assignment3

March 28, 2019

1 CSC412/2506 Assignment 3: Variational Auto Encoders

In this assignment we will learn how to preform efficient inference and learning in directed graphical models with continuous latent variables.We will use stochastic variational inference with automatic differentiation (SADVI) to approximate intractible posterior distributions. We will implement the two gradient estimators discussed in lecture, Score Function and Reparamterization, and experimentally demonstrate their properties such as biasedness and variance. We will use the reparameterization gradient estimators to optimize the ELBO of our latent variable model.

You can use automatic differentiation in your code. You may also use a machine learning framework to specify the encoder and decoder neural networks, and provide gradientent optimizers such as ADAM. However, you may not use any probabilistic modelling elements for these frameworks. In particular, sampling from and evaluating densities under distributions must be written by you.

2 Implementing the VAE [20pts]

In this assignment we will implement and investigate the Variational Auto Encoder on Binarized MNIST digits detailed in Auto-Encoding Variational Bayes by Kingma and Welling (2013). Before starting, read this paper. In particular, we will implement model as described in Appendix C.

2.1 Load and Prepare Data

Load the MNIST dataset, binarize the images, split into a training dataset of 10000 images and a test set of 10000 images. Also partition the training set into minibatches of size M=100.

```
In [ ]: # You may use the script provided in A2 or dataloaders provided by framework
```

2.2 Distributions [5pts]

Implement code to sample from and evaluate the log-pdf of diagonal multivariate gaussians $\mathcal{N}(x|\mu, \sigma^2 I)$ and Bernoulli distributions. For sampling from these distributions, you have access to samples from uniform and unit Gaussians, (`rand` and `randn`). Make sure you test you've implemented these correctly by comparing to standard packages!

```
In [ ]: # sampler from Diagonal Gaussian x~N(,^2 I) (hint: use reparameterization trick here)

        # sampler from Bernoulli
        # log-pdf of x under Diagonal Gaussian N(x|,^2 I)

        # log-pdf of x under Bernoulli
```

2.3 Defining Model Architecture [5pts]

Implement the model as defined in Appendix C. The MLPs will have a single hidden layer with Dh=500 hidden units. The dimensionality of the latent space will be Dz=2 for visualization purposes later.

Note that the output of the encoder will be $[\mu, \log \sigma^2]$. Why not ouput σ^2 directly? Keep this in mind when you sample from the distribution using your Diagonal Gaussian sampler.

```
In [ ]: # Set latent dimensionality=2 and number of hidden units=500.

        # Define MLP for recognition model / "encoder"
        # Provides parameters for q(z|x)

        # Define sample from recognition model
        # Samples z ~ q(z|x)

        # Define MLP for generative model / "decoder"
        # Provides parameters for distribution p(x|z)
```

2.4 Variational Objective [7pts]

Here we will use the log-pdfs, the encoder, gaussian sampler, and decoder to define the Monte Carlo estimator for the mean of the ELBO over the minibatch.

```
In [ ]: # log_q(z|x) logprobability of z under approximate posterior N(,^2)

        # log_p_z(z) log probability of z under prior

        # log_p(x|z) - conditional probability of data given latents.

        # Monte Carlo Estimator of mean ELBO with Reparameterization over M minibatch samples.
        # This is the average ELBO over the minibatch
        # Unlike the paper, do not use the closed form KL between two gaussians,
        # Following eq (2), use the above quantities to estimate ELBO as discussed in lecture
```

2.5 Optimize with Gradient Descent

Minimize the -ELBO with ADAM optimizer. You may use the optimizer provided by your framework

```
In [ ]: # Load Saved Model Parameters (if you've already trained)
```

```
In [ ]: # Set up ADAM optimizer

        # Train for ~200 epochs (1 epoch = all minibatches in traindata)
```

```
In [ ]: # Save Optimized Model Parameters
```

2.6 Report ELBO on Training and Test Set [3pts]

```
In [ ]: # ELBO on training set

        # ELBO on test set
```

3 Numerically Computing Intractable Integrals [10pts]

3.1 Numerical Integration over Latent Space [5pts]

Since we chose a low dimensional latent space, we are able to perform numerical integration to evaluate integrals which are intractible in higher dimension.

For instance, we will use this to integrate over the latent space. e.g. the

$$p(z|x) = \frac{p(x|z) * p(z)}{p(x)} = \frac{p(x|z) * p(z)}{\int p(x|z) * p(z) dz}$$

We want to numerically compute that integral. However, since we are parameterizing $\log p(x|z)$ and $\log p(z)$ we will have

$$\log p(z|x) = \log p(x|z) + \log p(z) - \log \int \exp[\log p(x|z) + \log p(z)] dz$$

You will write code which computes $\log \int \exp \log f(z) dz$ given an equally spaced grid of $\log f(z)$s as input. Note that if we approximate that integral with a numerical sum, in order for it to be numerically stable we will need `logsumexp`.

```
In [ ]: # Implement log sum exp

        # Implement stable numerical integration
        # over a 2d grid of equally spaced (delta_z) evaluations logf(x)
```

3.2 Compare Numerical Log-Likelihood to ELBO [5pts]

We can use the numerical integration to compute the log-likeihood of a element in our dataset under our model. We can then compare the numerical integration to the estimate given by the ELBO.

```
In [ ]: # Define the delta_z to be the spacing of the grid  (I used delta_z = 0.1)

        # Define a grid of delta_z spaced points [-4,4]x[-4,4]

        # Sample an x from the data to evaluate the likelhiood

        # Compute log_p(x|z)+log_p(z) for every point on the grid

        # Using your numerical integration code
        # integrate log_p(x|z)+log_p(z) over z to find log_p(x)
        # Check that your numerical integration is correct
        # by integrating log_p(x|z)+log_p(z) - log_p(x)
        # If you've successfully normalized this should integrate to 0 = log 1

        # Now compute the ELBO on x

        # Try this for multiple samples of x
        # note that the ELBO is a lower bound to the true log_p(x)!
```

4 Data Space Visualizations [10pts]

In this section we will investigate our model by visualizing the distributions over data given by the generative model, samples from these distributions, and reconstructions of the data.

```
In [ ]: # Write a function to reshape 784 array into a 28x28 image for plotting
```

4.1 Samples from the generative model [5pts]

Here you will sample from the generative model using ancestral sampling.

- First sample a z from the prior.
- Then use the generative model to parameterize a bernoulli distribution over x given z. Plot this distribution.
- Then sample x from the distribution. Plot this sample.

Do this for 10 samples z from the prior.

Concatenate all your plots into one 10x2 figure where the first column is the distribution over x and the second column is a sample from this distribution. Each row will be a new sample from the prior.

```
In [ ]: # Sample 10 z from prior

        # For each z, plot p(x|z)

        # Sample x from p(x|z)

        # Concatenate plots into a figure
```

4.2 Reconstructions of data [5pts]

Here we will investigate the VAEs ability to reconstruct 10 inputs from the data. For each input ou will

- Plot the input x
- Use the recognition network to encode x to the parameters for a distribution $q(z|x)$
- Sample $z \sim q(z|x)$
- Use the generative model to decode to the parameters for distribution $p(x|z)$. Plot this

- Sample $\tilde{x} \sim p(x|z)$. Plot this

Then you will concatenate all your plots into a 10x3 figure where the first column is the input data, the second column is the distribution over x, the third column is a reconstruction of the input. Each row will be a new sample from the data.

```
In [ ]: # Sample 10 xs from the data, plot.

        # For each x, encode to distribution q(z|x)

        # For each x, sample distribution z ~ q(z|x)

        # For each z, decode to distribution p(x|z), plot.

        # For each x, sample from the distribution x ~ p(x|z), plot.

        # Concatenate all plots into a figure.
```

5 Latent Space Visualizations [15pts]

In this section we will investigate our model by visualizing the latent space through various methods. These will include encoding the data, decoding along a grid, and linearly interpolating between encdoded data.

5.1 Latent embedding of data [5pts]

One way to understand what is represented in the latent space is to consider where it encodes elements of the data. Here we will produce a scatter plot in the latent space, where each point in the plot will be the mean vector for the distribution $q(z|x)$ given by the encoder. Further, we will colour each point in the plot by the class label for the input data.

Hopefully our latent space will have learned to distinguish between elements from different classes, even though we never provided class labels to the model!

```
In [ ]: # Encode the training data

        # Take the mean vector of each encoding

        # Plot these mean vectors in the latent space with a scatter
        # Colour each point depending on the class label
```

5.2 Decoding along a lattice [5pts]

We can also understand the "learned manifold" by plotting the generative distribution $p(x|z)$ for each point along a grid in the latent space. We will replicate figure 4b in the paper.

```
In [ ]: # Create a 20x20 equally spaced grid of z's
        # (use the  previous figure to help you decide appropriate bounds for the grid)
```

```
         # For each z on the grid plot the generative distribution over x

         # concatenate these plots to a lattice of distributions
```

5.3 Interpolate between two classes [5pts]

A common technique to assess latent representations is to interpolate between two points.

Here we will encode 3 pairs of data points with different classes. Then we will linearly interpolate between the mean vectors of their encodings. We will plot the generative distributions along the linear interpolation.

```
In [ ]: # Function which gives linear interpolation z_ between za and zb

         # Sample 3 pairs of data with different classes

         # Encode the data in each pair, and take the mean vectors

         # Linearly interpolate between these mean vectors

         # Along the interpolation, plot the distributions p(x|z_)

         # Concatenate these plots into one figure
```

6 Posteriors and Stochastic Variational Inference [20pts]

Here we will use numerical integration to plot the "true" posterior $p(z|x)$ which is generally intractable. We will compare the intractable true posterior to the variational approximate posterior given by the recognition model $q(z|x)$.

Then we will use the generative model to perform inference other inference tasks. In particular, we will see that the purpose of the encoder was only to make training the generative model tractable, and that we can do inference using the generative model completely without the encoder. To illustrate this we will perform the inference task of producing a generative distribution over the bottom half of the digit conditioned on the top half. We will achieve this with stochastic variational inference.

6.1 Plotting Posteriors [5pts]

Here we will plot the true posterior by evaluating $\log p(x|z) + \log p(z)$ on an equally spaced grid over z then numerically integrating over this grid to find the log-normalizer $\log p(x)$. This will give us the intractable true posterior $p(z|x)$.

Then we will compare the true posterior to the approximate posterior given by the recognition model $q(z|x)$. Does the recognition model produce a good approximate posterior to the intractable true posterior?

```
In [ ]: # Sample an element x from the dataset to plot posteriors for

         # Define a grid of equally spaced points in z
         # The grid needs to be fine enough that the plot is nice
```

```
        # To keep the integration tractable
        # I reccomend centering your grid at the mean of q(z|x)

        # Evaluate log_p(x|z) + log_p(z) for every z on the grid

        # Numerically integrate log_p(x|z) + log_p(z) to get log_p(x)

        # Produce a grid of normalized log_p(z|x)

        # Plot the contours of p(z|x) (note, not log)

        # Evaluate log_q(z|x) recognition network for every z on grid

        # Plot the contours of q(z|x) on previous plot
```

6.2 True posterior for top of digit [5pts]

In this question we will plot the "true" posterior given only the top of the image, $p(z|x_{top})$.

Realize that the generative model gives a Bernoulli distribution over each pixel in the image. We can easily evaluate the likelihood of only the top of an image by evaluating under only those corresponding dimensions of the generative model.

```
In [ ]: # Function which returns only the top half of a 28x28 array
        # This will be useful for plotting, as well as selecting correct bernoulli params

        # log_p(x_top | z) (hint: select top half of 28x28 bernoulli param array)

        # Sample an element from the data set and take only its top half: x_top

        # Define a grid of equally spaced points in z

        # Evaluate log_p(x_top | z) + log_p(z) for every z on grid

        # Numerically integrate to get log_p(x_top)

        # Normalize to produce grid of log_p(z|x_top)

        # Plot the contours of p(z|x_top)
```

6.3 Learn approximate posterior for top of digit with Stochastic Variational Inference [10 pts]

In this question we will see how we can use SVI to learn an approximate posterior to $p(z|x_{top})$ which we just obtained through intractable integration.

Note that we can't just use our recognition model, because our encoder doesn't know what to do with only top halfs of images. Instead, we will initialize a variational distribution $q(z) = \mathcal{N}(z|\mu, \sigma^2 I)$ and optimize the ELBO to minimize the KL divergence between it and the true distribution.

```
In [ ]: # Initialize parameters  and log for variational distribution q(z)

        # Define mean ELBO over M samples z ~ q(z)
        # using log_p(z), log_p(x_top | z), and q(z|x_top)

        # Loss for SVI is -1*ELBO

        # Set up ADAM to optimize  and log^2

        # Optimize for a few iterations until convergence (you can use a larger stepsize here)
```

```
In [ ]: # On previous plot of contours of p(z|x_top) plot the optimized q(z)
```

```
In [ ]: # Sample z ~ q(z)

        # Use generative model p(x|z) to produce distribution over x

        # Extract the bottom half of this generative distribution: p(x_bot| z)

        # Concatenate the x_top and p(x_bot | z) and plot.
```

7 Investigating Gradient Estimators [Bonus 5pts]

In this part we will experimentally investigate the difference in variances between the gradient estimates given by the Reparameterization and Score-Function gradient estimators.

Comment on their mean and variances

```
In [ ]: # Use Reparameterization Gradient Estimator
        # to estimate gradient of mean ELBO wrt  over M minibatch samples
        # hint: this will involve just taking gradients through the code used to train

        # Use Score-Function Gradient Estimator
        # to estimate gradient of mean ELBO wrt  over M minibatch samples
        # make sure you are not useing the reparameterization trick to sample z from q
        # you should only be taking gradients through log_q(z|x), no gradients through ELBO or z

        # Consider the gradients wrt the first component of
        # Produce two histograms in two different subplots
        # First show the distribution of gradients given by Reparameterization estimator
        # Second show the distribution of gradients given by Score Function Estimator
```