STA414 Assignment 3

Jeff Blair: 1002177057
blairjef

Collaborator: Jai Aggarwal

April 2020

1 Implementing the Model

a. `log_prior(z) = factorized_gaussian_log_density(0, 0, z)`

b. `decoder = Chain(Dense(Dz, Dh, tanh), Dense(Dh, Ddata))`

c.
```
function log_likelihood(x,z)
  """ Compute log likelihood log_p(x|z)"""
  theta = decoder(z)
  return  sum(bernoulli_log_density(theta, x), dims=1) # return likelihood for each element in batch
end
```

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

2 Amortized Approximate Inference and training

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

b. `log_q(q_mu, q_logsigma, z) = factorized_gaussian_log_density(q_mu, q_logsigma, z)`

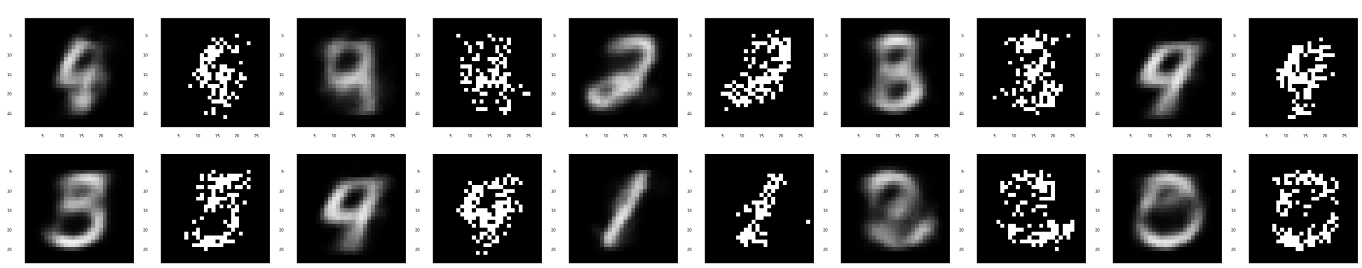
c.
```
function elbo(x)
  q_mu, q_logsigma = encoder(x)
  z = sample_diag_gaussian(q_mu, q_logsigma)
  joint_ll = joint_log_density(x, z)
  log_q_z = log_q(q_mu, q_logsigma, z)
  elbo_estimate = mean(joint_ll .- log_q_z)
  return elbo_estimate
end
```

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

e.
```
function train_model_params!(loss, encoder, decoder, train_x, test_x; nepochs=10)
  # model params
  ps = Flux.params(encoder, decoder)
  # ADAM optimizer with default parameters
  opt = ADAM()
  # over batches of the data
  for i in 1:nepochs
    for d in batch_x(train_x)
     # compute gradients with respect to variational loss over batch
      gs = Flux.gradient(ps) do
                return loss(d)
            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)
 Info: Test loss at epoch 100: 155.76130769378668
 @ Main In[144]:20
 Info: Parameters of encoder and decoder trained!
 @ Main In[144]:23
```

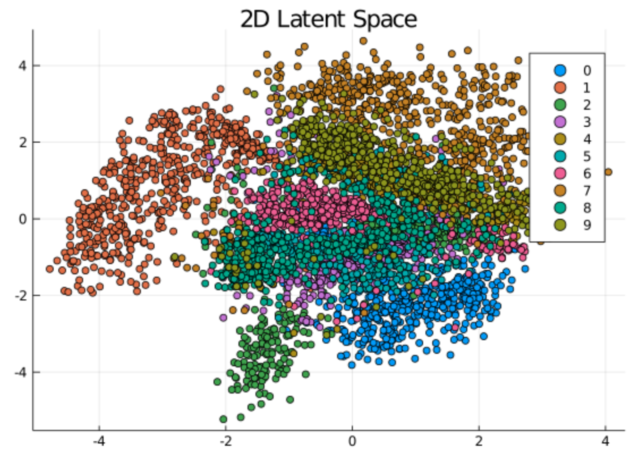
3 Visualizing Posteriors and Exploring the Model

a.
```
plot_list = Any[]
for i in 1:10
    z = sample_diag_gaussian([0, 0],0)
    # Use generative model to compute the bernoulli means over the pixels of x given z.
    theta = decoder(z)
    means = exp.(theta)./(1 .+ exp.(theta))
    # Plot as a greyscale image
    push!(plot_list, plot(mnist_img(means)))
    # Sample a binary image x from this product of Bernoullis. Plot this sample
    out = sample_bernoulli.(means)
    push!(plot_list, plot(mnist_img(out), size=(4000, 800)))
end
display(plot(plot_list..., layout=grid(2,10)))
```



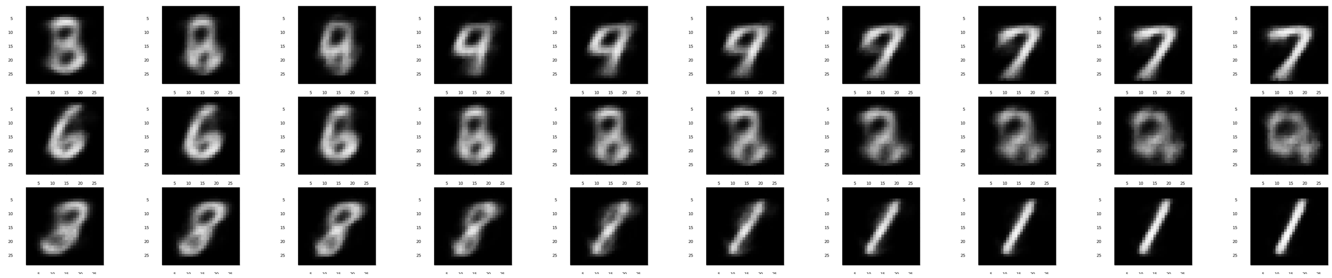
(a) Means vs Binary

b.
```julia
encodings = encoder(train_x)
means = encodings[1]
scatter(means[1,:], means[2,:], group=train_label, title = "2D Latent Space")
```



(a) 2D latent space

c.
```julia
function li(za, zb, alpha=0.5)
    return (alpha .* za) .+ ((1 .- alpha) .* zb)
end

labels = []
for i in 1:3
    push!(labels, sample([0,1,2,3,4,5,6,7,8,9], 2; replace=false))
end

# 3 pairs of images
images = []
for pair in labels
    s1 = train_x[:, train_label .== pair[1]]
    s2 = train_x[:, train_label .== pair[2]]
    image1 = s1[:,rand(1:size(s1)[2])]
    image2 = s2[:,rand(1:size(s2)[2])]
    push!(images, [encoder(image1)[1], encoder(image2)[1]])
end

# 10 interpolations per pair
plot_list = []
for pair in images
    for alpha in 0.1:0.1:1
        interp = li(pair[2], pair[1], alpha)
        theta = decoder(interp[:,1])
        means = exp.(theta)./(1 .+ exp.(theta))
        # Plot as a greyscale image
        push!(plot_list, plot(mnist_img(means), size=(4000,800)))
    end
end
print(labels)
display(plot(plot_list..., layout=grid(3,10)))
```



(a) Linear Interpolation

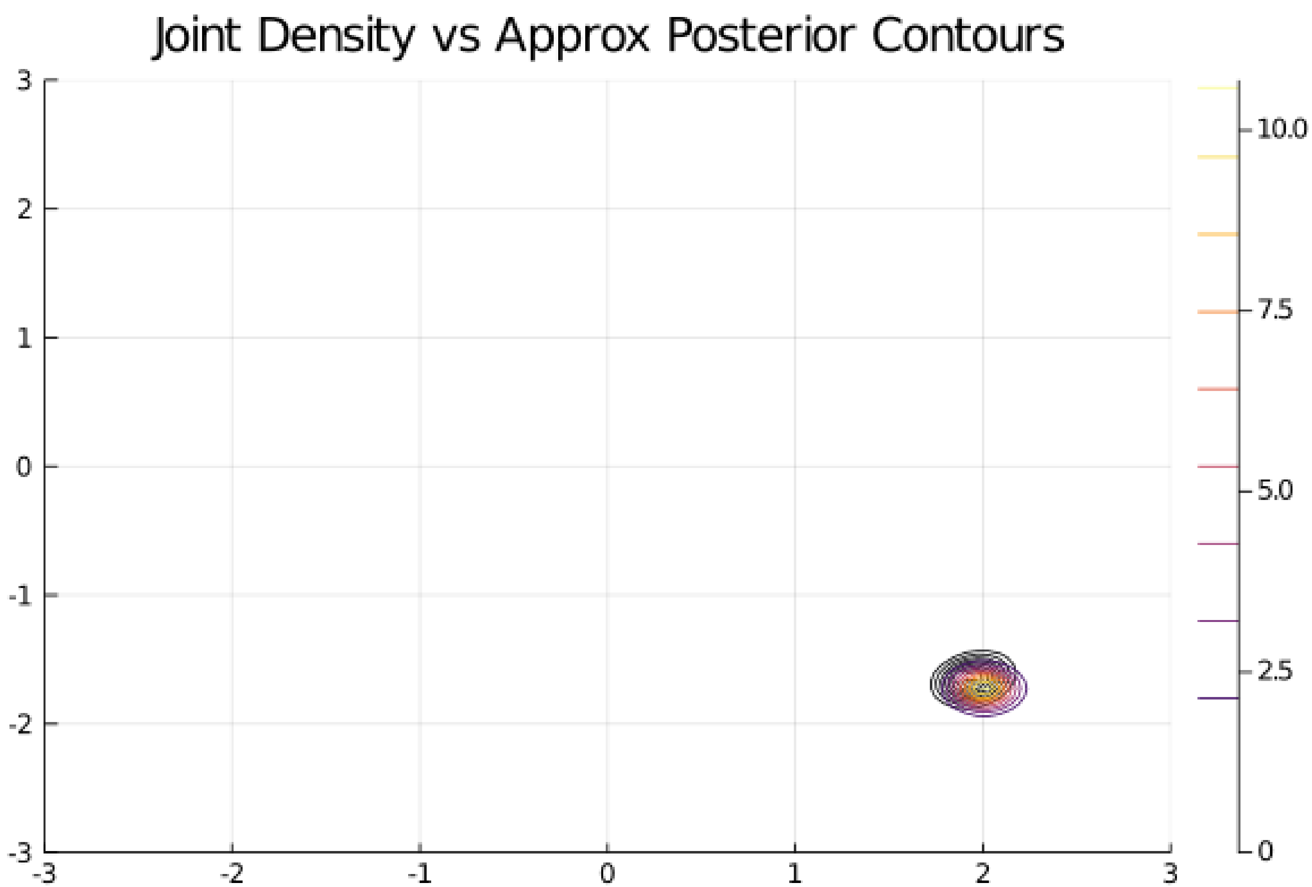
Labels for the first row were $8 \to 7$, second row $6 \to 2$, third row $3 \to 1$

4 Predicting the Bottom of Images given the Top

a.
```
# Write a funciton that computes p(z|top half of the image)
function top_half(x)
    y = reshape(x, 28, 28,:)
    z = y[:,1:14,:]
    return reshape(z, (392,:))
end

# Write a function that computes log p(top half of image x | z)
function top_log_px_z(x, z)
    theta = decoder(z)
    return sum(bernoulli_log_density(top_half(theta), top_half(x)), dims=1)
end

# Combine this likelihood with the prior to get a function that computes the joint log density
function joint_log_density_top(x, z)
    return log_prior(z) + top_log_px_z(x, z)
end
```

b.
```
# Initialize mu and sigma
mu = randn(2)
logsigma = randn(2)

# Write a function that computes ELBO over K samples
function elbo_th(x, opts, k)
    mu, logsigma = opts
    z = sample_diag_gaussian(repeat(mu, 1, k), logsigma)
    joint_ll = joint_log_density_top(x, z)
    log_q_z = log_q(mu, logsigma, z)
    elbo_estimate = mean(joint_ll .- log_q_z)
    return elbo_estimate
end

function loss_th(x, opts, k)
    return -elbo_th(x, opts, k)
end

# Training
s = train_x[:, rand(1:1000)]
function train(opts, t_x, iters=200, lr=1e-2, k=10)
    opts_cur = opts
#     x = t_x[:, rand(1:1000)]
    x = t_x
```

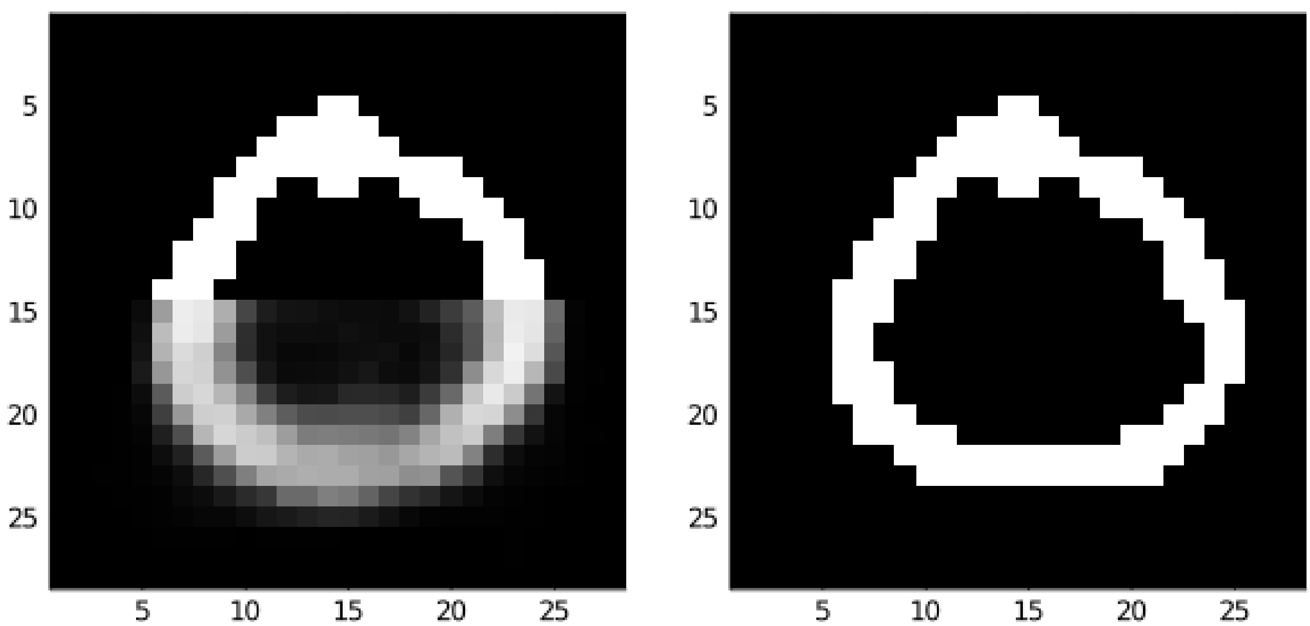
```
    for i in 1:iters
        opts_grad = gradient(opts_cur -> loss_th(x, opts_cur, k), opts_cur)
        opts_cur[1] .-= lr .* opts_grad[1][1]
        opts_cur[2] .-= lr .* opts_grad[1][2]
        @info "Elbo:" elbo_th = elbo_th(x, opts_cur, k)
    end
    return opts_cur
end

# Train Paramters
opts = mu, logsigma
opts = train(opts, s)

# Plotting Isocontours
mu, logsigma = opts
joint(z) = exp(joint_log_density_top(s, z))
post(z) = exp(log_q(mu, logsigma, z))
plot(title="Joint Density vs Approx Posterior Contours")
skillcontour!(joint)
skillcontour!(post)
```



(a) Contours

```
z = sample_diag_gaussian(mu, logsigma)

# Use generative model to compute the bernoulli means over the pixels of x given z.
theta = decoder(z)
means = exp.(theta)./(1 .+ exp.(theta))
```

```
# Concatenate top half and predicted bottom half
top = top_half(s)
bot = reshape(means, 28, 28, :)
bot = bot[:,15:end,:]
bot = reshape(bot, (392,:))

img = vcat(top, bot)
plot_list = [plot(mnist_img(img[:,1])), plot(mnist_img(s))]
display(plot(plot_list...))
savefig("4be")
```



(a) Predicted bottom half

c. a) True (Yes)
   b) False (No)
   c) True (Yes)
   d) False (No)
   e) True (Yes)