Exercises
Deep Learning
Fall 2018

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Fernando Pérez-Cruz
Web http://www.da.inf.ethz.ch/teaching/2018/DeepLearning/

Series Monday, Dec 10, 2018
(Deep Learning, Exercise series 10)

Problem 1 (Evidence Lower Bound):

In the lecture you have seen the following ELBO (or a variant thereof) derived via Jensen's inequality:

$$\mathcal{L} := \log p_\theta(\mathbf{x}) \geq \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z} - \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} d\mathbf{z} =: \mathcal{F} \tag{1}$$

Our goal now is to characterize how tight the above bound is, i.e. we want to derive an expression for the difference between the log-loss $\mathcal{L}$ and the ELBO $\mathcal{F}$. Starting from the expression for $\mathcal{L}$ and inserting $1 = \int q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z}$ (sometimes called partition of unity), show that

$$\mathcal{L} = \mathcal{F} + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) || \, p_\theta(\mathbf{z}|\mathbf{x})) \tag{2}$$

where D_{KL} denotes the KL-divergence. You may assume that all the expressions involved are well-defined.

Problem 2 (Variational Autoencoders):

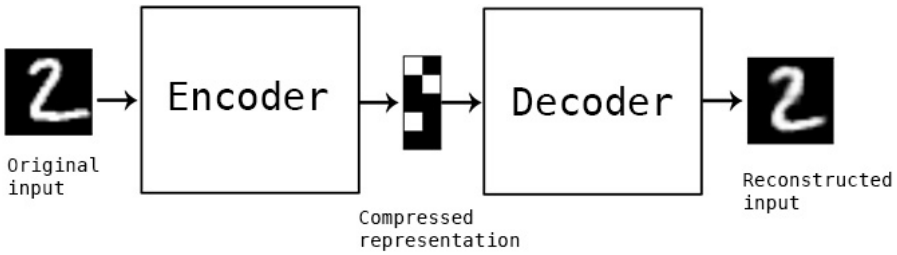
Read the original VAE publications (you can click on the embedded hyperlinks in the pdf) and answer the following questions:

- Kingma D. P., and Welling M. "Auto-encoding variational bayes." arXiv:1312.6114 (2013).
- Rezende D. J., Mohamed S., and Wierstra D. "Stochastic backpropagation and approximate inference in deep generative models." arXiv:1401.4082 (2014).

(i) Elaborate on the difference between variational and classical autoencoders. (ii) Explain the reparameterization trick, in particular explain why it is essential for backpropagation. (iii) What happens if the dimension of the latent space is larger than the dimension of the input space? (iv) In the image domain, do you think L_2 loss function is a "good" or "bad" proxy for visual similarity between original and reconstructed images?

Problem 3 (Practical: Autoencoders in Tensorflow):

In this exercise you will be implementing an Autoencoder on the MNIST dataset. You will be given a template that you need to fill and adapt (See the jupyter notebook:*mnist-autoencoder-template.ipynb* or the python script:*mnist-autoencoder-template.py*). Our Autoencoder comprises a compression (encoder) function and a decompression (decoder) function (See Fig. below). Our encoding/decoding functions are based on CNNs and their parameters are going to be optimized to minimize the reconstruction loss using Stochastic Gradient Descent.



More formally, an autoencoder takes an input $x \in \mathbb{Z}^d$ and maps it through an encoder to a hidden representation $y \in \mathbb{Z}^{d'}$ in the form: $y = s(Wx + b)$, where s is a non-linear function such as a *tanh*, or a *sigmoid*. The variable W is the weight matrix and b is the bias.

The latent representation y is then mapped back through a decoder into the reconstruction z of x. The mapping through the decoder follows a similiar transformation: $z = s(W^{'}y + b^{'})$. The model parameters: $W, W^{'}, b, b^{'}$ are optimized such that the average reconstruction error is minimized. There are different types of reconstruction errors, such as the *squared error loss:* $L_{SE}(x, z) = ||x - z||^2$ or the *cross-entropy loss:* $L_H(x, z) := -\sum_{k=1}^{d}[x_k \log(z_k) + (1 - x_k) \log(1 - z_k)]$.

Your tasks are:

1. Create an encoder and a decoder of 2 hidden layers, i.e. $y = s\big(W^2 \cdot s(W^1 x + b^1) + b^2\big)$ and $z = s\big(W^{2^{'}} \cdot s(W^{1^{'}} y + b^{1^{'}}) + b^{2^{'}}\big)$. In the template, you will have to fill the functions: *encoder* and *decoder*. The first hidden-layer will have 256 neurons and the second 128 neurons. You can use as activation function a *sigmoid*. (Hint: Use the *tf.nn.sigmoid()* function).

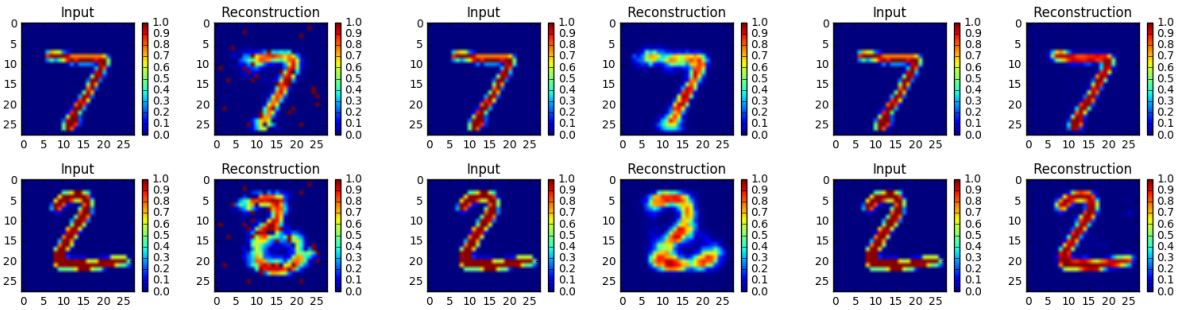
2. Initialize the weights of your weight matrices $(W^2, W^1, W^{2^{'}}, W^{1^{'}})$ and biases $(b^2, b^1, b^{2^{'}}, b^{1^{'}})$ by sampling random values from a normal distribution. (Hint: Use the *tf.random_normal()* function).

3. Implement a *mean_squared* loss function and train your Autoencoder. You can use the Adam Optimizer. By using the plot function given at the end of the template, display some test images with their reconstruction. (Hint: Use the *tf.train.AdamOptimizer* and set the learning rate to 0.001). After 20 training iterations, your results shall be similar to the two left-most columns in the figure below.

4. Implement the *cross-entropy* loss function and replace the *mean_squared* loss function. Train again your Autoencoder. Display some reconstructed images. The reconstruced images shall be similar to the ones in the middle of the figure below.

5. Implement the Xavier function for initializing the Weight matrices $\left(W^2, W^1, W^{2^{'}}, W^{1^{'}}\right)$. Use the *cross-entropy* loss function and train your Autoencoder. Visualize some of the reconstructed images, they should be similar to the two right-most columns of the figure below.

6. Going deeper: add an additional hidden layer to your encoder/decoder. You can set the number of neurons to be 64.



mean-squared error cross-entropy error Xavier initialization