COMPUTER SCIENCE
INDIANA UNIVERSITY
School of Informatics and Computing
Bloomington

Hidden variable models

Reminders/Comments

- Your linear kernel on SUSY should perform ok
- The linear kernel is a dot product between input and center
  - Any confusion about centers?
- Issues?

Optimization approach

Algorithm 2: Batch Gradient Descent(Err, $\mathbf{X}, \mathbf{y}$)

// A non-optimized, basic implementation of batch gradient descent
$\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
err $\leftarrow \infty$
tolerance $\leftarrow 10e^{-4}$
while $|\text{Err}(\mathbf{w}) - \text{err}| > \text{tolerance}$ **do**
    err $\leftarrow \text{Err}(\mathbf{w})$
    $\mathbf{g} \leftarrow \nabla\text{Err}(\mathbf{w})$ ▷ for linear regression, $\nabla\text{Err}(\mathbf{w}) = \frac{1}{n}\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$
    // The step-size η could be chosen by line-search
    $\eta \leftarrow \text{line search}(\mathbf{w}, \mathbf{g}, \text{Err})$
    $\mathbf{w} \leftarrow \mathbf{w} - \eta\mathbf{g}$
return $\mathbf{w}$

- What happens if you just do 10 iterations?
- What happens if you just pick a fixed step-size, instead of using a better mechanism to pick the step-size?

Step-size approaches

- If you did the second-order update for logistic regression, you are already picking a fantastic step-size

- If you did standard batch (first-order) gradient descent, then you should use line search

- If you did stochastic gradient descent, you may have to fiddle a bit more, but likely should pick a smaller step-size and do more iterations (say at least 100x more iterations)

Optimization Questions

- If you initialize logistic regression randomly (entries in w samples from a normal Gaussian) and run gradient descent to within a tolerance of 10^-4, will it converge to the same solution across two runs on the same data?
- How about for an NN?
- What if you initialize NN with same random initialization, and use same order of data, will it converge to the same solution?
- Does each step of gradient descent decrease the objective?
- How about stochastic gradient descent?

Example of using an NN for digit recognition (MNIST)

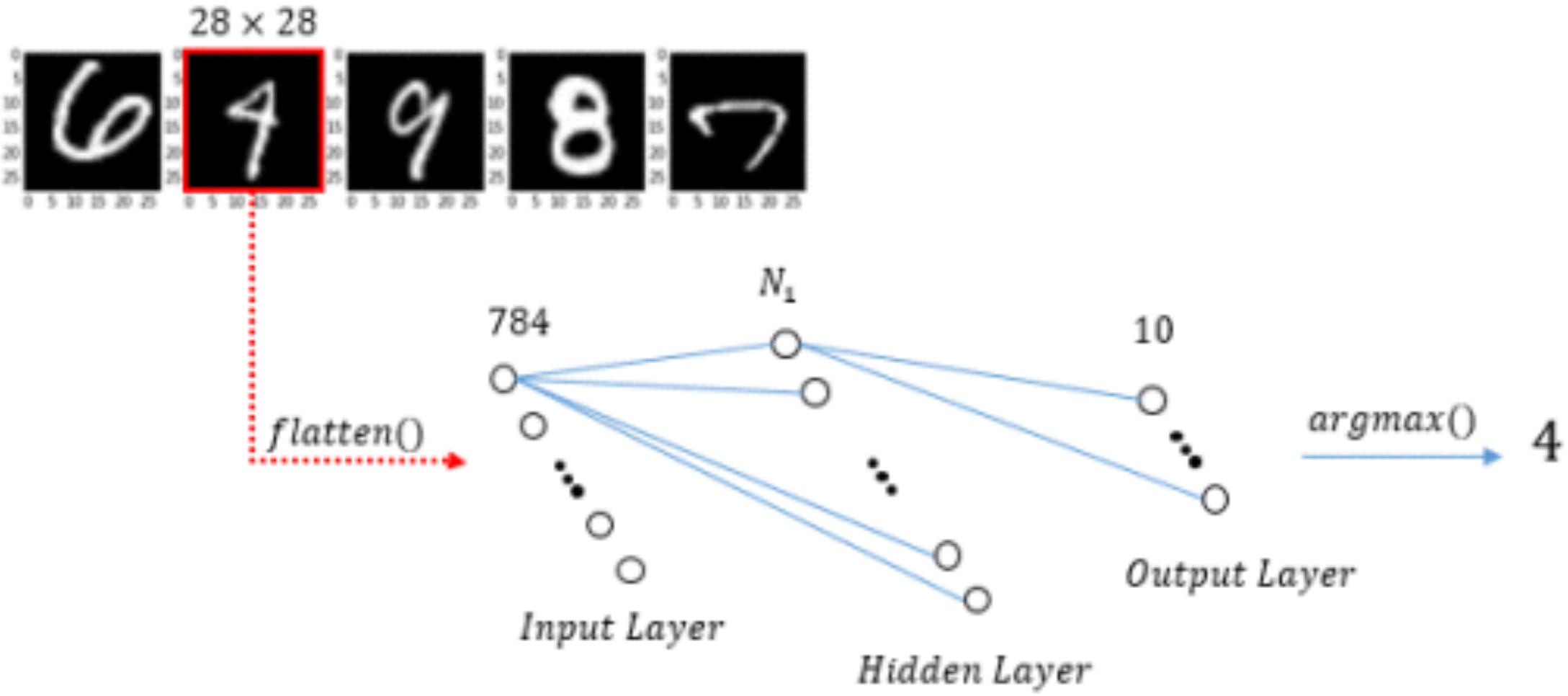
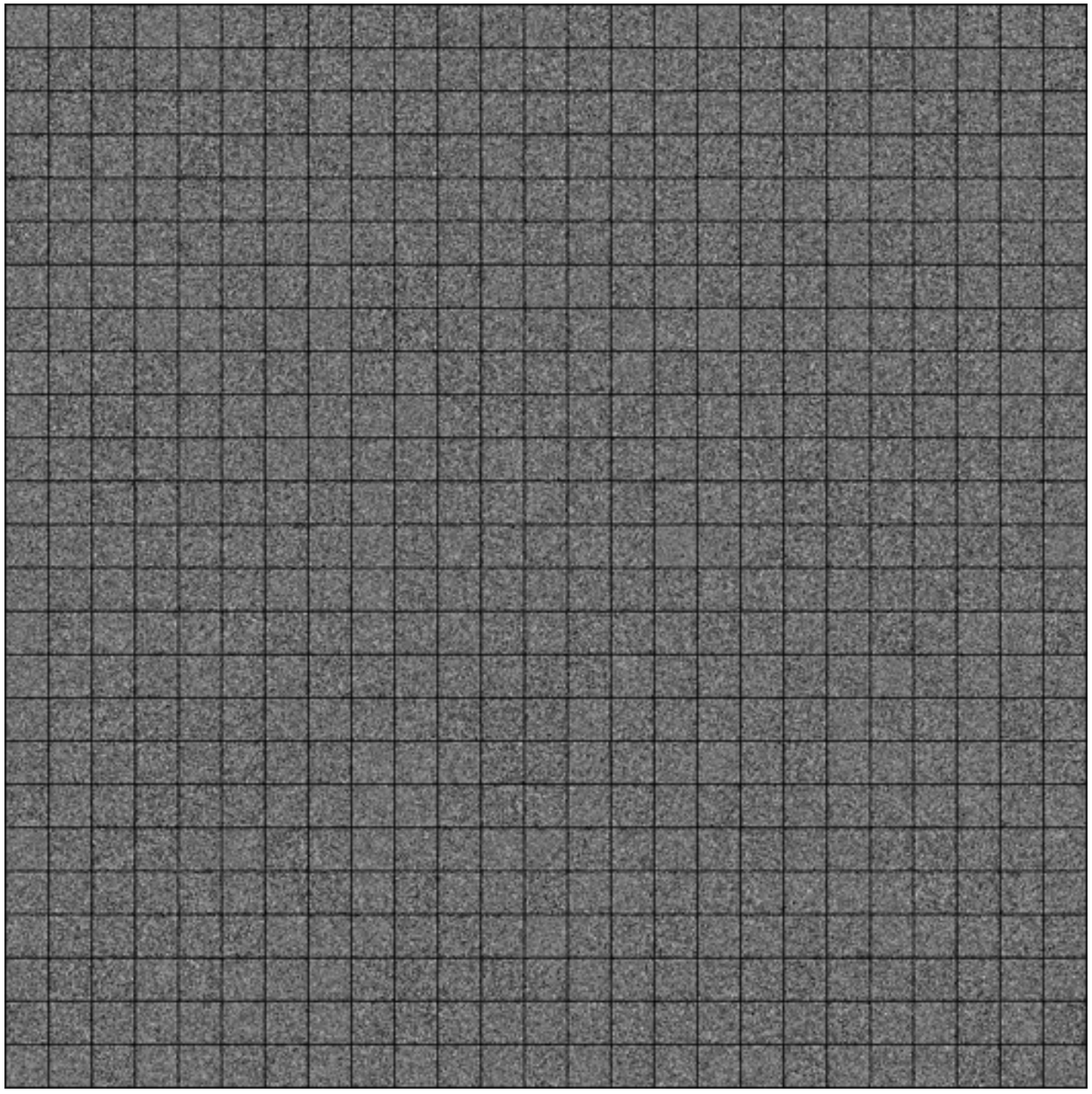


*image from: https://guillaumebrg.wordpress.com/2016/01/21/first-assignment-mlp-implementation-applied-to-digits-recognition-mnist-database/

Learned hidden layer



Weights initialized randomly

*from: http://nsphera.com/multi-layer-neural-network-in-octave/

Multi-variate output

- Can predict more than one value
- All the deltas for each of the output node is just then propagated back to the previous hidden layer
- Hidden layer adjusted based on delta for all output nodes, so representation that is learned is based on all outputs
- For MNIST, 10 outputs node, where node i = {0, …, 9} corresponds to the binary prediction “Is it digit i or not?”
- To predict the digit, pick p(y = i l x) that is largest
  - e.g., output for digits [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] is [0.8, 0.05, 0.1, 0.06, 0.15, 0.2, 0.1, 0.01, 0.85, 0.6]

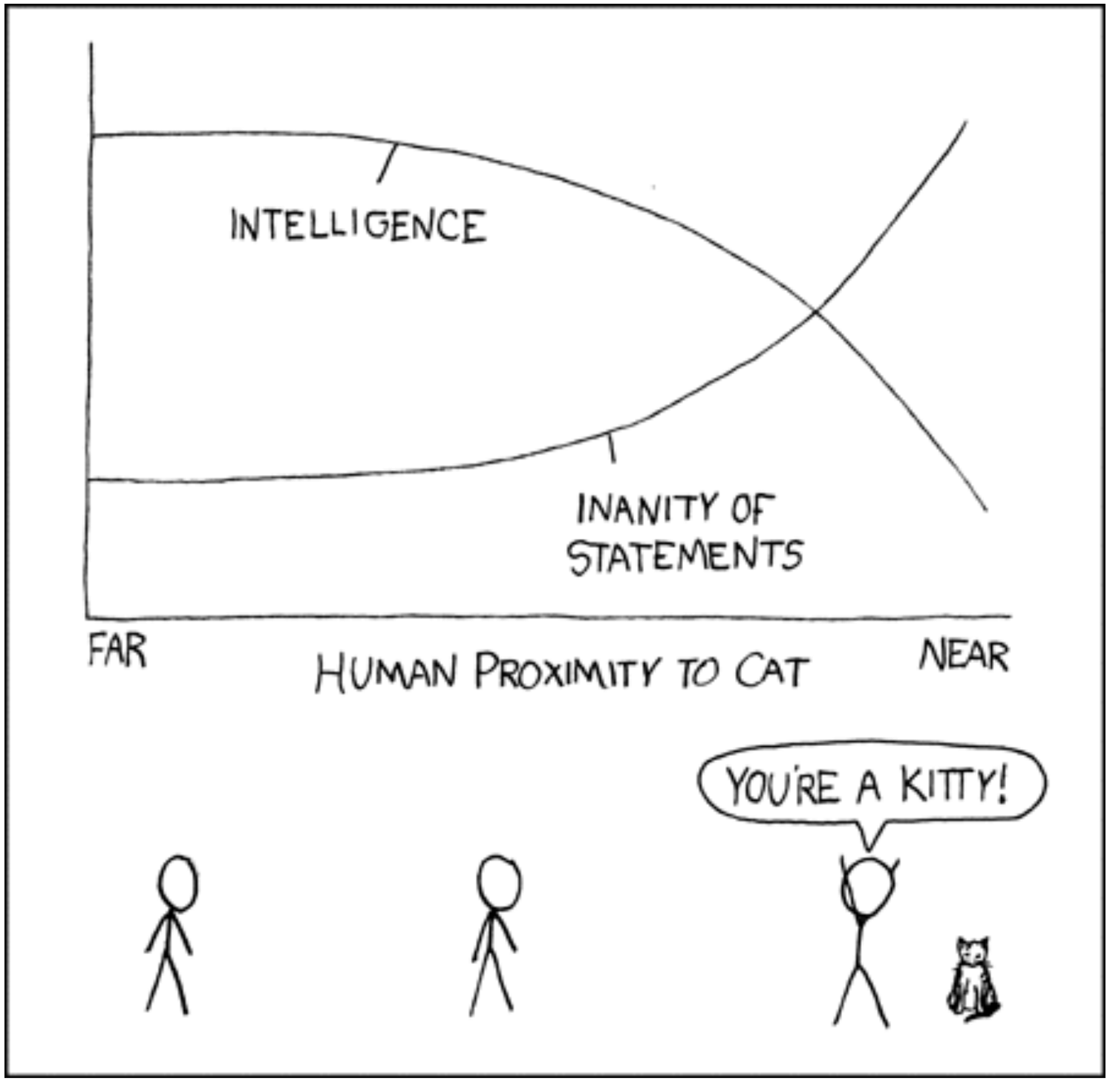
Hidden variables

- Different from missing variables, in the sense that we *could* have observed the missing information
  - e.g., if the person had just filled in the box on the form
- Hidden variables are never observed; rather they are useful for model description
  - e.g., hidden, latent representation
  - e.g., hidden state that drives dynamics
- Hidden variables make specification of distribution simpler
  - p(x | D) = \int_h p(x | D, h) p(h)
  - p(x | D, h) is often much simpler to specify

Intuitive example

- Underlying “state” influencing what we observe; partial observability makes what we observe difficult to interpret

- Imagine we can never see that a kitten is present; but it clearly helps to explain the data



Hidden variable models

- Probabilistic PCA and factor analysis
  - common in psychology
- Mixture models
- Hidden Markov Models
  - commonly used for NLP and modeling dynamical systems

Gaussian mixture model

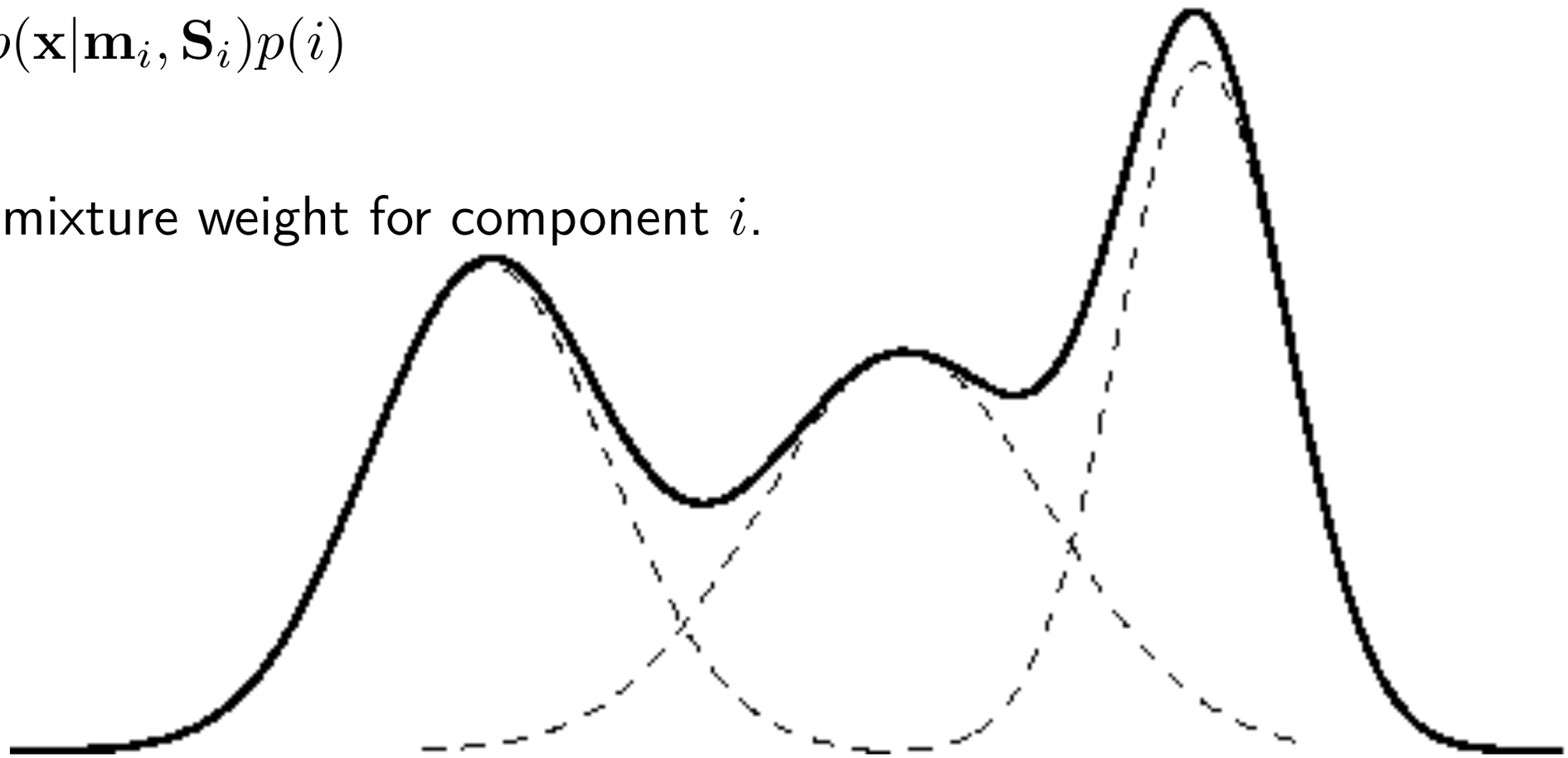
A D dimensional Gaussian distribution for a continuous variable $\mathbf{x}$ is

$$p(\mathbf{x}|\mathbf{m},\mathbf{S}) = \frac{1}{\sqrt{\det(2\pi\mathbf{S})}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^{\mathsf{T}}\mathbf{S}^{-1}(\mathbf{x}-\mathbf{m})\right\}$$

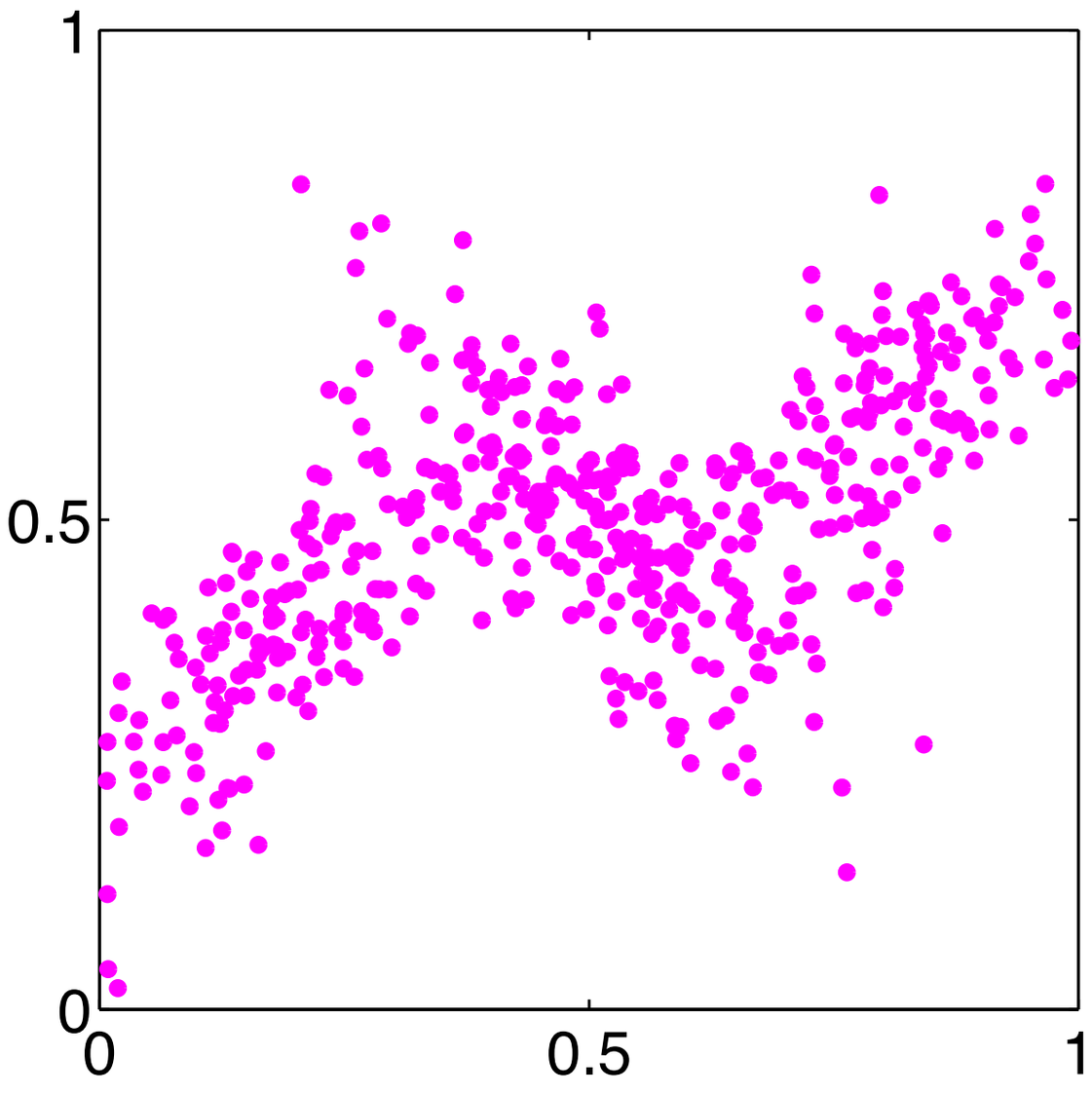
where $\mathbf{m}$ is the mean and $\mathbf{S}$ is the covariance matrix. A mixture of Gaussians is then

$$p(\mathbf{x}) = \sum_{i=1}^{H} p(\mathbf{x}|\mathbf{m}_i,\mathbf{S}_i)p(i)$$

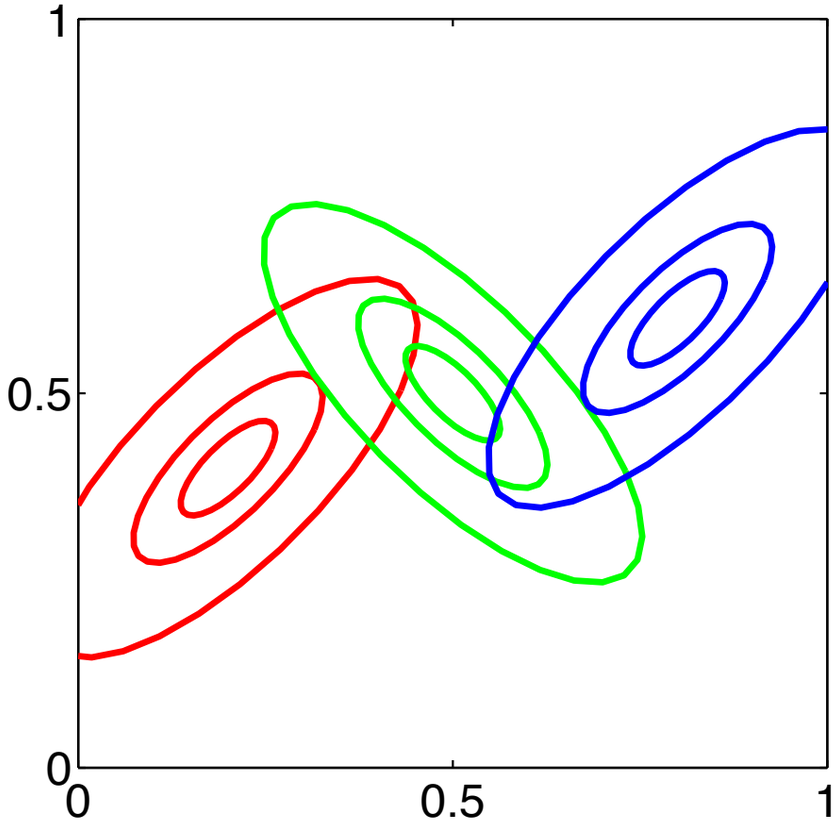
where $p(i)$ is the mixture weight for component i.



Example of 2-d data

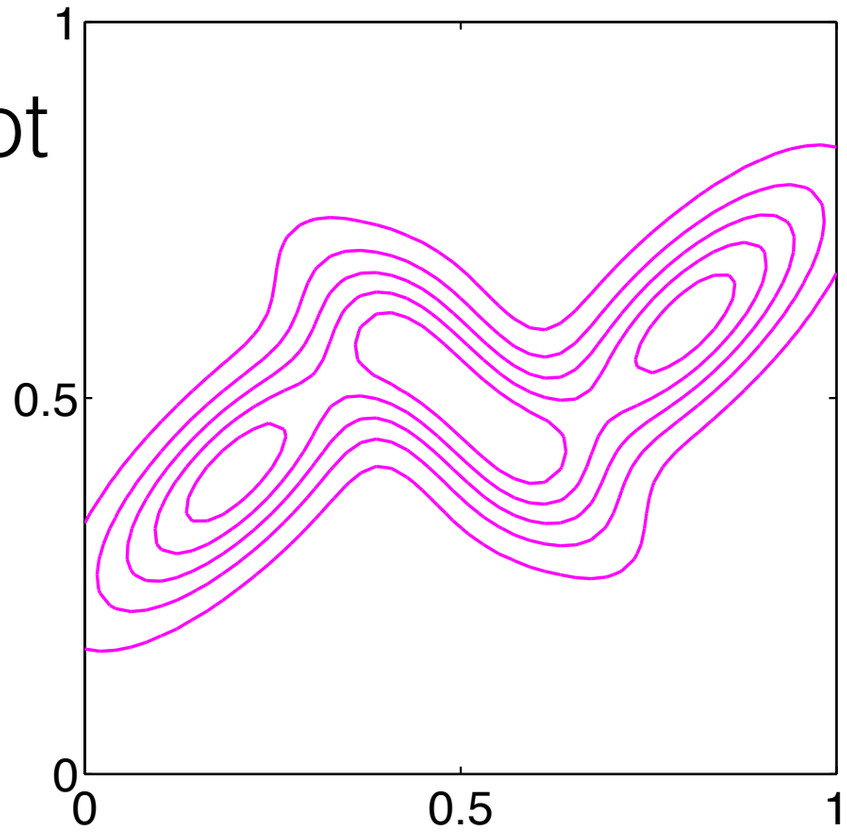


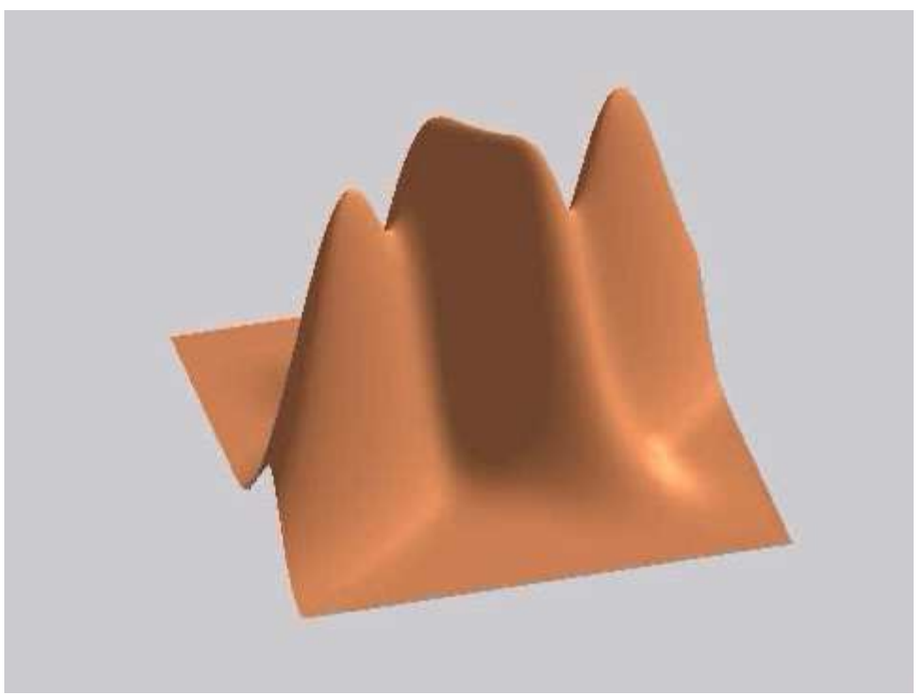
Mixture of 3 Gaussians



3 contours

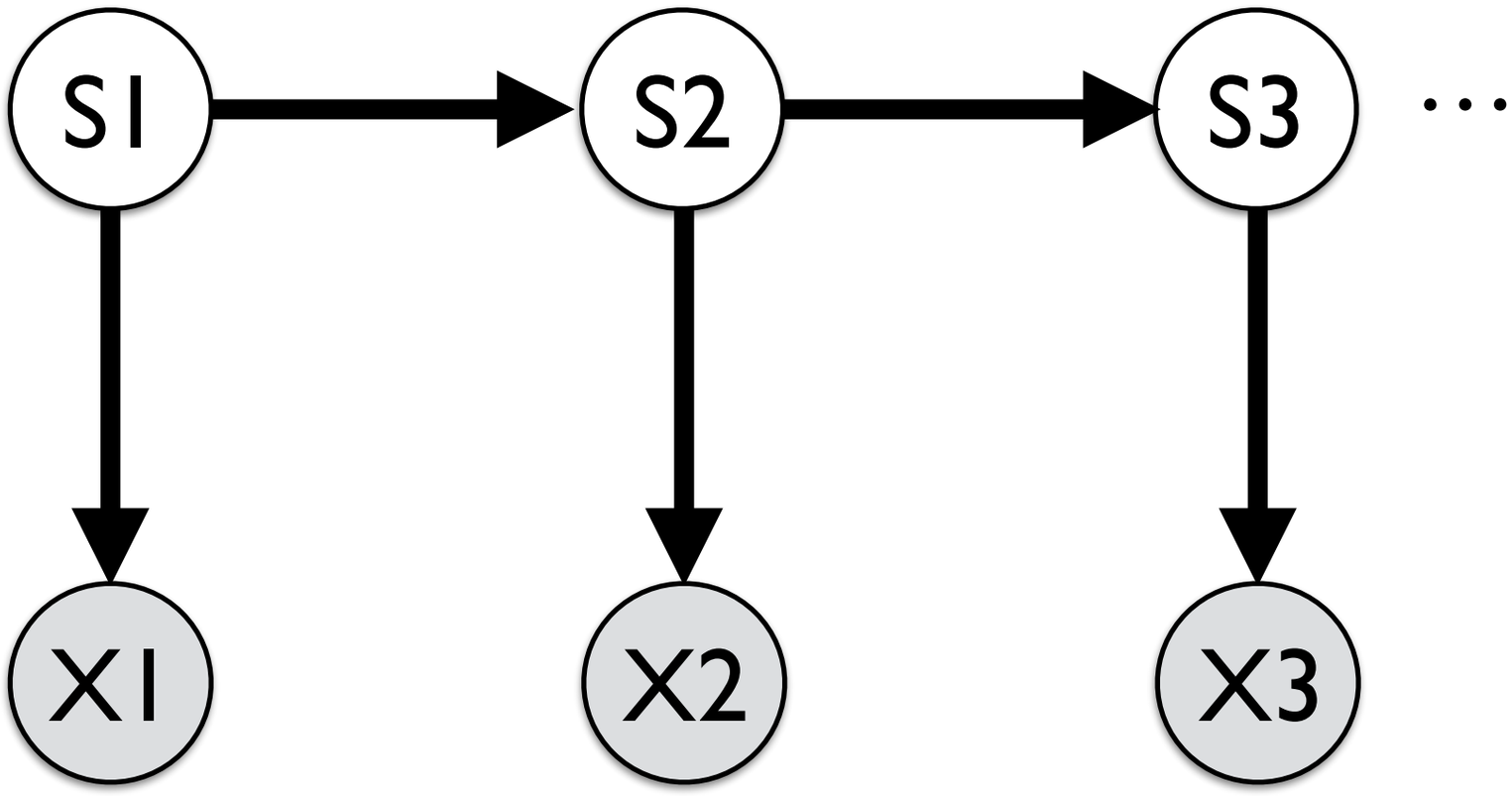
Contour plot





Surface plot

Hidden Markov Model



The observations are x1, x2, x3
Temporally related
Dynamics driven by hidden state

Closed-form solutions

- For some hidden variable models, have a closed form solution
  - probabilistic PCA and factor analysis
- For others, no closed form solution, still want to maximize likelihood of the data
  - e.g., mixture models

$$p(\mathbf{x}|\mathbf{m},\mathbf{S}) = \frac{1}{\sqrt{\det(2\pi\mathbf{S})}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^{\mathsf{T}}\mathbf{S}^{-1}(\mathbf{x}-\mathbf{m})\right\}$$

$$p(\mathbf{x}) = \sum_{i=1}^{H} p(\mathbf{x}|\mathbf{m}_i,\mathbf{S}_i)p(i) \qquad \log p(\mathbf{x})?$$

Expectation-maximization

- We can use an expectation-maximization approach instead to incrementally compute the solution (rather than a closed form)
- Similar to alternating descent approach taken for RFMs
- Enables logarithm and sum over hidden variables to be swapped, by minimizing instead a lower bound

$$\log p(\theta|\mathbf{x}) \geq \mathbb{E}[\log p(\mathbf{x}, \mathbf{h}|\theta) - \log p(\mathbf{h})]$$

$$\log p(\mathbf{x}, \mathbf{h}|\theta) = \log p(\mathbf{x}|\mathbf{h}, \theta) + \log p(\mathbf{h}|\theta)$$

If expectation w.r.t. $p(\mathbf{h}|\mathbf{x}, \theta)$ then equal, rather than $\geq$

Expectation-maximization

1. Approximate $p(\mathbf{h}|\mathbf{x}, \theta)$

2. Optimize theta for

$$\log p(\mathbf{x}, \mathbf{h}|\theta) = \log p(\mathbf{x}|\mathbf{h}, \theta) + \log p(\mathbf{h}|\theta)$$

e.g. mixture model

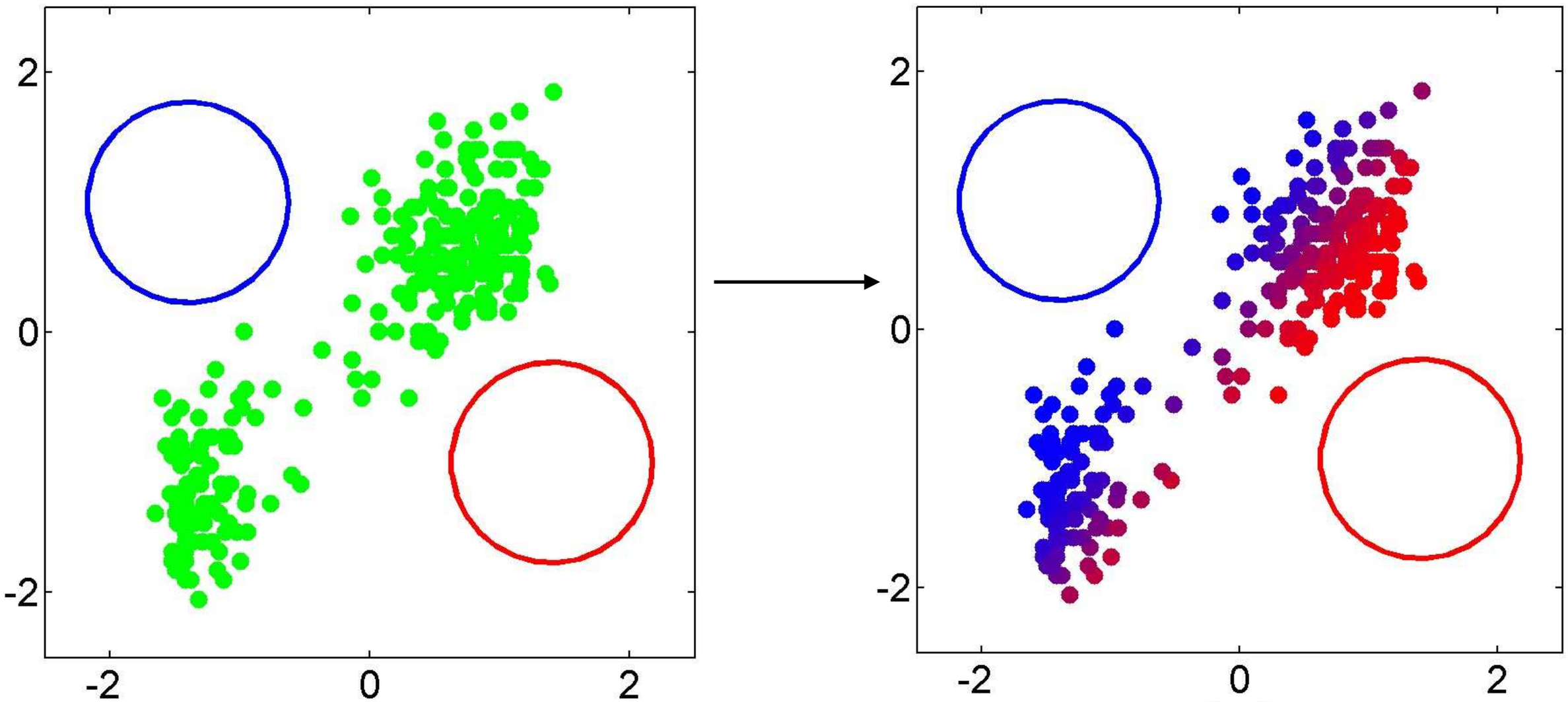
$$p(\mathbf{x}|\mathbf{m}, \mathbf{S}) = \frac{1}{\sqrt{\det(2\pi\mathbf{S})}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^{\mathsf{T}}\mathbf{S}^{-1}(\mathbf{x}-\mathbf{m})\right\}$$

$$p(\mathbf{x}) = \sum_{i=1}^{H} p(\mathbf{x}|\mathbf{m}_i, \mathbf{S}_i)p(i) \qquad \log p(\mathbf{x}|h=i) \text{ simple}$$

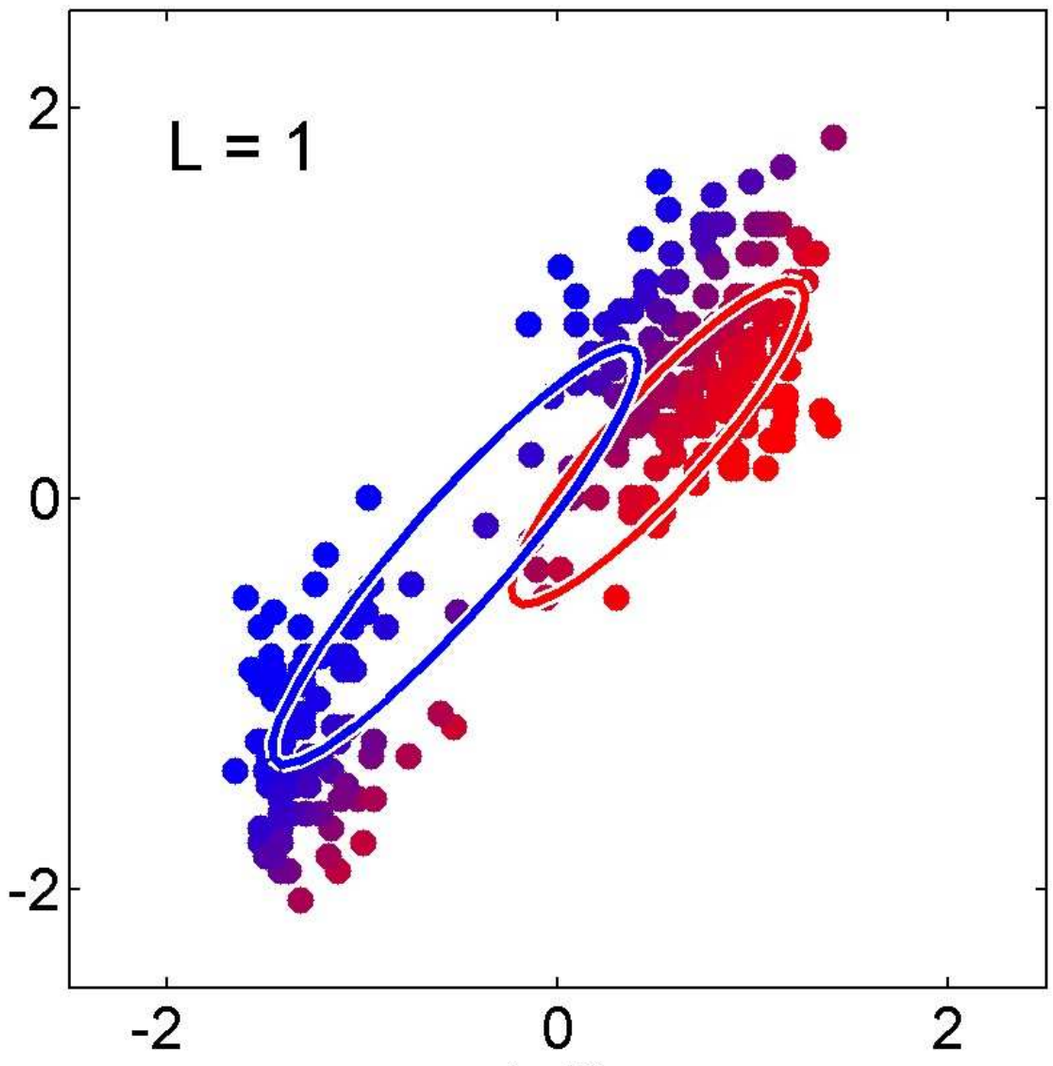
EM algorithm for mixtures

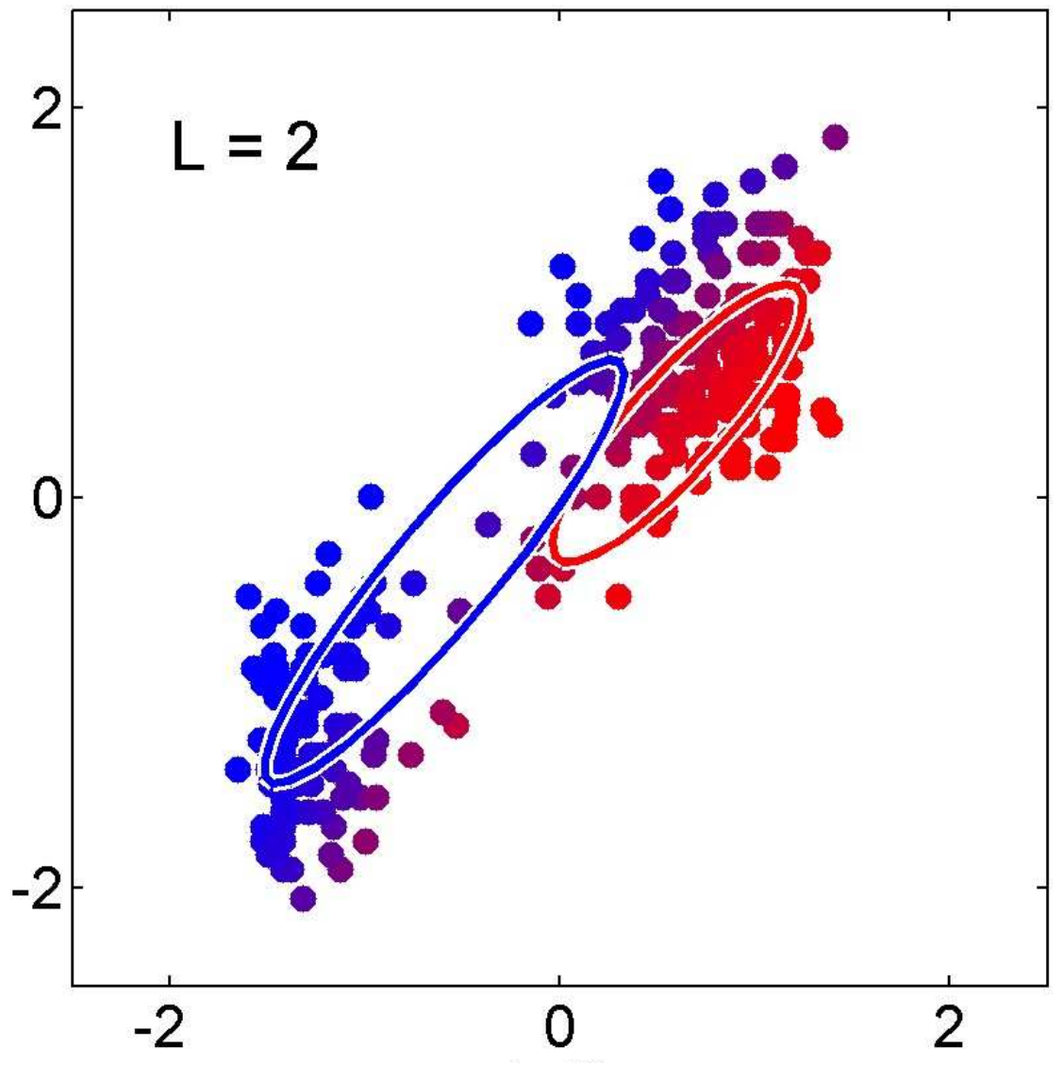
- Procedure: initialize parameters to some initial guess/random
- Alternate between:
  - E-step: fix parameter, approximate p(h l x, theta)
  - M-step: fix p(h l x, theta) obtaining maximum likelihood parameters for means, covariances and weights on each distribution
- Each cycle guaranteed not to decrease likelihood, converge to a local minimum

Simulation of EM for mixtures

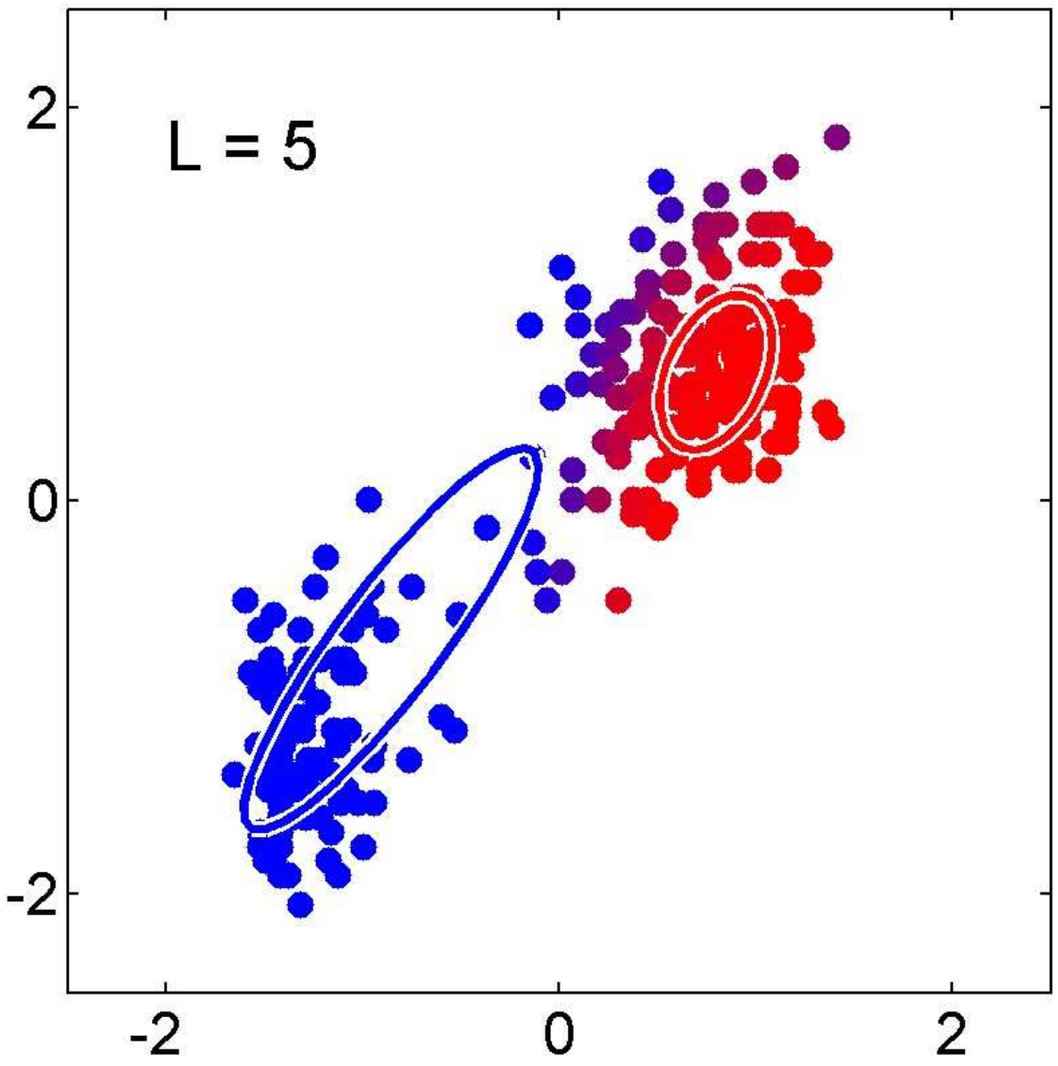


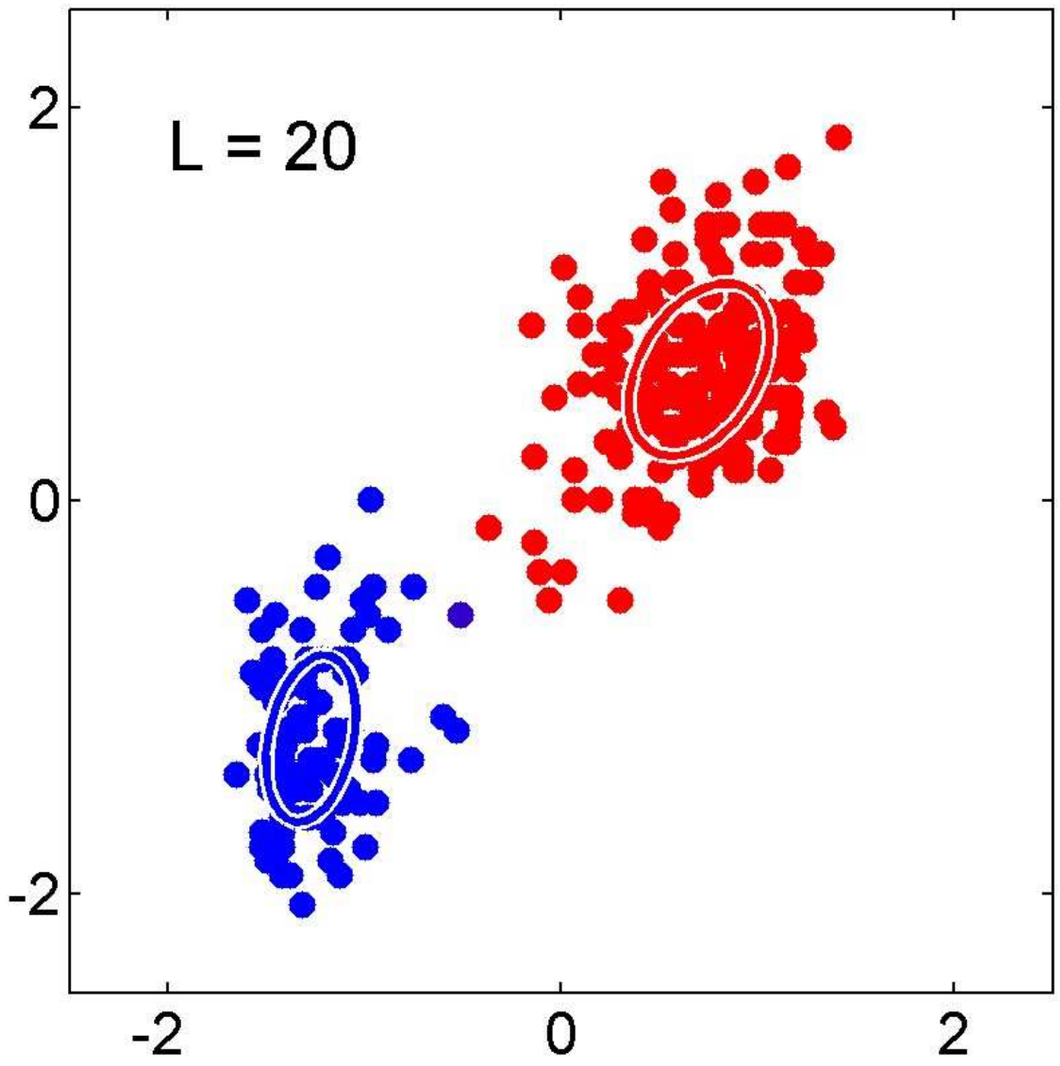
Simulation of EM for mixtures





Simulation of EM for mixtures





Question

- Are the nodes in hidden layer in a neural network considered to be hidden variables?

- In k-means clustering, the hidden variables are the cluster centers —never get to observe cluster centers

- Many models extracting structure use hidden variables, but do not use explicit distributions

- Key difference: explicitly maintaining a distribution over hidden variables (as in mixture-models) versus simply picking best hidden variables (as we usually do in NNs, though there are probabilistic NN models)

Why would we keep a distribution over values?

- For mixture models, incorporates a belief or uncertainty about which group you might be in

- If you just pick the most likely group, you throw away uncertainty information

- This is true for other models we have looked at: can keep a distribution over our parameters, rather than just picking the best

Bayesian learning

- Goal is to keep distribution over parameters
  - p(w | D) rather than w*
- Frequentist approach: find the most likely (“best”) parameters
  - this is what we have been doing so far with ML and MAP
- Still use Bayes rule to compute posterior p(w | Data), but now not taking argmax p(w | Data), but rather keeping distribution
- Key difference: can no longer ignore p(Data)

$$p(w|\text{Data}) = \frac{p(\text{Data}|w)p(w)}{p(\text{Data})}$$

Efficiently obtaining posterior

- Assume that p(x | theta) is Gaussian
- If we carefully select prior p(theta), then we get a nice known form for p(theta | x)
  - this is called a conjugate prior
  - e.g., Gaussian p(x | theta) and Gaussian p(theta) result in a Gaussian distribution p(theta | Data) with closed form mean, covariance
- If we do not carefully select p(theta), then we have to try to compute the integral of p(Data) numerically
  - and things start to get messy

Bayesian linear regression

- The likelihood is a normal distribution, with unknown mean and known variance (i.e., given hyper-parameter)
- The conjugate prior for that likelihood is the Gaussian with hyper-parameters mean and variance
- To simplify, assume $w \sim \mathcal{N}(\mu_w = 0, \sigma_w^2)$ and let $\alpha = \frac{1}{\sigma_w^2}$ then the posterior hyperparameters are

$$\mathbf{S} = \left(\alpha \mathbf{I} + \frac{1}{\sigma^2} \sum_{n=1}^{N} \boldsymbol{\phi}(\mathbf{x}^n) \boldsymbol{\phi}^T(\mathbf{x}^n) \right)^{-1}$$

$$\mathbf{m} = \frac{1}{\sigma^2} \mathbf{S} \sum_{n=1}^{N} y^n \boldsymbol{\phi}(\mathbf{x}^n)$$

Relation to linear regression

$\mathbf{m}$ is the solution to the Gaussian regularized linear regression problem, with regularization weight alpha

$$\mathbf{S} = \left(\alpha \mathbf{I} + \frac{1}{\sigma^2} \sum_{n=1}^{N} \boldsymbol{\phi}(\mathbf{x}^n) \boldsymbol{\phi}^T(\mathbf{x}^n) \right)^{-1}$$

$$\mathbf{m} = \frac{1}{\sigma^2} \mathbf{S} \sum_{n=1}^{N} y^n \boldsymbol{\phi}(\mathbf{x}^n)$$

The mean prediction for an input $\mathbf{x}$ is then given by

$$\bar{f}(\mathbf{x}) \equiv \int f(\mathbf{x}; \mathbf{w}) p(\mathbf{w}|\mathcal{D}, \Gamma) d\mathbf{w} = \mathbf{m}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{x}) .$$