CPSC532W HW1

Alan Milligan

August 2021

1. We aim to show that Gamma is conjugate to Poisson. I am assuming it is okay to do so with one observation k.

$$\begin{aligned}\text{Poisson}(k;\lambda) &= e^{-\lambda}\frac{\lambda^k}{k!}\\ \text{Gamma}(\lambda;\alpha,\beta) &= \frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}\end{aligned}$$

So if we multiply these in the way you would with a prior and likelihood, we get

$$\begin{aligned}\text{Poisson}(k;\lambda)\text{Gamma}(\lambda;\alpha,\beta) &= e^{-\lambda}\frac{\lambda^k}{k!}\frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}\\ &\propto e^{-\lambda}\lambda^k\lambda^{\alpha-1}e^{-\beta\lambda}\\ &= \lambda^{\alpha+k-1}e^{-(\beta+1)\lambda}\\ &\propto \frac{(\beta+1)^{\alpha+k}}{\Gamma(\alpha+k)}\lambda^{\alpha+k-1}e^{-(\beta+1)\lambda}\\ &= \text{Gamma}(\lambda;\alpha+k,\beta+1)\end{aligned}$$

So we can see the posterior is Gamma as desired. □

2. First, let $\mathbf{x} = (x_1,\ldots,x_i,\ldots,x_d)$ and $\mathbf{x}_{-i} = (x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_d)$. The transition operator $T(\mathbf{x},\mathbf{x}')$ in Gibbs sampling involves one entry i changing between $\mathbf{x}$ and $\mathbf{x}'$, so we can write it as $T(\mathbf{x},\mathbf{x}') = p(x_i'|\mathbf{x}_{-i})$. Also note that $\mathbf{x}_{-i} = \mathbf{x}'_{-i}$ given that Gibbs changes one variable at a time. Now we show this operator fullfills detailed balance.

$$\begin{aligned}p(\mathbf{x})T(\mathbf{x},\mathbf{x}') &= p(\mathbf{x})p(x_i'|\mathbf{x}_{-i})\\ &= p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i'|\mathbf{x}_{-i})\\ &= p(x_i|\mathbf{x}'_{-i})p(\mathbf{x}'_{-i})p(x_i'|\mathbf{x}'_{-i})\\ &= p(x_i|\mathbf{x}'_{-i})p(x_i',\mathbf{x}'_{-i})\\ &= T(\mathbf{x}',\mathbf{x})p(\mathbf{x}')\end{aligned}$$

□

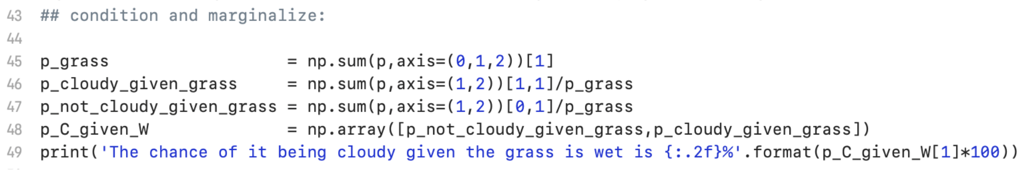
As for the acceptance ratio, we can recall $\mathbf{x}_{-i} = \mathbf{x}'_{-i}$ and see the following.

$$\begin{aligned}A(\mathbf{x},\mathbf{x}') &= \min\{1, \frac{p(\mathbf{x})T(\mathbf{x},\mathbf{x}')}{p(\mathbf{x}')T(\mathbf{x}',\mathbf{x})}\}\\ &= \min\left\{1, \frac{p(\mathbf{x})p(x_i'|\mathbf{x}_{-i})}{p(\mathbf{x}')p(x_i|\mathbf{x}'_{-i})}\right\}\\ &= \min\left\{1, \frac{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i'|\mathbf{x}_{-i})}{p(x_i'|\mathbf{x}'_{-i})p(\mathbf{x}'_{-i})p(x_i|\mathbf{x}'_{-i})}\right\}\\ &= \min\left\{1, \frac{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i'|\mathbf{x}_{-i})}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i'|\mathbf{x}_{-i})}\right\}\end{aligned}$$

$$= \min\{1, 1\}$$
$$= 1$$

□

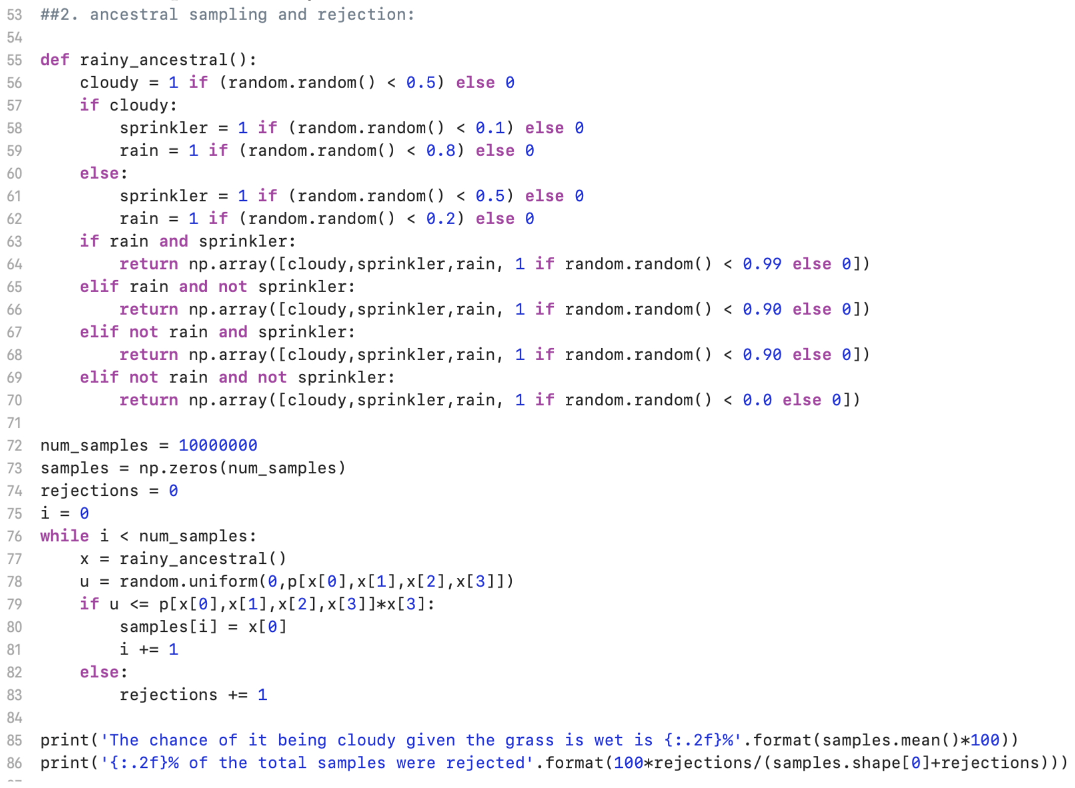
3. (a) Using the following code the chance of it being cloudy given the grass is wet is 57.58%.



```python
## condition and marginalize:

p_grass                  = np.sum(p,axis=(0,1,2))[1]
p_cloudy_given_grass     = np.sum(p,axis=(1,2))[1,1]/p_grass
p_not_cloudy_given_grass = np.sum(p,axis=(1,2))[0,1]/p_grass
p_C_given_W              = np.array([p_not_cloudy_given_grass,p_cloudy_given_grass])
print('The chance of it being cloudy given the grass is wet is {:.2f}%'.format(p_C_given_W[1]*100))
```

(b) Using the following code the chance of it being cloudy given the grass is wet is 57.57%. 35.28% of the total samples were rejected.



```python
##2. ancestral sampling and rejection:

def rainy_ancestral():
    cloudy = 1 if (random.random() < 0.5) else 0
    if cloudy:
        sprinkler = 1 if (random.random() < 0.1) else 0
        rain = 1 if (random.random() < 0.8) else 0
    else:
        sprinkler = 1 if (random.random() < 0.5) else 0
        rain = 1 if (random.random() < 0.2) else 0
    if rain and sprinkler:
        return np.array([cloudy,sprinkler,rain, 1 if random.random() < 0.99 else 0])
    elif rain and not sprinkler:
        return np.array([cloudy,sprinkler,rain, 1 if random.random() < 0.90 else 0])
    elif not rain and sprinkler:
        return np.array([cloudy,sprinkler,rain, 1 if random.random() < 0.90 else 0])
    elif not rain and not sprinkler:
        return np.array([cloudy,sprinkler,rain, 1 if random.random() < 0.0 else 0])

num_samples = 10000000
samples = np.zeros(num_samples)
rejections = 0
i = 0
while i < num_samples:
    x = rainy_ancestral()
    u = random.uniform(0,p[x[0],x[1],x[2],x[3]])
    if u <= p[x[0],x[1],x[2],x[3]]*x[3]:
        samples[i] = x[0]
        i += 1
    else:
        rejections += 1

print('The chance of it being cloudy given the grass is wet is {:.2f}%'.format(samples.mean()*100))
print('{:.2f}% of the total samples were rejected'.format(100*rejections/(samples.shape[0]+rejections)))
```

(c) Using the following code the chance of it being cloudy given the grass is wet is 57.57%.

```
#3: Gibbs
# we can use the joint above to condition on the variables, to create the needed
# conditional distributions:


#we can calculate p(R|C,S,W) and p(S|C,R,W) from the joint, dividing by the right marginal distribution
#indexing is [c,s,r,w]
p_R_given_C_S_W = p/p.sum(axis=2, keepdims=True)
p_S_given_C_R_W = p/p.sum(axis=1, keepdims=True)


# but for C given R,S,W, there is a 0 in the joint (0/0), arising from p(W|S,R)
# but since p(W|S,R) does not depend on C, we can factor it out:
#p(C | R, S) = p(R,S,C)/(int_C (p(R,S,C)))

#first create p(R,S,C):
p_C_S_R = np.zeros((2,2,2)) #c,s,r
for c in range(2):
    for s in range(2):
        for r in range(2):
            p_C_S_R[c,s,r] = p_C(c)*p_S_given_C(s,c)*p_R_given_C(r,c)

#then create the conditional distribution:
p_C_given_S_R = p_C_S_R[:,:,:]/p_C_S_R[:,:,:].sum(axis=(0),keepdims=True)


##gibbs sampling
num_samples = 10000000
samples = np.zeros(num_samples)
state = np.zeros(4,dtype='int')
#c,s,r,w, set w = True

state[3] = 1
i = 0
while i < num_samples:
    state[0] = np.random.choice([0,1], p=p_C_given_S_R[:, state[1],state[2]])
    state[1] = np.random.choice([0,1], p=p_S_given_C_R_W[state[0],:,state[2],state[3]])
    state[2] = np.random.choice([0,1], p=p_R_given_C_S_W[state[0],state[1],:,state[3]])
    samples[i] = state[0]
    i += 1

print('The chance of it being cloudy given the grass is wet is {:.2f}%'.format(samples.mean()*100))
```

4. (a) First, we will need the posterior for the block $\mathbf{w}$. This can be derived using Theorem 4.4.1 from MLAPP as follows. We note that

$$p(\mathbf{w}|\alpha) = p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \alpha I)$$
$$p(\hat{\mathbf{t}}|\mathbf{w}, \mathbf{X}, \sigma^2) = \mathcal{N}(\hat{\mathbf{t}}|\mathbf{X}\mathbf{w}, \sigma^2 I)$$

Now, we apply 4.4.1 to get

$$p(\mathbf{w}|\hat{\mathbf{t}}, \mathbf{X}, \sigma^2, \alpha) = \mathcal{N}(\mathbf{w}|\mu_{\mathbf{w}}, \mathbf{\Sigma}_{\mathbf{w}})$$

where

$$\mathbf{\Sigma}_{\mathbf{w}} = (\alpha^{-1} I + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1}$$
$$\mu_{\mathbf{w}} = \mathbf{\Sigma}_{\mathbf{w}}(\sigma^{-2}\mathbf{X}^T\hat{\mathbf{t}})$$

Then, we need to re-sample $\hat{\mathbf{t}}$ given some new weights $\mathbf{w}'$. Given fixed weights, this is the (vectorized) likelihood

$$p(\hat{\mathbf{t}}|\mathbf{w}, \mathbf{X}, \sigma^2, \alpha) = \mathcal{N}(\mathbf{t}|\mathbf{X}\mathbf{w}, \sigma^2 I)$$

Now, we will want to use Metropolis-Hastings to sample from each of these conditionals. Given that there is no single choice for a proposal distribution q, I am just going to use the classic random-walk MH and write our proposals as follows

$$\mathbf{w}' \sim \mathcal{N}(\mathbf{w}^t, \sigma^2_{\mathbf{w}} I)$$

$$\hat{\mathbf{t}}' \sim \mathcal{N}(\hat{\mathbf{t}}^t, \sigma^2_{\hat{\mathbf{t}}} I)$$

Where the two σ^2 can be some chosen values such as 1. After sampling these proposals, we will do the following update step.

$$\begin{aligned}
u_{\hat{\mathbf{t}}} &\sim \text{Uniform}(0,1) \\
A(\hat{\mathbf{t}}^t, \hat{\mathbf{t}}') &= \min\left\{1, \frac{p(\hat{\mathbf{t}}'|\mathbf{w}^t)q(\hat{\mathbf{t}}'|\hat{\mathbf{t}}^t)}{p(\hat{\mathbf{t}}^t|\mathbf{w}^t)q(\hat{\mathbf{t}}^t|\hat{\mathbf{t}}')}\right\} \\
&= \min\left\{1, \frac{\mathcal{N}(\hat{\mathbf{t}}'|\mathbf{X}\mathbf{w}^{\mathbf{t}}, \sigma^2 I)\mathcal{N}(\hat{\mathbf{t}}'|\hat{\mathbf{t}}^t, \sigma^2_{\hat{\mathbf{t}}} I)}{\mathcal{N}(\hat{\mathbf{t}}^t|\mathbf{X}\mathbf{w}^{\mathbf{t}}, \sigma^2 I)\mathcal{N}(\hat{\mathbf{t}}^t|\hat{\mathbf{t}}', \sigma^2_{\hat{\mathbf{t}}} I)}\right\} \\
\hat{\mathbf{t}}^{t+1} &= \begin{cases} \hat{\mathbf{t}}' & u_{\hat{\mathbf{t}}} < A(\hat{\mathbf{t}}^t, \hat{\mathbf{t}}') \\ \hat{\mathbf{t}}^t & \text{otherwise} \end{cases}
\end{aligned}$$

$$\begin{aligned}
u_{\mathbf{w}} &\sim \text{Uniform}(0,1) \\
A(\mathbf{w}^t, \mathbf{w}') &= \min\left\{1 \frac{p(\mathbf{w}'|\hat{\mathbf{t}}^{t+1})q(\mathbf{w}'|\mathbf{w}^t)}{p(\mathbf{w}^t|\hat{\mathbf{t}}^{t+1})q(\mathbf{w}^t|\mathbf{w}')}\right\} \\
&= \min\left\{1 \frac{\mathcal{N}(\mathbf{w}'|\mathbf{\Sigma}_{\mathbf{w}}(\sigma^{-2}\mathbf{X}^T\mathbf{t}^{t+1}), \mathbf{\Sigma}_{\mathbf{w}})\mathcal{N}(\mathbf{w}'|\mathbf{w}^t, \sigma^2_{\mathbf{w}} I)}{\mathcal{N}(\mathbf{w}^t|\mathbf{\Sigma}_{\mathbf{w}}(\sigma^{-2}\mathbf{X}^T\mathbf{t}^{t+1}), \mathbf{\Sigma}_{\mathbf{w}})\mathcal{N}(\mathbf{w}^t|\mathbf{w}', \sigma^2_{\mathbf{w}} I)}\right\} \\
\mathbf{w}^{t+1} &= \begin{cases} \mathbf{w}' & u_{\mathbf{w}} < A(\mathbf{w}^t, \mathbf{w}') \\ \mathbf{w}^t & \text{otherwise} \end{cases}
\end{aligned}$$

(b) (Yes I am borrowing my own LaTeX from the previous question given we need the same stuff) First, we will need the posterior for the block $\mathbf{w}$. This can be derived using Theorem 4.4.1 from MLAPP as follows. We note that

$$\begin{aligned}
p(\mathbf{w}|\alpha) = p(\mathbf{w}) &= \mathcal{N}(\mathbf{w}|0, \alpha I) \\
p(\hat{\mathbf{t}}|\mathbf{w}, \mathbf{X}, \sigma^2) &= \mathcal{N}(\hat{\mathbf{t}}|\mathbf{X}\mathbf{w}, \sigma^2 I)
\end{aligned}$$

Now, we apply 4.4.1 to get

$$p(\mathbf{w}|\hat{\mathbf{t}}, \mathbf{X}, \sigma^2, \alpha) = \mathcal{N}(\mathbf{w}|\mu_{\mathbf{w}}, \mathbf{\Sigma}_{\mathbf{w}})$$

where

$$\begin{aligned}
\mathbf{\Sigma}_{\mathbf{w}} &= (\alpha^{-1} I + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1} \\
\mu_{\mathbf{w}} &= \mathbf{\Sigma}_{\mathbf{w}}(\sigma^{-2}\mathbf{X}^T\hat{\mathbf{t}})
\end{aligned}$$

Then, we need to re-sample $\hat{\mathbf{t}}$ given some new weights $\mathbf{w}'$. Given fixed weights, this is the (vectorized) likelihood

$$p(\hat{\mathbf{t}}|\mathbf{w}, \mathbf{X}, \sigma^2, \alpha) = \mathcal{N}(\hat{\mathbf{t}}|\mathbf{X}\mathbf{w}, \sigma^2 I)$$

So, now that we have the two distributions we will want to sample from with our blocked Gibbs, we can properly write the update samples as follows.

$$\begin{aligned}
\hat{\mathbf{t}}^{t+1} &\sim \mathcal{N}(\mathbf{X}\mathbf{w}^t, \sigma^2 I) \\
\mathbf{w}^{t+1} &\sim \mathcal{N}(\mathbf{\Sigma}_{\mathbf{w}}(\sigma^{-2}\mathbf{X}^T\hat{\mathbf{t}}^{t+1}), (\alpha^{-1} I + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1})
\end{aligned}$$

(c) We want the posterior predictive for new data $\hat{\mathbf{t}}$ from $\hat{\mathbf{X}}$ which is the likelihood of that data times the posterior over $\mathbf{w}$ with $\mathbf{w}$ integrated out. I am going to use Theorem 4.4.1 from

MLAPP repeatedly here, and use a multivariate formulation of the likelihood and prior. As distributions, this is

$$\int_{\mathbf{w}} p(\hat{\mathbf{t}}|\hat{\mathbf{X}}, \mathbf{t}, \mathbf{X}, \sigma^2, \alpha) p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \alpha) d\mathbf{w}$$

First, we will need the posterior $p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \alpha)$. We note that

$$\begin{aligned} p(\mathbf{w}|\alpha) = p(\mathbf{w}) &= \mathcal{N}(\mathbf{w}|0, \alpha I) \\ p(\mathbf{t}|\mathbf{w}, \mathbf{X}, \sigma^2) &= \mathcal{N}(\mathbf{t}|\mathbf{X}\mathbf{w}, \sigma^2 I) \end{aligned}$$

Now, we apply 4.4.1 to get

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \alpha) = \mathcal{N}(\mathbf{w}|\mu_{\mathbf{w}}, \mathbf{\Sigma}_{\mathbf{w}})$$

where

$$\begin{aligned} \mathbf{\Sigma}_{\mathbf{w}} &= (\alpha^{-1} I + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1} \\ \mu_{\mathbf{w}} &= \mathbf{\Sigma}_{\mathbf{w}}(\sigma^{-2}\mathbf{X}^T\mathbf{t}) \end{aligned}$$

Now, given the observed data we can write

$$\begin{aligned} p(\mathbf{w}) = p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \alpha) &= \mathcal{N}(\mathbf{w}|\mu_{\mathbf{w}}, \mathbf{\Sigma}_{\mathbf{w}}) \\ p(\hat{\mathbf{t}}|\mathbf{w}) = p(\hat{\mathbf{t}}|\mathbf{w}, \hat{\mathbf{X}}, \mathbf{t}, \mathbf{X}, \sigma^2, \alpha) &= \mathcal{N}(\hat{\mathbf{t}}|\hat{\mathbf{X}}\mathbf{w}, \sigma^2 I) \end{aligned}$$

Then, again by Theorem 4.4.1 we find the posterior predictive as follows.

$$\begin{aligned} \int_{\mathbf{w}} p(\hat{\mathbf{t}}, |\mathbf{w}, \hat{\mathbf{X}}, \mathbf{t}, \mathbf{X}, \sigma^2, \alpha) p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \alpha) d\mathbf{w} &= \int_{\mathbf{w}} p(\hat{\mathbf{t}}, \mathbf{w}|\hat{\mathbf{X}}, \mathbf{t}, \mathbf{X}, \sigma^2, \alpha) d\mathbf{w} \\ &= p(\hat{\mathbf{t}}|\hat{\mathbf{X}}, \mathbf{t}, \mathbf{X}, \sigma^2, \alpha) \\ &= \mathcal{N}(\hat{\mathbf{t}}|\hat{\mathbf{X}}\mu_{\mathbf{w}}, \sigma^2 I + \hat{\mathbf{X}}\mathbf{\Sigma}_{\mathbf{w}}\hat{\mathbf{X}}^{\mathbf{T}}) \end{aligned}$$

□

5. As suggested in class, I will show how we analytically integrate out the β parameters (and the same is done with the θ parameters) in order to make a more efficiently sampler. We note that the full joint is as follows

$$p(\mathbf{Z}, \mathbf{X}, \boldsymbol{\theta}, \boldsymbol{\beta}, \alpha, \gamma) = \prod_{i=1}^{N}\prod_{d=1}^{D}\prod_{k=1}^{K} p(\mathbf{Z}_{d,i}|\boldsymbol{\theta}_d) p(\mathbf{W}_{d,i}|\boldsymbol{\beta}_{Z_{i,j}}) p(\boldsymbol{\theta}_d|\alpha) p(\boldsymbol{\beta}_k|\gamma)$$

We would like to exploit the fact that $p(\boldsymbol{\beta}_k|\gamma)$ is a Dirichlet distribution and $p(\mathbf{W}_{d,i}|\boldsymbol{\beta}_{Z_{i,j}})$ is a Discrete (or n=1 multinomial) and this is indeed a conjugate pair. We also note that that given we know their hyper-parameter, each β_k is conditionally independent so the integral of their product is the product of their integral. The following integral will be half of the collapsed joint liklihood, which would be multiplied by the result of integrating out $\boldsymbol{\theta}$ by a very similar computation.

$$\begin{aligned} \int_{\boldsymbol{\beta}} \prod_{i=1}^{N}\prod_{d=1}^{D}\prod_{k=1}^{K} p(\mathbf{W}_{d,i}|\boldsymbol{\beta}_{Z_{i,j}}) p(\boldsymbol{\beta}_k|\gamma) d\boldsymbol{\beta} &= \prod_{k=1}^{K} \int_{\boldsymbol{\beta}_k} \prod_{i=1}^{N}\prod_{d=1}^{D} p(\mathbf{W}_{d,i}|\boldsymbol{\beta}_{Z_{d,i}}) \frac{\prod_{w=1}^{W} \beta_{k,w}^{\gamma_w - 1}}{B(\gamma)} d\boldsymbol{\beta}_k \\ &= \prod_{k=1}^{K} \int_{\boldsymbol{\beta}_k} \prod_{i=1}^{N}\prod_{d=1}^{D}\prod_{w=1}^{W} \beta_{Z_{d,i},w} \frac{\prod_{w=1}^{W} \beta_{k,w}^{\gamma_w - 1}}{B(\gamma)} d\boldsymbol{\beta}_k \\ &= \prod_{k=1}^{K} \int_{\boldsymbol{\beta}_k} \prod_{w=1}^{W} \beta_{k,w}^{|\{Z_{i,d}=k,\ W_{i,d}=w\ \ (i,d)\in N\times D\}|} \frac{\prod_{w=1}^{W} \beta_{k,w}^{\gamma_w - 1}}{B(\gamma)} d\boldsymbol{\beta}_k \end{aligned}$$

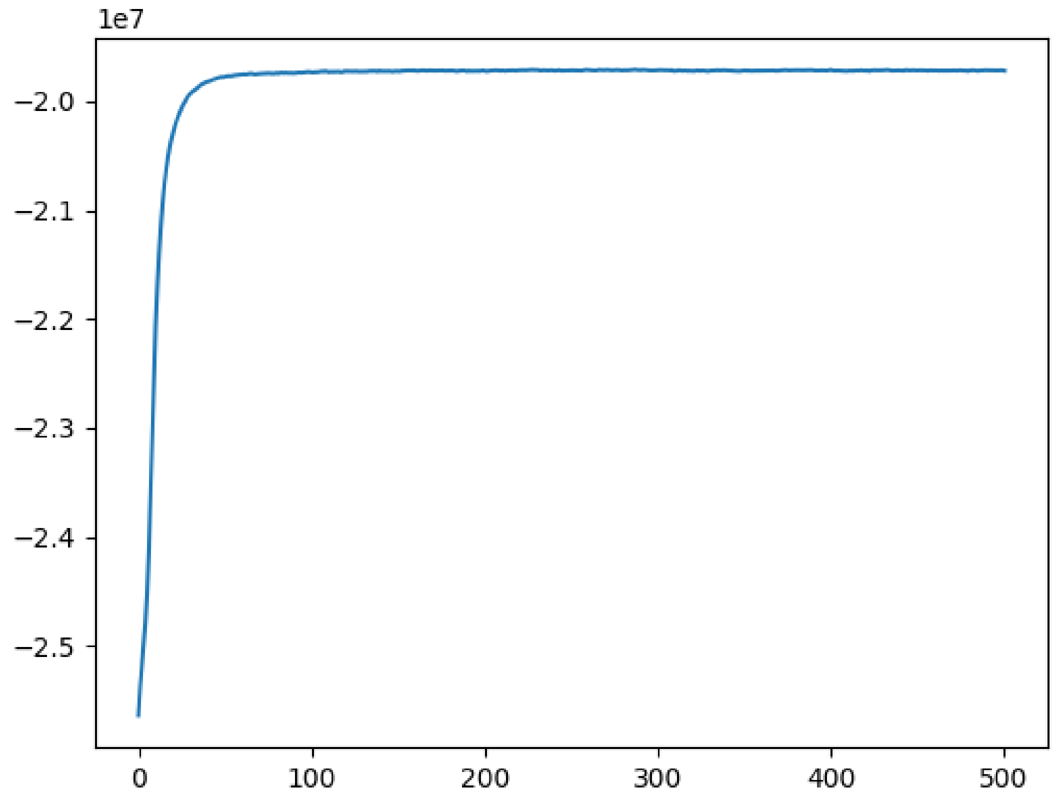
$$= \prod_{k=1}^{K} \frac{1}{B(\gamma)} \int_{\boldsymbol{\beta}_k} \prod_{w=1}^{W} \beta_{k,w}^{|\{Z_{i,d}=k\ W_{i,d}=w\}|} \prod_{w=1}^{W} \beta_{k,w}^{\gamma_w - 1} d\boldsymbol{\beta}_k$$

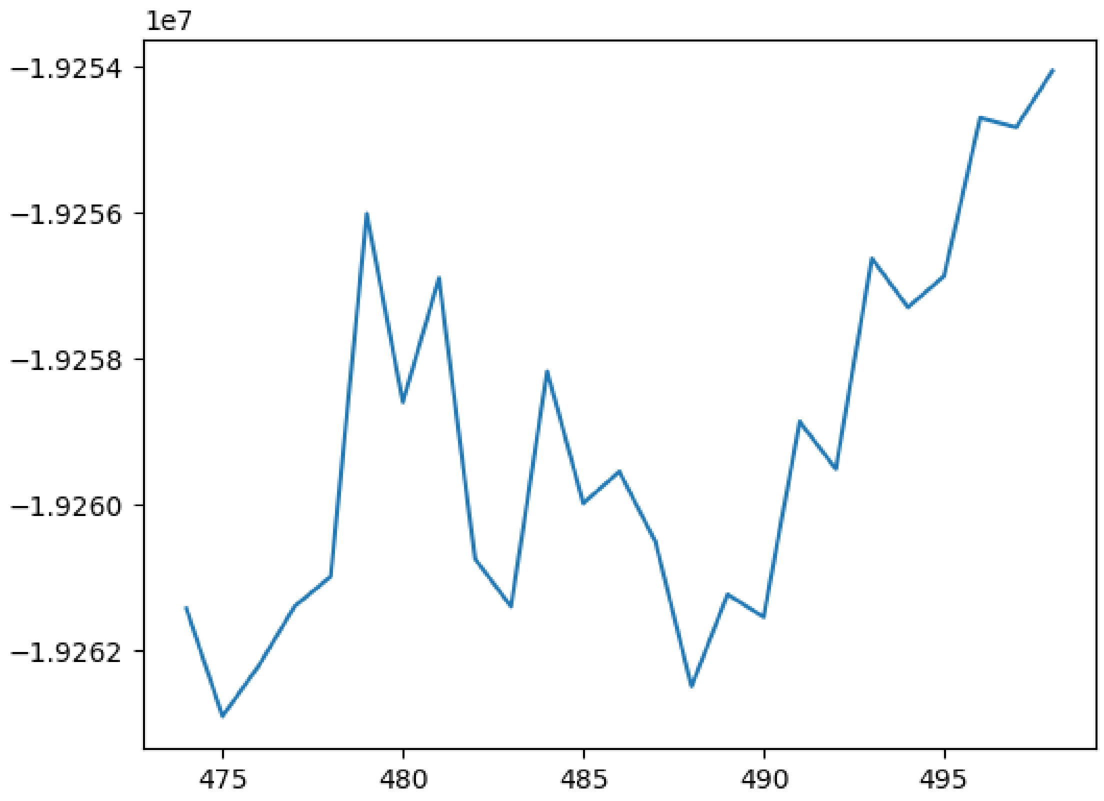
$$= \prod_{k=1}^{K} \frac{1}{B(\gamma)} \int_{\boldsymbol{\beta}_k} \prod_{w=1}^{W} \beta_{k,w}^{|\{Z_{i,d}=k\ W_{i,d}=w\}|+\gamma_w - 1} d\boldsymbol{\beta}_k$$

$$= \prod_{k=1}^{K} \frac{B(\vec{\mathbf{1}_W}(|\{Z_{i,d}=k\ W_{i,d}=w\}| + \gamma_w))}{B(\gamma)}$$

Where B is the beta function and $|\{Z_{i,d} = k\ W_{i,d} = w\}|$ is an "indicator set" that will end up being a count of all the times topic of the token i, d was k and the vocabulary token i, d was w. If we did the same thing with $\boldsymbol{\theta}$ we could get a similar mess of beta functions and take the product as our collapsed joint distribution. From there, Murphy's book uses a few gamma function tricks to turn this into the distribution over a single topic conditioned all the others, which was implemented in the program below.

I ran the following code with hyper-parameters $\alpha = 0.1$ and $\gamma = 0.001$ because Wikipedia told me those were reasonable values, and used 150 iterations of Gibbs sampling. Below is a plot of the total JLL as well as a plot of it over the final 25 samples, showing it is sufficiently spikey as a sampling method should be. Following that are the most probably words of each topic and the most similar documents to document zero.





Topic Words:

Topic 0: time cell cells frequency firing spike response stimulus signal rate temporal channel auditory information noise input stimuli figure channels spikes

Topic 1: training error learning set generalization performance examples data test prediction number results average neural networks large size figure sets weight

Topic 2: neurons neuron network synaptic model input neural activity connections patterns synapses pattern cortex inhibitory dynamics excitatory cortical function networks system

Topic 3: model system motion direction motor control eye velocity target position feedback head movement figure sensory subjects movements response gain human

Topic 4: model data distribution gaussian models probability parameters likelihood density mixture bayesian log variables prior estimate estimation posterior em variance distributions

Topic 5: neural weight threshold weights number input function size binary bit time functions networks network output bits inputs vector single transfer

Topic 6: function functions case networks set theorem bound class theory probability number result defined loss dimension results bounds paper approximation finite

Topic 7: noise systems figure system real point neural range work coding noisy shows robust complex process solutions processing ica approach results

Topic 8: speech neural network time training networks system output input signal hmm context mlp performance models sequence speaker trained hidden series

Topic 9: network units input networks output hidden layer unit weights learning neural training net architecture propagation back patterns information trained pattern

Topic 10: action reinforcement policy algorithm optimal robot actions task face function environment search based problem goal algorithms state reward performance path

Topic 11: recognition word system words character module segmentation modules set model characters digit level hand figure information segment digits systems segments

Topic 12: information analysis order component components independent source signals natural signal line low theory sources curve high processing fig separation higher

Topic 13: learning state time control model states system dynamics dynamic figure controller stochastic trajectory space learn adaptive transition rule initial systems

Topic 14: memory representation structure representations recurrent sequence time distributed state sequences context order science language level represented represent rules parallel length

Topic 15: data algorithm set tree learning problem node model algorithms based clustering nodes cluster models number experts expert trees local table

Topic 16: classification feature data set features class classifier pattern distance performance vectors vector space classifiers test patterns number training nearest classes

Topic 17: function algorithm linear vector matrix problem gradient space local solution point method points algorithms vectors optimization nonlinear optimal functions dimensional

Topic 18: analog circuit chip figure network output current neural vlsi voltage energy input node networks time implementation design circuits hardware neuron

Topic 19: image visual images object figure field map orientation receptive spatial objects local fields vision feature maps features center space motion

Similar Documents: 10 most similar documents to document 0: Connectivity Versus Entropy are:

1st most similar is document 11: A Computer Simulation of Olfactory Cortex with Functional Implications for Storage and Retrieval of Olfactory Information

2nd most similar is document 1: Stochastic Learning Networks and their Electronic Implementation

3rd most similar is document 2: Learning on a General Network

4th most similar is document 4: On Properties of Networks of Neuron-Like Elements

5th most similar is document 14: Speech Recognition Experiments with Perceptrons

6th most similar is document 7: Analysis and Comparison of Different Learning Algorithms for Pattern Association Problems

7th most similar is document 13: On the Power of Neural Networks for Solving Hard Problems

8th most similar is document 12: Neural Network Implementation Approaches for the Connection Machine

9th most similar is document 17: A Neural Network Classifier Based on Coding Theory

10th most similar is document 19: Phase Transitions in Neural Networks

```python
def sample_topic_assignment(topic_assignment,
                            topic_counts,
                            doc_counts,
                            topic_N,
                            doc_N,
                            alpha,
                            gamma,
                            words,
                            document_assignment):
    """
    Sample the topic assignment for each word in the corpus, one at a time.

    Args:
        topic_assignment: size n array of topic assignments
        topic_counts: n_topics x alphabet_size array of counts per topic of unique words
        doc_counts: n_docs x n_topics array of counts per document of unique topics

        topic_N: array of size n_topics count of total words assigned to each topic
        doc_N: array of size n_docs count of total words in each document, minus 1

        alpha: prior dirichlet parameter on document specific distributions over topics
        gamma: prior dirichlet parameter on topic specific distribuitons over words.

        words: size n array of words
        document_assignment: size n array of assignments of words to documents
    Returns:
        topic_assignment: updated topic_assignment array
        topic_counts: updated topic counts array
        doc_counts: updated doc_counts array
        topic_N: updated count of words assigned to each topic
    """
    for word, doc, i in tqdm(zip(words,document_assignment, range(words.size)), total=words.size):

        topic = topic_assignment[i]
        topic_counts[topic, word] -= 1
        doc_counts[doc, topic] -= 1
        topic_N[topic] -= 1

        topic_dist = (doc_counts[doc,:] + alpha)*(topic_counts[:,word] + gamma[word])/(topic_N + np.sum(gamma))
        topic_dist /= np.sum(topic_dist)
        topic = np.random.choice(topic_N.size, p=topic_dist)

        topic_assignment[i] = topic
        topic_counts[topic, word] += 1
        doc_counts[doc, topic] += 1
        topic_N[topic] += 1

    return (topic_assignment, topic_counts, doc_counts, topic_N)
```

```python
def joint_log_lik(doc_counts, topic_counts, alpha, gamma):
    """
    Calculate the joint log likelihood of the model

    Args:
        doc_counts: n_docs x n_topics array of counts per document of unique topics
        topic_counts: n_topics x alphabet_size array of counts per topic of unique words
        alpha: prior dirichlet parameter on document specific distributions over topics
        gamma: prior dirichlet parameter on topic specific distribuitons over words.
    Returns:
        ll: the joint log likelihood of the model
    """
    n_docs, n_topics = np.shape(doc_counts)
    _, alphabet_size = np.shape(topic_counts)

    add = (n_docs*sp.gammaln(np.sum(alpha)) +
           np.sum(sp.gammaln(doc_counts + alpha.reshape((1,n_topics)))) +
           n_topics*sp.gammaln(np.sum(gamma)) +
           np.sum(sp.gammaln(topic_counts + gamma.reshape((1,alphabet_size)))))
    sub = (n_docs*np.sum(sp.gammaln(alpha)) +
          np.sum(sp.gammaln(np.sum(doc_counts + alpha.reshape((1,n_topics)), axis=1))) +
          n_topics*np.sum(sp.gammaln(gamma)) +
          np.sum(sp.gammaln(np.sum(topic_counts + gamma.reshape((1,alphabet_size)), axis=1))))
    return add - sub
```

```
...
### find the 10 most probable words of the 20 topics:

fstr = ''
for i in range(n_topics):
    words = WO[np.argsort(-1*best_topic_counts[i,:])[0:20]]
    fstr += f'Topic {i}: {" ".join([w[0] for w in words])}\n'

with open('most_probable_words_per_topic','w') as f:
    f.write(fstr)

#most similar documents to document 0 by cosine similarity over topic distribution:
#normalize topics per document and dot product:
n_sim = 10

fstr = f'{n_sim} most similar documents to document 0: {titles[0][0]} are:\n'
ordinal = lambda n: "%d%s" % (n,"tsnrhtdd"[(n//10%10!=1)*(n%10<4)*n%10::4])

sim_docs = np.argsort(-1*np.dot(best_doc_counts[:,0].transpose()/np.linalg.norm(best_doc_counts[:,0]), doc_counts/np.linalg.norm(best_doc_counts, axis=0)))[0:n_sim]
for i in range(n_sim):
    fstr += f'{ordinal(i+1)} most similar is document {sim_docs[i]}: {titles[sim_docs[i]][0]}\n'

with open('most_similar_titles_to_0','w') as f:
    f.write(fstr)
```