Assignment 5

Callum Lau, 19102521

Approximate Inference

January 15, 2020

1 Mean-field Learning

1.1 Part (a)

The binary latent factor model is as follows-

· vector $\boldsymbol{s}$ of K latent binary factors

· observed vector $\boldsymbol{x}$

· parameters $\boldsymbol{\theta} = \{\{\boldsymbol{\mu}_i, \pi_i\}_{i=1}^K, \sigma^2\}$

With the model described by-

$$p(\boldsymbol{s}|\boldsymbol{\pi}) = \prod_{i=1}^K p(s_i|\pi_i) = \prod_{i=1}^K \pi_i^{s_i}(1-\pi_i)^{(1-s_i)} \tag{1}$$

$$p(\boldsymbol{x}|\boldsymbol{s}, \boldsymbol{\mu}, \sigma^2) = \mathcal{N}\left(\sum_i s_i \boldsymbol{\mu}_i, \sigma^2 I\right) \tag{2}$$

We also make the mean field approximation-

$$q_n(\boldsymbol{s}^{(n)}) = \prod_{i=1}^K \lambda_{in}^{s_i^{(n)}}(1-\lambda_{in}^{(1-s_i^{(n)})}) \tag{3}$$

From which it can be shown that-

$$\langle s_i^{(n)} \rangle = \lambda_{in}, \quad \langle s_i^{(n)} s_j^{(n)} \rangle = \lambda_{in}\lambda_{jn} + \delta_{ij}(\lambda_{in} - \lambda_{in}^2) \tag{4}$$

We now write down the free energy and derive the Variational E-step using the Mean-Field approximation-

$$\mathcal{F}(\boldsymbol{q}, \boldsymbol{\theta}) = \langle \log p(\boldsymbol{s}, \boldsymbol{x}|\boldsymbol{\theta}) \rangle_{q(\boldsymbol{s})} - \langle \log q(\boldsymbol{s}) \rangle_{q(\boldsymbol{s})} \tag{5}$$

Consider the $\log p(\boldsymbol{s}, \boldsymbol{x}|\boldsymbol{\theta})$ term:

$$\begin{aligned}
\log p(\boldsymbol{s}, \boldsymbol{x}|\boldsymbol{\theta}) &= \log p(\boldsymbol{x}|\boldsymbol{s}, \boldsymbol{\theta}) + \log p(\boldsymbol{s}|\boldsymbol{\theta}) \\
&= \log p(\boldsymbol{x}|\boldsymbol{s}, \boldsymbol{\mu}, \sigma^2) + \log p(\boldsymbol{s}|\boldsymbol{\pi}) \\
&\propto \sum_{n=1}^{N}\sum_{i=1}^{K} \Big(s_i^{(n)} \log \pi_i + (1 - s_i^{(n)}) \log(1 - \pi_i) \Big) - ND \log \sigma \\
&\quad - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (\boldsymbol{x}^{(n)} - \sum_i s_i^{(n)} \boldsymbol{\mu_i})^\top (\boldsymbol{x}^{(n)} - \sum_i s_i^{(n)} \boldsymbol{\mu_i}) \\
&= \sum_{n=1}^{N}\sum_{i=1}^{K} \Big(s_i^{(n)} \log \pi_i + (1 - s_i^{(n)}) \log(1 - \pi_i) \Big) - ND \log \sigma \\
&\quad - \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left(\boldsymbol{x}^{(n)\top} \boldsymbol{x}^{(n)} - 2 \sum_i s_i^{(n)} \boldsymbol{\mu_i}^\top \boldsymbol{x}^{(n)} + \sum_i \sum_j s_i^{(n)} s_j^{(n)} \boldsymbol{\mu_i}^\top \boldsymbol{\mu_j} \right)
\end{aligned}$$

Which upon taking the expectation w.r.t $q(\boldsymbol{s})$, i.e. using equation (4) we get:

$$\begin{aligned}
\langle \log p(\boldsymbol{s}, \boldsymbol{x}|\boldsymbol{\theta}) \rangle_{q(\boldsymbol{s})} &= \sum_{n=1}^{N}\sum_{i=1}^{K} (\lambda_{in} \log \pi_i + (1 - \lambda_{in}) \log(1 - \pi_i)) - ND \log \sigma \\
- \frac{1}{2\sigma^2} \sum_{n=1}^{N} &\left(\boldsymbol{x}^{(n)\top} \boldsymbol{x}^{(n)} - 2 \sum_i \lambda_{in} \boldsymbol{\mu_i}^\top \boldsymbol{x}^{(n)} + \sum_i \sum_j \lambda_{in} \lambda_{jn} \boldsymbol{\mu_i}^\top \boldsymbol{\mu_j} + \sum_i (\lambda_{in} - \lambda_{in}^2) \boldsymbol{\mu_i}^\top \boldsymbol{\mu_i} \right)
\end{aligned} \tag{6}$$

Consider the $\log q(\boldsymbol{s})$ term:

$$\langle \log q(\boldsymbol{s}) \rangle_{q(\boldsymbol{s})} = \sum_n \sum_i \lambda_{in} \log \lambda_{in} + (1 - \lambda_{in}) \log(1 - \lambda_{in}) \tag{7}$$

Combining equations (6) and (7), we can write down the Free Energy:

$$\begin{aligned}
\mathcal{F}(\boldsymbol{q}, \boldsymbol{\theta}) &= \sum_{n=1}^{N}\sum_{i=1}^{K} \left(\lambda_{in} \log(\frac{\pi_i}{\lambda_{in}}) + (1 - \lambda_{in}) \frac{\log(1 - \pi_i)}{\log(1 - \lambda_{in})} \right) \\
&\quad - ND \log \sigma - \frac{1}{2\sigma^2} \sum_n (\boldsymbol{x}^{(n)} - \sum_i \lambda_{in} \boldsymbol{\mu_i})^\top (\boldsymbol{x}^{(n)} - \sum_i \lambda_{in} \boldsymbol{\mu_i}) \\
&\quad - \frac{1}{2\sigma^2} \sum_n \sum_i (\lambda_{in} - \lambda_{in}^2) \boldsymbol{\mu_i}^\top \boldsymbol{\mu_i} - \frac{ND}{2} \log(2\pi)
\end{aligned} \tag{8}$$

In order to find the $\boldsymbol{\lambda}^{(n)}$'s that maximise $\mathcal{F}$, we differentiate w.r.t λ_{in}:

$$\frac{\partial \mathcal{F}}{\partial \lambda_{in}} = \log \frac{\pi_i}{1 - \pi_i} - \log \frac{\lambda_{in}}{1 - \lambda_{in}} + \frac{1}{\sigma^2} (\boldsymbol{x}^{(n)} - \sum_{j \neq i} \lambda_{jn} \boldsymbol{\mu}_j)^\top \boldsymbol{\mu}_i - \frac{1}{2\sigma^2} \boldsymbol{\mu}_i \top \boldsymbol{\mu}_i \tag{9}$$

Which setting to zero gives a fixed point equation for λ_{in}:

$$\lambda_{in} = f\left(\log\frac{\pi_i}{1-\pi_i} + \frac{1}{\sigma^2}(\boldsymbol{x}^{(n)} - \sum_{j\neq i}\lambda_{jn}\boldsymbol{\mu}_j)^\top\boldsymbol{\mu}_i - \frac{1}{2\sigma^2}\boldsymbol{\mu}_i{}^\top\boldsymbol{\mu}_i\right) \tag{10}$$

Where f is the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$

We can therefore write a python function:
`mean_field(X,mu,sigma,pie,lambda0,maxsteps)` which performs the variational E-step by solving the fixed point equation (10) and computing the corresponding free energy via equation (8).

```
def mean_field(X, mu, sigma, pie, lambda0, maxsteps):
    """
    -- lambda0 is N x K
    -- F is lower bound on likelihood
    -- X is N x D data matrix
    -- mu is D x K matrix of means
    -- pie is 1 x K vector of priors
    """
    N, D = X.shape
    K = mu.shape[1]
    Lambda = lambda0.copy()
    f = []
    eps = 1
    cIter = 0
    constant = 1e-15

    while abs(eps) > 1e-15 and cIter <= maxsteps:

        # Account for zero and one values for lambdas (to avoid NaNs)
        Lambda[np.isclose(Lambda, 0)] = constant
        Lambda[np.isclose(Lambda, 1)] = 1 - constant

        # Calculate Free Energy according to equation (8)
        term_1 = np.sum(np.multiply(Lambda, np.log(pie/Lambda)))
        term_2 = np.sum(np.multiply((1-Lambda), np.log((1-pie)/(1-Lambda))
                                                  ))
        term_3 = -N*D*np.log(sigma)
        weights = (X - Lambda @ mu.T)
        term_4 = -np.trace((2*sigma**2)**-1*(weights @ weights.T))
        term_5 = -np.sum((2*sigma**2)**-1*(Lambda - Lambda**2) @ np.diag(
                                                  mu.T @ mu))
        term_6 = -N*(D/2)*np.log(2*np.pi)
        f_ = np.sum(term_1 + term_2 + term_3 + term_4 + term_5 + term_6)

        # Terminate iteration if Free Energy is unchanged
        if(cIter > 0):
            eps = f_ - f[cIter - 1]
        f.append(f_)
        # Estimate lambdas according to fixed point equation (10)
        for i in range(K):
            term_1 = np.repeat(np.log(pie[:,i])-np.log((1-pie[:,i])), N)
            weight = np.delete(Lambda, i, 1) @ np.delete(mu, i, 1).T
            term_2 = (sigma**-2)*(X - weight) @ mu[:,i]
            term_3 = -np.repeat(1/(2*sigma**2)*np.dot(mu[:,i], mu[:,i]),N)
            Lambda[:,i] = sigmoid(term_1 + term_2 + term_3)
        cIter += 1

    # Return the updated Lambda estimates and final Free Energy value
    return Lambda, f[-1]
```

Note that looping over the data points is avoided in all calculations and only a loop over the K binary latents is required.

1.2 Part (b)

We form the free energy expression (8) partly by evaluating the expected value of $p(\boldsymbol{x}|\boldsymbol{s}, \boldsymbol{\mu}, \sigma^2)$ - the likelihood of the data - which is a Gaussian expression. Hence, after taking the logarithm, we differentiate in the M step a squared 'loss' term:

$$-\frac{1}{2\sigma^2}\sum_n(\boldsymbol{x}^{(n)} - \sum_i \lambda_{in}\boldsymbol{\mu_i})^\top(\boldsymbol{x}^{(n)} - \sum_i \lambda_{in}\boldsymbol{\mu_i})$$

w.r.t to the 'weights' $\boldsymbol{\mu}$, in order to find the $\hat{\boldsymbol{\mu}}$ which minimise this expression, which in turn increases the free energy (since the expression is negative). The solution takes the form:

$$\hat{\boldsymbol{\mu}_j} = \sum_i \left(\sum_{n=1}^N \langle \boldsymbol{s}^{(n)}\boldsymbol{s}^{(n)\top}\rangle_{q(\boldsymbol{s}^{(n)})}\right)^{-1}_{ji} \sum_{n=1}^N \langle \boldsymbol{s}^{(n)}\rangle_{q(\boldsymbol{s}^{(n)})}\boldsymbol{x}^{(n)}$$

which is analogous to:

$$\hat{\boldsymbol{\beta}} = (SS^\top)^{-1}SX$$

i.e. the solution to the linear regression problem $X = S^\top\beta + \epsilon$ for output X, input S, weights β and Gaussian distributed error ϵ. In fact we see that in finding this solution we also have to minimise a squared loss term $(X - S^\top\beta)^\top(X - S^\top\beta)$ in order to find the optimal weights $\hat{\beta}$.

1.3 Part (c)

Firstly note-

· ESS is K x K

· ES is N x K

· mu is D x K

· X is N x D

· pie is 1 x K

We now evaluate each expression in the code - mu, sigma and pie - solutions to evaluate the computational complexity:

mu
- inv(ESS) is inversion of K x K matrix is O(K^3)
- inv(ESS)*ES' is a (K x K) x (K x N) matrix mult: O(K^2N)
- (inv(ESS)*ES')*X is a (K x N) x (N x D) matrix mult: O(KND)

sigma
- X'*X is a (D x N) x (N x D) matrix mult: O(D^2N)
- mu'*mu is a (K x D) x (D x K) matrix mult: O(K^2D)
- (mu'*mu)*ESS is a (K x K) x (K x K) matrix mult: O(K^3)
- ES'*X is a (K x N) x (N x D) matrix mult: O(KND)
- (ES'*X)*mu is a (K x D) x (D x K) matrix mult: O(K^2D)

Aggregating all these terms, the total complexity is

$$O(K^3 + K^2N + KND + D^2N + K^2D)$$

1.4 Part (d)

It appears that there 8 features, given by:

1. A vertical line in the 2nd column
2. A vertical line in the 4th column
3. A horizontal line in the 1st (top) row
4. A cross centred at the 2nd row, 3rd column
5. A (2 x 2) box in the center
6. A (2 x 2) box in the bottom left
7. A 'frame' covering all pixels apart from the (2 x 2) center
8. A diagonal line from top left to bottom right

An analysis of the suitable models is given below:

1. Factor Analysis would not be a good model for this data. The key assumption for FA is that the factors are Gaussian distributed. However, upon analysing the 8 different features, it seems much more likely that the factors $\boldsymbol{s}$ are not Gaussian distributed and instead take binary values, and are perhaps multivariate bernoulli distributed. These factors randomly 'activate' the different sources μ that we see in the images, along with some Gaussian noise. Therefore FA would not be a good model for the data.

2. In ICA we are trying to separate non-Gaussian and linearly combined sources. In our case, this would mean the K latents $\boldsymbol{s}$ are non-Gaussian and instead multivariate Bernoulli distributed. These then dictate the mixing proportions of the sources $\boldsymbol{\mu}$. Hence, the assumptions are exactly the same as ICA. Indeed, upon examining the generated images this seems to be the case. In our previous analysis we have found 8 independent sources, and each seems to be some random and linear combination of these sources. Therefore we conclude that ICA would do a good job modelling our data.

3. Again we observe that the images seen to be generated by linearly combining independent sources $\boldsymbol{\mu}$, and hence all images seem to be some mixture of the different sources with added noise. On the other hand, in a MoG model we would assume that each image was generated by a single multinomial draw from all the different mixtures, and hence each image would correspond to just one of these mixtures. However, by inspecting the images this is clearly not the case, since again, it appears much more likely that each image corresponds to some combination of different mixtures/sources. Therefore MoG would be a poor model for this data. If we were to perform MoG we would see that the algorithm will cluster images which look similar together, rather than separating out the different features.

1.5 Part (e)

Using the M-step code provided, we produce the code for `learn_bin_factors(X,K,iterations)` below:

```python
def learn_bin_factors(X, K, iterations):

    N, D = X.shape
    Lambda, mu, pie, sigma = init_params(N,D,K)
    maxsteps = 30
    free_energies = []

    for cIter in range(iterations):

        Lambda, f = mean_field(X, mu, sigma, pie, Lambda, maxsteps)
        free_energies.append(f)
        print(f)
        ESS = compute_ESS(Lambda)
        mu, sigma, pie = m_step(X, Lambda, ESS)

    return mu, sigma, pie

def init_params(N, D, K):
    Lambda = 0.25 + np.random.rand(N,K)/2.0
    mu = 0.25 + np.random.rand(D,K)/2.0
    pie = 0.25 + np.random.rand(1,K)/2.0
    sigma = 1

    return Lambda, mu, pie, sigma
```

```
def compute_ESS(Lambda):
    """
    Computes E_q[ss'] (N, K, K), using Lambda matrix.
    """
    N, K = Lambda.shape
    ESS = np.zeros(shape=(N,K,K))

    for n in range(N):
        lambda_n = Lambda[n,:]
        for k_1 in range(K):
            for k_2 in range(K):
                ESS[n,k_1,k_2] = lambda_n[k_1]*lambda_n[k_2]
        diag = lambda_n - lambda_n**2
        ESS[n,:,:] += np.diag(diag)
    return ESS
```

1.6 Part (f)

We then ran the algorithm on the generated images. Firstly to check the log-likelihood was increasing at all times we plotted the free energy:

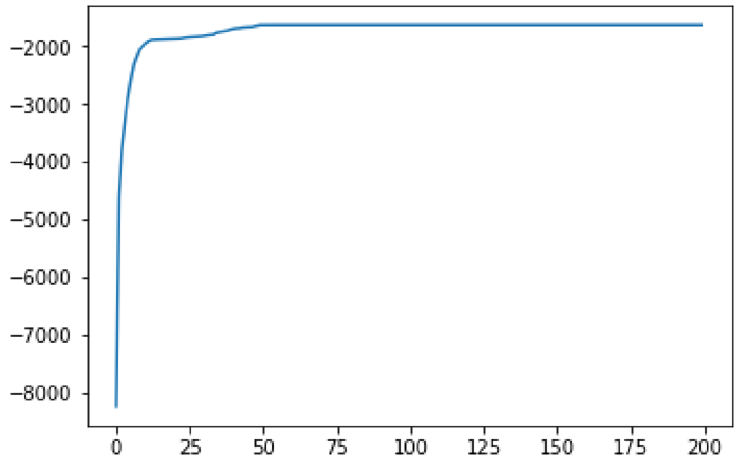


Figure 1: Plot of the Free Energy per iteration

The features that were found are given below:

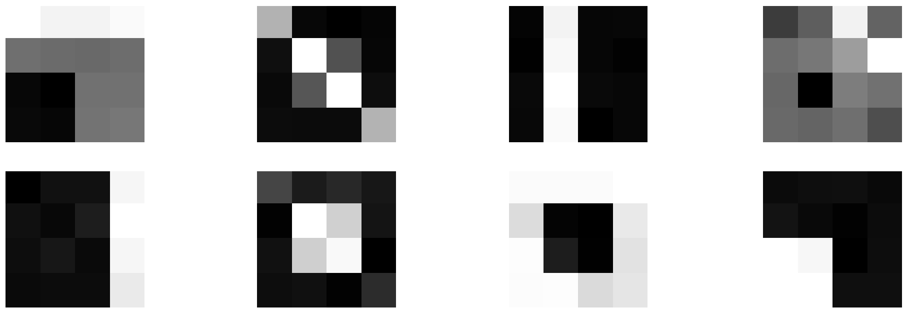


Figure 2: Features for K = 8

- As we can see the algorithm has correctly identified 7 of the 8 different features we were looking for (with the 'cross' feature not present). On multiple runs it seemed to be the case that at least one feature was not found.

- We set K = 8 since in our analysis in part (d) we observed that there seemed to be 8 features present throughout all the generated data. However, we could have improved the algorithm by setting K to a much higher value e.g. K = 32, and implementing an automatic relevance detection algorithm to determine the number of features automatically. This algorithm would automatically eliminate sources that were redundant, and so, instead of relying upon inspection to determine the value of K, the algorithm could determine it itself.

- The algorithm could also be improved by using thresholding to hard assign the values of **mu**. We know we are trying to recover binary sources so thresholding should give us clearer results. For values of **mu** below 0.4, the pixel was assigned to 0, otherwise the pixel was assigned to 1. The following **mu**'s had thresholding applied:

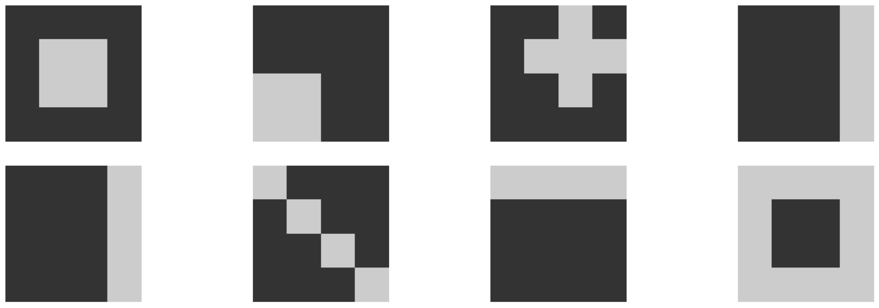


Figure 3: Features with thresholding

- The Lambdas, pies, mus and sigmas were initialised uniformly at random in the range 0.25 to 0.75. The rationale behind this was that the images composed of random features had on average roughly a mean of 0.5. Clearly the only constraint for the Lambda, pie and mu values is that they should be between 0 and 1. However, initialising their values away from the extremal values of 0 and 1 gave much better results, possibly because it avoids the possibility assigning an extremely value for a pixel leading to some hard to 'get out of' local minima during the EM iterations.

1.7 Part (g)

The plots of $\mathcal{F}$ and $\mathcal{F}(t) - \mathcal{F}(t-1)$ for $\sigma = 1, 2, 5$ are given below:

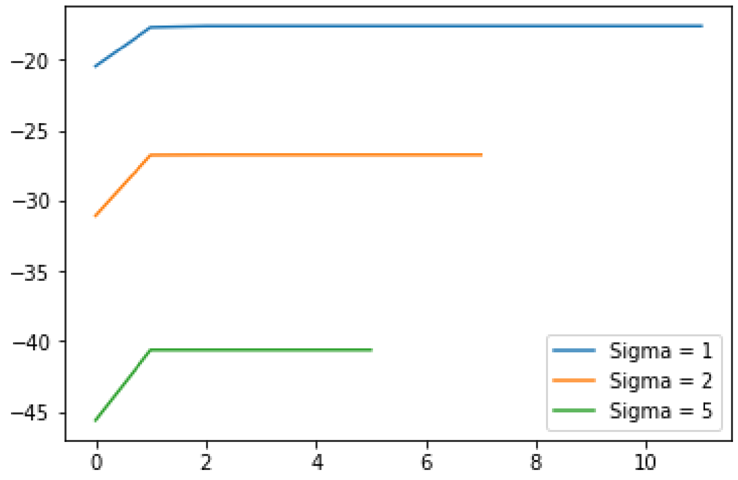


Figure 4: $\mathcal{F}$

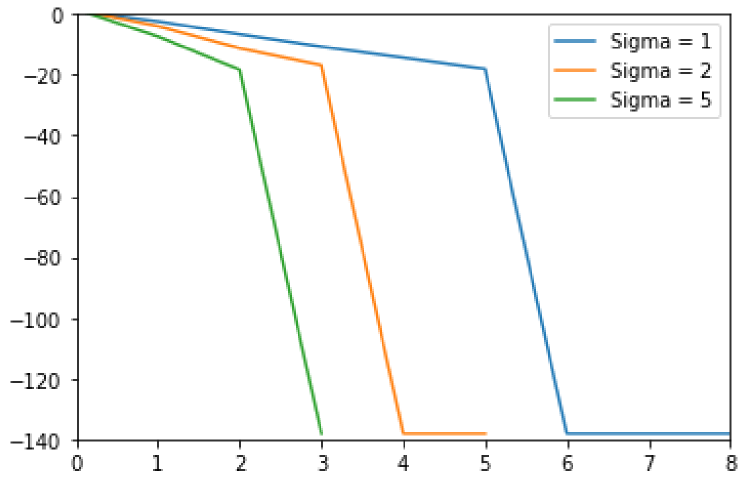


Figure 5: Plot of free energy differences $\log(\mathcal{F}(t) - \mathcal{F}(t-1))$

- The rate of convergence depends on the value of sigma. As shown by Figure 5, for larger sigma, the faster the free energy converges. For example for Sigma = 5, the free energy converges in 3 iterations, whereas for sigma = 1, it converges in 6 iterations. This may be because of equation (9) - the fixed point equation for lambda - where sigma appears in the denominator. For larger sigma, therefore, we expect the inverse sigma terms to be less impactful, and the solution to the fixed point equation to converge much faster.

- As seen by figure 4, the free energy is higher for smaller sigma, which is likely because sigma = 1 is much closer to the true value of sigma that generated the data, and the model is much closer to representing the data

The reason for the differences can be explained mathematically in equation (9) in Part (a), which is the fixed point equation for lambda:

$$\frac{\partial \mathcal{F}}{\partial \lambda_{in}} = \log \frac{\pi_i}{1-\pi_i} - \log \frac{\lambda_{in}}{1-\lambda_{in}} + \frac{1}{\sigma^2}(\boldsymbol{x}^{(n)} - \sum_{j\neq i} \lambda_{jn}\boldsymbol{\mu}_j)^\top \boldsymbol{\mu}_i - \frac{1}{2\sigma^2}\boldsymbol{\mu}_i{}^\top \boldsymbol{\mu}_i \tag{11}$$

If sigma is small, we expect the inverse sigma term to contribute much more to the term. Since this is the only non-constant term in the equation, (due to the $\sum_{j\neq i} \lambda_{jn}\boldsymbol{\mu}_j$ term), we would expect the solution to take longer to converge as these terms become more important. There is also an intuitive reason. With larger σ, the convergent solution the algorithm gives has a much wider scope to be explained by the noise, whereas for smaller σ this noise is very small, and so the model parameters have to be tuned much more accurately to make the free energy converge.

2 Noisy ICA

2.1 Part (a)

$$\begin{aligned}
p(z_k) &= \int p(z_k|u_k)p(u_k)du_k = \int \frac{1}{(2\pi u_k^{-1})^{\frac{1}{2}}} e^{(-\frac{1}{2}z_k^2 u_k)} \frac{(\frac{\alpha}{2})^{\frac{\alpha}{2}}}{\Gamma(\frac{\alpha}{2})} u_k^{(\frac{\alpha}{2}-1)} e^{(-\frac{\alpha}{2}u_k)} du_k \\
&= \frac{1}{\sqrt{2\pi}} \frac{(\frac{\alpha}{2})^{\frac{\alpha}{2}}}{\Gamma(\frac{\alpha}{2})} \int u_k^{\frac{\alpha-1}{2}} e^{-\frac{1}{2}u_k(z_k^2+\alpha)} du_k \\
&= \frac{1}{\sqrt{2\pi}} \frac{(\frac{\alpha}{2})^{\frac{\alpha}{2}}}{\Gamma(\frac{\alpha}{2})} \frac{\Gamma(\frac{\alpha+1}{2})}{(\frac{z_k^2+\alpha}{2})^{(\frac{\alpha+1}{2})}} \int \frac{(\frac{z_k^2+\alpha}{2})^{(\frac{\alpha+1}{2})}}{\Gamma(\frac{\alpha+1}{2})} u_k^{\frac{\alpha-1}{2}} e^{-\frac{1}{2}u_k(z_k^2+\alpha)} du_k \\
&= \frac{1}{\sqrt{2\pi}} \frac{(\frac{\alpha}{2})^{\frac{\alpha}{2}}}{\Gamma(\frac{\alpha}{2})} \frac{\Gamma(\frac{\alpha+1}{2})}{(\frac{z_k^2+\alpha}{2})^{(\frac{\alpha+1}{2})}} \int \frac{\beta^\alpha}{\Gamma(\alpha)} u_k^{\alpha-1} e^{-\beta u_k} du_k
\end{aligned}$$

where $\alpha = \frac{\alpha+1}{2}$ and $\beta = \frac{z_k^2+\alpha}{2}$. Therefore we are integrating the Gamma distribution over its support, and so it integrates to one:

$$\begin{aligned}
p(z_k) &= \frac{1}{\sqrt{2\pi}} \frac{(\frac{\alpha}{2})^{\frac{\alpha}{2}}}{\Gamma(\frac{\alpha}{2})} \frac{\Gamma(\frac{\alpha+1}{2})}{(\frac{z_k^2+\alpha}{2})^{(\frac{\alpha+1}{2})}} \\
&= \frac{1}{\sqrt{2\pi}} \left(\frac{2}{\alpha}\right)^{\frac{1}{2}} \left(\frac{2}{\alpha}\right)^{-\frac{\alpha+1}{2}} \frac{\Gamma(\frac{\alpha+1}{2})}{\Gamma(\frac{\alpha}{2})} \left(\frac{z_k^2+\alpha}{2}\right)^{-\frac{\alpha+1}{2}}
\end{aligned}$$

$$= \frac{1}{\sqrt{\pi\alpha}} \frac{\Gamma(\frac{\alpha+1}{2})}{\Gamma(\frac{\alpha}{2})} \left(1 + \frac{z_k^2}{\alpha}\right)^{-\frac{\alpha+1}{2}}$$

Which we recognise as a Student t-distribution. A nice plot is given by Wikipedia:

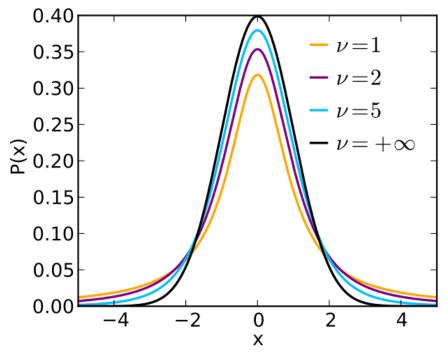


Figure 6: Student t-distribution

This model is related to ICA, since the marginal latents $P(z_k)$ are not Gaussian distributed. and similar to ICA we take a linear combination of these non-Gaussian sources via $A\boldsymbol{z}$. However in this model we are assuming that the outputs $\boldsymbol{x}$ are Gaussian distributed.

2.2 Part (b)

Firstly we note down the Graphical Model structure:

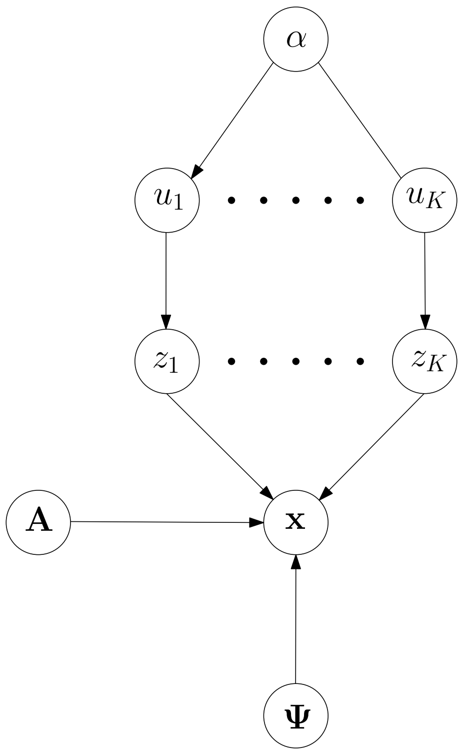


Figure 7: Noisy ICA Graphical Model

Upon observing $\boldsymbol{x}$, moralisation of the parents will introduce loopiness in the graph structure. A minimal VB factorisation therefore would be to break the $\boldsymbol{Z}$ and $\boldsymbol{U}$ variables into separated chains so that $Q(Z, U) = \prod_k Q(z_k, u_k)$ (note we do not include α as it is a hyperparameter). The full factorisation over the latents and parameters is:

$$Q(Z, U, A, \Psi) = Q(\Psi, A) \prod_k Q(z_k, u_k) \tag{12}$$

2.3 Part (c)

We can use the formula in the lectures to find the update for $Q(\Psi, A)$:

$$\begin{aligned} Q(\Psi, A) &\propto P(\Psi, A) \exp \langle \log p(X, \boldsymbol{Z}, \boldsymbol{U} | \Psi, A) \rangle_{\prod_k Q(z_k, u_k)} \\ &= P(\Psi, A) \exp \langle \log p(X | \boldsymbol{Z}, \Psi, A) + \log p(\boldsymbol{Z}, \boldsymbol{U}) \rangle_{\prod_k Q(z_k, u_k)} \end{aligned}$$

ignoring constant terms w.r.t. Ψ, A

$$\propto P(\Psi, A) \exp \langle \log p(X | \boldsymbol{Z}, \Psi, A) \rangle_{\prod_k Q(z_k, u_k)}$$

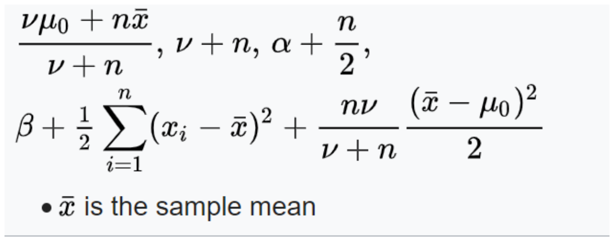
Note that the model of ICA: $x_k = \sum_d A_{kd} z_d + \Psi_{kk}$ has independent components z_d. This implies the further factorisation $Q(\Psi, A) = \prod_{k=1} Q(\Psi_{kk}, A_k)$ where $A_k^\top$ is the kth column of A. Note that we can also write (from the graphical model) $Q(z_k, u_k) = Q(z_k | u_k) Q(u_k)$. Following the formula in the lectures therefore we know that:

$$\begin{aligned} Q(\Psi_{kk}, A_k) &\propto P(\Psi_{kk}, A_k) \exp \langle \log p(X | \boldsymbol{x}, \Psi_{kk}, A_k) \rangle_{\prod_k Q(z_k | u_k)} \\ &\propto P(\Psi_{kk}, A_k) \frac{1}{\Psi_{kk}^{\frac{N}{2}}} \exp \sum_{n=1}^{N} \left\langle -\frac{1}{2\Psi_{kk}^2} (x_k^{(n)} - A_k \boldsymbol{z})^2 \right\rangle_{\prod_k Q(z_k | u_k)} \end{aligned} \tag{13}$$

The likelihood therefore is just a normal with unknown mean $A_k \langle \boldsymbol{z} \rangle$ and variance Ψ_{kk}. From Wikipedia, we know that the conjugate prior for such a distribution is the 4-parameter distribution the normal-inverse gamma, i.e. we want $P(\Psi_{kk}, A_k)$ to be a normal inverse gamma prior:

$$\Psi_{kk}, A_k \sim \text{NIG}(\mu_0, \nu, \alpha, \beta) \tag{14}$$

Below is an image taken from Wikipedia of the posterior updates for a normal inverse gamma prior multiplied by a Gaussian likelihood:



$$\frac{\nu\mu_0 + n\bar{x}}{\nu + n}, \; \nu + n, \; \alpha + \frac{n}{2},$$
$$\beta + \tfrac{1}{2} \sum_{i=1}^{n} (x_i - \bar{x})^2 + \frac{n\nu}{\nu + n} \frac{(\bar{x} - \mu_0)^2}{2}$$

- $\bar{x}$ is the sample mean

Figure 8: Posterior updates

Applying this to our case we get the updates for $Q(\Psi_{kk}, A_k)$:

$$\tilde{\mu} \leftarrow \frac{\nu\mu_0 + N\bar{x}_k}{\nu + N} \tag{15}$$

$$\tilde{\nu} \leftarrow \nu + N \tag{16}$$

$$\tilde{\alpha} \leftarrow \alpha + \frac{N}{2} \tag{17}$$

$$\tilde{\beta} \leftarrow \beta + \frac{1}{2}\sum_{i=1}^{N}(x_k^{(n)} - \bar{x}_k)^2 + \frac{N\nu}{\nu + N}\frac{(\bar{x}_k - \mu_0)^2}{2} \tag{18}$$

2.4 Part (d)

2.4.1 (i)

The shape of the marginal distribution $P(z_k)$ is student-t distribution parametrised by the α parameter (as derived in Part a). Hence if we want to learn the shape of this distribution we must perform hyperparameter optimisation on α. Hence if we now make the VB free energy of this model now dependent on this parameter we get (where $\theta = \Psi, A$):

$$\mathcal{F}(Q_{z|u}, Q_u, Q_\theta, \alpha) = \int\int\int dz du d\theta Q_{z|u} Q_u Q_\theta \log \frac{p(X, \boldsymbol{z}, \boldsymbol{u}, \theta|\alpha)}{Q_{z|u} Q_u Q_\theta} \leq p(X|\alpha) \tag{19}$$

Note that the denominator of the integral expression is not dependent on α, and hence we get the optimisation rule:

$$\alpha \leftarrow \underset{\alpha}{\text{argmax}} \int\int\int dz du d\theta Q_{z|u} Q_u Q_\theta (X, \boldsymbol{z}, \boldsymbol{u}, \theta|\alpha) \tag{20}$$

From the graphical model of this distribution we can further decompose:

$$\begin{aligned} P(X, \boldsymbol{z}, \boldsymbol{u}, \theta|\alpha) &= P(X|\boldsymbol{z}, \cancel{\boldsymbol{u}}, \Psi, A, \cancel{\alpha}) P(\boldsymbol{z}, \boldsymbol{u}, \Psi, A|\alpha) \\ &= P(X|\boldsymbol{z}, \Psi, A) P(\boldsymbol{z}, \boldsymbol{u}|\cancel{\Psi}, \cancel{A}, \alpha) P(\Psi, A|\cancel{\alpha}) \\ &= P(X|\boldsymbol{z}, \Psi, A) P(\boldsymbol{z}|\boldsymbol{u}, \cancel{\alpha}) P(\boldsymbol{u}|\alpha) P(\Psi, A) \\ \implies \log P(X, \boldsymbol{z}, \boldsymbol{u}, \theta|\alpha) &= \log P(X|\boldsymbol{z}, \Psi, A) + \log P(\boldsymbol{z}|\boldsymbol{u}) + \log P(\boldsymbol{u}|\alpha) + \log P(\Psi, A) \end{aligned}$$

We see that the only term dependent on α is $\log P(\boldsymbol{u}|\alpha)$ and therefore we write the simplified optimisation rule as:

$$\alpha \leftarrow \underset{\alpha}{\text{argmax}} \int Q_u \log P(\boldsymbol{u}|\alpha) du \tag{21}$$

Now note:

$$\begin{aligned} \langle \log P(\boldsymbol{u}|\alpha) \rangle_{Q_u} &= \langle \prod_{k=1}^{K} p(u_k|\alpha) \rangle_{Q_{u_k}} = \sum_{k=1}^{K} \left\langle \log \left(\frac{(\frac{\alpha}{2})^{\frac{\alpha}{2}}}{\Gamma(\frac{\alpha}{2})} u_k^{\frac{\alpha}{2}-1} e^{-\frac{\alpha}{2}u_k} \right) \right\rangle_{Q_{u_k}} \\ &= \sum_{k=1}^{K} \left(\frac{\alpha}{2} \log \frac{\alpha}{2} - \log \Gamma(\frac{\alpha}{2}) + (\frac{\alpha}{2} - 1) \langle \log u_k \rangle_{Q_{u_k}} - \frac{\alpha}{2} \langle u_k \rangle_{Q_{u_k}} \right) \end{aligned}$$

Hence to find the argmax we differentiate w.r.t. α:

$$\frac{\partial}{\partial \alpha} = \frac{K}{2}\log\frac{\alpha}{2} + \frac{K}{2} - \frac{K}{2}\frac{\Gamma'(\frac{\alpha}{2})}{\Gamma(\frac{\alpha}{2})} + \frac{1}{2}\sum_{k=1}^{K}\langle \log u_k\rangle_{Q_{u_k}} - \frac{1}{2}\sum_{k=1}^{K}\langle u_k\rangle_{Q_{u_k}} = 0$$
$$\frac{\partial}{\partial \alpha} = K\log\frac{\alpha}{2} + K - K\frac{\Gamma'(\frac{\alpha}{2})}{\Gamma(\frac{\alpha}{2})} + \sum_{k=1}^{K}\langle \log u_k\rangle_{Q_{u_k}} - \sum_{k=1}^{K}\langle u_k\rangle_{Q_{u_k}} = 0$$

After finding the Q_{u_k} distribution we can then evaluate the expectation $\langle u_k\rangle_{Q_{u_k}}$ and use a numerical solver to find $\langle \log u_k\rangle_{Q_{u_k}}$. Solving this equation gives the maximising value of α.

2.4.2 (ii)

We alter the model parametrisation so that

$$u_k \sim \text{Gamma}(\frac{\alpha_k}{2}, 1) \tag{22}$$

This enables us to learn the number of features since we can then have that the precision u_k^{-1} for each z_k can individually tend to zero. Note that this happens when $\alpha_k \to \infty$, since the probability mass of this Gamma prior becomes skewed towards ever increasing values (i.e. the mean tends to infinity). Effectively this means we can learn the number of factors, as we eliminate z_k with zero variances. Noting the factorisation stemming from the graphical model, we seek the following update:

$$\alpha_k \leftarrow \text{argmax}\langle \log P(u_k|\alpha_k)\rangle_{Q(u_k)}$$

Note that since we have the two-stage representation we know that we can write $Q(z_k, u_k) = Q(z_k|u_k)Q(u_k)$ and therefore we are able to find the marginal posterior on $Q(u_k)$

$$\begin{aligned}\langle \log P(u_k|\alpha_k)\rangle_{Q(u_k)} &= \left\langle \log\left(\frac{1}{\Gamma(\frac{\alpha_k}{2})}u_k^{\frac{\alpha_k}{2}-1}e^{-u_k}\right)\right\rangle_{Q(u_k)} \\ &= \Big(-\log\Gamma(\frac{\alpha_k}{2}) + (\frac{\alpha_k}{2}-1)\langle \log u_k\rangle_{Q(u_k)} - \langle u_k\rangle_{Q(u_k)}\Big)\end{aligned}$$

Hence to find the argmax we differentiate w.r.t. α_k:

$$\begin{aligned}\frac{\partial}{\partial \alpha_k} &= -\frac{1}{2}\frac{\Gamma'(\frac{\alpha_k}{2})}{\Gamma(\frac{\alpha_k}{2})} + \frac{1}{2}\langle \log u_k\rangle_{Q(u_k)} = 0 \\ &= -\frac{\Gamma'(\frac{\alpha_k}{2})}{\Gamma(\frac{\alpha_k}{2})} + \langle \log u_k\rangle_{Q(u_k)} = 0\end{aligned}$$

After finding the $Q(u_k)$ distribution we can then evaluate the expectation $\langle \log u_k\rangle_{Q(u_k)}$. Solving this equation using a numerical solver would then give the maximising value of α. The algorithm would then learn the number of factors as we observe some of th $alpha_k$ tending to infinity via this update rule.

3 Variational Bayes for binary factors

3.1 Part a

In order to perform hyperparameter optimisation we impose a prior on the columns of μ:

$$P(\mu_i|\alpha_i) = \mathcal{N}(0, \alpha_i^{-1}I) \tag{23}$$

Then, following the presentation in the notes we know that the optimisation rule is:

$$\begin{aligned}
\alpha_i &= \underset{\alpha_i}{\operatorname{argmax}} \langle P(\mu_i|\alpha_i)\rangle_{Q_{\mu_i}} \\
&= \underset{\alpha_i}{\operatorname{argmax}} \left(\log(\frac{1}{(2\pi)^{\frac{D}{2}}|\alpha_i^{-1}I|^{\frac{1}{2}}}) - \frac{1}{2}\alpha_i \langle \mu_i^\top \mu_i\rangle_{Q_{\mu_i}} \right) \\
&= \underset{\alpha_i}{\operatorname{argmax}} \left(\frac{D}{2}\log(\alpha_i) - \frac{\alpha_i}{2}\langle \mu_i^\top \mu_i\rangle_{Q_{\mu_i}} \right)
\end{aligned}$$

differentiating w.r.t. α_i:

$$\begin{aligned}
\frac{\partial}{\partial \alpha_i} &= \frac{D}{2\alpha_i} - \frac{1}{2}\langle \mu_i^\top \mu_i\rangle_{Q_{\mu_i}} = 0 \\
\implies \alpha_i &= \frac{D}{\langle \mu_i^\top \mu_i\rangle_{Q_{\mu_i}}}
\end{aligned} \tag{24}$$

Therefore we just need to find the approximating distribution Q_{μ_i}. Note that our variational approximation of the latent distributions now becomes: $Q(Z) = Q(s)\prod_{k=1}^{K} Q(\mu_k)$. And therefore to find Q_{μ_i}, we must take the expectation of the log joint w.r.t $Q(Z) = Q(s)\prod_{k\neq i}^{K} Q(\mu_i)$.
Note the updated binary latent factor model- X is the observed data, latents $Z = \{s, \mu\}$, $\theta = \{\pi, \sigma, \alpha\}$. We can expand the log joint as follows:

$$\begin{aligned}
\log p(X, Z, \theta) &= \log P(X, S, \mu, \pi, \sigma, \alpha) \\
&= \log p(X|s, \mu, \pi, \sigma) + \log p(s|\sigma, \pi) + \log p(\mu|\alpha) + \log p(\sigma) + \log p(\pi)
\end{aligned} \tag{25}$$

Let $Q_{\neg i}(\mu) = \prod_{k\neq i} Q(\mu_k)$ and following the formula in the lectures:

$$Q(\mu_i) = \exp\langle \log p(X, Z, \theta)\rangle_{Q(s)Q_{\neg i}(\mu)} \tag{26}$$

Hence using the equation (23) and dropping all terms constant w.r.t μ:

$$Q(\mu_i) = \exp\langle \log p(X|s, \mu, \pi, \sigma) + \log p(\mu|\alpha)\rangle_{Q(s)Q_{\neg i}(\mu)} \tag{27}$$

We wish to expand this equation, keeping only terms involving μ_i. Firstly, we find that:

$$\langle \log p(\mu|\alpha)\rangle_{Q(s)Q_{\neg i}(\mu)} = \sum_{k=1}^{K} \langle \log p(\mu_k|\alpha_k)\rangle_{Q(s)Q_{\neg i}(\mu)} \propto \sum_{k=1}^{K} \langle \frac{D}{2}\log\alpha_k - \frac{1}{2}\alpha_i \mu_k^\top \mu_k\rangle_{Q(s)Q_{\neg i}(\mu)}$$

$$\propto -\frac{1}{2}\alpha_i \mu_i^\top \mu_i \qquad \text{(since all other terms are constant w.r.t } \mu_i) \tag{28}$$

Secondly we find $\langle \log p(X|s,\mu,\pi,\sigma)\rangle_{Q(s)Q_{\neg i}(\mu)}$.

Note that the model specifies $p(x|s,\mu,\pi,\sigma) = \mathcal{N}(\sum_k \mu_k s_k, \sigma^2 I)$, and therefore:

$$\langle \log p(X|s,\mu,\pi,\sigma)\rangle_{Q(s)Q_{\neg i}(\mu)} = \sum_n \left\langle -\frac{D}{2}\log(2\pi) - D\log\sigma - \frac{1}{2\sigma^2}(x^n - \sum_k s_k^n \mu_k)^\top (x^n - \sum_k s_k^n \mu_k)\right\rangle_{Q(s)Q_{\neg i}(\mu)}$$

Only keeping terms involving μ:

$$\begin{aligned}\langle \log p(X|s,\mu,\pi,\sigma)\rangle_{Q(s)Q_{\neg i}(\mu)} &\propto \sum_n \left\langle -\frac{1}{2\sigma^2}(x^n - \sum_k s_k^n \mu_k)^\top (x^n - \sum_k s_k^n \mu_k)\right\rangle_{Q(s)Q_{\neg i}(\mu)} \\ &\propto -\frac{1}{2\sigma^2}\sum_n \left\langle x^{n\top}x^n - 2x^{n\top}\sum_k s_k^n \mu_k + \sum_j \sum_k s_j^n s_k^n \mu_j^\top \mu_k \right\rangle_{Q(s)Q_{\neg i}(\mu)}\end{aligned}$$

dropping $x^{n\top}x^n$ which has no μ_i dependence:

$$\propto -\frac{1}{2\sigma^2}\sum_n \left\langle -2x^{n\top}\sum_k s_k^n \mu_k + \sum_j \sum_k s_j^n s_k^n \mu_j^\top \mu_k \right\rangle_{Q(s)Q_{\neg i}(\mu)}$$

Again we drop a $\sum_k$ sum since we are only interested in the specific case where $\mu_k = \mu_i$, since all the non μ_i terms are constant w.r.t the distribution $Q(\mu_i)$.

$$\propto -\frac{1}{2\sigma^2}\sum_n \left\langle -2x^{n\top}s_i^n \mu_i + 2\sum_k s_i^n s_k^n \mu_i^\top \mu_k - s_i^n s_i^n \mu_i^\top \mu_i \right\rangle_{Q(s)Q_{\neg i}(\mu)}$$

(note the extra $-s_i^n s_i^n \mu_i^\top \mu_i$ term comes from the double counting of the $\mu_i^\top \mu_i$ term in the sum $2\sum_k s_i^n s_k^n \mu_i^\top \mu_k$.)

$$\begin{aligned}&= \frac{\mu_i^\top}{\sigma^2}\sum_n \left\langle x^n s_i^n - \sum_k s_i^n s_k^n \mu_k + \frac{1}{2}s_i^n s_i^n \mu_i \right\rangle_{Q(s)Q_{\neg i}(\mu)} \\ &= \frac{\mu_i^\top}{\sigma^2}\sum_n \left(x^n \langle s_i^n\rangle - \sum_k \langle s_i^n s_k^n\rangle \langle \mu_k\rangle_{Q(\mu_k)} + \frac{1}{2}\langle s_i^n s_i^n\rangle \mu_i \right) \\ &= \frac{1}{\sigma^2}\sum_n \left(\mu_i^\top x^n \langle s_i^n\rangle - \sum_{k\neq i} \langle s_i^n s_k^n\rangle \mu_i^\top \langle \mu_k\rangle_{Q(\mu_k)} - \frac{1}{2}\langle s_i^n s_i^n\rangle \mu_i^\top \mu_i \right)\end{aligned}$$

Now if we substitute in for the expected values of s_i: $\langle s_i^n\rangle = \lambda_{in}$ and $\langle s_i^n s_k^n\rangle = \lambda_{in}\lambda_{kn} - \delta_{ik}(\lambda_i n - \lambda_{in}^2)$, we get:

$$= \frac{1}{\sigma^2}\sum_n \left(\mu_i^\top x^n \lambda_{in} - \sum_{k\neq i} \lambda_{in}\lambda_{kn}\mu_i^\top \langle \mu_k\rangle_{Q(\mu_k)} - \frac{1}{2}\lambda_{in}\mu_i^\top \mu_i \right) \tag{29}$$

Hence combining equations (26) and (27) we get the result:

$$
\begin{aligned}
Q(\mu_i) &= \exp\left(\frac{1}{\sigma^2}\sum_n \mu_i^\top x^n \lambda_{in} - \frac{1}{\sigma^2}\sum_n\sum_{k\neq i}\lambda_{in}\lambda_{kn}\mu_i^\top\langle\mu_k\rangle_{Q(\mu_k)} - \frac{1}{2\sigma^2}\sum_n \lambda_{in}\mu_i^\top\mu_i - \frac{1}{2}\alpha_i\mu_i^\top\mu_i\right)\\
&= \exp -\frac{1}{2}\left(\mu_i^\top\left[\frac{1}{\sigma^2}\sum_n\lambda_i + \alpha_i\right]\mu_i + \frac{2\mu_i^\top}{\sigma^2}\left[\sum_n\sum_{k\neq i}\lambda_{in}\lambda_{kn}\langle\mu_k\rangle_{Q(\mu_k)} - \sum_n\lambda_{in}x^n\right]\right)
\end{aligned}
\tag{30}
$$

Which we recognise as the form of a Gaussian so that $Q(\mu_i) \sim \mathcal{N}(M_i, C_i I)$. By completing the square we can find its mean and variance. Therefore it must be the case:

$$
\boxed{C_i = \left[\frac{1}{\sigma^2}\sum_n \lambda_i + \alpha_i\right]^{-1}} \tag{31}
$$

$$
\boxed{M_i = \frac{C_i}{\sigma^2}\left[\sum_n \lambda_{in}x^n - \sum_n\sum_{k\neq i}\lambda_{in}\lambda_{kn}\langle\mu_k\rangle_{Q(\mu_k)}\right] = C_i A_i} \tag{32}
$$

Where we have $A_i \overset{def}{=} \frac{1}{\sigma^2}\left[\sum_n\sum_{k\neq i}\lambda_{in}\lambda_{kn}\langle\mu_k\rangle_{Q(\mu_k)} - \sum_n\lambda_{in}x^n\right]$ for later convenience. Now we are equipped to solve equation (22): $\alpha_i = \frac{D}{\langle\mu_i^\top\mu_i\rangle_{Q_{\mu_i}}}$ explicitly. We note that:

$$
\begin{aligned}
\mathbf{E}_{Q(\mu_i)}[\mu_i^\top\mu_i] &= \mathbf{E}_{Q(\mu_i)}[Tr(\mu_i^\top\mu_i)] = \mathbf{E}_{Q(\mu_i)}[Tr(\mu_i\mu_i^\top)]\\
&= Tr(\mathbf{E}_{Q(\mu_i)}[\mu_i\mu_i^\top]) = Tr(\mathbf{Var}_{Q(\mu_i)}[\mu_i]) + Tr(\mathbf{E}_{Q(\mu_i)}[\mu_i^\top]\mathbf{E}_{Q(\mu_i)}[\mu_i])\\
&= Tr(C_i) + Tr(M_i^\top M_i) = DC_i + C_i^2 A_i^\top A_i
\end{aligned}
\tag{33}
$$

Finally, using equations (22) and (31) we get VB hyperparameter update:

$$
\boxed{\alpha_i = \frac{D}{DC_i + C_i^2 A_i^\top A_i}} \tag{34}
$$

3.2 Part b

A few changes to the original algorithm must be made. Firstly we remove the μ calculation from the M-step and replace it with an E-step to find the expected moments of μ as per equations (29) and (30). Secondly, remember that we computed the expected moments of s_i in Question 1 via the fixed point equation (9), however we must now replace the μ_i values with their expected moments:

$$
\begin{aligned}
\frac{\partial\mathcal{F}}{\partial\lambda_{in}} &= \log\frac{\pi_i}{1-\pi_i} - \log\frac{\lambda_{in}}{1-\lambda_{in}} + \frac{1}{\sigma^2}(\boldsymbol{x}^{(n)} - \sum_{j\neq i}\lambda_{jn}\langle\boldsymbol{\mu}_j\rangle)^\top\langle\boldsymbol{\mu}_i\rangle - \frac{1}{2\sigma^2}\langle\boldsymbol{\mu}_i\top\boldsymbol{\mu}_i\rangle\\
&= \log\frac{\pi_i}{1-\pi_i} - \log\frac{\lambda_{in}}{1-\lambda_{in}} + \frac{1}{\sigma^2}(\boldsymbol{x}^{(n)} - \sum_{j\neq i}\lambda_{jn}M_j)^\top M_i - \frac{1}{2\sigma^2}\left(DC_i + C_i^2A_i^\top A_i\right)
\end{aligned}
\tag{35}
$$

Thirdly, in the M-step for the σ and π updates, the π update remains unchanged since it does not contain μ, but the σ update must now also use the expected moments of μ:

$$\sigma^2 = \frac{1}{ND}\left[\sum_{n=1}^{N} \boldsymbol{x}^{(n)\top}\boldsymbol{x}^{(n)} + \sum_{i,j}\langle \boldsymbol{\mu}_i^\top \boldsymbol{\mu}_j\rangle \sum_{n=1}^{N}\langle s_i^{(n)} s_j^{(n)}\rangle_{q(\boldsymbol{s}^{(n)})} - 2\sum_i \langle \mu_i^\top\rangle \sum_{n=1}^{N}\langle s_i^{(n)}\rangle_{q(\boldsymbol{s}^{(n)})}\boldsymbol{x}^{(n)}\right]$$

$$= \frac{1}{ND}\left[\sum_{n=1}^{N} \boldsymbol{x}^{(n)\top}\boldsymbol{x}^{(n)} + \sum_{i\neq j} M_i^\top M_j \sum_{n=1}^{N}\lambda_{in}\lambda_{jn} + \sum_i \left(DC_i + C_i^2 A_i^\top A_i\right)\sum_{n=1}^{N}\lambda_{in} - 2\sum_i M_i^\top \sum_{n=1}^{N}\lambda_{in}\boldsymbol{x}^{(n)}\right] \tag{36}$$

Finally we add an additional hyper-M step for the hyperparameter optimisation of α according to equation (32). We can summarise our updated algorithm as follows:

VB binary latent factor	
Input	Data X, latents and parameters $S, \mu, \pi, \sigma, \alpha$
Initialisation	Randomly initialise parameters
E-step	1. Compute expected moments of s_i^n by iterating fixed point equations (10) until convergence. Use these to form ES and ESS. 2. Compute expected moments of μ_i by iterating fixed point equations (29) and (30) until convergence. Use these to form mu_ES and mu_ESS .
M-Step	1. Compute optimising values of σ and π using equations (7) and (10) given in the question sheet. However, replace μ_i values with their expected moments given in mu_ES and mu_ESS. 2. Perform an hyper-M step to find the optimising value of α_i using equation (32)

We implemented this algorithm and provide sample plots below. The first plot shows the free energy per iteration for different values of K:

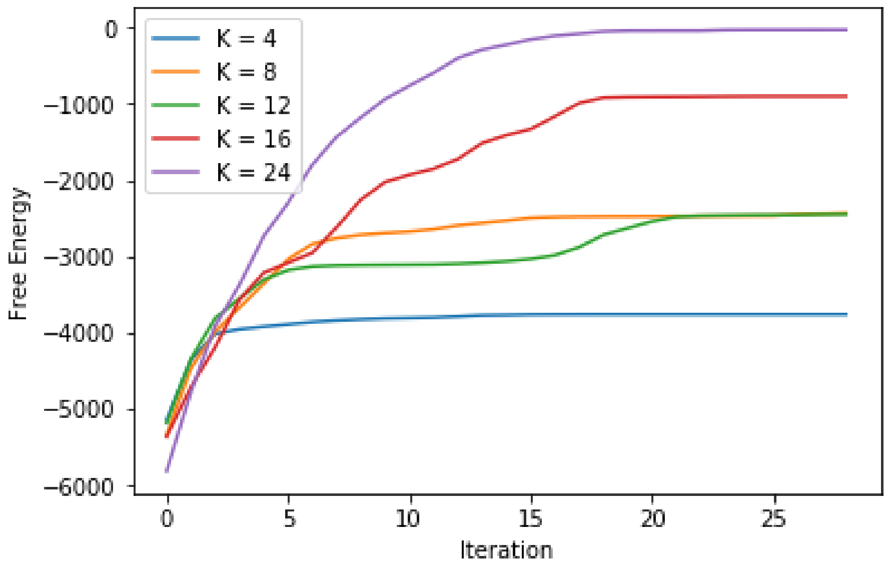


Figure 9: Free energy vs iteration

We also provide a sample plot of the effective number of factors (using some threshold on α_i to indicate shut off factors):

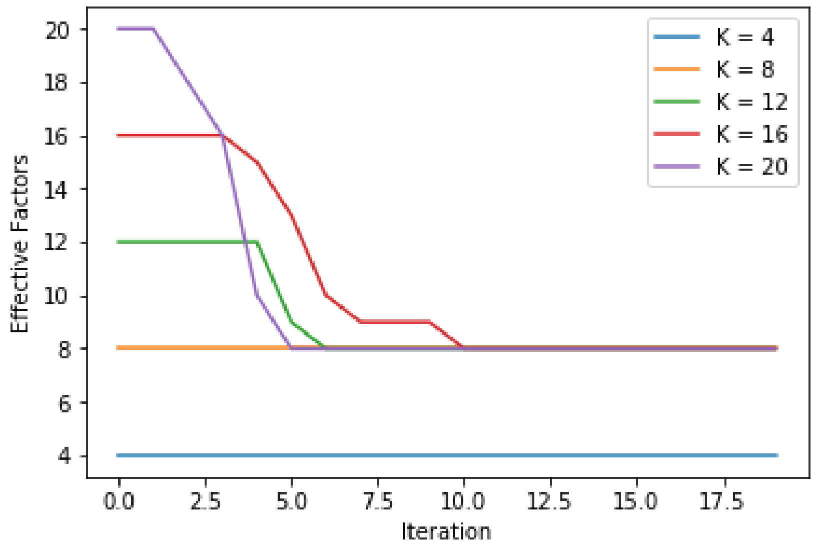


Figure 10: Effective number of factors vs iteration

- Firstly note that for K = 4 we get the lowest free energy. This conforms to our expectations because we know that there are 8 features, and so the algorithm will be unable to find the features. However, note more generally that models with a greater value of K (possible number of components), have a higher free energy. The ARD algorithm works by selecting the relevant K value which is determined by the number of α_k values which diverge. Intuitively therefore, for higher values of K, the algorithm is more likely to find the independent features - producing a high value for the free energy - and then effectively shut off the components which do not correspond to likely features, which in turn dampens their negative contribution to the free energy. Essentially we believe that for larger values of K, there is a larger 'search space' to find the independent features and thus produce a higher free energy value.

- This corresponds well to our plot of the effective number of features. Firstly we note that for K = 4 and K = 8 the effective number of features does not decrease which is what we expect (since there are 8 features). For K > 8 we see that as the number of iterations increases, superfluous components get 'shut off', and the effective number of components eventually decrease to 8. As these components get shut off, they contribute nothing to the free energy and therefore are unable to reduce it.

We also provide sample plots of the α_i values to show how they diverge:

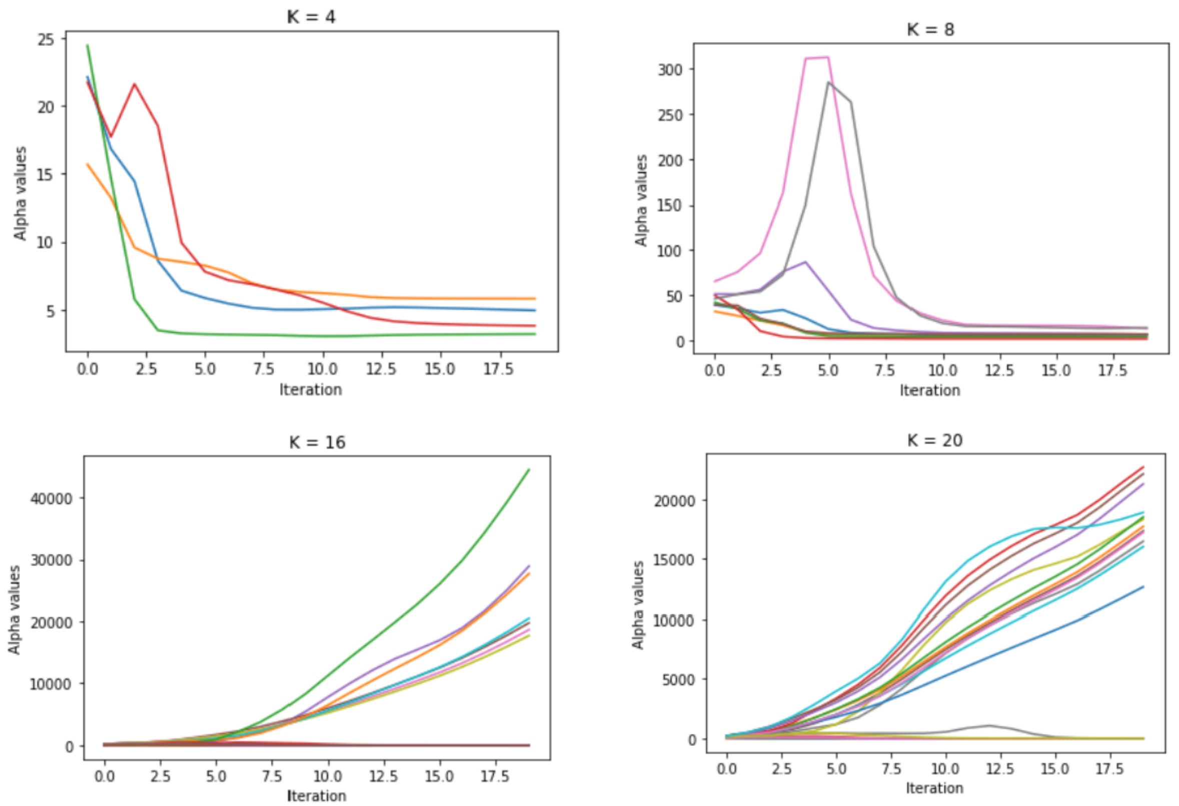


Figure 11: Alpha Divergences

Finally we provide the plot of the learnt factors for $K = 24$ on a good run, where all 8 of the factors are found, and the rest are simply repeats of a single μ_i whose α_i values have diverged:

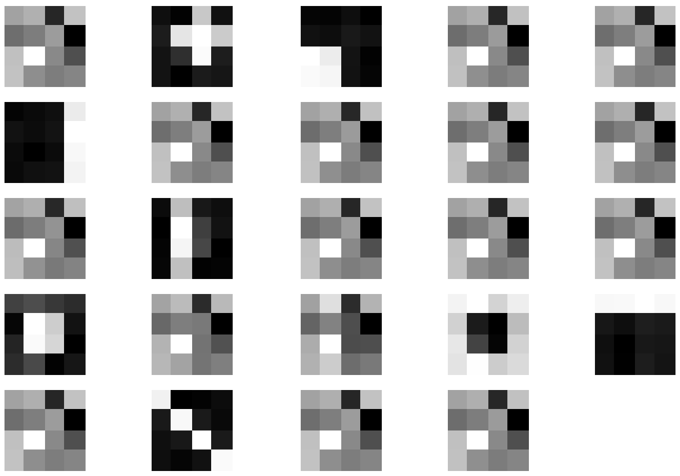


Figure 12: Factors

The code for this algorithm is given below. This first function is used to find the expected moments of μ - mu_ES and mu_ESS:

```
def compute_moments(X, mu_ES, mu_ESS, ESS, sigma, pie, lambda0, alpha,
                                                maxsteps):

    # C_ is K vector
    # M_ is D x K
    N, D = X.shape
    D, K = mu_ES.shape

    mu_ESS_ = mu_ESS.copy()
    mu_ES_ = mu_ES.copy()

    C_ = (1/sigma**2 * np.sum(lambda0, axis = 0) + alpha)**(-1) ## (K,)

    for step in range(maxsteps):

        for i in range(K):

            t_1 = X.T @ lambda0[:, i][:, None]

            temp = np.hstack((lambda0[:, :i], lambda0[:, i+1:])).T @
                                                  lambda0[:, i][:, None]

            t_2 = np.hstack((mu_ES_[:, :i], mu_ES_[:, i+1:])) @ temp

            mu_ES_[:, i] = C_[i] * np.squeeze((1/sigma**2)*(t_1 - t_2))

    mu_ESS_ = (D * C_ + np.diag(mu_ES_.T @ mu_ES_))

    return mu_ES_, mu_ESS_
```

The second function is the updated M-step:

```
def m_step(X, ES, ESS, mu_ES, mu_ESS):

    N, D = X.shape
    if ES.shape[0] != N:
        raise TypeError('ES must have the same number of rows as X')
    K = ES.shape[1]
    if ESS.shape == (N, K, K):
        ESS = np.sum(ESS, axis=0)
    if ESS.shape != (K, K):
        raise TypeError('ESS must be square and have the same number of
                                             columns as ES')

    t_1 = np.trace(np.dot(X.T, X))
    t_2 =  np.trace(np.dot(np.dot(mu_ES.T, mu_ES), ESS))
    t_3 = - np.dot(np.dot(mu_ES.T, mu_ES).diagonal(), (ESS.diagonal()))
    t_4 = np.dot(mu_ESS, ESS.diagonal())
    t_5 = -2*np.trace(np.dot(np.dot(ES.T, X), mu_ES))
    sigma = np.sqrt((t_1 + t_2 + t_3 + t_4 + t_5)/(N*D))

    pie = np.mean(ES, axis=0, keepdims=True)

    return sigma, pie
```

We also have the hyper-M step for α

```
def hyper_m(D, mu_ESS):

    K = np.size(mu_ESS)
    alpha = np.zeros(K)

    for i in range(K):
        alpha[i] = D/(mu_ESS[i])

    return alpha
```

The mean-field function remains largely the same (computed according to equation (33))

```
def mean_field(X, mu_ES, mu_ESS, sigma, pie, lambda0, maxsteps):
    """
    -- lambda0 is N x K
    -- F is lower bound on likelihood
    -- X is N x D data matrix
    -- mu_ES is D x K matrix of means
    -- pie is 1 x K vector of priors
    """
    N, D = X.shape
    K = mu_ES.shape[1]
    Lambda = lambda0.copy()
    f = []
    eps = 1
    cIter = 0
    constant = 1e-15

    while abs(eps) > 1e-15 and cIter <= maxsteps:
        Lambda[np.isclose(Lambda, 0)] = constant
        Lambda[np.isclose(Lambda, 1)] = 1 - constant

        term_1 = np.sum(np.multiply(Lambda, np.log(pie/Lambda)))
        term_2 = np.sum(np.multiply((1-Lambda), np.log((1-pie)/(1-Lambda))
                                             ))
        term_3 = -N*D*np.log(sigma)
        weights = (X - Lambda @ mu_ES.T)
        term_4 = -np.trace((2*sigma**2)**-1*(weights @ weights.T))
        term_5 = -np.sum((2*sigma**2)**-1*(Lambda - Lambda**2) @ np.diag(
                                             mu_ESS))
        term_6 = -N*(D/2)*np.log(2*np.pi)
        f_ = np.sum(term_1 + term_2 + term_3 + term_4 + term_5 + term_6)

        if(cIter > 0):
            eps = f_ - f[cIter - 1]
        f.append(f_)

        cIter += 1

        for i in range(K):
            term_1 = np.repeat(np.log(pie[:,i])-np.log((1-pie[:,i])), N)
            weight = np.delete(Lambda, i, 1) @ np.delete(mu_ES, i, 1).T
            term_2 = (sigma**-2)*(X - weight) @ mu_ES[:,i]
            term_3 = -np.repeat(1/(2*sigma**2)*mu_ESS[i], N)
            Lambda[:,i] = sigmoid(term_1 + term_2 + term_3)

    return Lambda, f[-1], f

def compute_ESS(Lambda):
    """
    Computes E_q[ss'] (N, K, K), using Lambda matrix.
    """

    N, K = Lambda.shape
    ESS = np.zeros(shape=(N,K,K))

    for n in range(N):

        lambda_n = Lambda[n,:]

        for k_1 in range(K):

            for k_2 in range(K):

                ESS[n,k_1,k_2] = lambda_n[k_1]*lambda_n[k_2]

        diag = lambda_n - lambda_n**2

        ESS[n,:,:] += np.diag(diag)

    return ESS
```

The rest of the code remains the same:

```
def init_params(N, D, K):
    Lambda = 0.20 + np.random.rand(N,K)*0.6
    mu_ES = 0.20 + np.random.rand(D,K)*0.6
    mu_ESS = 0.20 + np.ones(K)*0.6
    pie = 0.20 + np.random.rand(1,K)*0.5
    sigma = np.random.uniform()

    return Lambda, mu_ES, mu_ESS, pie, sigma

def learn_bin_factors(X, K, iterations):

    N, D = X.shape
    Lambda, mu_ES, mu_ESS, pie, sigma = init_params(N,D,K)

    alpha_val = 120
    alpha = np.ones(K)*alpha_val
    THRESHOLD = alpha_val*10
    maxsteps = 30
    free_energies = []
    alphas = []
    nfactors = np.zeros(iterations)

    ESS = compute_ESS(Lambda)
    mu_ES, mu_ESS = compute_moments(X, mu_ES, [], ESS, sigma, pie, Lambda,
                                       alpha, 5)

    for cIter in range(iterations):


        Lambda, f, f_l = mean_field(X, mu_ES, mu_ESS, sigma, pie, Lambda,
                                      maxsteps)
        free_energies.append(f)

        ESS = compute_ESS(Lambda)
        mu_ES, mu_ESS = compute_moments(X, mu_ES, mu_ESS, ESS, sigma, pie,
                                          Lambda, alpha, 5)

        sigma, pie = m_step(X, Lambda, ESS, mu_ES, mu_ESS)


        alpha = hyper_m(D, mu_ESS)
        alphas.append(alpha)

        factors =  [a for a in alpha if a < THRESHOLD]
        nfactors[cIter] = len(factors)

    return mu_ES, sigma, pie, Lambda, free_energies, alphas, nfactors
```