**CS 7140 Advanced Machine Learning**
Northeastern University, Spring 2020

# Homework 4

Due 11:59 p.m. Tue, April 7th, 2020

Please submit your written solutions as a PDF file using the provided LaTeX template. (`http://roseyu.com/CS7140/latex_template.tex`). For problems requiring coding, additionally submit all necessary code. Code must run and produce the results reported in the PDF for full credit.

## 1 Deep Structured Prediction

### Problem A [2 points]: Convolutional Neural Network

**i. [1 points]:** Given two functions

$$a(x) = \begin{cases} 1 & -1 \leq x \leq 1 \\ 0 & else \end{cases} \qquad b(x) = \begin{cases} 1+x & -1 \leq x \leq 0 \\ 1-x & 0 \leq x \leq 1 \\ 0 & else \end{cases} \tag{1}$$

Compute $c(x) = a(x) \star b(x)$, i.e. the convolution of function $a(x)$ and function $b(x)$ . Show all steps of the derivation for your answer.

**ii. [1 points]:** Using the definition of convolution, explain why a convolutional neural network is translation invariant with mathematical equations.

### Problem B [2 points]: Message Passing Neural Network

Message passing neural network (MPNN) [1] is a modern generalization of Hopfield network. Answering the following questions for MPNN.

**i. [1 points]:** Message passing update can be viewed as a generalization of a convolution from a regular grid to a graph (irregular). Explain this relationship using a few sentences.

**ii. [1 points]:** What is the benefit of using a Message Passing Neural Network over a vector representation of the graphs (e.g stacking all the features of the nodes/edges together indiscriminately) and a fully connected network?

# 2 Deep Approximate Inference

**Problem A [3 points]: Variational AutoEncoder**

Use PyTorch to implement the variational autoencoder (VAE) and learn a probabilistic model of the MNIST dataset of handwritten digits. Formally, given a sequence of binary pixels $\mathbf{x} \in \{0,1\}^d$, and let $\mathbf{z} \in \mathbb{R}^k$ denote a set of latent variables. Our goal is to learn a latent variable model $p_\theta(\mathbf{x})$ of the high-dimensional data distribution $p_{\text{data}}(\mathbf{x})$. From variational inference, we learned to maximize the lower bound of the marginal log-likelihood to obtain an expression known as the evidence lower bound (ELBO):

$$\log p_\theta(\mathbf{x}) \geq \text{ELBO}(\mathbf{x}; \theta, \phi) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

**i. [1 points]:** Implement the reparameterization trick in the function `sample_gaussian` of `utils.py`. Specifically, your answer will take in the mean $\mathbf{m}$ and variance $\mathbf{v}$ of the Gaussian distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and return a sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Implement `negative_elbo_bound` in the file `vae.py`, as PyTorch optimizers *minimize* the loss function. Additionally, since we are computing the negative ELBO over a mini-batch of data $\{\mathbf{x}^{(i)}\}_{i=1}^n$, make sure to compute the *average* $-\frac{1}{n}\sum_{i=1}^n \text{ELBO}(\mathbf{x}^{(i)}; \theta, \phi)$ over the mini-batch. Note that the ELBO itself cannot be computed exactly, we ask that you estimate the reconstruction term via Monte Carlo sampling

$$-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \approx -\log p_\theta(\mathbf{x}|\mathbf{z}^{(1)})$$

where $\mathbf{z}^{(1)} \sim q_\theta(\mathbf{z}|\mathbf{x})$ denotes a single sample. The function `kl_normal` in `utils.py` will be helpful for this question.

**ii. [1 points]:** Run `python vae.py` to train the VAE. Once the run is complete (20000 iterations), it will output (assuming your implementation is correct): the average (1) negative ELBO, (2) KL term, and (3) reconstruction loss as evaluated on a test subset that we have selected. Report the three numbers you obtain as part of the write-up. Since we're using stochastic optimization, you may wish to run the model multiple times and report each metric's mean and corresponding standard error. (Hint: the negative ELBO on the test subset should be somewhere around 100.)

**iii. [1 points]:** Visualize 200 digits (generate a single image tiled in a grid of $10 \times 20$ digits) sampled from $p_\theta(\mathbf{x})$.

**Problem B [3 points]: Gaussian Mixture VAE**

Recall that in Problem A, the VAE's prior distribution was a parameter-free isotropic Gaussian $p(Vz) = N(\mathbf{z}|0, I)$. While this original setup works well, there are settings in which we desire more expressivity to better model our data. In this problem we will implement the GMVAE, which has a mixture of Gaussians as the prior distribution. Specifically:

$$p_\theta(\mathbf{z}) = \sum_{i=1}^{k} \frac{1}{k} \mathcal{N}(\mathbf{z}|\mu_i, \mathrm{diag}(\sigma_i^2))$$

where $i \in \{1, \cdots, k\}$ denotes the $i$th cluster index. For notational simplicity, we shall subsume our mixture of Guassian parameters ${\mu_i, \sigma_i}_{i=1}^{k}$ into our generative model parameters $\theta$. For simplicity, we have also assumed fixed uniform weights 1/k over the possible different clusters. Apart from the prior, the GMVAE shares an identical setup as the VAE.

**i. [1 points]:** Implement (1) `log_normal` and (2) `log_normal_mixture` functions in `utils.py`, and the function `negative_elbo_bound` in `gmvae.py`. The function `log_mean_exp` in `utils.py` will be helpful for this problem.

**ii. [1 points]:** Run `python run_gmvae.py` to train the GMVAE. Once the run is complete (20000 iterations), it will output: the average (1) negative ELBO, (2) KL term, and (3) reconstruction loss as evaluated on a test subset. Report the three numbers you obtain as part of the write-up. Report the three metrics you obtain as part of the write-up. Since we're using stochastic optimization, you may wish to run the model multiple times and report each metric's mean and the corresponding standard error.

**iii. [1 points]:** Visualize 200 digits (generate a single image tiled in a grid of $10 \times 20$ digits) sampled from $p_\theta(x)$.

# References

[1] Gilmer, Justin, et al. "Neural message passing for quantum chemistry." *Proceedings of the 34th International Conference on Machine Learning*-Volume 70. JMLR. org, 2017.