Exercises
Computational Intelligence Lab
SS 2020

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Prof. Dr. Thomas Hofmann
Web http://da.inf.ethz.ch/cil

Series 9, May 7-8, 2020 (Generative models)

Problem 1 (Variational lower bound for generative models):

This exercise follows the main concepts explained by Kingma D. P., and Welling M. in "Auto-encoding variational bayes." (2013), read this paper for further details.
Consider the following general setting for a generative model with continuous latent variables: We have available a training set $\mathcal{X} := \{\mathbf{x}^{(i)}\}_{i=1}^N$ that is assumed to be generated from a hidden code $\mathbf{z} \in \mathbb{R}^J$. Each training sample is generated by first sampling the latent variable $\mathbf{z}^{(i)}$ from the true prior $p_{\theta^*}(\mathbf{z})$ and then drawing $\mathbf{x}^{(i)}$ from $p_{\theta^*}(\mathbf{x}|\mathbf{z}^{(i)})$, which could be a complex distribution. A relevant task is to estimate the optimal parameters θ^* from the training data. A well known strategy is to maximize the complete-data log-likelihood which is given by

$$\log p_\theta(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}) = \sum_{i=1}^N \log p_\theta(\mathbf{x}^{(i)})$$

1. Show that the likelihood for a single data-point $\log p_\theta(\mathbf{x}^{(i)})$ is given by

$$E_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x}^{(i)})}[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})] - D_{\mathsf{KL}}(q_\Phi(\mathbf{z}|\mathbf{x}^{(i)}) \,||\, p_\theta(\mathbf{z})) + D_{\mathsf{KL}}(q_\Phi(\mathbf{z}|\mathbf{x}^{(i)}) \,||\, p_\theta(\mathbf{z}|\mathbf{x}^{(i)})).$$

   where $q_\Phi(\mathbf{z}|\mathbf{x}^{(i)})$ is the variational approximation to the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$. *Hint: Use Bayes rule and $E_X[f(Y)] = f(Y)$ if the random variable Y does not depend on X.*

2. Group this expression into two terms, and identify the variational lower bound (ELBO) $\mathcal{L}(\theta, \Phi; \mathbf{x}^{(i)})$.

3. In the ELBO which of the 2 terms quantifies reconstruction quality and which one acts as a regularizer?

4. Recall that the ELBO can also be written as

$$\mathbb{E}_{\mathbf{z}\sim q_\phi(z|\mathbf{x}^{(i)})}[-\log q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) + \log p_\theta(\mathbf{x}^{(i)}, \mathbf{z})].$$

   The goal is to maximize this lower bound w.r.t. the parameters θ and ϕ using Monte Carlo sampling to estimate the expectation. However sampling z can be impractical for our purpose, the reparameterization trick assigns

$$\mathbf{z} = g_\phi(\boldsymbol{\epsilon}, \mathbf{x}) \quad \text{with} \quad \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}).$$

   Write down the Monte Carlo estimate of the lower bound using this reparameterization trick. Why is this trick essential for backpropagation ?

5. Let's consider a univariate Gaussian distribution, assume $z \sim p(z \mid x) = \mathcal{N}(\mu, \sigma^2)$. We want to use the reparameterization $z = g_\phi(\epsilon, x)$ with $\epsilon \sim \mathcal{N}(0, 1)$. What is g in this case ?

6. Briefly explain how in the VAE model the log-likelihood could be maximized in practice and summarize the main differences with a classical autoencoder.

7. Show that for the special case that $p_\theta(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$ is isotropic Gaussian (J dimensions), and $q_\Phi(\mathbf{z}|\mathbf{x}^{(i)})$ is multivariate Gaussian with a diagonal covariance matrix, the $-D_{\mathsf{KL}}$ term in the above expression analytically integrates to

$$\frac{1}{2}\sum_{j=1}^J \left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right).$$

Problem 2 (Variational Autoencoder):

In this exercise, you will build a variational autoencoder and compare it to a "traditional" autoencoder. For both experiments, we will encode the digits of the MNIST dataset in a low-dimensional space and look at its structure.

We have provided a notebook `exercise09.ipynb` with the code template, which includes the model architecture. It is a very simple convolutional architecture for the encoder and a deconvolutional architecture for the decoder. Follow the instructions below and answer the questions:

1. Non-linear autoencoder: write the training code for a simple autoencoder, using a binary log-likelihood loss. You should encode the MNIST training digits into a 2-dimensional vector and reconstruct them. We suggest using the Adam optimizer and a minibatch size of 256.
   Hint: for consistency with the ELBO, you should at least sum the loss across pixels, but we also suggest averaging it across the batch dimension (instead of summing again) to make it independent of the batch size and learning rate.
2. Encode all digits of the MNIST test set and visualize the latent space with a scatter plot. What does it look like?
3. Variational autoencoder: let us now enforce an isotropic Gaussian prior on the latent code. How do we need to modify the encoder? Do we also need to modify the decoder?
   Info: to avoid dealing with positivity constraints, we parameterize σ^2 as $\log \sigma^2$ (which is defined across the entire real domain) and take its exponential to retrieve σ^2.
4. Write the code for the reparameterization trick. Why do we restrict ourselves to a diagonal Gaussian, as opposed to a full covariance matrix?
5. Add the KL term to the previous loss function, and train the model as before.
6. Encode the test set and look again at the latent space. What does it look like? What are the differences with respect to the previous experiment?
7. Generation: how do you sample from this VAE? Try to sample some digits and visualize them.

Problem 3 (Generative Adversarial Networks):

In this exercise, you will implement a *deep convolutional GAN* (DCGAN) for generating handwritten digits using the MNIST dataset. A DCGAN is a convolutional architecture that relies on the GAN framework for training. The discriminator is a standard convolutional neural network trained for binary classification (real/fake), and the generator is a deconvolutional network (i.e. uses transposed convolutions to progressively upsample the output image).

Follow the instructions in the TensorFlow tutorial:

https://github.com/tensorflow/tensorflow/blob/r1.13/tensorflow/contrib/eager/python/examples/generative_examples/dcgan.ipynb

Problem 4 (Road Extraction from Satellite Images):

For the third choice of semester project task, we provide a set of satellite/aerial images acquired from GoogleMaps. We also provide ground-truth images where each pixel is labeled as {road, background}. Your goal is to train a classifier to segment roads in these images, i.e. assign a label {road=1, background=0} to each pixel.

1. Download the training data from the competition website

   inclass.kaggle.com/c/cil-road-segmentation-2020

2. Obtain the python notebook `segment_aerial_images.ipynb` from

   github.com/dalab/lecture_cil_public/tree/master/exercises/2020/ex09

   to see example code on how to extract the images as well as pixel labels.

   The notebook shows how to use `scikit learn` to generate features from each pixel, and finally train a linear classifier to predict whether each pixel is road or background. It also provides helper functions to visualize the images, labels and predictions.

3. As a more advanced approach, try `tf_aerial_images.py`, which demonstrates the use of a convolutional neural network in TensorFlow for the same prediction task.

Problem 5 (Galaxy image classification):

For the fourth choice of semester project task, we provide a set of cosmology images observed by astronomical telescopes, corrupted cosmology images and images with other content. To get you started in exploring the data, we will consider here the problem of discriminating real cosmology images from the other two groups.

1. Download the supplementary data from the competition website.

   inclass.kaggle.com/c/cil-cosmology-2020

2. Read documentation on the supplementary data as needed for this task.
3. Obtain the Python script `features_cosmology_project.py` from

   github.com/dalab/lecture_cil_public/tree/master/exercises/2020/ex9

   which provides example code for loading the images, extracting basic features from them, and a skeleton for evaluating machine learning models.
4. Get familiar with the cosmology data by solving a simple binary classification problem: Each image should be assigned a label from the set $\{\text{real cosmology} = 1, \text{non real cosmology} = 0\}$. For this, divide the images in the folder `labeled` into training and test set. Implement hand-crafted image features + baseline classifiers with `Scikitlearn`, or learn image features automatically with a deep learning approach using `Tensorflow`. Evaluate and contrast your models' predictive performance and get a feeling for which features are informative.