (K-means, K-medoids)*

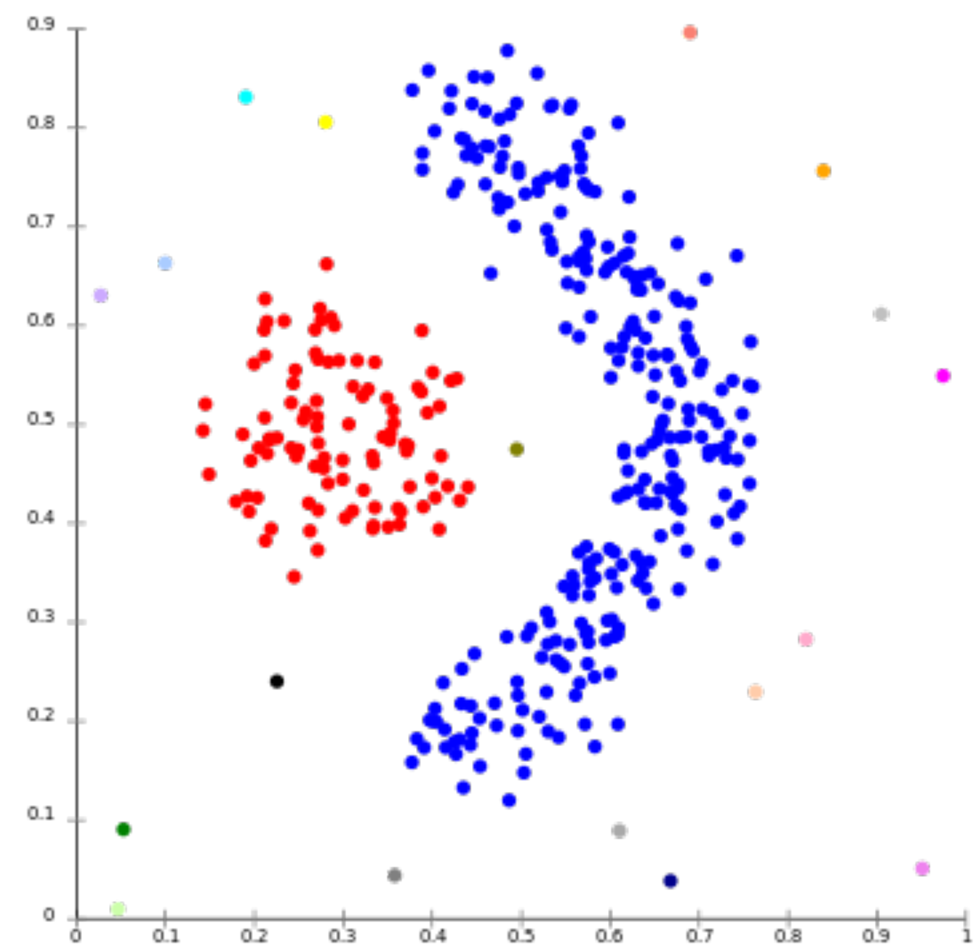




Four Types of Clustering

2. *Connectivity-based (Hierarchical)*

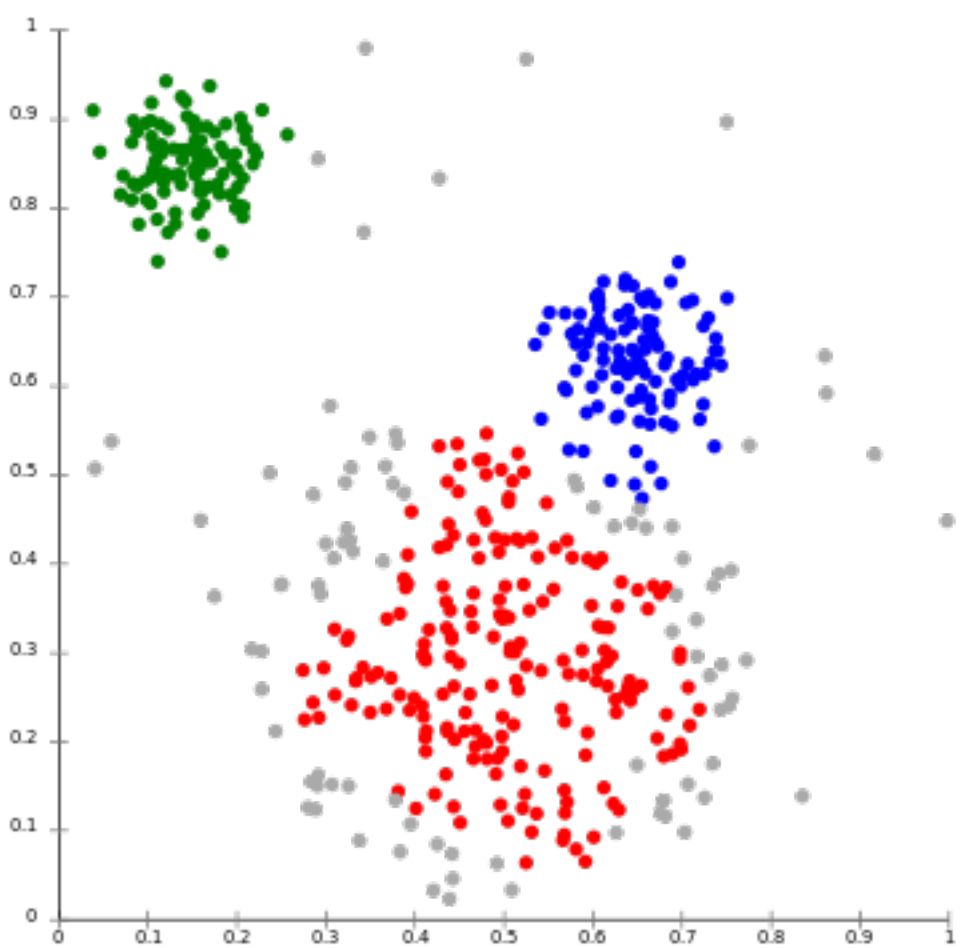


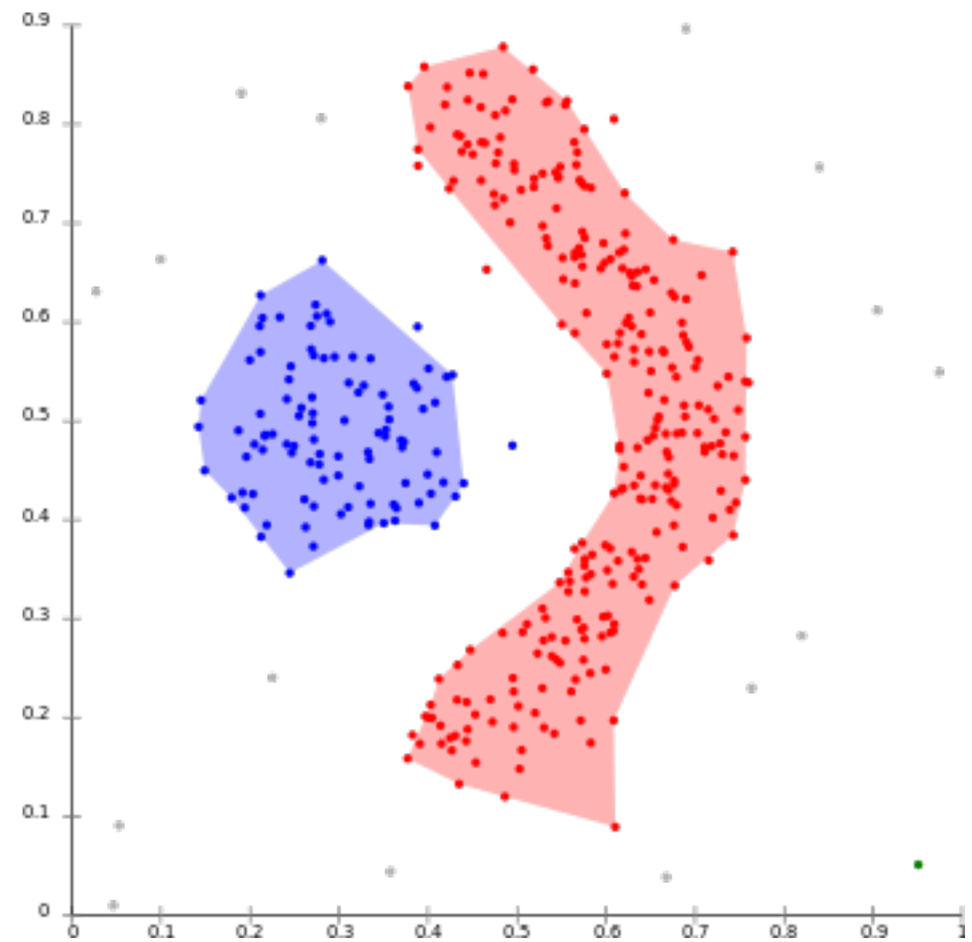


Notion of Clusters: Cut off dendrogram at some depth

Four Types of Clustering

3. *Density-based (DBSCAN, OPTICS)*

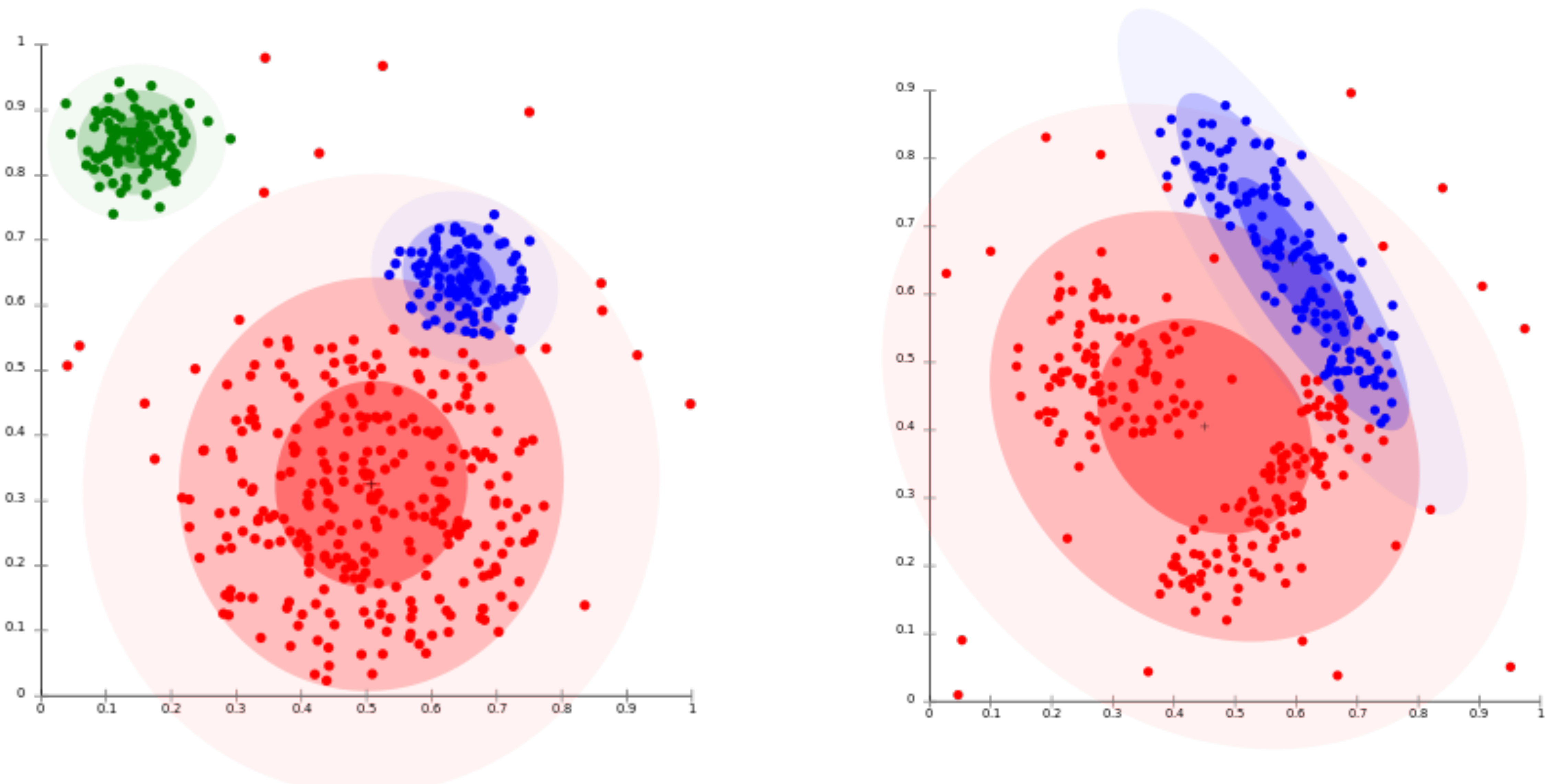




Notion of Clusters: Connected regions of high density

Four Types of Clustering

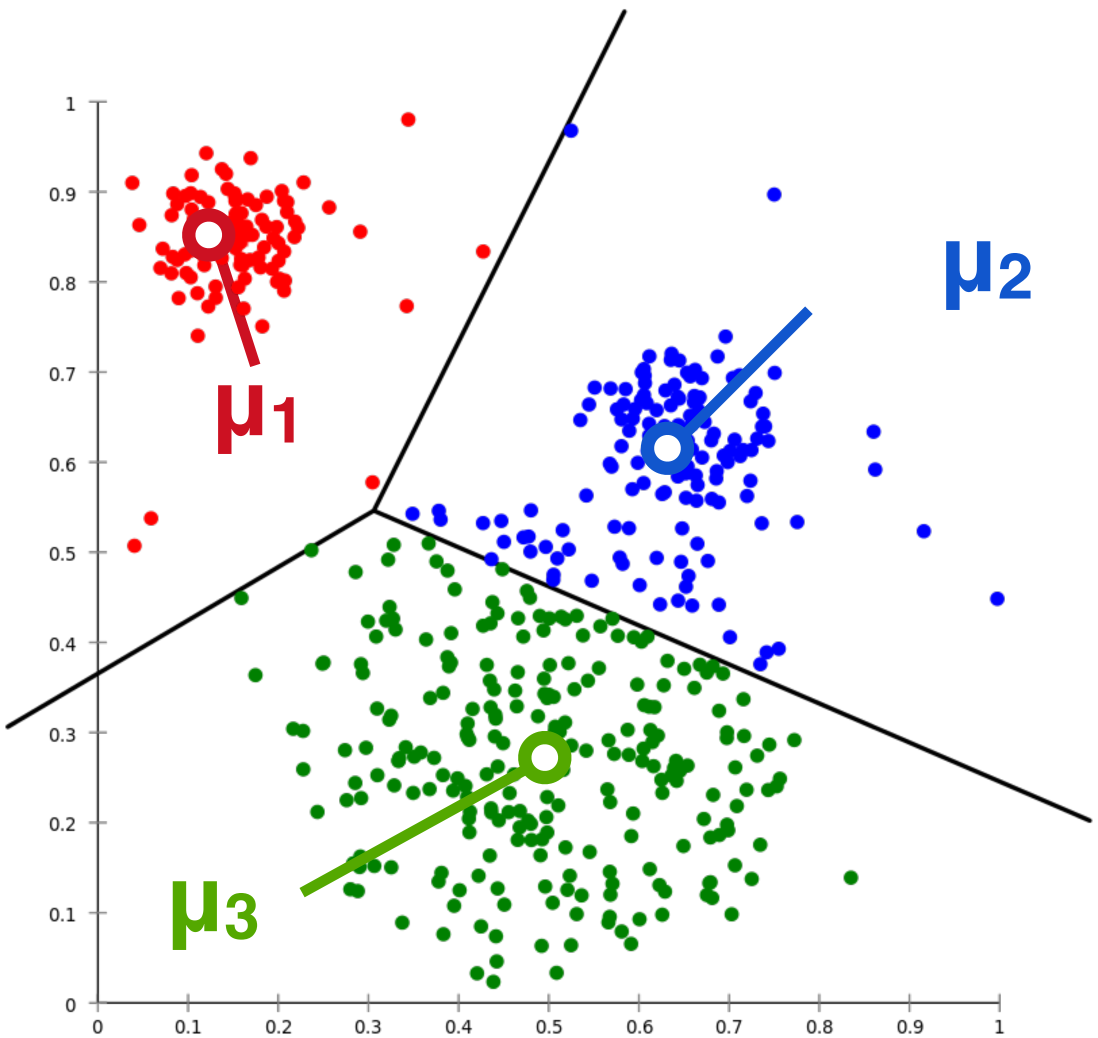
4. *Distribution-based (Mixture Models)*

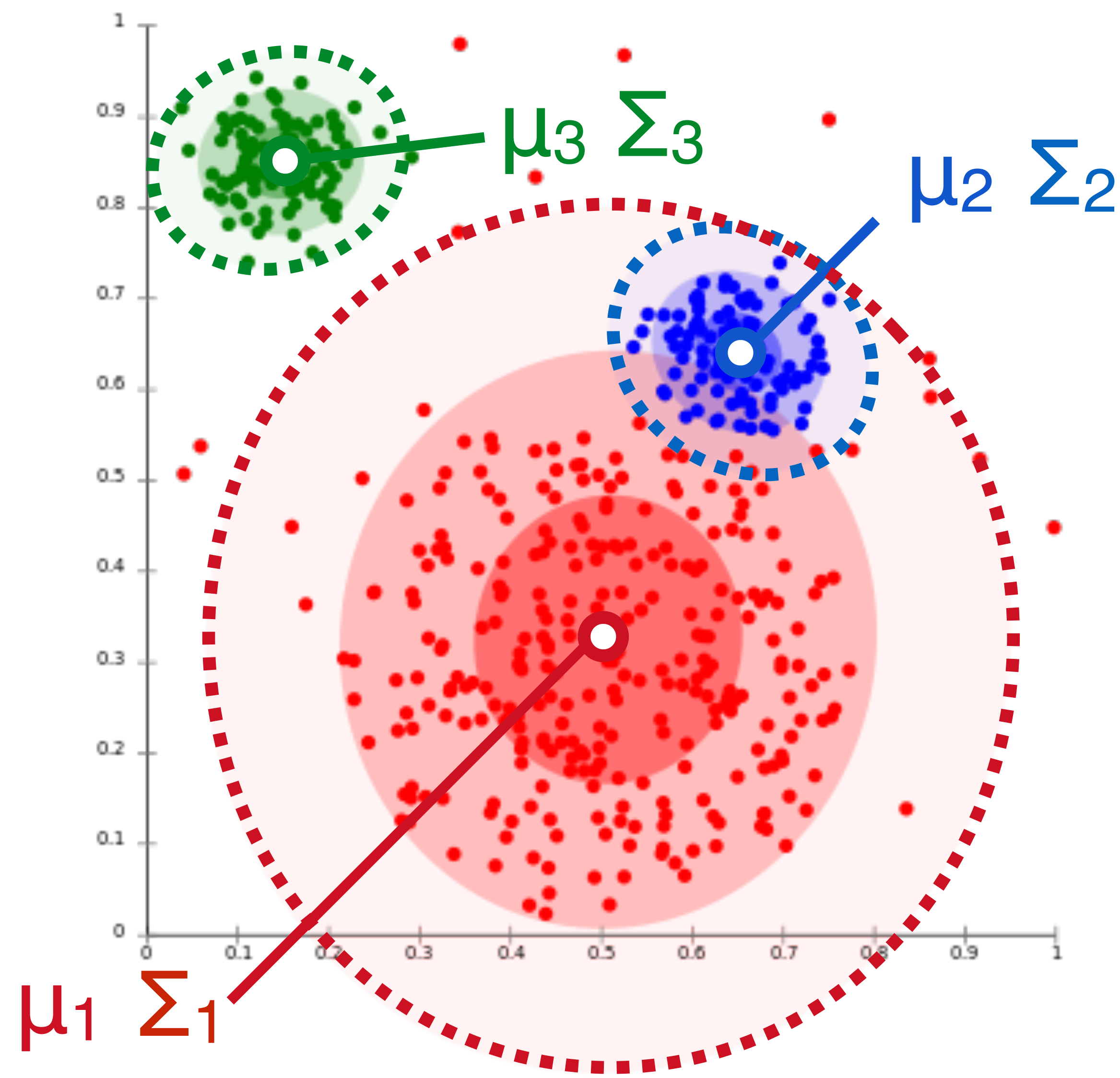


Notion of Clusters: Distributions on features

From K-Means —> *Gaussian Mixture Models*

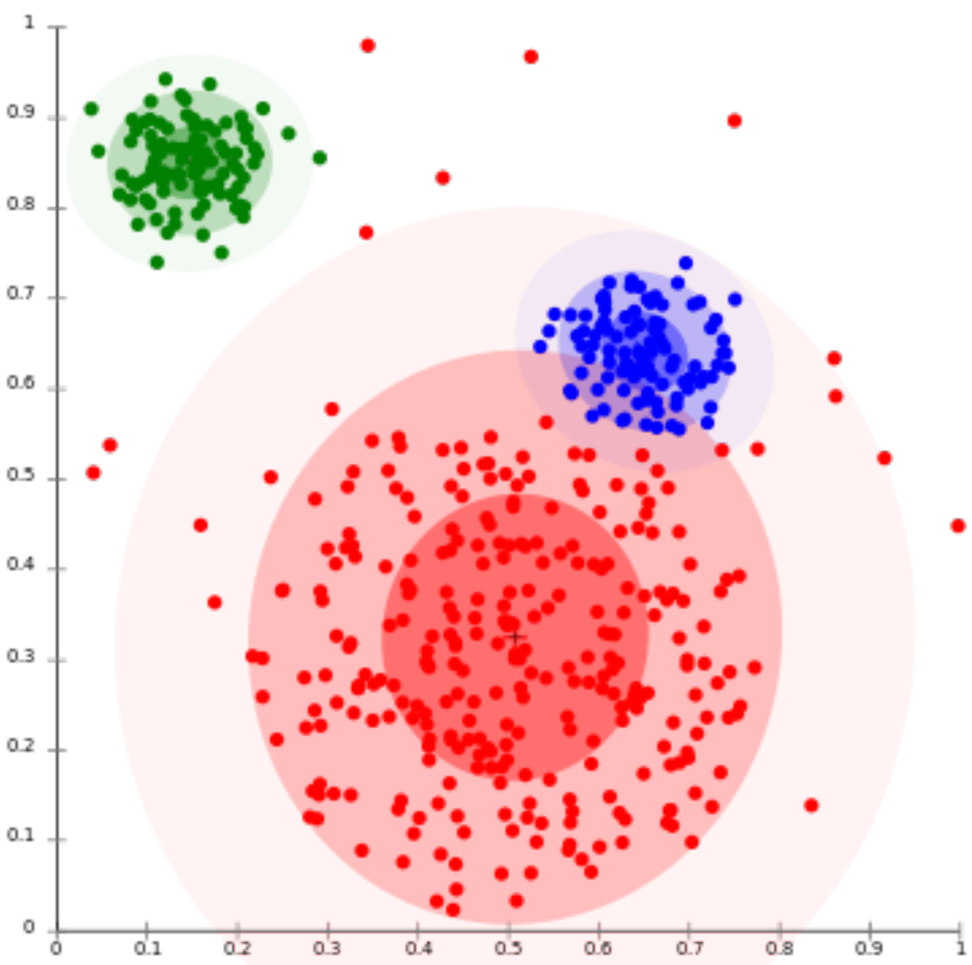
Idea: Learn both means μ_k and covariances Σ_k





Don't just learn *where* the center of the cluster is, but also *how big it is*, and *what shape it has*.

"Hard EM" with Gaussians



Assignment Update

$$z_n = \underset{k}{\operatorname{argmax}}\, p(z_n = k \,|\, \boldsymbol{x}_n, \boldsymbol{\theta})$$

Parameter Updates

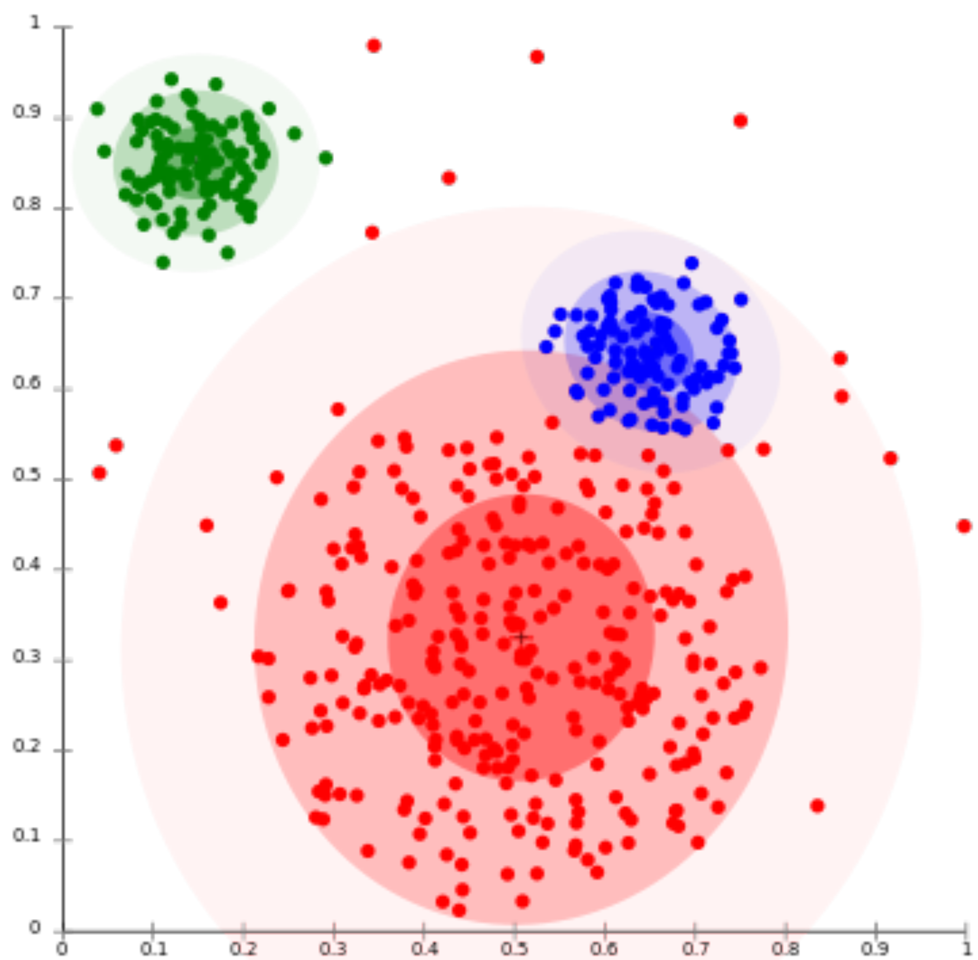
$$N_k := \sum_{n=1}^{N} z_{nk} \qquad z_{nk} := I[z_n = k]$$

$$\boldsymbol{\pi} = (N_1/N, \ldots, N_K/N)$$

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} z_{nk}\, \boldsymbol{x}_n$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} z_{nk}\, (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top$$

Gaussian Mixture Models



Soft **Assignment Update**

$$\boxed{\gamma_{nk} := p(z_n = k \mid \boldsymbol{x}_n, \boldsymbol{\theta})}$$

Parameter Updates

$$N_k := \sum_{n=1}^{N} \boxed{\gamma_{nk}}$$

$$\boldsymbol{\pi} = (N_1/N, \ldots, N_K/N)$$

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \boxed{\gamma_{nk}} \boldsymbol{x}_n$$

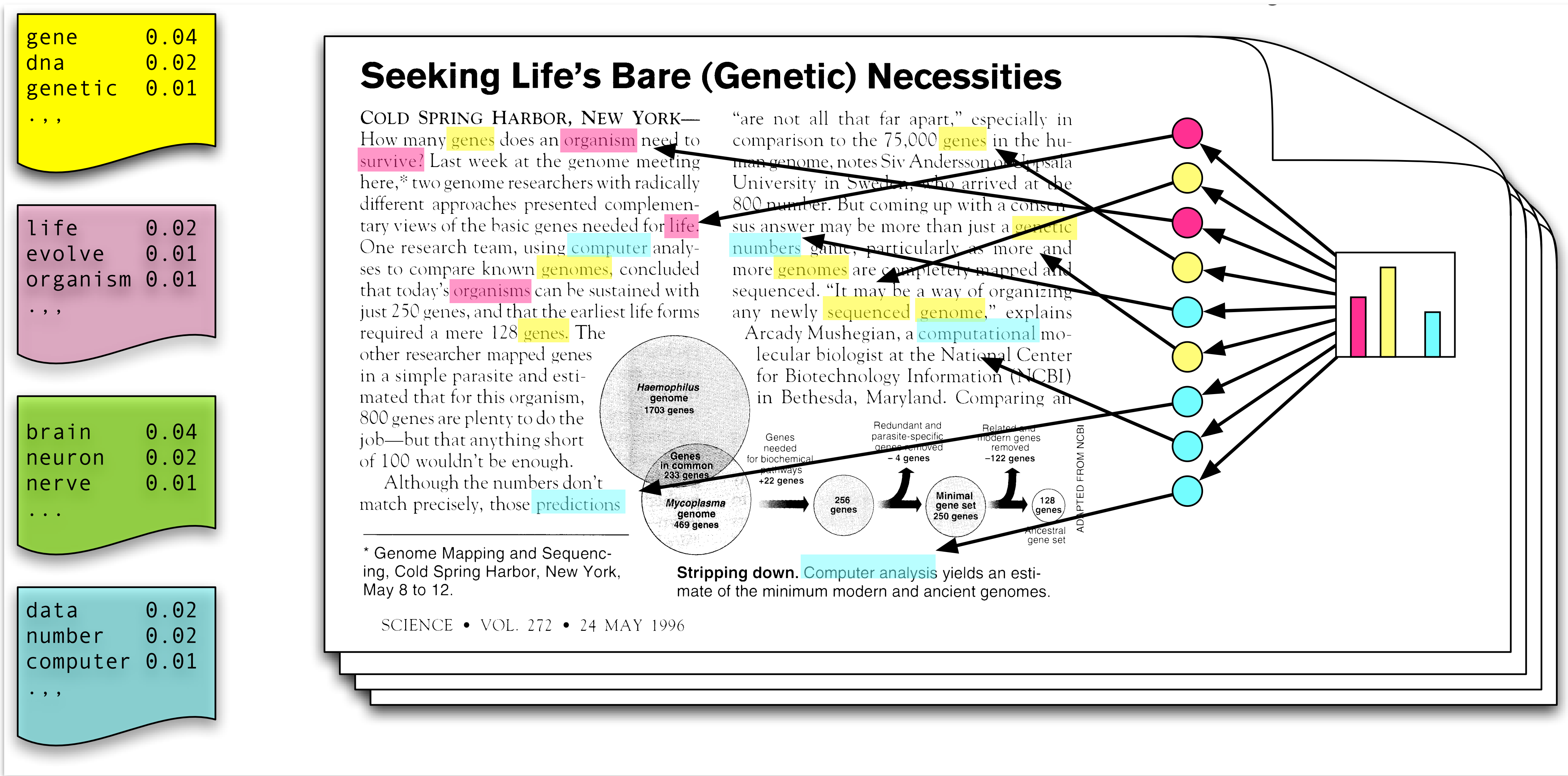
$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \boxed{\gamma_{nk}} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\top}$$

Idea: Replace **hard** assignments with **soft** assignments

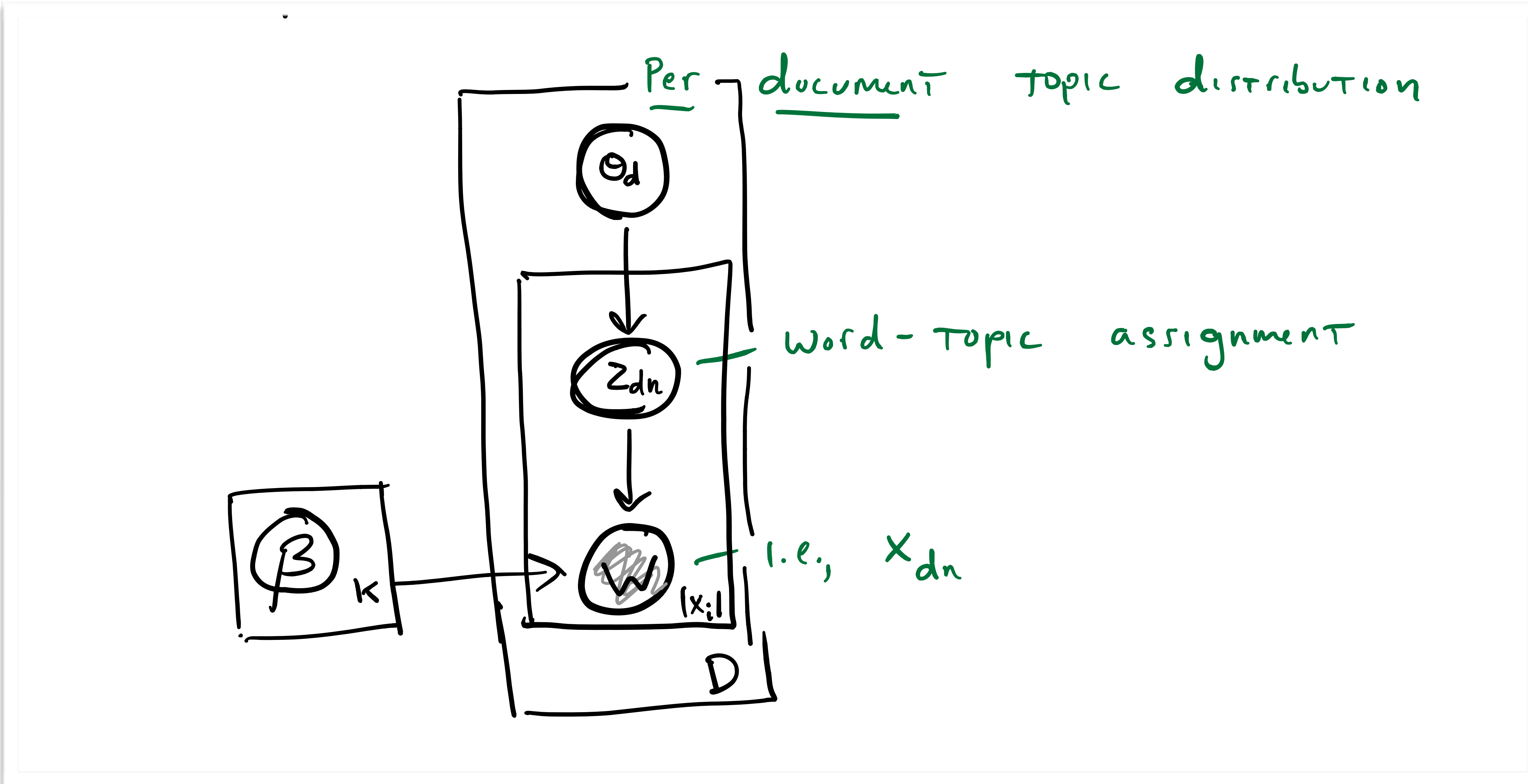
Topic modeling

Topic Modeling

Topics (shared) | **Words in Document** (mixture over topics) | **Topic Proportions** (document-specific)



- Each **topic** is a distribution over words
- Each **document** is a mixture over topics
- Each **word** is drawn from one topic distribution



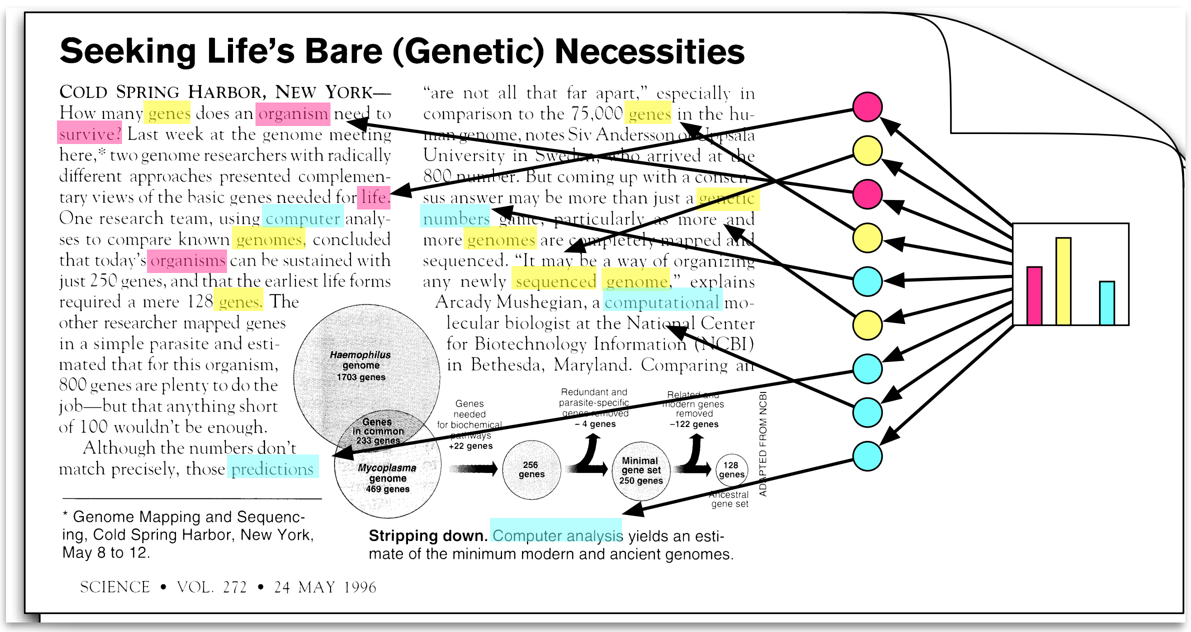
EM for topic models —> PLSA

EM for Word Mixtures (PLSA)

Generative Model

$$z_n \sim \mathrm{Discrete}(\boldsymbol{\theta})$$

$$x_n | z_n = k \sim \mathrm{Discrete}(\boldsymbol{\beta}_k)$$



Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

E-step: Update assignments

$$\phi_{nk} = \frac{\theta_k \beta_{kv}}{\sum_l \theta_l \beta_{lv}} \qquad x_v = v$$

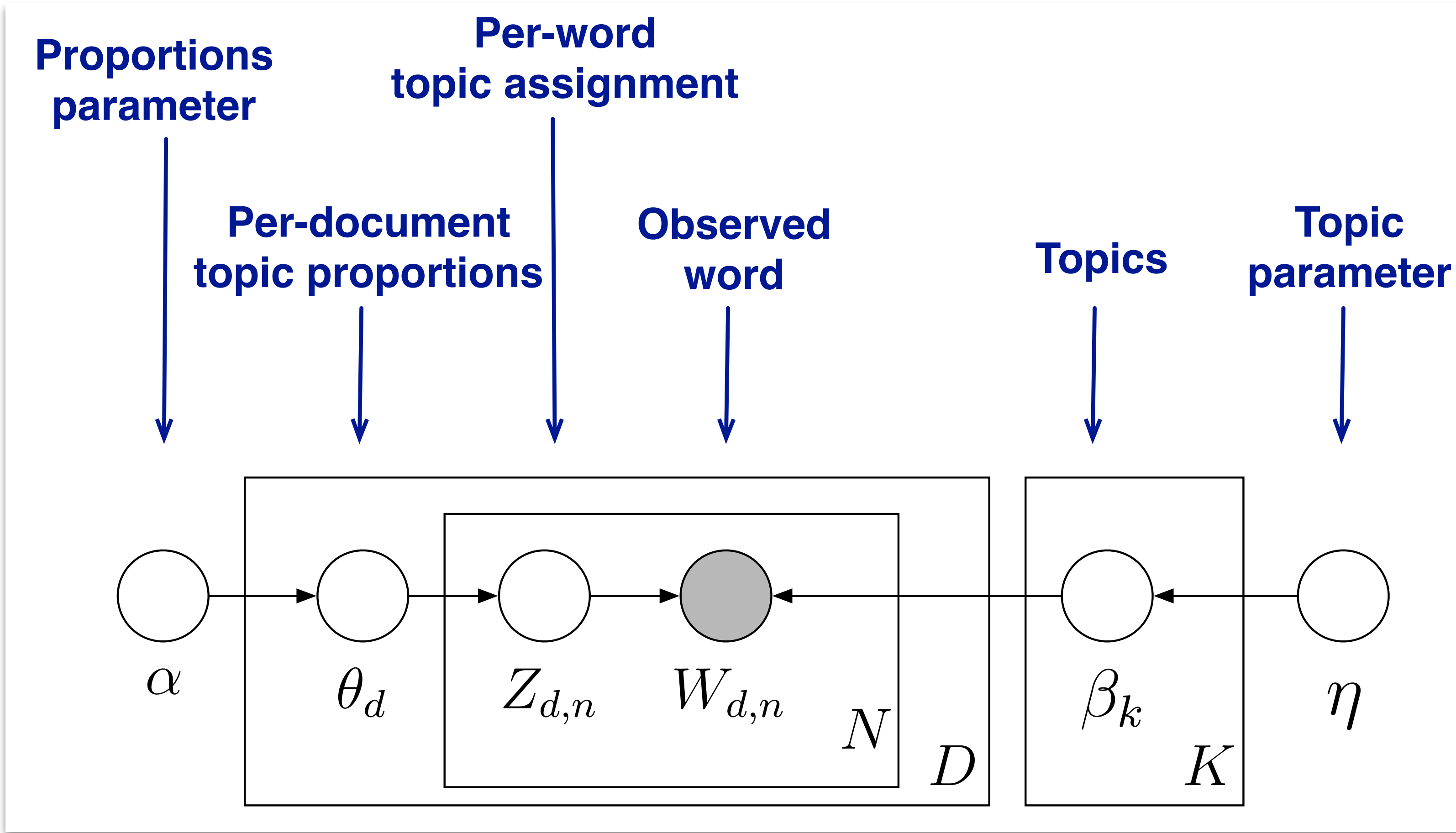
M-step: Update parameters

$$\beta_{kv} = \frac{N_{kv}}{\sum_w N_{kw}} \qquad N_{kv} := \sum_{n=1}^{N} \phi_{nk} x_{nv}$$

$$\theta_k = \frac{N_k}{\sum_l N_l} \qquad N_k := \sum_{n=1}^{N} \phi_{nk}$$

Latent Dirichlet Allocation

(a.k.a. PLSI/PLSA with priors)



$$\beta_k \sim \text{Dirichlet}(\eta) \quad k = 1, \ldots, K$$

$$\theta_d \sim \text{Dirichlet}(\alpha) \quad d = 1, \ldots, D$$

$$Z_{d,n} \sim \text{Discrete}(\theta_d) \quad n = 1, \ldots, N_d$$

$$W_{d,n} | Z_{d,n} = k \sim \text{Discrete}(\beta_k) \quad n = 1, \ldots, N_d$$

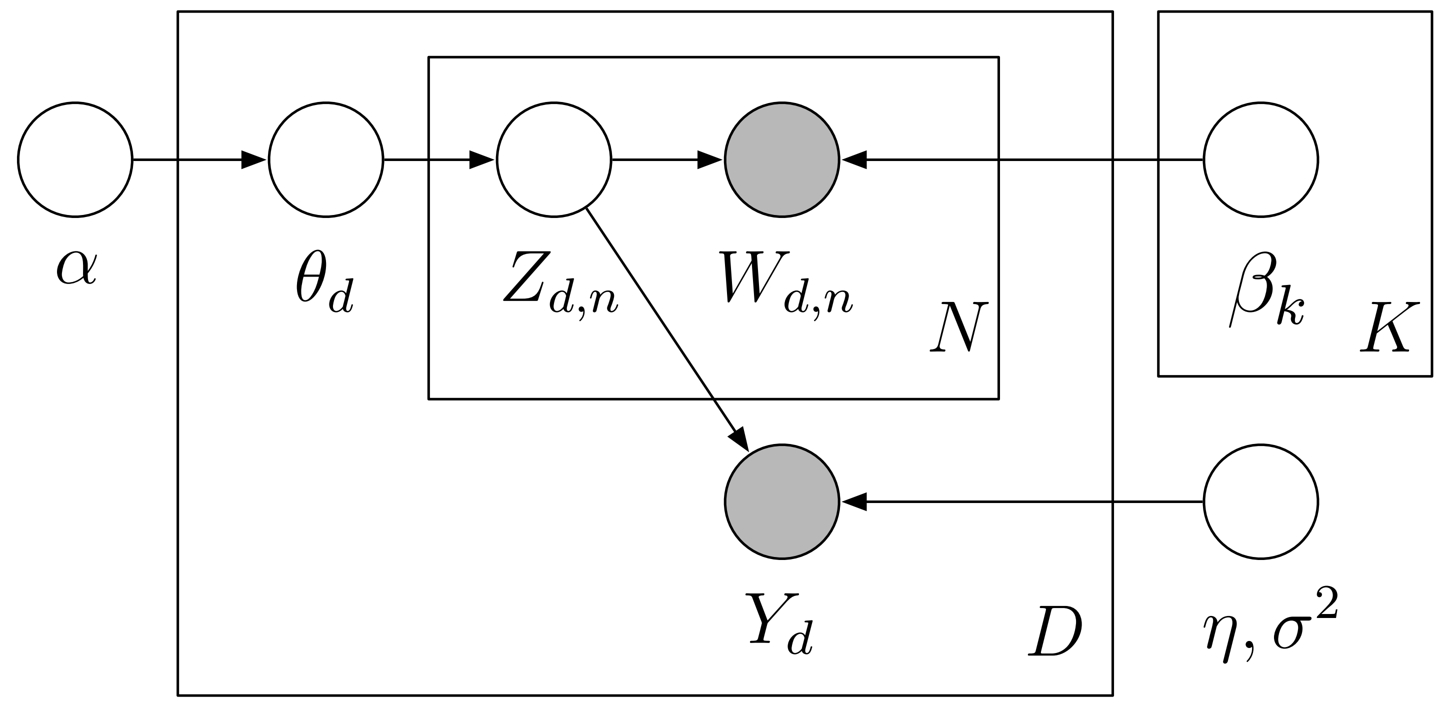
Estimation: Gibbs sampling

Initialization: Initialize $\mathbf{x}^{(0)} \in \mathcal{R}^D$ and number of samples N

- **for** $i = 0$ to $N-1$ **do**
- $x_1^{(i+1)} \sim p(x_1 | x_2^{(i)}, x_3^{(i)}, ..., x_D^{(i)})$
- $x_2^{(i+1)} \sim p(x_2 | x_1^{(i+1)}, x_3^{(i)}, ..., x_D^{(i)})$
- $\vdots$
- $x_j^{(i+1)} \sim p(x_j | x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, ..., x_D^{(i)})$
- $\vdots$
- $x_D^{(i+1)} \sim p(x_D | x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{D-1}^{(i+1)})$

return $(\{\mathbf{x}^{(i)}\}_{i=0}^{N-1})$

Extensions: Supervised LDA



1. Draw topic proportions $\theta \mid \alpha \sim \mathrm{Dir}(\alpha)$.
2. For each word
   - Draw topic assignment $z_n \mid \theta \sim \mathrm{Mult}(\theta)$.
   - Draw word $w_n \mid z_n, \beta_{1:K} \sim \mathrm{Mult}(\beta_{z_n})$.
3. Draw response variable $y \mid z_{1:N}, \eta, \sigma^2 \sim \mathrm{N}\left(\eta^\top \bar{z}, \sigma^2\right)$, where

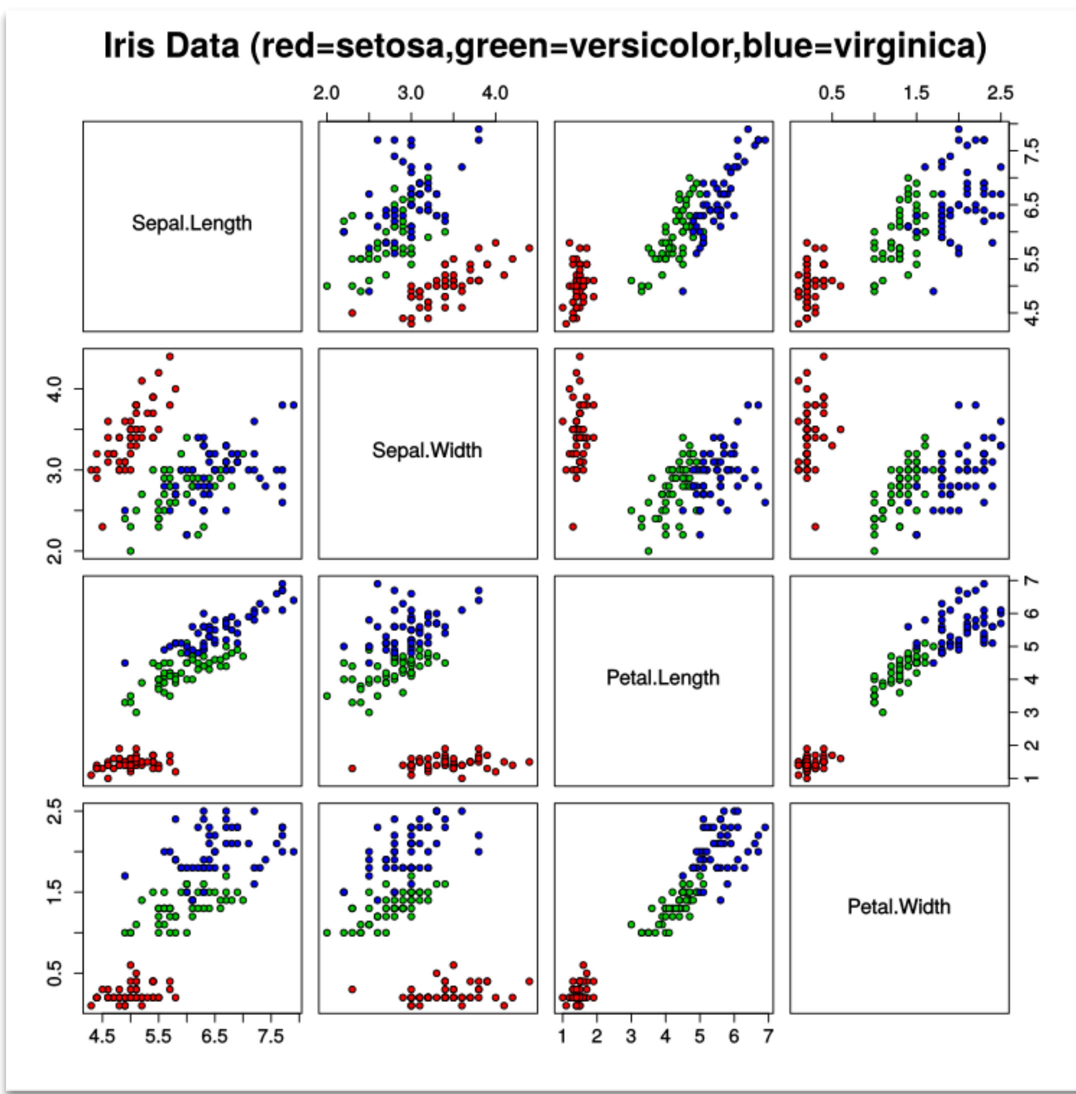
$$\bar{z} = (1/N) \sum_{n=1}^{N} z_n.$$

Dimensionality reduction

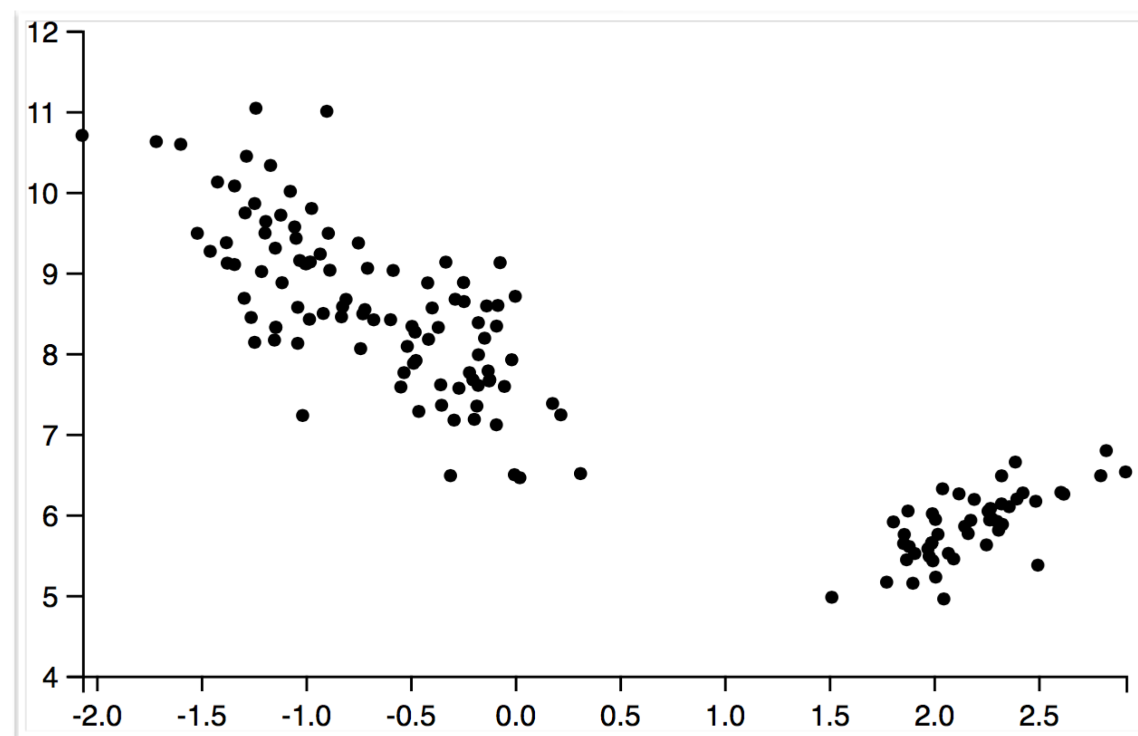
Dimensionality reduction

Goal: Map high dimensional data onto lower-dimensional data in a manner that preserves *distances/similarities*

Original Data (4 dims)



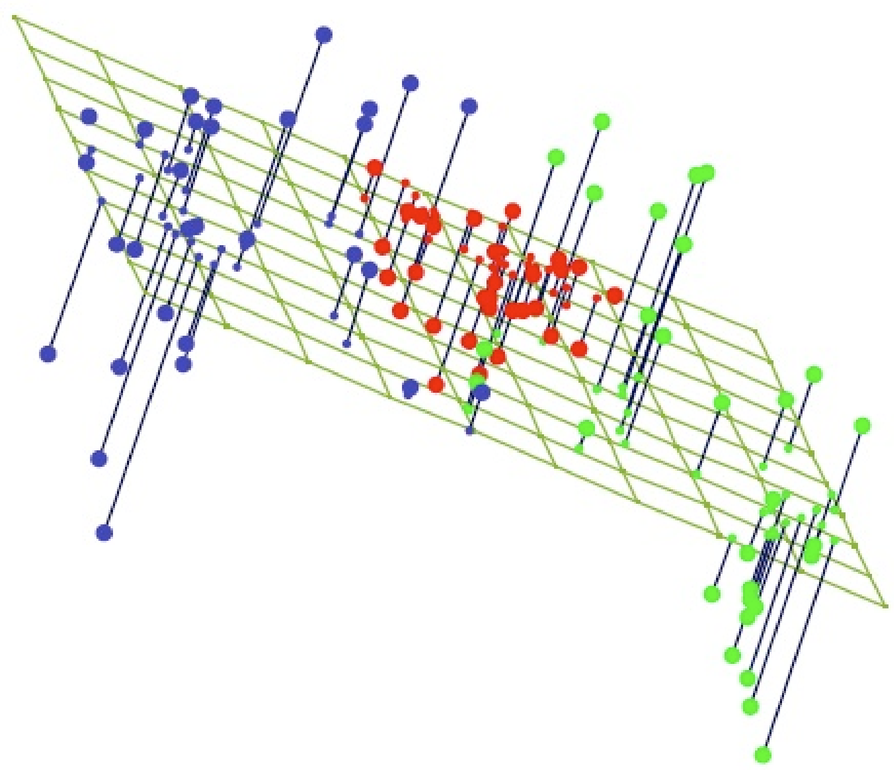
Projection with PCA (2 dims)



Objective: projection should "preserve" relative distances

Linear dimensionality reduction

Idea: Project high-dimensional vector
onto a lower dimensional space



In Sum: Principal Component Analysis

Data

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \cdots\cdots & \mathbf{x}_n \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times n}$$

Orthonormal Basis

$$\mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdot\cdot & \mathbf{u}_d \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times d}$$

Eigenvectors of Covariance

$$\mathbf{C} = \frac{1}{n} \sum_{j=1}^{n} \mathbf{x}_j \mathbf{x}_j^\top = \frac{1}{n} \mathbf{X}\mathbf{X}^\top$$

$$\mathbf{C}\mathbf{u}_j = \lambda_j \mathbf{u}_j$$

Eigen-decomposition

$$\boldsymbol{C} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top$$

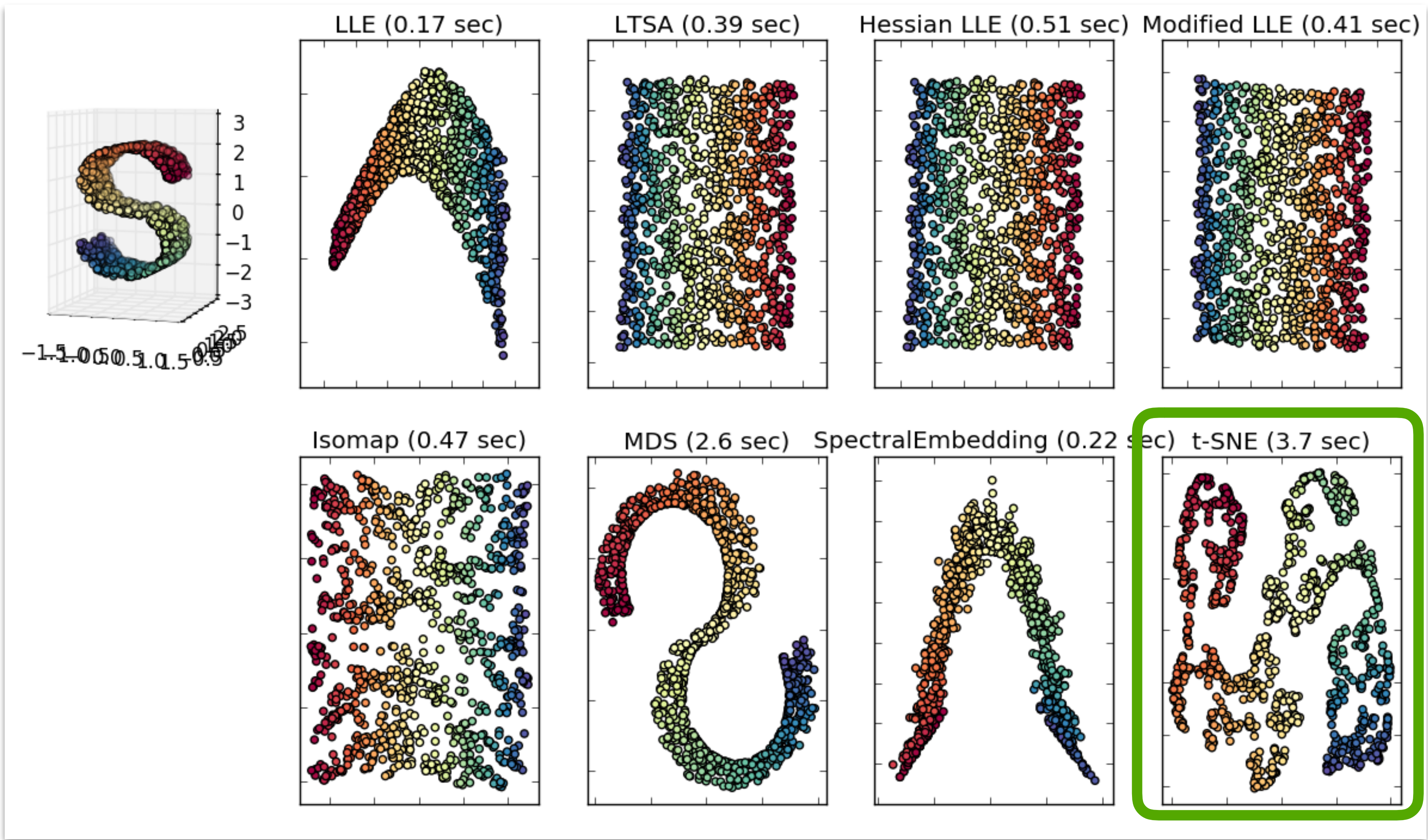
$$\boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ldots & \\ & & & \lambda_d \end{pmatrix}$$

Idea: Take **top-*k*** eigenvectors to maximize variance

Probabilistic PCA

- If we define a *prior* over *z* then we can **sample** from the latent space and hallucinate images

Non-linear reduction



Visualizing data using t-SNE

L Maaten, G Hinton - Journal of machine learning research, 2008 - jmlr.org Paperpile

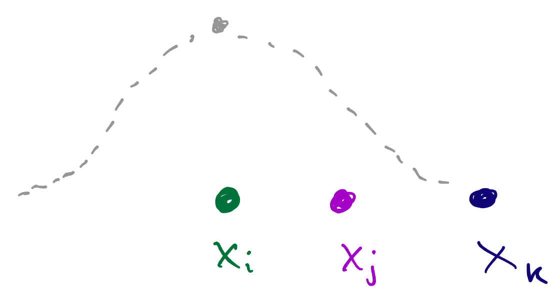
We present a new technique called" **t-SNE**" that visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map. The technique is a variation of Stochastic Neighbor Embedding (Hinton and Roweis, 2002) that is much easier to optimize …

Cited by 11621 Related articles All 39 versions Import into BibTeX

Define a conditional probability that encodes similarity

$$P_{j|i} = \frac{\exp\{-\|x_i - x_j\|^2 / 2\sigma_i^2\}}{\sum_{k \neq i} \exp\{-\|x_i - x_k\|^2 / 2\sigma_i^2\}}$$

This is in high dim space



Similarly in the map y

$$q_{j|i} = \frac{\exp\{-\|y_i - y_j\|^2\}}{\sum_{k \neq i} \exp\{-\|y_i - y_k\|^2\}}$$

Ideally: $P_{j|i} \stackrel{\sim}{=} q_{j|i} \quad \forall i,j$

Auto-encoders

Auto-Encoders

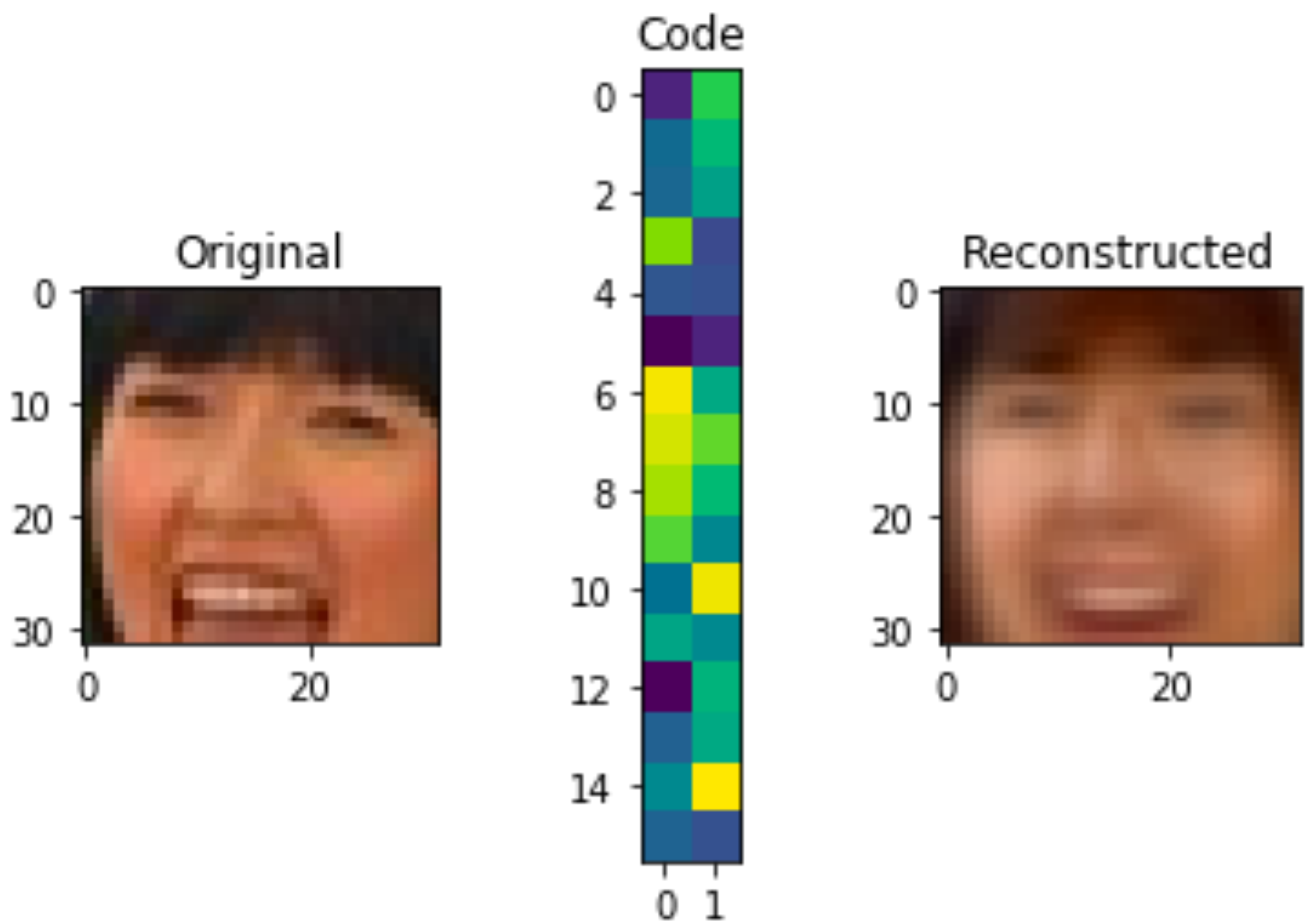
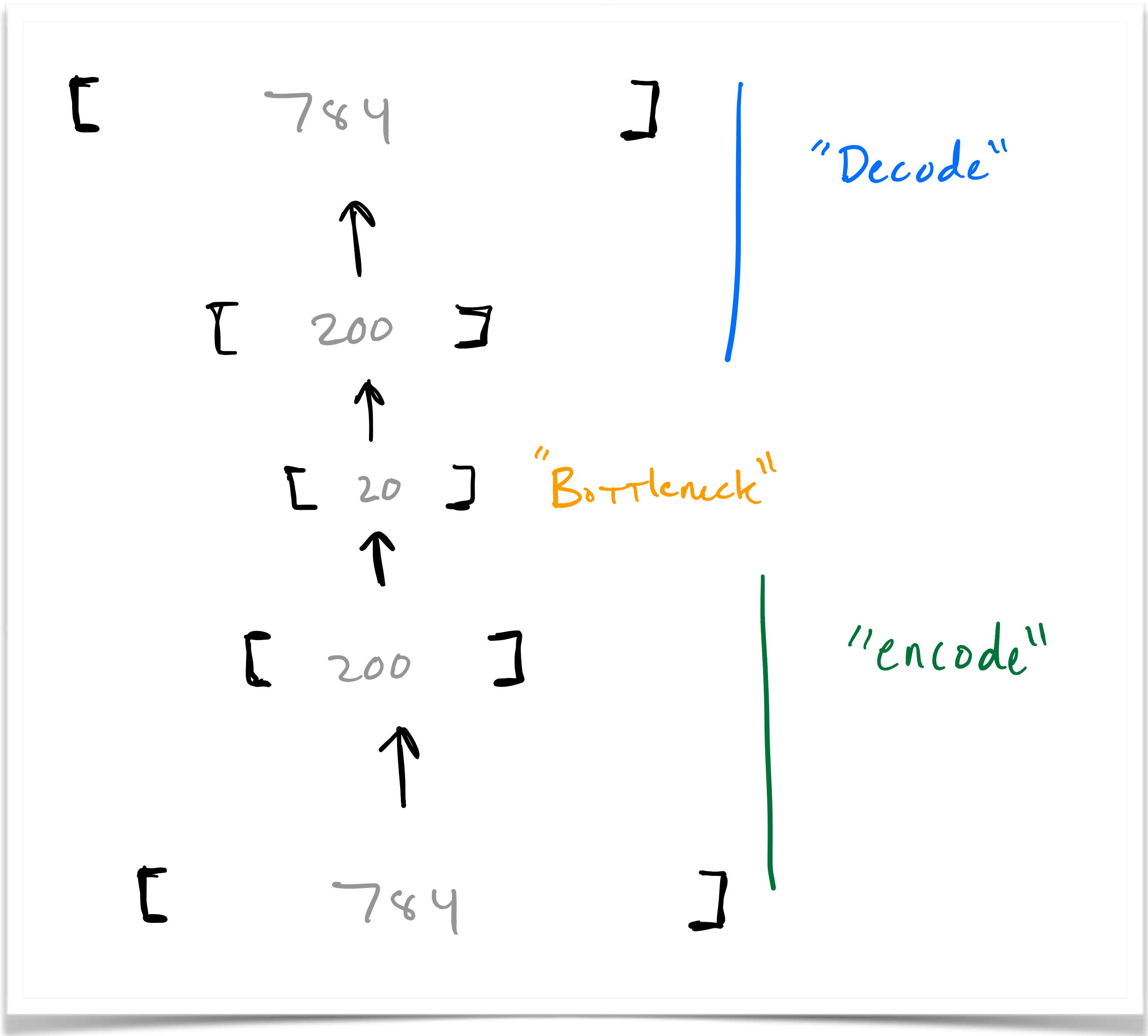


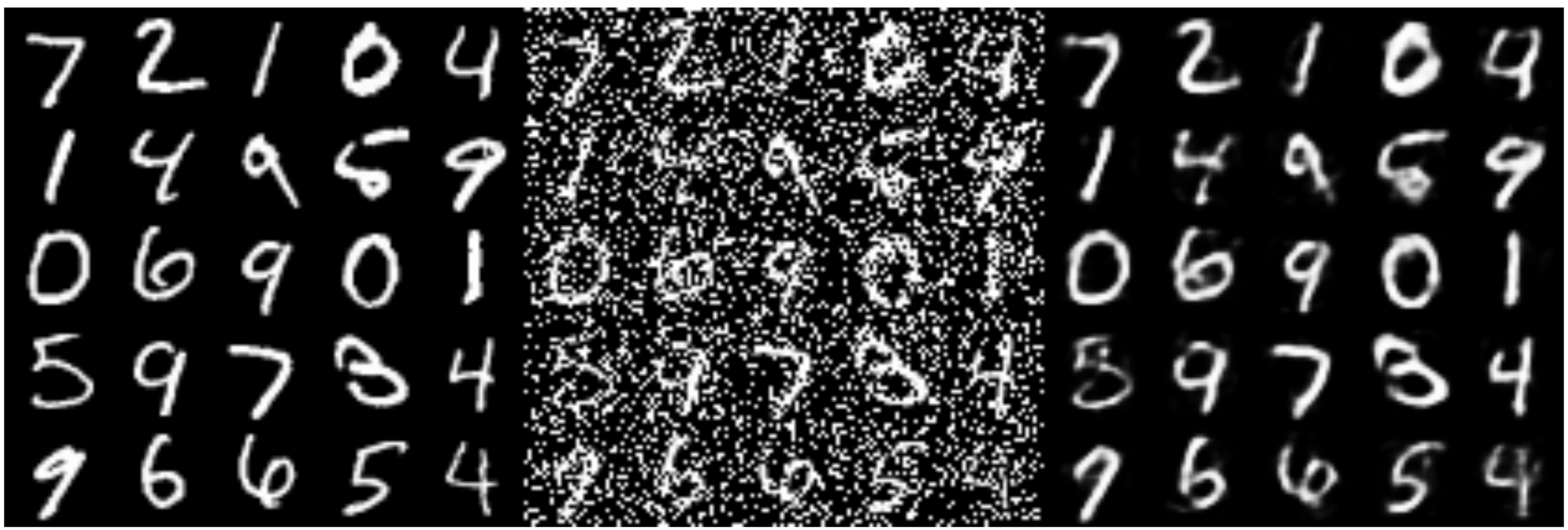
Figure credit: https://stackabuse.com/autoencoders-for-image-reconstruction-in-python-and-keras/



$$L(\boldsymbol{x}, g(f(\boldsymbol{x})))$$

Denoising auto-encoders

x $\quad x'$ $\quad g(f(x))$



Copyright by opendeep.org.

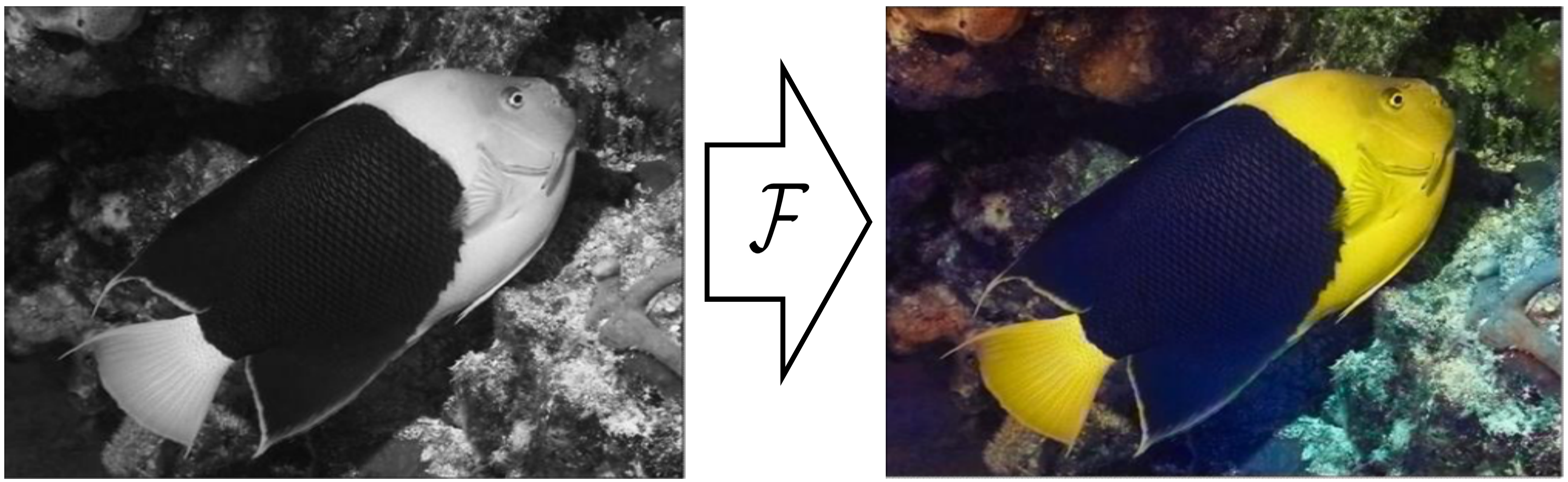
$$L(x, g(f(x')))$$

Self-supervision

- Self-supervision: A form of **unsupervised** learning in which the data itself provides the **supervision**
- Generally: Hide some aspect of the data, attempt to reconstruct it from the rest
- Formulating “good” self-training objectives is an active area of research!

Example: Colorizing

Train network to predict pixel colour from a monochrome input

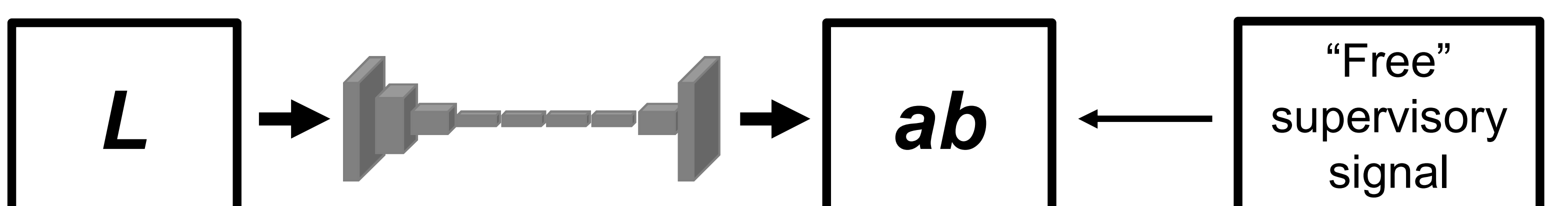


Grayscale image: *L* channel

$$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$$

Concatenate (L,ab)

$$(\mathbf{X}, \widehat{\mathbf{Y}})$$



Structured prediction

Structured output spaces

Source: http://cocodataset.org/

Structured output spaces

John	lives	in	New	York	and	works	for	the	European	Union
B-PER	O	O	B-LOC	I-LOC	O	O	O	O	B-ORG	I-ORG

Source: https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html

Designing features

$$\boldsymbol{x} = \text{“ monsters eat tasty bunnies “}$$

$$\boldsymbol{y} = \quad \text{noun} \;\; \text{verb adj} \quad \text{noun}$$

Want to design $\phi(x, y)$

Some possibilities

- # of times *w* gets label *l* (for all *w*, *l*) **Unary**
- # of times *l* is adjacent to *l’* (for all *l* and *l’*) **Markov**

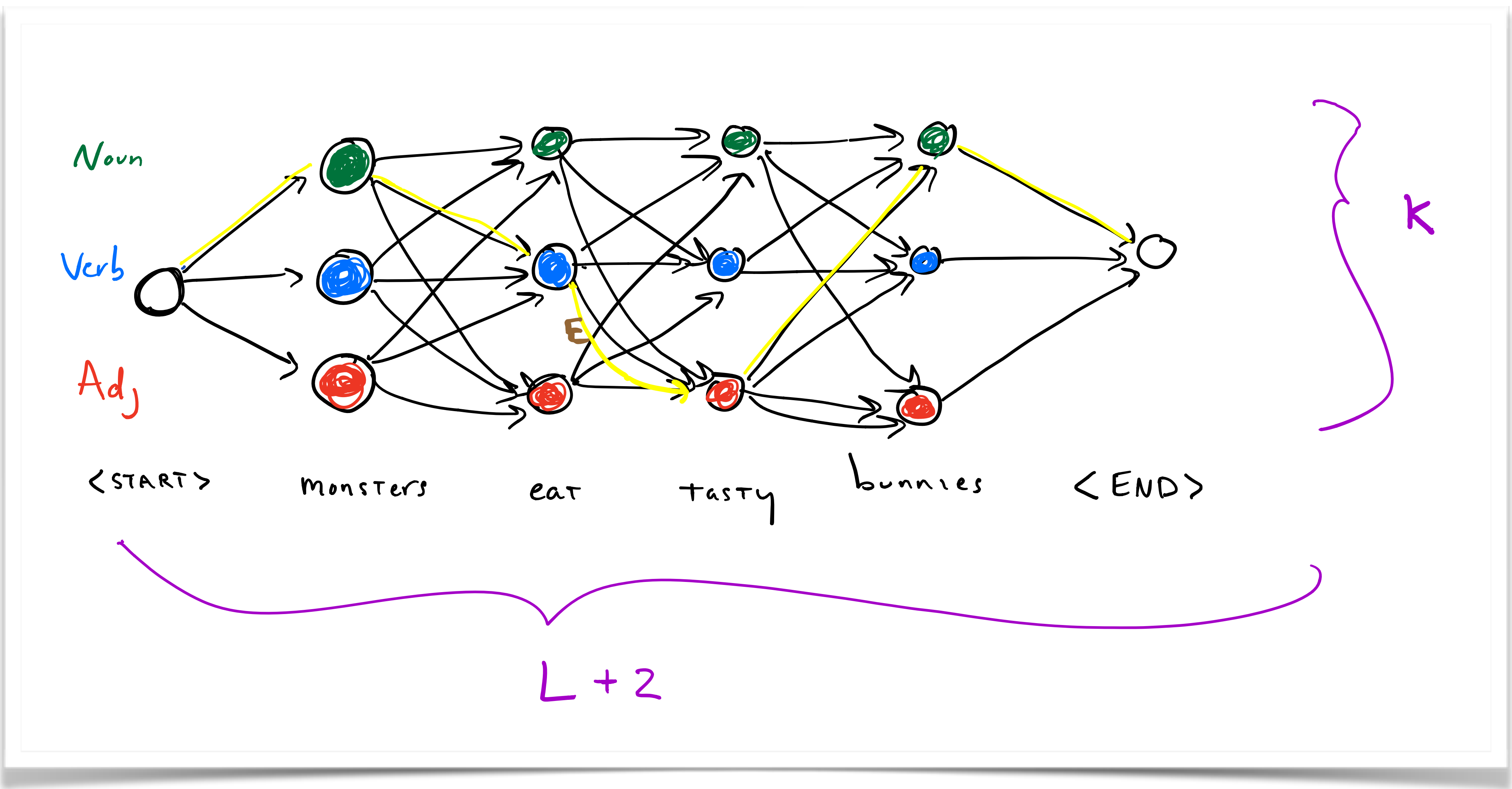
Algorithm 40 STRUCTUREDPERCEPTRONTRAIN($\mathbf{D}$, *MaxIter*)

```
w ← 0                                              // initialize weights
for iter = 1 ... MaxIter do
   for all (x,y) ∈ D do
      ŷ ← argmax_{ŷ∈𝒴(x)} w · φ(x, ŷ)              // compute prediction
      if ŷ ≠ y then
         w ← w + φ(x, y) − φ(x, ŷ)                  // update weights
      end if
   end for
end for
return w                                           // return learned weights
```

Algorithm 40 StructuredPerceptronTrain($\mathbf{D}$, *MaxIter*)

$w \leftarrow \mathbf{0}$ // initialize weights
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $\boxed{\hat{y} \leftarrow \operatorname{argmax}_{\hat{y} \in \mathcal{Y}(x)} w \cdot \phi(x, \hat{y})}$ // compute prediction
    if $\hat{y} \neq y$ **then**
      $w \leftarrow w + \phi(x, y) - \phi(x, \hat{y})$ // update weights
    end if
  end for
end for
return w // return learned weights

Viterbi



Modeling Sequences

$$P(X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_n = y_n)$$
$$= \prod_{i=1}^{n+1} P(y_i | y_{i-1}) \prod_{i=1}^{n} P(x_i | y_i)$$

Transition probability

Emission probability

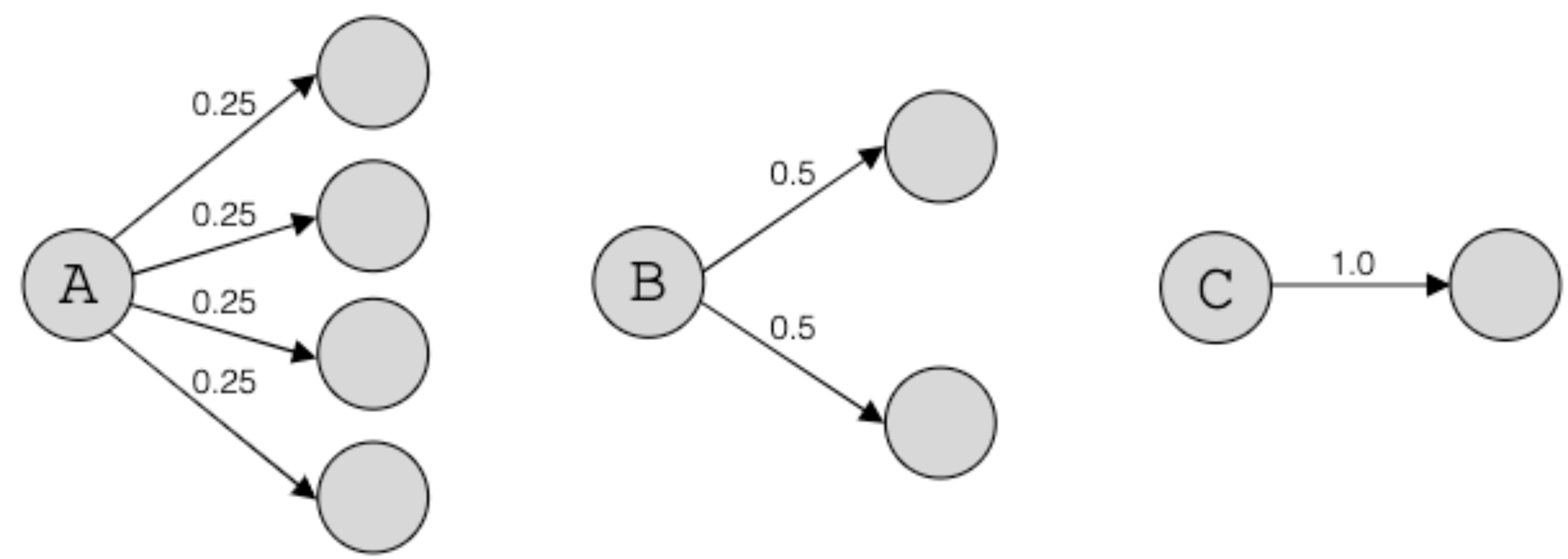
HMMs v MEMMs

HMM $p(y_i|y_{i-1})p(x_i|y_i)$

MEMM $$\frac{\exp(w \cdot \phi(x_1, ..., x_m, y_{i-1}, y_i))}{\sum_{y' \in \mathcal{Y}} \exp(w \cdot \phi(x_1, ..., x_m, y_{i-1}, y'))}$$

ϕ permits richer representations!

The “label bias” problem



An example of three states, A, B and C, which have uniform outgoing transition distributions. Label bias will cause the inference procedure to favor paths which go through state C.

Figure from Awni Hannun, https://awni.github.io/label-bias/

MEMMs vs CRFs

MEMMs *locally* normalize, chain together transition probabilities:

$$p(y|x) = \prod_{i}^{m} p(y_i|y_{i-1}, x_1, ...x_m)$$

$$\frac{\exp(w \cdot \phi(x_1, ..., x_m, y_{i-1}, y_i))}{\sum_{y' \in \mathcal{Y}} \exp(w \cdot \phi(x_1, ..., x_m, y_{i-1}, y'))}$$

CRFs *globally* normalize

$$p(y|x) = \frac{\exp\{\sum_i s(y_i, x_i, y_{i-1})\}}{\sum_{y'} \exp\{\sum_i s(y'_i, x_i, y'_{i-1})\}}$$

Beyond linear-chains

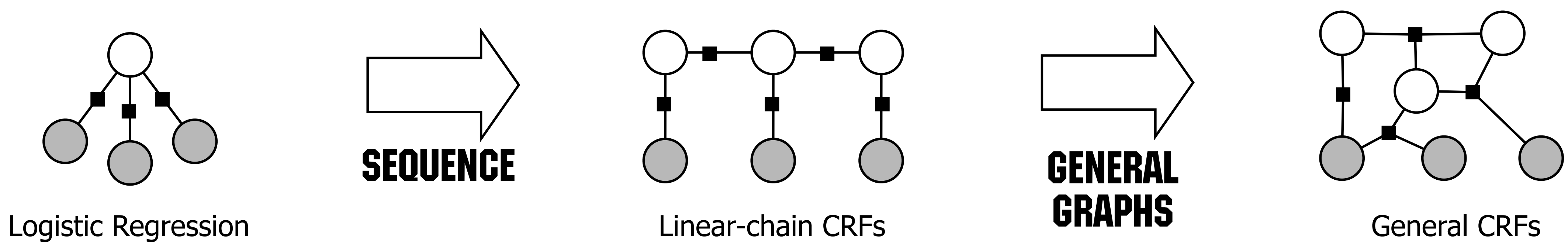


Figure from Sutton and McCallum, 2011