1. (a)
The code to compute the log of the prior is

log_prior(z) = factorized_gaussian_log_density(0,0,z)

1. (b)
The code to produce a 784-dimensional mean vector of a product of Bernoulli distributions is

```
Dz, Dh = 2, 500
Ddata = 28^2
decoder = Chain(Dense(Dz,Dh,tanh), Dense(Dh, Ddata))
```

1. (c)
The code to compute the log-likelihood log $p(x|z)$ is

```
function log_likelihood(x,z)
  logitmean = decoder(z)
  return  sum(bernoulli_log_density(logitmean,x) , dims = 1)
end
```

1. (d)
The code to compute the joint log density $logp(z,x)$ is

joint_log_density(x,z) = log_prior(z) .+ log_likelihood(x,z)

2. (a)
The code to output the mean and log-standard deviation of a factorized Gaussian is

encoder = Chain(Dense(Ddata,Dh,tanh),Dense(Dh,2*Dz),unpack_gaussian_params)

2. (b)
The code to evaluate the likelihood of z under the variational distribution is

log_q(q_μ, q_logσ, z) = factorized_gaussian_log_density(q_μ, q_logσ,z)

2. (c)
The code to compute an unbiased estimate of the mean variational evidence lower bound on a batch of images is

```
function elbo(x)
  q_μ =  encoder(x)[1]
  q_logσ = encoder(x)[2]
  z = exp.(q_logσ) .* randn(size(q_μ)) .+ q_μ
  joint_ll = joint_log_density(x,z)
  log_q_z = log_q (q_μ, q_logσ, z)
  elbo_estimate = sum(joint_ll.-log_q_z) / size(q_μ)[2]
  return elbo_estimate
end
```

2. (d)

The code to compute the negative elbo estimate over a batch of data is

```
function loss(x)
  return -elbo(x)
end
```

2. (e)

The code to initialize and optimize the encoder and decoder parameters jointy on the training set is

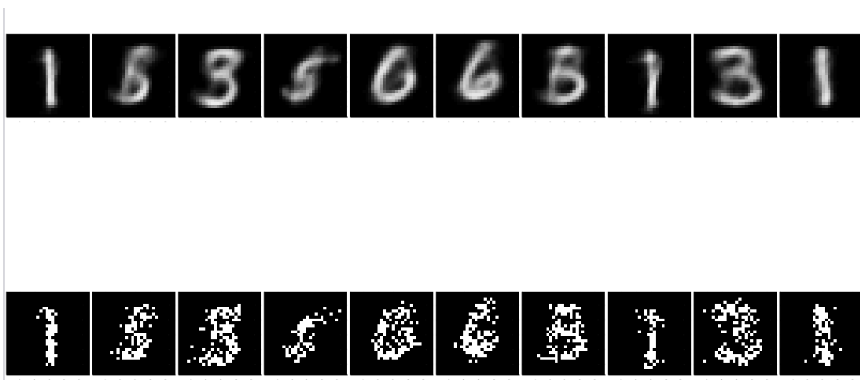
```
function train_model_params!(loss, encoder, decoder, train_x, test_x; nepochs=10)
  ps = Flux.params(encoder,decoder)
  opt = ADAM()
  for i in 1:nepochs
    for d in batch_x(train_x)
      gs = Flux.gradient(ps) do
        batch_loss = loss(d)
        return batch_loss
      end
      Flux.Optimise.update!(opt,ps,gs)
    end
    if i%1 == 0 # change 1 to higher number to compute and print less frequently
      @info "Test loss at epoch $i: $(loss(batch_x(test_x)[1]))"
    end
  end
  @info "Parameters of encoder and decoder trained!"
end

## Train the model
train_model_params!(loss,encoder,decoder,train_x,test_x, nepochs=100)
```

We train the data for 100 epochs and find the final ELBO on the test set is 150.64.

3. (a)

The following code generates the 2x10 plots . The first row plots are the Bernoulli means of $p(x|z)$ and the second row plots are the binary images, sampled from the distribution above it.

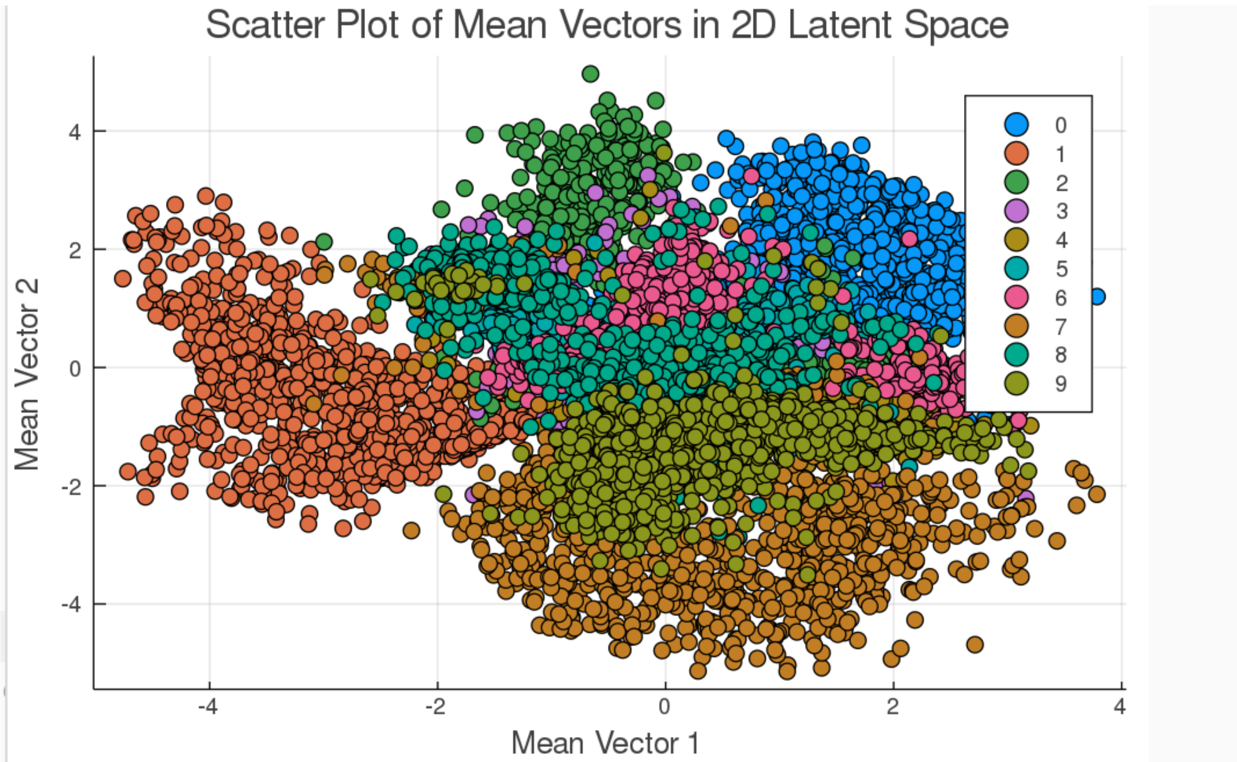
```
z = randn(2,10)
logitp = decoder(z)
p = (1 .+ exp.(-logitp)) .^ (-1)
x = sample_bernoulli(p)
plot_list_mean = []
for i in 1:10
  push!(plot_list_mean,plot(mnist_img(θ[:,i])))
end
plot_list_images = []
for j in 1:10
  push!(plot_list_images, plot(mnist_img(x[:,j])))
end
plot_list = [plot_list_mean; plot_list_images]
display(plot(plot_list..., layout = grid(2,10), size = (10000,6000)))
```



Generative Sample Plots (1st Row: mean, 2nd Row: binarized)

3. (b)

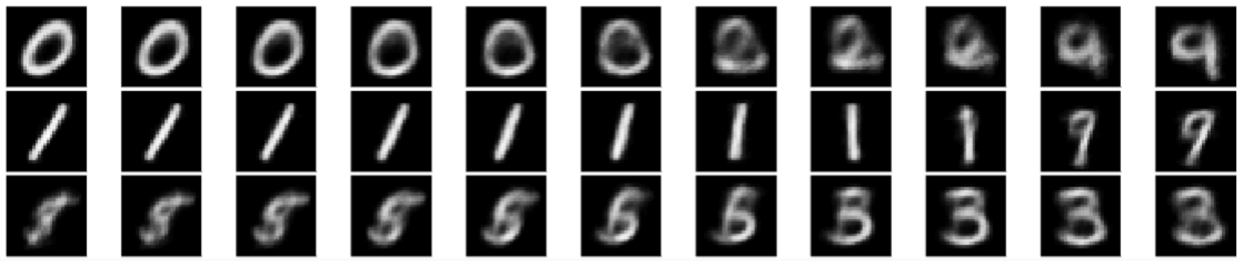
q_μ, q_logσ = encoder(train_x)[1], encoder(train_x)[2]
scatter(q_μ[1,:], q_μ[2,:], group = train_label)



Mean Vectors in 2D Latent Space

3. (c)
In the figure below, each row represents the plots of the corresponding pairs.

```
function interpolation(za,zb,α)
  return α * za .+ (1-α) * zb
end
# Manually pick the data with different classes and  check by train_label
train_label[21:30] # 4, 0, 9, 1, 1, 2, 4, 3
x1 = train_x[:,21] # Number 4
x2 = train_x[:,22] # Number 0
x3 = train_x[:,23] # Number 9
x4 = train_x[:,24] # Number 1
x5 = train_x[:,26] # Number 2
x6 = train_x[:,30] # Number 3

logitmean1 = encoder(x1)[1]
logitmean2 = encoder(x2)[1]
logitmean3 = encoder(x3)[1]
logitmean4 = encoder(x4)[1]
logitmean5 = encoder(x5)[1]
logitmean6 = encoder(x6)[1]

bernmean1 = []
for i in 0:10
  logitmeans = decoder(interpolation(logitmean1, logitmean2, i/10))
  means = exp.(logitmeans) ./ (1 .+ exp.(logitmeans))
  push!(bernmean1, plot(mnist_img(means[:,1])))
end

bernmean2 = []
for i in 0:10
  logitmeans = decoder(interpolation(logitmean3, logitmean4, i/10))
  means = exp.(logitmeans) ./ (1 .+ exp.(logitmeans))
  push!(bernmean2, plot(mnist_img(means[:,1])))
end

bernmean3 = []
for i in 0:10
  logitmeans = decoder(interpolation(logitmean5, logitmean6,  i / 10))
  means = exp.(logitmeans) ./ (1 .+ exp.(logitmeans))
  push!(bernmean3, plot(mnist_img(means[:,1])))
end
bernmean = []
bernmean = [bernmean1; bernmean2; bernmean3]
display(plot(bernmean..., layout=grid(3,11), size =(10000, 2000), axis=nothing))
```



Bernoulli Means from the Latent Space Interpolation

4. (a) (a)

```
function top_image(x)
    " return only the top half of a 28*28 array"
 end
```

4. (a) (b)

```
function log_likelihood_top(x,z)
  logitmean = decoder(z)
  logitmean_top # gives the logit mean of the top half images
  sum(bernoulli_log_density(logitmean_top, x), dims=1)
end
```

4. (a) (c)

```
log_joint_top(x,z) =  log_prior(z) .+ log_likelihood_top(x,z)
```

4. (b)

```
encoder_2 = Chain(Dense(392,Dh, tanh), Dense(Dh,4), unpack_gaussian_params())

function elbo_2(x)
  q_μ =  encoder_2(x)[1]
  q_logσ = encoder_2(x)[2]
  z = exp.(q_logσ) .* randn(size(q_μ)) .+ q_μ
  joint_ll = log_joint_top(x,z)
  log_q_z = factorized_gaussian_log_density(q_μ, q_logσ, z)
  elbo_estimate_2 = sum(joint_ll.-log_q_z) / size(q_μ)[2]
  return elbo_estimate_2
end
```

```
function loss_2(x)
  return –elbo_2(x)
end


function train_model_params_2!(loss, encoder, decoder, train_x, test_x; nepochs=10)
  ps = Flux.params(encoder,decoder)
  opt = ADAM()
  for i in 1:nepochs
    for d in batch_x(train_x)
      gs = Flux.gradient(ps) do
        batch_loss = loss(d)
        return batch_loss
      end
      Flux.Optimise.update!(opt,ps,gs)
    end
    if i%1 == 0 # change 1 to higher number to compute and print less frequently
      @info "Test loss at epoch $i: $(loss(batch_x(test_x)[1]))"
    end
  end
  @info "Parameters of encoder and decoder trained!"
end end
@info "Parameters of encoder and decoder trained!"
      end

train_model_params_2!(loss_2,encoder_2,decoder,train_x, test_x, nepochs
=100)
```

4. (c)
(a) True
(b) False
(c) False
(d) False
(e) True