Assignment 3

Student Name Sheng Gao and #1002093584

April 16, 2020

```
using Flux
using MLDatasets
using Statistics
using Logging
using Test
using Random
using StatsFuns: log1pexp
Random.seed!(412414);

#### Probability Stuff
# Make sure you test these against a standard implementation!

# log-pdf of x under Factorized or Diagonal Gaussian N(x|μ,σI)
function factorized_gaussian_log_density(mu, logsig,xs)
  """
  mu and logsig either same size as x in batch or same as whole batch
  returns a 1 x batchsize array of likelihoods
  """
  σ = exp.(logsig)
  return sum((-1/2)*log.(2π*σ.^2) .+ -1/2 * ((xs .- mu).^2)./(σ.^2),dims=1)
end

# log-pdf of x under Bernoulli
function bernoulli_log_density(logit_means,x)
  """Numerically stable log_likelihood under bernoulli by accepting μ/(1-μ)"""
  b = x .* 2 .- 1 # {0,1} -> {-1,1}
  return - log1pexp.(-b .* logit_means)
end
## This is really bernoulli
@testset "test stable bernoulli" begin
  using Distributions
  x = rand(10,100) .> 0.5
  μ = rand(10)
  logit_μ = log.(μ./(1 .- μ))
  @test logpdf.(Bernoulli.(μ),x) ≈ bernoulli_log_density(logit_μ,x)
  # over i.i.d. batch
  @test sum(logpdf.(Bernoulli.(μ),x),dims=1) ≈
sum(bernoulli_log_density(logit_μ,x),dims=1)
end

Test Summary:          | Pass  Total
test stable bernoulli |    2      2

# sample from Diagonal Gaussian x~N(μ,σI) (hint: use reparameterization trick here)
sample_diag_gaussian(μ,logσ) = (ϵ = randn(size(μ)); μ .+ exp.(logσ).*ϵ)
```

```
# sample from Bernoulli (this can just be supplied by library)
sample_bernoulli(θ) = rand.(Bernoulli.(θ))

# Load MNIST data, binarise it, split into train and test sets (10000 each) and
partition train into mini-batches of M=100.
# You may use the utilities from A2, or dataloaders provided by a framework
function load_binarized_mnist(train_size=10000, test_size=10000)
  train_x, train_label = MNIST.traindata(1:train_size);
  test_x, test_label = MNIST.testdata(1:test_size);
  @info "Loaded MNIST digits with dimensionality $(size(train_x))"
  train_x = reshape(train_x, 28*28,:)
  test_x = reshape(test_x, 28*28,:)
  @info "Reshaped MNIST digits to vectors, dimensionality $(size(train_x))"
  train_x = train_x .> 0.5; #binarize
  test_x = test_x .> 0.5; #binarize
  @info "Binarized the pixels"
  return (train_x, train_label), (test_x, test_label)
end

function batch_data((x,label)::Tuple, batch_size=100)
  """
  Shuffle both data and image and put into batches
  """
  N = size(x)[end] # number of examples in set
  rand_idx = shuffle(1:N) # randomly shuffle batch elements
  batch_idx = Iterators.partition(rand_idx,batch_size) # split into batches
  batch_x = [x[:,i] for i in batch_idx]
  batch_label = [label[i] for i in batch_idx]
  return zip(batch_x, batch_label)
end
# if you only want to batch xs
batch_x(x::AbstractArray, batch_size=100) =
first.(batch_data((x,zeros(size(x)[end])),batch_size))


### Implementing the model

## Load the Data
train_data, test_data = load_binarized_mnist();
train_x, train_label = train_data;
test_x, test_label = test_data;

## Test the dimensions of loaded data
@testset "correct dimensions" begin
  @test size(train_x) == (784,10000)
  @test size(train_label) == (10000,)
  @test size(test_x) == (784,10000)
  @test size(test_label) == (10000,)
end
```

```
Test Summary:      | Pass  Total
correct dimensions |    4      4
```

```
## Model Dimensionality
# #### Set up model according to Appendix C (using Bernoulli decoder for Binarized
MNIST)
# Set latent dimensionality=2 and number of hidden units=500.
Dz, Dh = 2, 500
Ddata = 28^2
```

```
# ## Generative Model
# This will require implementing a simple MLP neural network
# See example_flux_model.jl for inspiration
# Further, you should read the Basics section of the Flux.jl documentation
# https://fluxml.ai/Flux.jl/stable/models/basics/
# that goes over the simple functions you will use.
# You will see that there's nothing magical going on inside these neural network
libraries
# and when you implemented a neural network in previous assignments you did most of the
work.
# If you want more information about how to use the functions from Flux, you can always
reference
# the internal docs for each function by typing `?` into the REPL:
# ? Chain
# ? Dense
```

Question 1.

1a.

```
## Model Distributions
function log_prior(z) #TODO
  return factorized_gaussian_log_density(0, 0, z)
end
```

```
log_prior (generic function with 1 method)
```

1b.

```
decoder = Chain(Dense(Dz,Dh, tanh), Dense(Dh, Ddata)) #TODO
```

```
Chain(Dense(2, 500, tanh), Dense(500, 784))
```

1c.

```
function log_likelihood(x,z)
  """ Compute log likelihood log_p(x|z)"""
  θ = decoder(z) # TODO: parameters decoded from latent z
  return sum(bernoulli_log_density(θ,x), dims=1) # return likelihood for each element in
batch
end
```

```
log_likelihood (generic function with 1 method)
```

1d.

```
function joint_log_density(x,z)
  return log_prior(z) .+ log_likelihood(x,z) #TODO
end
```

```
joint_log_density (generic function with 1 method)
```

```
## Amortized Inference
function unpack_gaussian_params(θ)
  μ, logσ = θ[1:2,:], θ[3:end,:]
  return  μ, logσ
end
```

```
unpack_gaussian_params (generic function with 1 method)
```

Question 2.

2a.

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

```
Chain(Dense(784, 500, tanh), Dense(500, 4), unpack_gaussian_params)
```

2b.

```
function log_q(q_μ, q_logσ, z)
  return factorized_gaussian_log_density(q_μ, q_logσ, z)
end
   #TODO: write log likelihood under variational distribution.

# Hint: last "layer" in Chain can be 'unpack_gaussian_params'
```

2c.

```
function elbo(x)
  q_μ, q_logσ = encoder(x) #TODO variational parameters from data
  z = sample_diag_gaussian(q_μ, q_logσ) #TODO: sample from variational distribution
  joint_ll = joint_log_density(x, z) #TODO: joint likelihood of z and x under model
  log_q_z = log_q(q_μ, q_logσ, z) #TODO: likelihood of z under variational distribution
  elbo_estimate = sum(joint_ll - log_q_z) / size(x)[2] #TODO: Scalar value, mean
variational evidence lower bound over batch
  return elbo_estimate
end
```

```
elbo (generic function with 1 method)
```

2d.

```
function loss(x)
  return -elbo(x) #TODO: scalar value for the variational loss over elements in the
batch
end

# Training with gradient optimization:
# See example_flux_model.jl for inspiration
```

2e.

```
function train_model_params!(loss, encoder, decoder, train_x, test_x; nepochs=10)
  # model params
  ps = Flux.params(encoder, decoder) #TODO parameters to update with gradient descent
  # ADAM optimizer with default parameters
  opt = ADAM()
  # over batches of the data
  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 # compute gradients with respect to variational loss over batch
      #TODO update the paramters with gradients
      Flux.Optimise.update!(opt,ps,gs)
    end
    if i%10 == 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_model_params! (generic function with 1 method)
```

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

### Save the trained model!
#using BSON:@save
#cd(@__DIR__)
#@info "Changed directory to $(@__DIR__)"
#save_dir = "trained_models"
#if !(isdir(save_dir))
#  mkdir(save_dir)
#  @info "Created save directory $save_dir"
#end
#@save joinpath(save_dir,"encoder_params.bson") encoder
#@save joinpath(save_dir,"decoder_params.bson") decoder
#@info "Saved model params in $save_dir"



## Load the trained model!
using BSON:@load
cd(@__DIR__)
@info "Changed directory to $(@__DIR__)"
load_dir = "trained_models"
@load joinpath(load_dir,"encoder_params.bson") encoder
@load joinpath(load_dir,"decoder_params.bson") decoder
@info "Load model params from $load_dir"
```

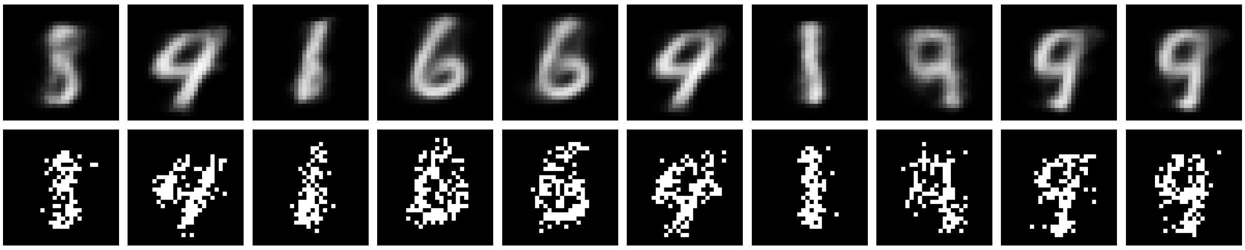
Question 3.

3a.

```
# Visualization
using Images
using Plots
## 3a
z1 = sample_diag_gaussian([0,0],[0,0])
m1 = decoder(z1)
x1 = sample_bernoulli(sigmoid.(m1))
mnist_img(m1) = ndims(m1)==2 ? Gray.(reshape(m1,28,28,:)) :
Gray.(transpose(reshape(m1,28,28)))
mnist_img(x1) = ndims(x1)==2 ? Gray.(reshape(x1,28,28,:)) :
Gray.(transpose(reshape(x1,28,28)))
# make vector of digits into images, works on batches also
# mnist_img(x) = ndims(x)==2 ? Gray.(reshape(x,28,28,:)) : Gray.(reshape(x,28,28))
z2 = sample_diag_gaussian([0,0],[0,0])
m2 = decoder(z2)
x2 = sample_bernoulli(sigmoid.(m2))
mnist_img(m2) = ndims(m2)==2 ? Gray.(reshape(m2,28,28,:)) :
Gray.(transpose(reshape(m2,28,28)))
mnist_img(x2) = ndims(x2)==2 ? Gray.(reshape(x2,28,28,:)) :
Gray.(transpose(reshape(x2,28,28)))
z3 = sample_diag_gaussian([0,0],[0,0])
m3 = decoder(z3)
x3 = sample_bernoulli(sigmoid.(m3))
mnist_img(m3) = ndims(m3)==2 ? Gray.(reshape(m3,28,28,:)) :
Gray.(transpose(reshape(m3,28,28)))
mnist_img(x3) = ndims(x3)==2 ? Gray.(reshape(x3,28,28,:)) :
Gray.(transpose(reshape(x3,28,28)))
```

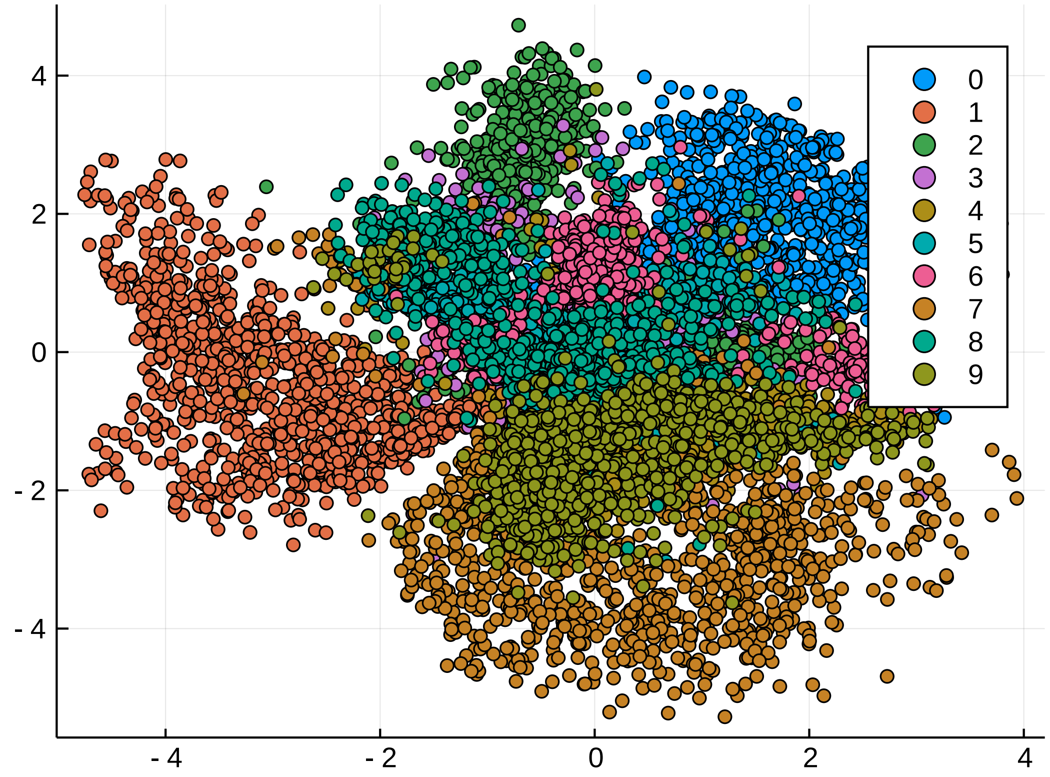
```
z4 = sample_diag_gaussian([0,0],[0,0])
m4 = decoder(z4)
x4 = sample_bernoulli(sigmoid.(m4))
mnist_img(m4) = ndims(m4)==2 ? Gray.(reshape(m4,28,28,:)) :
Gray.(transpose(reshape(m4,28,28)))
mnist_img(x4) = ndims(x4)==2 ? Gray.(reshape(x4,28,28,:)) :
Gray.(transpose(reshape(x4,28,28)))
z5 = sample_diag_gaussian([0,0],[0,0])
m5 = decoder(z5)
x5 = sample_bernoulli(sigmoid.(m5))
mnist_img(m5) = ndims(m5)==2 ? Gray.(reshape(m5,28,28,:)) :
Gray.(transpose(reshape(m5,28,28)))
mnist_img(x5) = ndims(x5)==2 ? Gray.(reshape(x5,28,28,:)) :
Gray.(transpose(reshape(x5,28,28)))
z6 = sample_diag_gaussian([0,0],[0,0])
m6 = decoder(z6)
x6 = sample_bernoulli(sigmoid.(m6))
mnist_img(m6) = ndims(m6)==2 ? Gray.(reshape(m6,28,28,:)) :
Gray.(transpose(reshape(m6,28,28)))
mnist_img(x6) = ndims(x6)==2 ? Gray.(reshape(x6,28,28,:)) :
Gray.(transpose(reshape(x6,28,28)))
z7 = sample_diag_gaussian([0,0],[0,0])
m7 = decoder(z7)
x7 = sample_bernoulli(sigmoid.(m7))
mnist_img(m7) = ndims(m7)==2 ? Gray.(reshape(m7,28,28,:)) :
Gray.(transpose(reshape(m7,28,28)))
mnist_img(x7) = ndims(x7)==2 ? Gray.(reshape(x7,28,28,:)) :
Gray.(transpose(reshape(x7,28,28)))
z8 = sample_diag_gaussian([0,0],[0,0])
m8 = decoder(z8)
x8 = sample_bernoulli(sigmoid.(m8))
mnist_img(m8) = ndims(m8)==2 ? Gray.(reshape(m8,28,28,:)) :
Gray.(transpose(reshape(m8,28,28)))
mnist_img(x8) = ndims(x8)==2 ? Gray.(reshape(x8,28,28,:)) :
Gray.(transpose(reshape(x8,28,28)))
z9 = sample_diag_gaussian([0,0],[0,0])
m9 = decoder(z9)
x9 = sample_bernoulli(sigmoid.(m9))
mnist_img(m9) = ndims(m9)==2 ? Gray.(reshape(m9,28,28,:)) :
Gray.(transpose(reshape(m9,28,28)))
mnist_img(x9) = ndims(x9)==2 ? Gray.(reshape(x9,28,28,:)) :
Gray.(transpose(reshape(x9,28,28)))
z10 = sample_diag_gaussian([0,0],[0,0])
m10 = decoder(z10)
x10 = sample_bernoulli(sigmoid.(m10))
mnist_img(m10) = ndims(m10)==2 ? Gray.(reshape(m10,28,28,:)) :
Gray.(transpose(reshape(m10,28,28)))
mnist_img(x10) = ndims(x10)==2 ? Gray.(reshape(x10,28,28,:)) :
Gray.(transpose(reshape(x10,28,28)))

## Example for how to use mnist_img to plot digit from training data
# plot(mnist_img(train_x[:,1]))
pl1 = Any[]
pl2 = Any[]
push!(pl1, plot(mnist_img(sigmoid.(m1))), plot(mnist_img(sigmoid.(m2))),
plot(mnist_img(sigmoid.(m3))), plot(mnist_img(sigmoid.(m4))),
plot(mnist_img(sigmoid.(m5))), plot(mnist_img(sigmoid.(m6))),
plot(mnist_img(sigmoid.(m7))), plot(mnist_img(sigmoid.(m8))),
plot(mnist_img(sigmoid.(m9))), plot(mnist_img(sigmoid.(m10))))
```

```
push!(pl2, plot(mnist_img(x1)), plot(mnist_img(x2)), plot(mnist_img(x3)),
plot(mnist_img(x4)), plot(mnist_img(x5)), plot(mnist_img(x6)), plot(mnist_img(x7)),
plot(mnist_img(x8)), plot(mnist_img(x9)), plot(mnist_img(x10)))
# push!(pl2, plot(mnist_img(log1pexp.(x3))), plot(mnist_img(log1pexp.(x4))))
# push!(pl3, plot(mnist_img(log1pexp.(x5))), plot(mnist_img(log1pexp.(x6))))
# push!(pl4, plot(mnist_img(log1pexp.(x7))), plot(mnist_img(log1pexp.(x8))))
# push!(pl5, plot(mnist_img(log1pexp.(x9))), plot(mnist_img(log1pexp.(x10))))
plot(pl1..., pl2..., axis = false, layout = grid(2, 10), size = (784*10, 784*2))
```



3b.

```
m, s = encoder(train_x[:,1:10000])
colorlist = train_label[1:10000]
scatter(m[1,:], m[2,:], group=colorlist)
```



3c.

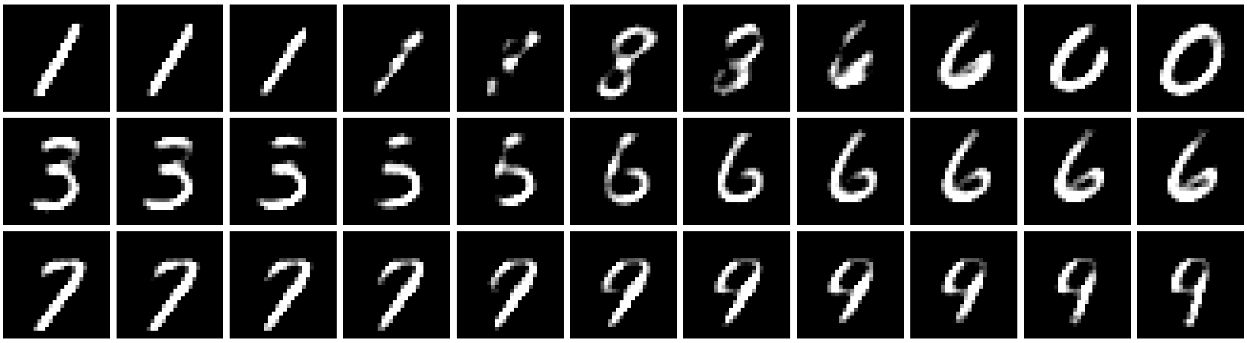
```
function linear_interpolation(z_a, z_b, α)
  return α * z_a + (1 - α) * z_b
end
```

```
label_0 = findall(x->x==0, train_label)
label_1 = findall(x->x==1, train_label)
label_6 = findall(x->x==6, train_label)
```

```
label_3 = findall(x->x==3, train_label)
label_9 = findall(x->x==9, train_label)
label_7 = findall(x->x==7, train_label)

encode0 = encoder(train_x[:,label_0[1]])
encode1 = encoder(train_x[:,label_1[1]])
encode6 = encoder(train_x[:,label_6[1]])
encode3 = encoder(train_x[:,label_3[1]])
encode9 = encoder(train_x[:,label_9[1]])
encode7 = encoder(train_x[:,label_7[1]])

pair01 = [encode0[1], encode1[1]]
pair63 = [encode6[1], encode3[1]]
pair97 = [encode9[1], encode7[1]]

li01_0 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0)))
li01_1 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.1)))
li01_2 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.2)))
li01_3 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.3)))
li01_4 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.4)))
li01_5 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.5)))
li01_6 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.6)))
li01_7 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.7)))
li01_8 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.8)))
li01_9 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 0.9)))
li01_10 = vec(decoder(linear_interpolation(pair01[1], pair01[2], 1)))
mnist_img(li01_0) = ndims(li01_0)==2 ? Gray.(reshape(li01_0,28,28,:)) :
Gray.(transpose(reshape(li01_0,28,28)))
mnist_img(li01_1) = ndims(li01_1)==2 ? Gray.(reshape(li01_1,28,28,:)) :
Gray.(transpose(reshape(li01_1,28,28)))
mnist_img(li01_2) = ndims(li01_2)==2 ? Gray.(reshape(li01_2,28,28,:)) :
Gray.(transpose(reshape(li01_2,28,28)))
mnist_img(li01_3) = ndims(li01_3)==2 ? Gray.(reshape(li01_3,28,28,:)) :
Gray.(transpose(reshape(li01_3,28,28)))
mnist_img(li01_4) = ndims(li01_4)==2 ? Gray.(reshape(li01_4,28,28,:)) :
Gray.(transpose(reshape(li01_4,28,28)))
mnist_img(li01_5) = ndims(li01_5)==2 ? Gray.(reshape(li01_5,28,28,:)) :
Gray.(transpose(reshape(li01_5,28,28)))
mnist_img(li01_6) = ndims(li01_6)==2 ? Gray.(reshape(li01_6,28,28,:)) :
Gray.(transpose(reshape(li01_6,28,28)))
mnist_img(li01_7) = ndims(li01_7)==2 ? Gray.(reshape(li01_7,28,28,:)) :
Gray.(transpose(reshape(li01_7,28,28)))
mnist_img(li01_8) = ndims(li01_8)==2 ? Gray.(reshape(li01_8,28,28,:)) :
Gray.(transpose(reshape(li01_8,28,28)))
mnist_img(li01_9) = ndims(li01_9)==2 ? Gray.(reshape(li01_9,28,28,:)) :
Gray.(transpose(reshape(li01_9,28,28)))
mnist_img(li01_10) = ndims(li01_10)==2 ? Gray.(reshape(li01_10,28,28,:)) :
Gray.(transpose(reshape(li01_10,28,28)))
pl01 = Any[]
push!(pl01, plot(mnist_img(li01_0)), plot(mnist_img(li01_1)), plot(mnist_img(li01_2)),
plot(mnist_img(li01_3)), plot(mnist_img(li01_4)), plot(mnist_img(li01_5)),
plot(mnist_img(li01_6)), plot(mnist_img(li01_7)), plot(mnist_img(li01_8)),
plot(mnist_img(li01_9)), plot(mnist_img(li01_10)))


li63_0 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0)))
li63_1 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.1)))
li63_2 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.2)))
li63_3 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.3)))
```

```
li63_4 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.4)))
li63_5 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.5)))
li63_6 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.6)))
li63_7 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.7)))
li63_8 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.8)))
li63_9 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 0.9)))
li63_10 = vec(decoder(linear_interpolation(pair63[1], pair63[2], 1)))
mnist_img(li63_0) = ndims(li63_0)==2 ? Gray.(reshape(li63_0,28,28,:)) :
Gray.(transpose(reshape(li63_0,28,28)))
mnist_img(li63_1) = ndims(li63_1)==2 ? Gray.(reshape(li63_1,28,28,:)) :
Gray.(transpose(reshape(li63_1,28,28)))
mnist_img(li63_2) = ndims(li63_2)==2 ? Gray.(reshape(li63_2,28,28,:)) :
Gray.(transpose(reshape(li63_2,28,28)))
mnist_img(li63_3) = ndims(li63_3)==2 ? Gray.(reshape(li63_3,28,28,:)) :
Gray.(transpose(reshape(li63_3,28,28)))
mnist_img(li63_4) = ndims(li63_4)==2 ? Gray.(reshape(li63_4,28,28,:)) :
Gray.(transpose(reshape(li63_4,28,28)))
mnist_img(li63_5) = ndims(li63_5)==2 ? Gray.(reshape(li63_5,28,28,:)) :
Gray.(transpose(reshape(li63_5,28,28)))
mnist_img(li63_6) = ndims(li63_6)==2 ? Gray.(reshape(li63_6,28,28,:)) :
Gray.(transpose(reshape(li63_6,28,28)))
mnist_img(li63_7) = ndims(li63_7)==2 ? Gray.(reshape(li63_7,28,28,:)) :
Gray.(transpose(reshape(li63_7,28,28)))
mnist_img(li63_8) = ndims(li63_8)==2 ? Gray.(reshape(li63_8,28,28,:)) :
Gray.(transpose(reshape(li63_8,28,28)))
mnist_img(li63_9) = ndims(li63_9)==2 ? Gray.(reshape(li63_9,28,28,:)) :
Gray.(transpose(reshape(li63_9,28,28)))
mnist_img(li63_10) = ndims(li63_10)==2 ? Gray.(reshape(li63_10,28,28,:)) :
Gray.(transpose(reshape(li63_10,28,28)))
pl63 = Any[]
push!(pl63, plot(mnist_img(li63_0)), plot(mnist_img(li63_1)), plot(mnist_img(li63_2)),
plot(mnist_img(li63_3)), plot(mnist_img(li63_4)), plot(mnist_img(li63_5)),
plot(mnist_img(li63_6)), plot(mnist_img(li63_7)), plot(mnist_img(li63_8)),
plot(mnist_img(li63_9)), plot(mnist_img(li63_10)))


li97_0 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0)))
li97_1 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.1)))
li97_2 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.2)))
li97_3 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.3)))
li97_4 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.4)))
li97_5 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.5)))
li97_6 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.6)))
li97_7 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.7)))
li97_8 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.8)))
li97_9 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 0.9)))
li97_10 = vec(decoder(linear_interpolation(pair97[1], pair97[2], 1)))
mnist_img(li97_0) = ndims(li97_0)==2 ? Gray.(reshape(li97_0,28,28,:)) :
Gray.(transpose(reshape(li97_0,28,28)))
mnist_img(li97_1) = ndims(li97_1)==2 ? Gray.(reshape(li97_1,28,28,:)) :
Gray.(transpose(reshape(li97_1,28,28)))
mnist_img(li97_2) = ndims(li97_2)==2 ? Gray.(reshape(li97_2,28,28,:)) :
Gray.(transpose(reshape(li97_2,28,28)))
mnist_img(li97_3) = ndims(li97_3)==2 ? Gray.(reshape(li97_3,28,28,:)) :
Gray.(transpose(reshape(li97_3,28,28)))
mnist_img(li97_4) = ndims(li97_4)==2 ? Gray.(reshape(li97_4,28,28,:)) :
Gray.(transpose(reshape(li97_4,28,28)))
mnist_img(li97_5) = ndims(li97_5)==2 ? Gray.(reshape(li97_5,28,28,:)) :
Gray.(transpose(reshape(li97_5,28,28)))
```

```
mnist_img(li97_6) = ndims(li97_6)==2 ? Gray.(reshape(li97_6,28,28,:)) :
Gray.(transpose(reshape(li97_6,28,28)))
mnist_img(li97_7) = ndims(li97_7)==2 ? Gray.(reshape(li97_7,28,28,:)) :
Gray.(transpose(reshape(li97_7,28,28)))
mnist_img(li97_8) = ndims(li97_8)==2 ? Gray.(reshape(li97_8,28,28,:)) :
Gray.(transpose(reshape(li97_8,28,28)))
mnist_img(li97_9) = ndims(li97_9)==2 ? Gray.(reshape(li97_9,28,28,:)) :
Gray.(transpose(reshape(li97_9,28,28)))
mnist_img(li97_10) = ndims(li97_10)==2 ? Gray.(reshape(li97_10,28,28,:)) :
Gray.(transpose(reshape(li97_10,28,28)))
pl97 = Any[]
push!(pl97, plot(mnist_img(li97_0)), plot(mnist_img(li97_1)), plot(mnist_img(li97_2)),
plot(mnist_img(li97_3)), plot(mnist_img(li97_4)), plot(mnist_img(li97_5)),
plot(mnist_img(li97_6)), plot(mnist_img(li97_7)), plot(mnist_img(li97_8)),
plot(mnist_img(li97_9)), plot(mnist_img(li97_10)))

display(plot(pl01..., pl63..., pl97..., axis = false, layout = grid(3, 11), size =
(784*11, 784*3)))
```



```
## The first row is encoding a pair of 0 and 1, the second row is encoding a pair of 6
and 3, and the third row is encoding a pair of 9 and 7.
```

Question 4.

4a.

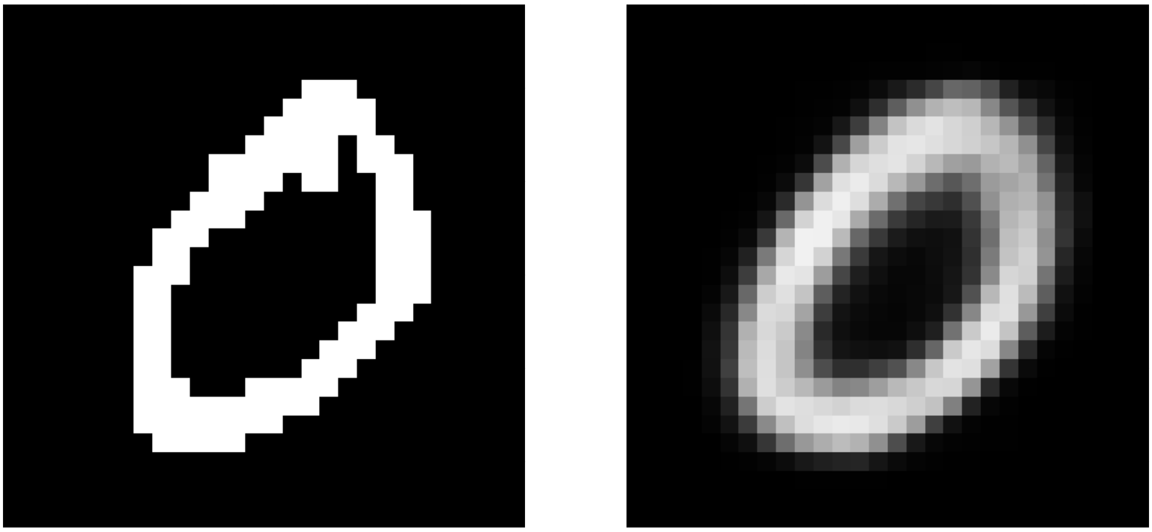
```
function top_half(whole_image)
  return whole_image[1:392]
end

function likelihood_factorizes(z, x)
  z = decoder(z)[1:392,:]
  return sum(bernoulli_log_density(z, x), dims=1)
end

function log_joint_density(z, x)
  return likelihood_factorizes(z, x) + log_prior(z)
end
```

```
log_joint_density (generic function with 1 method)
```

4b.

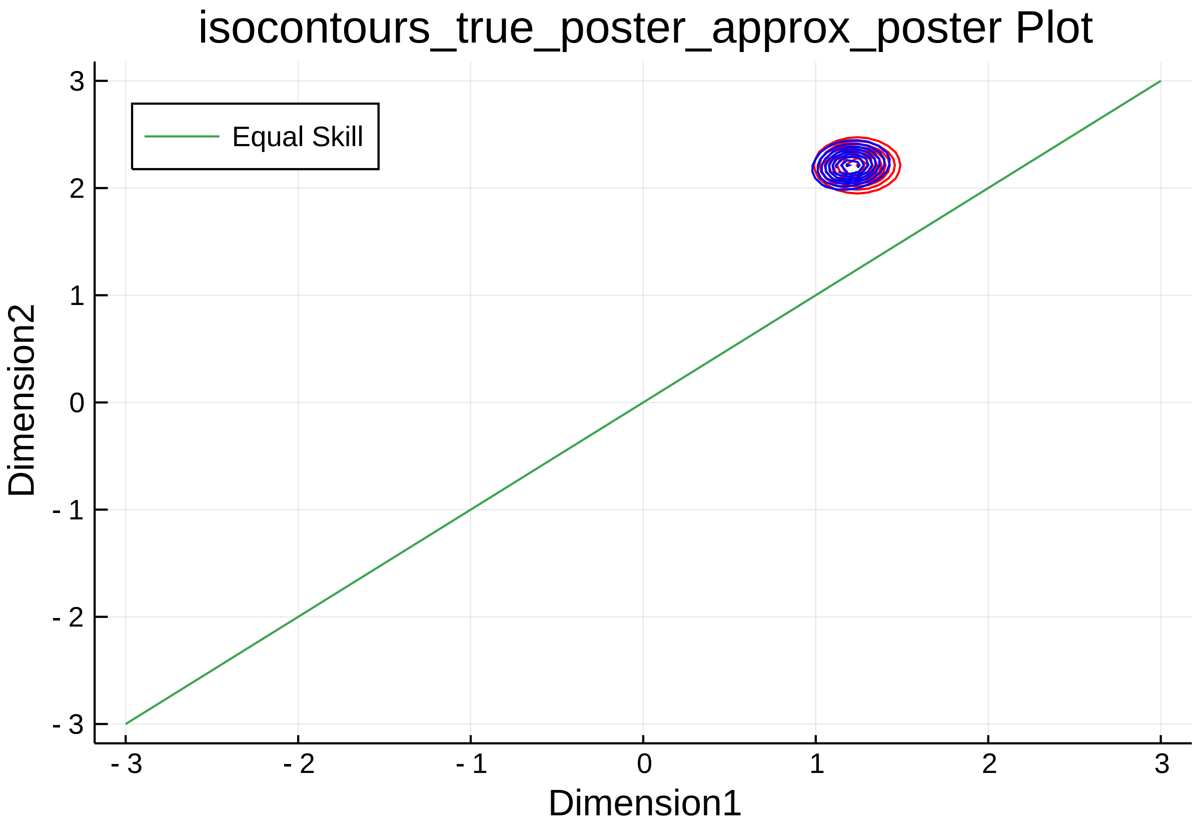
```
## choosing the second image, number 0
m1, s1 = encoder(train_x[:,2])
z11 = sample_diag_gaussian(m1, s1)
ss1 = vec(decoder(z11))
mnist_img(ss1) = ndims(ss1)==2 ? Gray.(reshape(ss1,28,28,:)) :
Gray.(transpose(reshape(ss1,28,28)))
```

```
plot(mnist_img(sigmoid.(ss1)))
choo = Any[]
push!(choo, plot(mnist_img(train_x[:,2])), plot(mnist_img(sigmoid.(ss1))))
display(plot(choo..., axis = false, layout = grid(1, 2), size = (784*2, 784*1)))
```

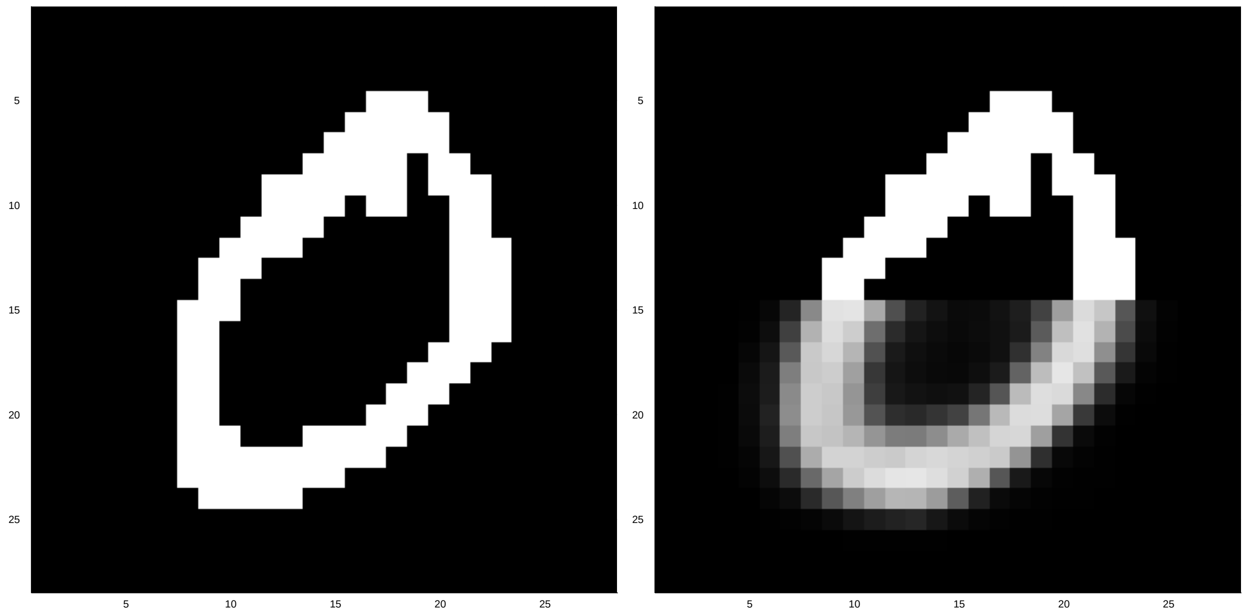


```
## 0 digit is modelled well, train data vs encoded and decoded data

ϕ_μ = [0.,0.]
ϕ_logσ = [0.,0.]
params_init = [ϕ_μ, ϕ_logσ]

function elbo(params; images=train_x[:,2], num_samples=50)
  a, b = params
  sample1 = sample_diag_gaussian(a, b)
  for i in 2:1:num_samples
      sample2 = sample_diag_gaussian(a, b)
      sample1 = hcat(sample1, sample2)
  end
  half_images = top_half(images)
  # logq
  lq = log_q(a, b, sample1)
  jld = log_joint_density(sample1, half_images)
  return mean(jld .- lq)
end




function loss_elbo(params; images=train_x[:,2], num_samples=50)
  return -elbo(params)
end


function optimize_μ_logσ(params, sample_images; num_itrs=100, lr= 1e-2, num_q_samples =
50)
  params_cur = params
  for i in 1:num_itrs
    grad_params = gradient(params_cur -> loss_elbo(params_cur;images=sample_images,
num_samples=num_q_samples), params_cur)#TODO: gradients of variational objective wrt
```

```
params
    params_cur[1] .=  params_cur[1] - lr * grad_params[1][1]
    params_cur[2] .=  params_cur[2] - lr * grad_params[1][2] #TODO: update parmaeters
wite lr-sized steps in desending gradient direction
    @info "loss: $(loss_elbo(params_cur;images=sample_images,
num_samples=num_q_samples))"#TODO: report objective value with current parameters
  end
  return params_cur
end

r, t = optimize_μ_logσ(params_init, train_x[:,2])

function skillcontour!(f; colour=nothing, label="sample gaussian")
  n = 100
  x = range(-3,stop=3,length=n)
  y = range(-3,stop=3,length=n)
  z_grid = Iterators.product(x,y) # meshgrid for contour
  z_grid = reshape.(collect.(z_grid),:,1) # add single batch dim
  z = f.(z_grid)
  z = getindex.(z,1)'
  max_z = maximum(z)
  levels = [.99, 0.9, 0.8, 0.7,0.6,0.5, 0.4, 0.3, 0.2] .* max_z
  if colour==nothing
  p1 = contour!(x, y, z, fill=false, levels=levels)
  else
  p1 = contour!(x, y, z, fill=false, c=colour,levels=levels,colorbar=false)
  end
  plot!(p1)
end

function plot_line_equal_skill!()
  plot!(range(-3, 3, length=200), range(-3, 3, length=200), label="Equal Skill")
end

plot(title="isocontours_true_poster_approx_poster Plot", xlabel = "Dimension1", ylabel =
"Dimension2", legend=:topleft);

approx_posterior(zs) = exp(log_q(r, t, zs))
true_posterior(zs) = exp.(log_joint_density(zs, top_half(train_x[:,2])))
skillcontour!(approx_posterior, colour=:red, label=:"approx_posterior")
skillcontour!(true_posterior, colour=:blue, label=:"true_posterior")
display(plot_line_equal_skill!())
```



```
## blue color isocontours is approximate posterior.
## red color isocontours is true posterior.

z4c = sample_diag_gaussian(r, t)
x4c = vec(decoder(z4c)[393:784,:])
mnist_img(x4c) = ndims(x4c)==2 ? Gray.(reshape(x4c,14,28,:)) :
Gray.(transpose(reshape(x4c,28,14)))
plot(mnist_img(sigmoid.(x4c)))
plot(mnist_img(train_x[1:392,2]))
pred_image = vcat(train_x[1:392,2], sigmoid.(x4c))
mnist_img(pred_image) = ndims(pred_image)==2 ? Gray.(reshape(pred_image,28,28,:)) :
Gray.(transpose(reshape(pred_image,28,28)))
plot(mnist_img(pred_image))

pl_compare = Any[]
push!(pl_compare, plot(mnist_img(train_x[:,2])), plot(mnist_img(pred_image)))

plot(pl_compare..., layout = grid(1, 2), size = (784*2, 784*1))
```



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