

*L**A**T*_E*X* command declarations here.

In [1]:

```

from __future__ import division

# scientific
%matplotlib inline
from matplotlib import pyplot as plt;
import matplotlib as mpl;
import numpy as np;
import sklearn as skl;
import sklearn.datasets;
import sklearn.cluster;
import sklearn.mixture;

# ipython
import IPython;

# python
import os;
import random;

#####
# image processing
import PIL;

# trim and scale images
def trim(im, percent=100):
    print("trim:", percent);
    bg = PIL.Image.new(im.mode, im.size, im.getpixel((0,0)))
    diff = PIL.ImageChops.difference(im, bg)
    diff = PIL.ImageChops.add(diff, diff, 2.0, -100)
    bbox = diff.getbbox()
    if bbox:
        x = im.crop(bbox)
    return x.resize(((x.size[0]*percent)//100, (x.size[1]*percent)//1
PIL.Image.ANTIALIAS);

#####
# daft (rendering PGMs)
import daft;

# set to FALSE to load PGMs from static images
RENDER_PGMs = True;

# decorator for pgm rendering
def pgm_render(pgm_func):
    def render_func(path, percent=100, render=None, *args, **kwargs):
        print("render_func:", percent);
        # render
        render = render if (render is not None) else RENDER_PGMs;

        if render:
            print("rendering");
            # render
            pgm = pgm_func(*args, **kwargs);
            pgm.render();

```

```

    pgm.figure.savefig(path, dpi=300);

    # trim
    img = trim(PIL.Image.open(path), percent);
    img.save(path, 'PNG');
else:
    print("not rendering");

    # error
if not os.path.isfile(path):
    raise Exception("Error: Graphical model image %s not found.
You may need to set RENDER_PGM=True." % path);

    # display
return IPython.display.Image(filename=path);#trim(PIL.Image.open
(path), percent);

return render_func;

#####

```

EECS 445: Machine Learning

Lecture 17: Clustering, Mixture Models, and EM

- Instructor: **Jacob Abernethy**
- Date: November 14, 2016

Lecture Exposition: Benjamin Bray

This lecture has notes!

They can be found [here](#) (`./eecs445-f16-em-notes.pdf`). Written by our very own Benjamin Bray!

References

- [MLAPP] Murphy, Kevin. *Machine Learning: A Probabilistic Perspective* (<https://mitpress.mit.edu/books/machine-learning-0>). 2012.
- [PRML] Bishop, Christopher. *Pattern Recognition and Machine Learning* (<http://research.microsoft.com/en-us/um/people/cmbishop/prml/>). 2006.
- [Koller & Friedman 2009] Koller, Daphne and Nir Friedman. *Probabilistic Graphical Models* (<https://mitpress.mit.edu/books/probabilistic-graphical-models>). 2009.
- [Neal & Hinton 1998] Neal, Radford and Geoffrey Hinton. "A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants" (<http://www.cs.toronto.edu/~fritz/absps/emk.pdf>). 1998.
- [Do 2008] Do, Chuong B. and Serafim Batzoglou. "What is the Expectation Maximization Algorithm?" (<http://www.nature.com/nbt/journal/v26/n8/full/nbt1406.html>). 2008.

Outline

- Clustering & Mixture Models
 - Latent Variable Models
 - Gaussian Mixture Models
- Review: Information Theory
- Expectation Maximization
 - Evidence Lower Bound
 - Variational Approximation
- Examples
 - Coin Flip Example
 - Gaussian Mixture Models

Clustering & Mixture Models

Uses material from [MLAPP]

Review: Latent Variable Models

In a **Latent Variable Model**, we have

- Observed variables \mathbf{X}
- Hidden variables \mathbf{Z} that influence \mathbf{X}
- Parameters θ governing the overall model

In general, our goal is to

Use what we know to make *inferences* about what we don't know.

Review: Latent Variable Models

The most general latent variable model for iid observations \mathcal{X} is

- \mathbf{X}_n and \mathbf{Z}_n may be sets of variables
- Parameters θ can be divided into variables θ_x influencing \mathbf{X} and θ_z influencing \mathbf{Z}

```
In [2]: @pgm_render
def pgm_latent():
    pgm = daft.PGM([4,4], origin=[-2,-1], node_unit=0.8, grid_unit=2.0);
    # nodes
    pgm.add_node(daft.Node("z", r"$z_n$", 0.7, 1));
    pgm.add_node(daft.Node("x", r"$x_n$", 1.3, 1, observed=True));
    pgm.add_node(daft.Node("theta", r"\theta", 1.3, 0.3));

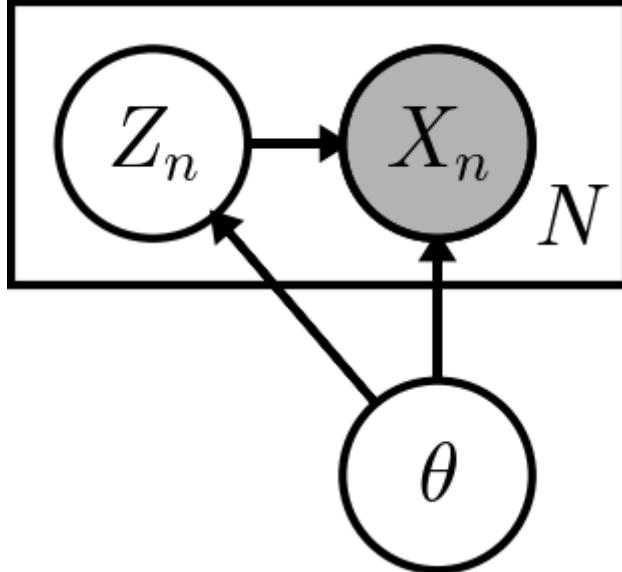
    # edges
    pgm.add_edge("z", "x", head_length=0.08);
    pgm.add_edge("theta", "x", head_length=0.08);
    pgm.add_edge("theta", "z", head_length=0.08);

    pgm.add_plate(daft.Plate([0.4,0.8,1.3,0.5], label=r"\quad\quad\quad",
                           shift=-0.1))

return pgm;
```

```
In [3]: %%capture
pgm_latent("images/pgm/latent.png")
```

Out[3]:



Review: Clustering

Goal: Partition data $\mathcal{X} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ into K disjoint **clusters**.

- Points within a cluster should be more similar to each other than to points in other clusters.
- Estimate **cluster centers** $\mu_k \in \mathbb{R}^d$ for $k = 1, \dots, K$
- Estimate **cluster assignments** $z_j \in \{1, \dots, K\}$ for each point x_j

Usually, we fix K beforehand! Use model selection to overcome this limitation.

Review: K-Means Clustering

First, pick random cluster centers μ_k . Then, repeat until convergence:

E-Step: Assign x_j to the nearest cluster center μ_k ,

$$z_j = \arg \min_k \|x_j - \mu_k\|^2$$

M-Step: Re-estimate cluster centers by averaging over assignments:

$$\mu_k = \frac{1}{\#\{j \mid z_j = k\}} \sum_{j=1}^N x_j \mathbb{I}(z_j = k)$$

Mixture Models

Clustering problems motivate the definition of a **mixture model**.

- Observe data from multiple **base distributions** $p_k(x|\theta_k)$ over the same space.
- Proportion of each distribution given by **mixing weights** $\pi = (\pi_1, \dots, \pi_K)$
- Don't know *which* distribution each point comes from.

$$p(x|\theta) = \sum_{k=1}^K \pi_k p_k(x|\theta_k)$$

Mixture Models for Clustering

In clustering, we represent mixtures via latent **cluster indicators**,

$$\begin{aligned} z_n &\sim \text{Categorical}[\pi] & \forall n = 1, \dots, N \\ x_n | z_n &\sim p_{z_n}(\cdot | \theta_{z_n}) & \forall n = 1, \dots, N \end{aligned}$$

In this form, mixture models are clearly applicable to clustering problems.

For a deeper understanding of mixture models, see [De Finetti's Theorem](#) (<http://www.cs.berkeley.edu/~jordan/courses/260-spring10/lectures/lecture1.pdf>).

Gaussian Mixture Models

In a **Gaussian Mixture Model**, the base distributions are Gaussian.

- Cluster k has mean μ_k , covariance Σ_k

$$\begin{aligned} z_n &\sim \text{Categorical}[\pi] & \forall n = 1, \dots, N \\ x_n | z_n &\sim \mathcal{N}(\mu_{z_n}, \Sigma_{z_n}) & \forall n = 1, \dots, N \end{aligned}$$

Gaussian Mixture Models: Visualization

Here are the contours of each base distribution plotted together:

```
In [4]: # example stolen from scikit-learn docs
def plot_gmm():
    n_samples = 300

    # generate random sample, two components
    np.random.seed(0)

    # generate spherical data centered on (20, 20)
    shifted_gaussian = np.random.randn(n_samples, 2) + np.array([20,
20])

    # generate zero centered stretched Gaussian data
    C = np.array([[0., -0.7], [3.5, .7]])
    stretched_gaussian = np.dot(np.random.randn(n_samples, 2), C)

    # concatenate the two datasets into the final training set
    X_train = np.vstack([shifted_gaussian, stretched_gaussian])

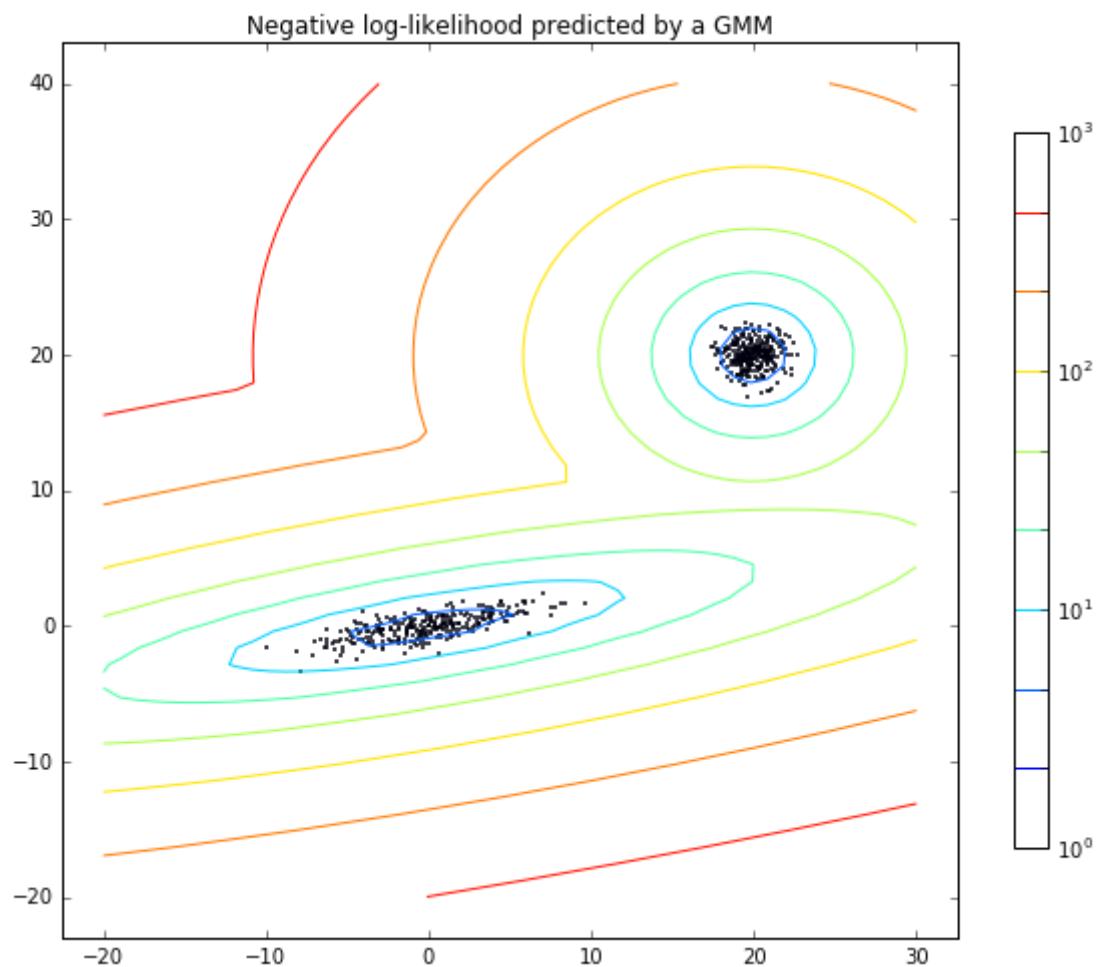
    # fit a Gaussian Mixture Model with two components
    clf = skl.mixture.GMM(n_components=2, covariance_type='full')
    clf.fit(X_train)

    # display predicted scores by the model as a contour plot
    x = np.linspace(-20.0, 30.0)
    y = np.linspace(-20.0, 40.0)
    X, Y = np.meshgrid(x, y)
    XX = np.array([X.ravel(), Y.ravel()]).T
    Z = -clf.score_samples(XX)[0]
    Z = Z.reshape(X.shape)

    plt.figure(figsize=(10,8))
    CS = plt.contour(X, Y, Z, norm=mpl.colors.LogNorm(vmin=1.0, vmax=100
0.0),
                      levels=np.logspace(0, 3, 10))
    CB = plt.colorbar(CS, shrink=0.8, extend='both')
    plt.scatter(X_train[:, 0], X_train[:, 1], .8)

    plt.title('Negative log-likelihood predicted by a GMM')
    plt.axis('tight')
```

```
In [5]: plot_gmm()
```



Gaussian Mixture Models: Inference & Learning

Goal: Since only the data points \mathbf{x}_n are observed, we wish to

1. Learn the cluster parameters μ_{z_n}, Σ_{z_n}
2. Infer the cluster assignments z_n for each point \mathbf{x}_n

We will use the **Expectation Maximization** algorithm.

- Same general idea as K-means, but *fuzzier* (probabilistic!)
- Alternate between *guessing* latent variables and *re-estimating* parameters.

Review of Information Theory

See **Lecture 13** or **[MLAPP]** §2.8 for more details.

Review: Entropy

The **entropy** of a discrete random variable \mathbf{X} with distribution \mathbf{p} is

$$H[\mathbf{X}] = H[\mathbf{p}] = E[I(\mathbf{p}(\mathbf{X}))] = - \sum_{x \in \mathcal{X}} p(x) \log p(x)$$

Entropy measures the **optimal code length**, in bits, for messages drawn from \mathbf{p} .

- Expected information received when sampling from \mathbf{X} .

Review: Cross Entropy

The **cross entropy** measures the average number of bits needed to encode messages drawn from \mathbf{p} when we use a code optimal for \mathbf{q} :

$$H(\mathbf{p}, \mathbf{q}) = - \sum_{x \in \mathcal{X}} p(x) \log q(x) = -E_p[\log q(x)]$$

Intuitively, $H(\mathbf{p}, \mathbf{q}) \geq H(\mathbf{p})$. The **relative entropy** is the difference $H(\mathbf{p}, \mathbf{q}) - H(\mathbf{p})$.

Review: Relative Entropy

The **relative entropy** or **Kullback-Leibler divergence** of \mathbf{q} from \mathbf{p} is

$$\begin{aligned} D_{KL}(\mathbf{p} || \mathbf{q}) &= \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \\ &= H(\mathbf{p}, \mathbf{q}) - H(\mathbf{p}) \end{aligned}$$

Measures the number of extra bits needed to encode messages from \mathbf{p} if we use a code optimal for \mathbf{q} .

Expectation Maximization

Uses material from [Neal & Hinton 1998], [Do 2008], and [MLAPP]

Theoretical Perspective

Most textbooks (including **[MLAPP]** and **[PRML]**) introduce EM without much justification.

- This is confusing and unsatisfying!

Instead, we will tackle the theory head-on, using ideas from **[Neal & Hinton 1998]**.

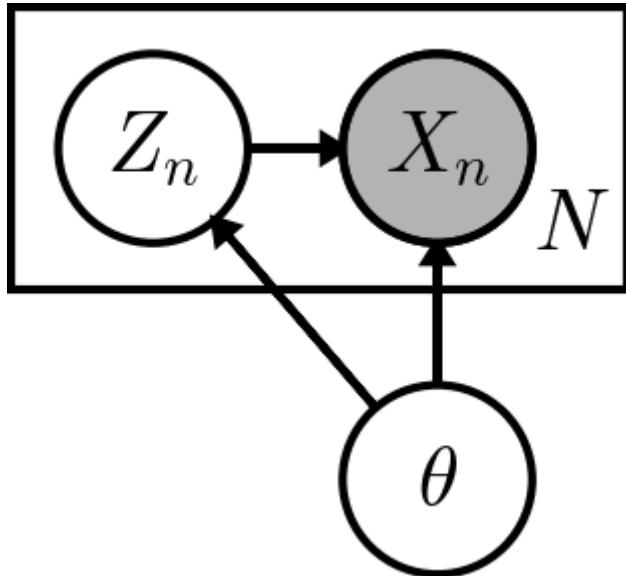
Question: How can we perform maximum likelihood estimation in latent variable models?

Parameter Estimation with Latent Variables

Question: How can we perform MLE/MAP in latent variable models?

```
In [6]: %%capture  
pgm_latent("images/pgm/latent.png")
```

Out[6]:



Complete Data Log-Likelihood

Suppose we observe both X and Z . The **complete data log-likelihood** is

$$\ell_c(\theta) = P(X, Z|\theta) = \sum_{n=1}^N \log p(x_n, z_n|\theta)$$

[MLAPP]: This likelihood function is convex when $p(x_n, z_n|\theta)$ is an exponential family distribution.

Observed Data Log-Likelihood

In reality, we only observe $\mathcal{X} = (x_1, \dots, x_n)$. The **observed data log-likelihood** is given by

$$\begin{aligned}\ell(\theta|\mathcal{X}) &= \log p(\mathcal{X}|\theta) = \sum_{k=1}^N \log p(x_k|\theta) \\ &= \sum_{k=1}^N \left[\log \sum_{z_k} p(x_k, z_k|\theta) \right]\end{aligned}$$

[MLAPP]: This likelihood function is **non-convex** even assuming exponential family distributions. Multiple modes make inference (NP) hard!

Wrangling a Non-Convex Likelihood

How can we optimize the observed likelihood $\ell(\theta|\mathcal{X})$?

- *Gradient-based methods?* Tricky to enforce constraints (e.g. positive-definiteness)

We need a different approach...

Wrangling a Non-Convex Likelihood

Observation: In most models, MLE/MAP is easy with fully observed data. We will alternate between

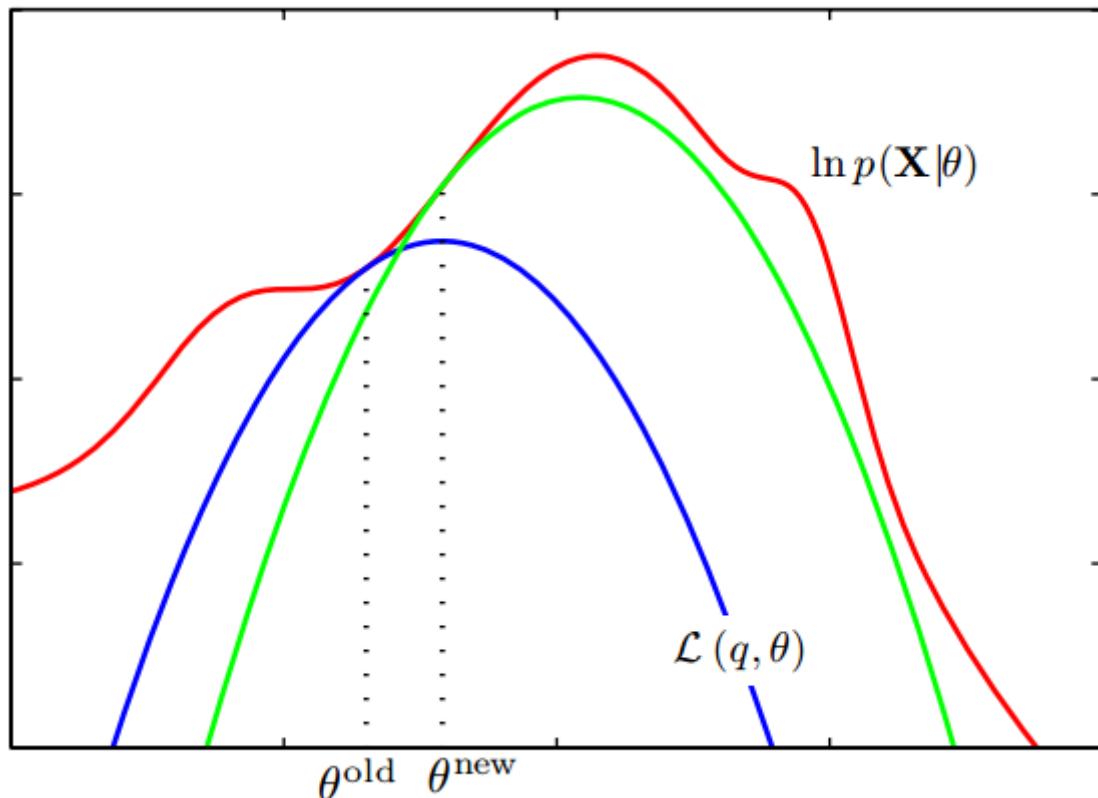
- **E-Step:** Inferring the values of the latent variables.
- **M-Step:** Re-estimating the parameters, assuming we have complete data.

Note the similarity to K-Means!

Optimizing via Lower-Bounds

From a theoretical standpoint, we will alternate between

1. Estimating a lower bound $\mathcal{L}(q, \theta)$ on $\ln p(\mathbf{X}|\theta)$
2. Maximizing this lower bound to obtain a new estimate of θ



(Figure 9.14 of **[PRML]**)

Expectation Maximization

Our general approach will be to reason about the hidden variables through a proxy distribution $q(Z)$.

- Distribution q is an approximation to $P(Z|\mathcal{X}, \theta)$
- We use q to compute a lower bound on the log-likelihood.

The q distribution will disappear later, but this perspective is useful for extending EM to new settings.

Review: Jensen's Inequality

For a convex function f ,

$$\sum_{k=1}^K \lambda_k f(x_k) \geq f\left(\sum_{k=1}^K \lambda_k x_k\right)$$

If $\lambda_k \geq 0$ and $\sum_{k=1}^K \lambda_k = 1$.

Evidence Lower Bound

Through Jensen's inequality, we obtain the **evidence lower bound (ELBO)** on the log-likelihood:

$$\begin{aligned}\ell(\theta|\mathcal{X}) &= \log \sum_z p(\mathcal{X}, z|\theta) \\ &= \log \sum_z q(z) \frac{p(\mathcal{X}, z|\theta)}{q(z)} \\ &\geq \sum_z q(z) \log \frac{p(\mathcal{X}, z|\theta)}{q(z)} \equiv \mathcal{L}(q, \theta)\end{aligned}$$

Important: the choice of distribution $q(z)$ is arbitrary, above holds for all q ! More on this soon

Evidence Lower Bound

We can rewrite the lower bound $\mathcal{L}(q, \theta)$ as follows:

$$\begin{aligned}\ell(\theta|\mathcal{X}) \geq \mathcal{L}(q, \theta) &= \sum_z q(z) \log \frac{p(\mathcal{X}, z|\theta)}{q(z)} \\ &= \sum_q q(z) \log p(\mathcal{X}, z|\theta) - \sum_z q(z) \log q(z) \\ &= E_q[\log p(\mathcal{X}, Z|\theta)] - E_q[\log q(z)] \\ &= E_q[\log p(\mathcal{X}, Z|\theta)] + H[q]\end{aligned}$$

ELBO: Relationship to Relative Entropy

We have just shown that

$$\ell(\theta|\mathcal{X}) \geq \mathcal{L}(q, \theta) = E_q[\log p(\mathcal{X}, Z|\theta)] - E_q[\log q(z)]$$

This very closely resembles the formula for **relative entropy**

$$\begin{aligned}D_{KL}(q||p) &= E_q[-\log p(Z)] + E_q[\log q(Z)] \\ &= H[q, p] - H[q]\end{aligned}$$

Except that the variables \mathbf{X} are clamped to our observations $\mathbf{X} = \mathcal{X}$.

- So $P(\mathcal{X}, Z)$ is an *unnormalized* distribution.

ELBO: Relationship to Relative Entropy

It is easy to show $\mathcal{L}(q, \theta)$ differs from a relative entropy by only a constant wrt θ :

$$\begin{aligned}D_{KL}(q||p(Z|\mathcal{X}, \theta)) &= H(q, p(Z|\mathcal{X}, \theta)) - H(q) \\ &= E_q[-\log p(Z|\mathcal{X}, \theta)] - H(q) \\ &= E_q[-\log p(Z, \mathcal{X}|\theta)] - E_q[-\log p(\mathcal{X}|\theta)] - H(q) \\ &= E_q[-\log p(Z, \mathcal{X}|\theta)] + \log p(\mathcal{X}|\theta) - H(q) \\ &= -\mathcal{L}(q, \theta) + \text{const.}\end{aligned}$$

ELBO: Tightness

This yields a second proof of the ELBO, since $D_{KL} \geq 0$,

$$\log p(\mathcal{X}|\theta) = D_{KL}(q||p(Z|\mathcal{X}, \theta)) + \mathcal{L}(q, \theta) \geq \mathcal{L}(q, \theta)$$

The quality of our bound $\mathcal{L}(q, \theta)$ depends heavily on the choice of proxy distribution $q(Z)$.

Remark: Maximizing $\mathcal{L}(q, \theta)$ with respect to q is equivalent to minimizing the relative entropy between q and the hidden posterior $P(Z|\mathcal{X}, \theta)$. Hence, the optimal q is exactly the hidden posterior, for which $\log p(\mathcal{X}|\theta) = \mathcal{L}(q, \theta)$.

The bound is **tight**--we can make it *touch* the log-likelihood.

Break time!



Expectation Maximization

Finding a global maximum of the likelihood is difficult in the presence of latent variables.

$$\hat{\theta}_{ML} = \arg \max_{\theta} \ell(\theta|\mathcal{X}) = \arg \max_{\theta} \log p(\mathcal{X}|\theta)$$

Instead, the **Expectation Maximization** algorithm gives an iterative procedure for finding a *local* maximum of the likelihood.

- Under the assumption that the hidden posterior $P(Z|\mathcal{X}, \theta)$ is tractable.
- Exploits the evidence lower bound $\ell(\theta|\mathcal{X}) \geq \mathcal{L}(q, \theta)$.

Expectation Maximization

Consider only proxy distributions of the form

$$q_\vartheta(Z) = p(Z|\mathcal{X}, \vartheta)$$

The optimal value for ϑ , in the sense that $\mathcal{L}(\vartheta, \theta) \equiv \mathcal{L}(q_\vartheta, \theta)$ is maximum, depends on θ .

- Similarly, the optimal θ depends on ϑ .

This suggests an *iterative* scheme.

Expectation Maximization: E-Step

Suppose we have an estimate θ_t of the parameters. To improve our estimate, we first compute a new lower bound on the observed log-likelihood,

$$\vartheta_{t+1} = \arg \max_{\vartheta} \mathcal{L}(\vartheta, \theta_t) = \theta_t$$

Since $q_\vartheta(Z) = p(Z|\mathcal{X}, \vartheta)$, we know from before that $\vartheta = \theta_t$ maximizes $\mathcal{L}(\vartheta, \theta)$.

Expectation Maximization: M-Step

Next, we estimate new parameters by optimizing over the lower bound.

$$\begin{aligned}\theta_{t+1} &= \arg \max_{\theta} \mathcal{L}(\vartheta_{t+1}, \theta) \\ &= \arg \max_{\theta} E_q[\log p(\mathcal{X}, Z|\theta)] - E_q[\log q_\vartheta(z)] \\ &= \arg \max_{\theta} E_q[\log p(\mathcal{X}, Z|\theta)]\end{aligned}$$

The last line optimizes over the **expected complete data log-likelihood**.

Expectation Maximization: Algorithm

Expectation maximization performs **coordinate ascent** on $\mathcal{L}(\vartheta, \theta)$.

E-Step: Compute a new lower bound on the observed log-likelihood,

$$\vartheta_{t+1} = \arg \max_{\vartheta} \mathcal{L}(\vartheta, \theta_t) = \theta_t$$

M-Step: Estimate new parameters by maximizing the bound.

$$\theta_{t+1} = \arg \max_{\theta} \mathcal{L}(\vartheta_{t+1}, \theta) = \arg \max_{\theta} E_q[\log p(\mathcal{X}, Z|\theta)]$$

Expectation Maximization: Standard Formulation

Some authors define the **auxiliary function** or **Q-function**

$$Q(\theta_t, \theta) = E_q[\log p(\mathcal{X}, Z|\theta)]$$

where $q(Z) = p(Z|\mathcal{X}, \theta_t)$ as before.

Some forgo the q function altogether!

Expectation Maximization: Standard Formulation

Expectation maximization performs updates on $Q(\theta_t, \theta)$.

E-Step: Write down an expression for

$$Q(\theta_t, \theta) = E_q[\log p(\mathcal{X}, Z|\theta)]$$

M-Step: Maximize the auxiliary function,

$$\theta_{t+1} = \arg \max_{\theta} Q(\theta_t, \theta)$$

Expectation Maximization: Convergence

Theorem: After a single iteration of Expectation Maximization, the observed data likelihood of the parameters has not decreased, that is,

$$\ell(\theta_t | \mathcal{X}) \leq \ell(\theta_{t+1} | \mathcal{X})$$

Proof: This result is a simple consequence of all our hard work so far!

$$\begin{aligned}\ell(\theta_t | \mathcal{X}) &= \mathcal{L}(q_{\theta_{t+1}}, \theta_t) && \text{(Tightness)} \\ &\leq \mathcal{L}(q_{\theta_{t+1}}, \theta_{t+1}) && \text{(M-Step)} \\ &\leq \ell(\theta_{t+1} | \mathcal{X}) && \text{(ELBO)}\end{aligned}$$

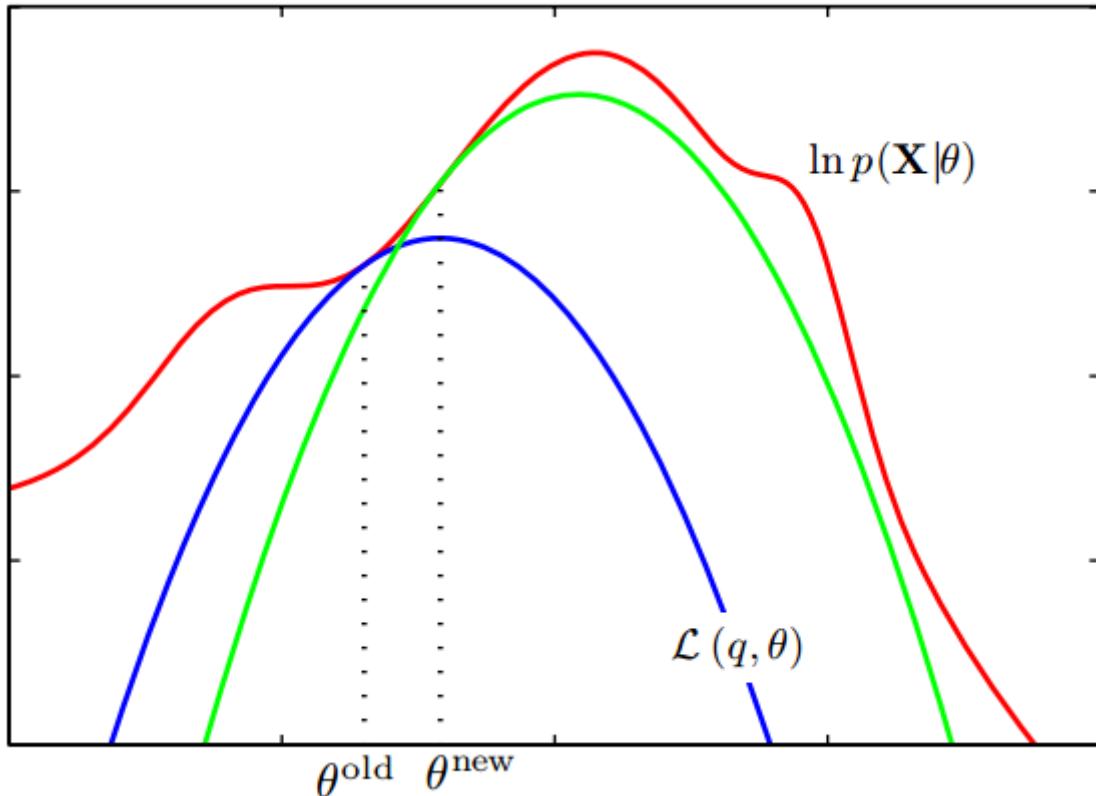
Expectation Maximization: Convergence

Theorem: (Neal & Hinton 1998, Thm. 2) Every local maximum of $\mathcal{L}(q, \theta)$ is a local maximum of $\ell(\theta | \mathcal{X})$.

Exercise! :)

Expectation Maximization: Interpretation

The graph from before should now make more sense:



(Figure 9.14 from **[PRML]**)

Expectation Maximization: Examples

We're done with the theory. Now for some examples:

- *Today:* Coin Flip Example, Gaussian Mixture Models
- *Later:* Hidden Markov Models

We'll also generalize to new settings:

- "Hard" Expectation Maximization
- Variational Expectation Maximization

Coin Flip: Problem

Suppose we have two coins, each with unknown bias θ_A and θ_B , and collect data in the following way:

1. Repeat for $n = 1, \dots, N$:
 - A. Choose a random coin z_n .
 - B. Flip this same coin M times.
 - C. Record only the total number x_n of heads.

Coin Flip: Sample Data

Below is sample data for $N = 5$ and $M = 10$

HTTTHHTH
HHHHTHHHHH
HTHHHHHTHH
HTHTTTTHHTT
THHHHTHHHTH

Coin Flip: Model

The corresponding probabilistic model is a **mixture of binomials**:

$$\begin{array}{lll} \theta = (\theta_A, \theta_B) & & \text{fixed coin biases} \\ Z_n \sim \text{Uniform}\{A, B\} & \forall n = 1, \dots, N & \text{coin indicators} \\ X_n | Z_n, \theta \sim \text{Binomial}[\theta_{Z_n}, M] & \forall n = 1, \dots, N & \text{head count} \end{array}$$

Coin Flip: Graphical Model

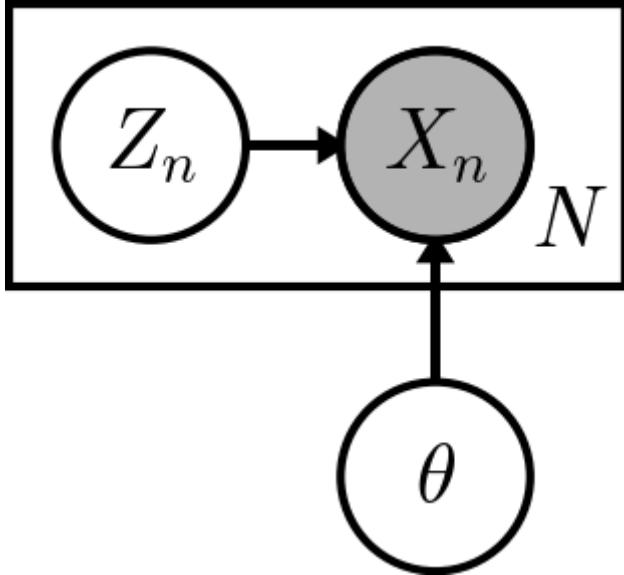
```
In [7]: @pgm_render
def pgm_coinflip():
    pgm = daft.PGM([4,4], origin=[-2,-1], node_unit=0.8, grid_unit=2.0);
    # nodes
    pgm.add_node(daft.Node("z", r"$Z_n$", 0.7, 1));
    pgm.add_node(daft.Node("x", r"$X_n$", 1.3, 1, observed=True));
    pgm.add_node(daft.Node("theta", r"$\theta$", 1.3, 0.3));

    # edges
    pgm.add_edge("z", "x", head_length=0.08);
    pgm.add_edge("theta", "x", head_length=0.08);

    pgm.add_plate(daft.Plate([0.4,0.8,1.3,0.5],
                           label=r"$\qquad\qquad\qquad; \; N$",
                           shift=-0.1))
    return pgm;
```

```
In [8]: %%capture
pgm_coinflip("images/pgm/coinflip.png")
```

Out[8]:



Coin Flip: Complete Data Log-Likelihood

The **complete data log-likelihood** for a single trial (x_n, z_n) is

$$\log p(x_n, z_n | \theta) = \log p(z_n) + \log p(x_n | z_n, \theta)$$

Here, $P(z_n) = \frac{1}{2}$. The remaining term is

$$\begin{aligned} \log p(x_n | z_n, \theta) &= \log \binom{M}{x_n} \theta_{z_n}^{x_n} (1 - \theta_{z_n})^{M-x_n} \\ &= \log \binom{M}{x_n} + x_n \log \theta_{z_n} + (M - x_n) \log(1 - \theta_{z_n}) \end{aligned}$$

Coin Flip: Likelihood Plot

There aren't many latent variables, so we can plot $\log P(\mathcal{X}|\theta_A, \theta_B)$ directly!

```
In [9]: def coin_likelihood(roll, bias):
    #  $P(X | z, \theta)$ 
    numHeads = roll.count("H");
    flips = len(roll);
    return pow(bias, numHeads) * pow(1-bias, flips-numHeads);

def coin_marginal_likelihood(rolls, biasA, biasB):
    #  $P(X | \theta)$ 
    trials = [];
    for roll in rolls:
        h = roll.count("H");
        t = roll.count("T");
        likelihoodA = coin_likelihood(roll, biasA);
        likelihoodB = coin_likelihood(roll, biasB);
        trials.append(np.log(0.5 * (likelihoodA + likelihoodB)));
    return sum(trials);

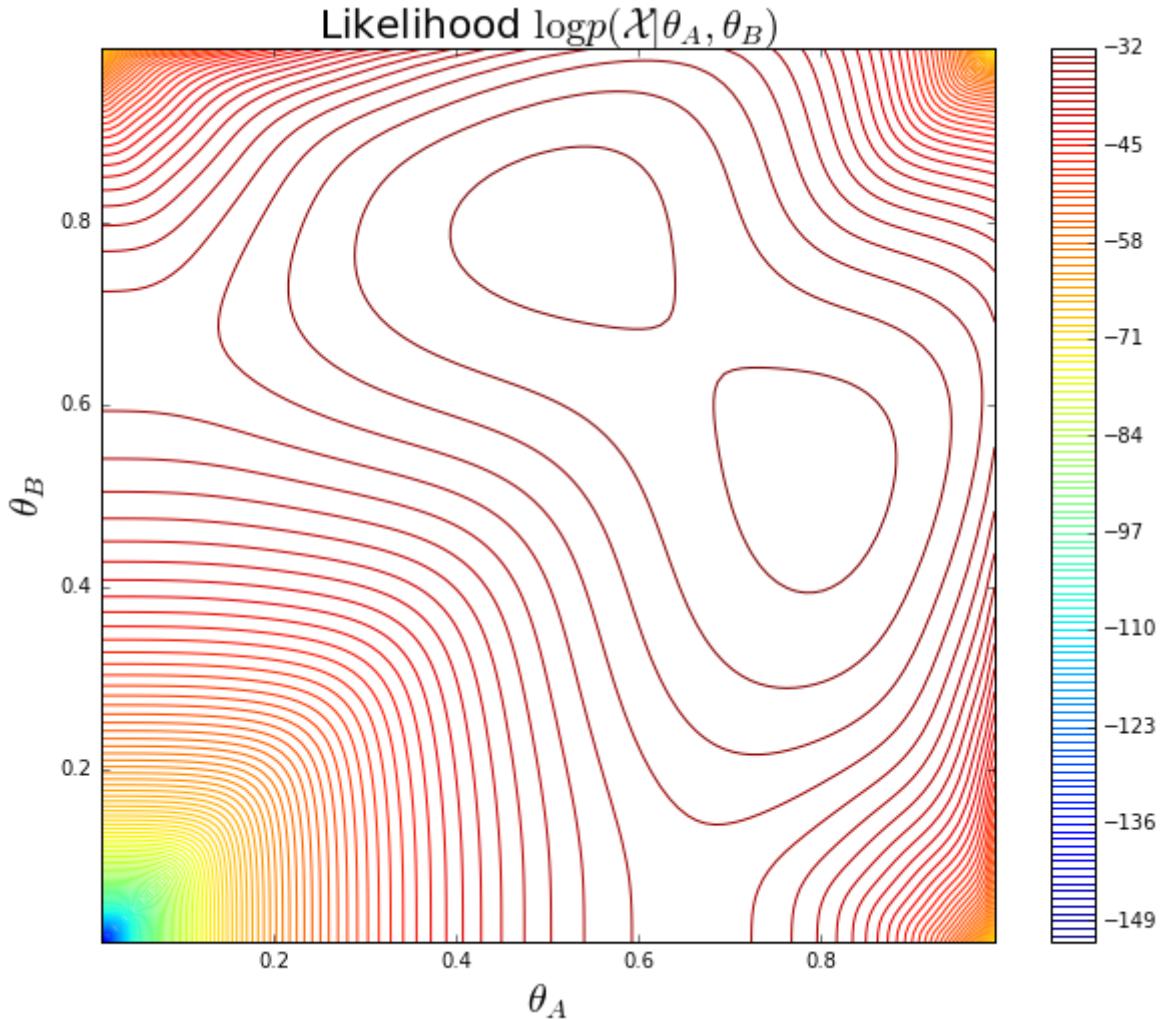
def plot_coin_likelihood(rolls, thetas=None):
    # grid
    xvals = np.linspace(0.01,0.99,100);
    yvals = np.linspace(0.01,0.99,100);
    X,Y = np.meshgrid(xvals, yvals);

    # compute likelihood
    Z = [];
    for i,r in enumerate(X):
        z = []
        for j,c in enumerate(r):
            z.append(coin_marginal_likelihood(rolls,c,Y[i][j]));
        Z.append(z);

    # plot
    plt.figure(figsize=(10,8));
    C = plt.contour(X,Y,Z,150);
    cbar = plt.colorbar(C);
    plt.title(r"Likelihood $\log p(\mathcal{X}|\theta_A, \theta_B)$", fontsize=20);
    plt.xlabel(r"$\theta_A$", fontsize=20);
    plt.ylabel(r"$\theta_B$", fontsize=20);

    # plot thetas
    if thetas is not None:
        thetas = np.array(thetas);
        plt.plot(thetas[:,0], thetas[:,1], '-k', lw=2.0);
        plt.plot(thetas[:,0], thetas[:,1], 'ok', ms=5.0);
```

```
In [10]: plot_coin_likelihood(["HTTTTHHTHTH", "HHHHHTHHHHH",
                            "HTHHHHHTHH", "HTHTTTHHTT", "THHHTHHHTH"]);
```



Coin Flip: E-Step

The **E-Step** involves writing down an expression for

$$\begin{aligned} E_q[\log p(\mathcal{X}, Z|\theta)] &= E_q[\log p(\mathcal{X}|Z, \theta)p(Z)] \\ &= E_q[\log p(\mathcal{X}|Z, \theta)] + \log p(Z) \end{aligned}$$

The $\log p(Z)$ term is constant wrt θ , so we ignore it.

Recall $q \equiv q_{\theta_t} = p(Z|\mathcal{X}, \theta_t)$

Coin Flip: E-Step

$$\begin{aligned}E_q[\log p(\mathcal{X}|Z, \theta)] &= \sum_{n=1}^N E_q \left[x_n \log \theta_{z_n} + (M - x_n) \log(1 - \theta_{z_n}) \right] + \text{const.} \\&= \sum_{n=1}^N q_\vartheta(z_n = A) \left[x_n \log \theta_A + (M - x_n) \log \theta_A \right] \\&\quad + \sum_{n=1}^N q_\vartheta(z_n = B) \left[x_n \log \theta_B + (M - x_n) \log \theta_B \right] + \text{const.}\end{aligned}$$

Coin Flip: M-Step

Let $a_n = q(z_n = A)$ and $b_n = q(z_n = B)$. Taking derivatives with respect to θ_A and θ_B , we obtain the following update rules:

$$\theta_A = \frac{\sum_{n=1}^N a_n x_n}{\sum_{n=1}^N a_n M} \quad \theta_B = \frac{\sum_{n=1}^N b_n x_n}{\sum_{n=1}^N b_n M}$$

Interpretation: For each coin, examples are *weighted* according to the probability that they belong to that coin. Observing M flips is equivalent to observing $a_n M$ effective flips.

Coin Flip: Python Code

```
In [11]: def coin_em(rolls, theta_A=None, theta_B=None, maxiter=10):
    # Initial Guess
    theta_A = theta_A or random.random();
    theta_B = theta_B or random.random();
    thetas = [(theta_A, theta_B)];
    # Iterate
    for c in range(maxiter):
        print("#%d:\t%0.2f %0.2f" % (c, theta_A, theta_B));
        # assign a coin to each trial
        heads_A, tails_A = 0,0;
        heads_B, tails_B = 0,0;

        for trial in rolls:
            likelihood_A = coin_likelihood(trial,theta_A);
            likelihood_B = coin_likelihood(trial,theta_B);
            p_A = likelihood_A / (likelihood_A + likelihood_B);
            p_B = likelihood_B / (likelihood_A + likelihood_B);
            heads_A += p_A * trial.count("H");
            tails_A += p_A * trial.count("T");
            heads_B += p_B * trial.count("H");
            tails_B += p_B * trial.count("T");
        # recompute thetas
        theta_A = heads_A / (heads_A + tails_A);
        theta_B = heads_B / (heads_B + tails_B);

    thetas.append((theta_A,theta_B));
    return thetas, (theta_A,theta_B);
```

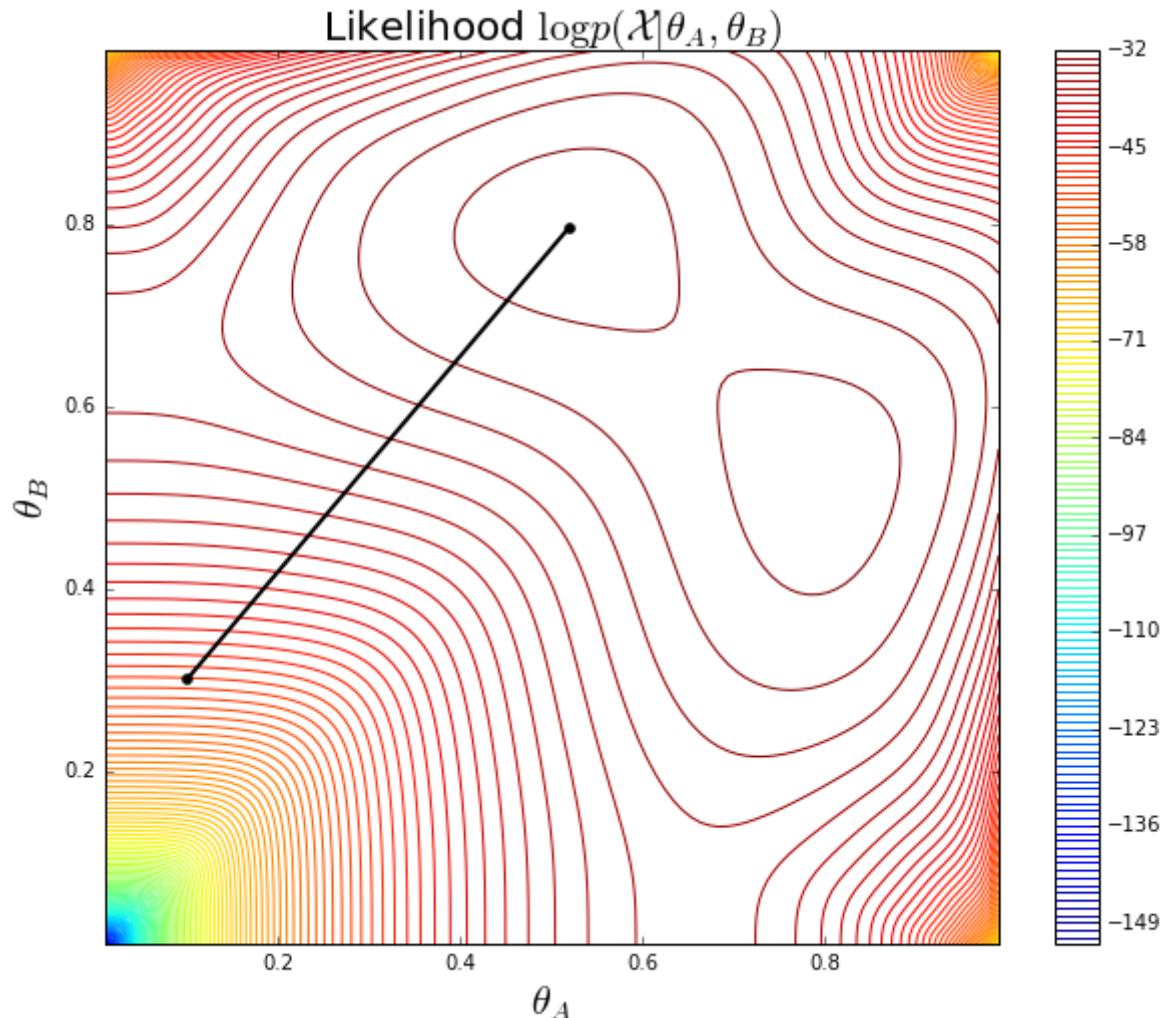
Coin Flip: Demo

```
In [12]: rolls = [ "HTTTHHTHTH", "HHHHTHHHHH", "HTHHHHHTHH",
                 "HTHTTTHHTT", "THHHTHHHTH" ];
thetas, _ = coin_em(rolls, 0.1, 0.3, maxiter=6);

#0: 0.10 0.30
#1: 0.43 0.66
#2: 0.50 0.75
#3: 0.51 0.78
#4: 0.52 0.79
#5: 0.52 0.79
```

Coin Flip: Plot of Convergence

```
In [13]: plot_coin_likelihood(rolls, thetas)
```



Gaussian Mixture Models: Specification

Recall from earlier the **Gaussian Mixture Model**:

$$\begin{array}{ll} \theta = (\pi, \mu, \Sigma) & \text{model parameters} \\ z_n \sim \text{Categorical}[\pi] & \text{cluster indicators} \\ x_n | z_n, \theta \sim \mathcal{N}(\mu_{z_n}, \Sigma_{z_n}) & \text{base distribution} \end{array}$$

GMM: Complete Likelihood

The complete data log-likelihood for a single datapoint (x_n, z_n) is

$$\begin{aligned}\log p(x_n, z_n | \theta) &= \log \prod_{k=1}^K \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)^{\mathbb{I}(z_n=k)} \\ &= \sum_{k=1}^K \mathbb{I}(z_n = k) \log \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)\end{aligned}$$

GMM: Hidden Posterior

The hidden posterior for a single point (x_n, z_n) can be found using Bayes' rule:

$$\begin{aligned}p(z_n = k | x_n, \theta) &= \frac{P(z_n = k | \theta) p(x_n | z_n = k, \theta)}{p(x_n | \theta)} \\ &= \frac{\pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)}{\sum_{k'=1}^K \pi_{k'} \mathcal{N}(x_n | \mu_{k'}, \Sigma_{k'})}\end{aligned}$$

GMM: E-Step

The **expected complete likelihood** is $\sum_{n=1}^N \sum_{k=1}^K r_{nk} \log \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)$

- $\sum_{n=1}^N \sum_{k=1}^K r_{nk} \log \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)$

$$r_{nk} = \frac{1}{\sum_{k'=1}^K \pi_{k'} \mathcal{N}(x_n | \mu_{k'}, \Sigma_{k'})}$$

$\sum_{n=1}^N \sum_{k=1}^K r_{nk} \log \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)$

where $r_{nk} \equiv p(z_n = k | x_n, \theta_t)$ is the **responsibility** that cluster k takes for datapoint x_n after step t .

GMM: M-Step

During the M-Step, we optimize the lower bound with respect to $\theta = (\pi, \mu, \Sigma)$. Verify that the correct updates are

$$\begin{aligned}\pi_k &= \frac{1}{N} \sum_{n=1}^N r_{nk} = \frac{r_k}{N} \quad \mu_k = \frac{\sum_{n=1}^N r_{nk} x_n}{r_k} \\ \Sigma_k &= \frac{\sum_{n=1}^N r_{nk} x_n x_n^T}{r_k} - \mu_k \mu_k^T\end{aligned}$$

where $r_k = \sum_{n=1}^N r_{nk}$ is the effective sample size for cluster k .

General Advice for EM

1. **Specify the model.** Identify the observed variables, hidden variables, and parameters. Draw a graphical model.
2. **Identify relevant probabilities.** This includes the complete data likelihood $p(X, Z|\theta)$ and hidden posterior $p(Z|X, \theta)$.
3. **Derive the E-Step.** Compute $E_q[\log p(\mathcal{X}, Z|\theta)]$.
4. **Derive the M-Step.** Take derivatives and set to zero, when possible. Otherwise, gradient-based methods can be used.