Uppsala University
Department of Information Technology
Division of Systems and Control
DW/NW 2019-06
Last rev. October 11, 2021 by DW

Advanced Probabilistic Machine Learning

Instruction to the laboratory work

Unsupervised Learning with Variational Autoencoders

Language: Python

Preparation:
Solve all preparatory exercises in Section 3

Reading instructions:

- Laboratory instructions: Section 1-3.
- Kingma, D. P., & Welling, M. (2019). *An introduction to variational autoencoders*. arXiv: 1906.02691 `[cs.LG]`.

Name		Assistant's comments
Program	Year of reg.	
Date		
Passed prep. ex.	Sign	
Passed lab.	Sign	

Contents

1 Introduction

Unsupervised learning tries to find unknown patterns in unlabeled data. In the lecture we have seen principal component analysis (PCA), probabilistic principal component analysis (PPCA), and variational autoencoders as three examples of unsupervised learning. In this lab we will try to find patterns in images of handwritten digits and train a model that can generate similarly looking images.

This laboratory work is based on lectures 10 to 13 and exercise sessions 10 and 11 about variational inference, unsupervised learning, and variational autoencoders. Therefore it is advisable to have the material from those lectures and exercise sessions fresh in mind before starting this laboratory work.

The goal of this laboratory work is to:

- Learn how to train a probabilistic PCA model.
- Learn how to train a variational autoencoder.
- Get a glimpse of a state-of-the-art machine learning framework.

Throughout the lab we will us a software library called *PyTorch*. This library is introduced in Section 2. Section 3 contains the preparatory exercises, and Section 4 contains the instructions for the lab session.

Important: Read Section 2 and answer the preparatory exercises in Section 3 *before* the lab session.

2 PyTorch

PyTorch is an open source software library for machine learning that is based on the machine learning library Torch which is no longer actively developed. PyTorch is developed primarily by Facebook's artificial intelligence research group and used by companies as well as academic research groups. It can be used for general computations with multidimensional arrays on CPUs and GPUs, but it is tailored especially to deep learning and neural networks.

PyTorch is natively written in Python and C++, and well documented APIs exist for these languages. It is not the only deep learning framework, some other state-of-the-art alternatives are TensorFlow and MXNet.

2.1 Installation

If you use your own computer for the computer lab, you need to have PyTorch properly installed *before* the lab[1]. The lab assistants will not be able to assist you with the installation process during the lab. Please consult the PyTorch documentation for more information about the installation procedure.

Another option is to use Google Colab to work with PyTorch online. This cloud platform also optionally provides access to GPUs which might speed up some computations.

On the Linux systems in the computer rooms where the laboratory session is scheduled only an old unsupported version of PyTorch is installed. However, you can use these computers to access and work with Google Colab.

2.2 Comparison with TensorFlow

As PyTorch, TensorFlow contains support for GPUs and builds a computational graph that is used for computing gradients via backpropagation. However, Tensorflow builds a static computational graph whereas PyTorch works with a dynamic computational graph. In TensorFlow the graph is created once and then executed multiple times, but in PyTorch every forward pass defines a new computational graph. Thus in TensorFlow the computational graph can be optimized before running any actual computations. Even if this optimization might be computationally expensive, it can pay off if the graph is run multiple times. On the other hand, an advantage of using dynamic graphs is the increased flexibility with respect to the control flow that they allow. Since the computational graph is rebuilt in each forward pass, one can use regular `if` statements and `for` loops (even without fixed numbers of iterations) and easily change the computations in different iterations of the training loop.

[1] The lab is written for the most recent PyTorch version, therefore make sure to install the latest version.

2.3 Introduction

A Jupyter notebook with an introduction to PyTorch can be downloaded from here. Alternatively, you can open it on Google Colab. Reading and running the notebook is highly recommended, since it introduces important concepts and commands that are required in the lab session.

The official PyTorch tutorials and the PyTorch documentation might be helpful additional resources, in particular if you are looking for a more general introduction to PyTorch. However, the introduction to PyTorch in the provided Jupyter notebook covers everything you should know for the lab session.

3 Preparatory exercises

3.1 Probabilistic principal component analysis

In constrast to (regular) PCA, the so-called probabilistic PCA (PPCA) (Tipping & Bishop, 1999) allows a probabilistic interpretation of the principal components. The probabilistic formulation of PCA also allows us to extend the method and alter its underlying assumptions quite easily, as we will learn during the course of this laboratory.

Let $\mathbf{x} \in \mathbb{R}^D$ represent a data sample that we want to decode from a lower dimensional representation $\mathbf{z} \in \mathbb{R}^M$ with $M < D$. The PPCA model assumes that $\mathbf{z}$ is standard normally distributed and $\mathbf{x}$ can be decoded by a noisy linear transformation of $\mathbf{z}$. Mathematically, the model is given by

$$\begin{aligned} p(\mathbf{x} \,|\, \mathbf{z}) &= \mathcal{N}\left(\mathbf{x}; \mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_D\right), \\ p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}_M), \end{aligned}$$

with parameters $\mathbf{W} \in \mathbb{R}^{D \times M}$, $\boldsymbol{\mu} \in \mathbb{R}^D$, and $\sigma^2 > 0$. Tipping and Bishop (1999) showed that for $\sigma^2 \to 0$ the model recovers the standard PCA (but the components of $\mathbf{z}$ might be permuted).

We assume that the data $\mathbf{x}_1, \ldots, \mathbf{x}_N$ is identically and independently distributed according to the PPCA model. In a maximum likelihood setting, one determines the parameters $\mathbf{W}$, $\boldsymbol{\mu}$, and σ^2 that maximize the likelihood

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N; \mathbf{W}, \boldsymbol{\mu}, \sigma^2) = \prod_{n=1}^{N} p(\mathbf{x}_n; \mathbf{W}, \boldsymbol{\mu}, \sigma^2),$$

or equivalently the log-likelihood

$$\log p(\mathbf{x}_1, \ldots, \mathbf{x}_N; \mathbf{W}, \boldsymbol{\mu}, \sigma^2) = \sum_{n=1}^{N} \log p(\mathbf{x}_n; \mathbf{W}, \boldsymbol{\mu}, \sigma^2).$$

Question 3.1: *Show that for the model of the probabilistic PCA*

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{C}),$$

where $\mathbf{C} = \mathbf{W}\mathbf{W}^\mathsf{T} + \sigma^2 \mathbf{I}_D$ *(cf. exercise 11.2a).*

Tip 3.1 Use Corollary 2 in lecture 2. ∘

Answer:

The PPCA is defined as a model for how to decode a latent vector $\mathbf{z}$ to the observed data $\mathbf{x}$. Of course, as in the regular PCA, we would like to encode data $\mathbf{x}$ in the lower dimensional latent space as well.

Question 3.2 (cf. exercise 11.2b) 3.2: *Show that the distribution of the latent variable* $\mathbf{z}$ *conditioned on* $\mathbf{x}$ *is Gaussian as well and given by*

$$p(\mathbf{z} \,|\, \mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \mathbf{M}^{-1}\mathbf{W}^{\mathsf{T}}(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M}^{-1}\right),$$

where $\mathbf{M} = \mathbf{W}^{\mathsf{T}}\mathbf{W} + \sigma^2\mathbf{I}_M$ *(cf. exercise 11.2b).*[2]

Tip 3.2 Use Corollary 1 in lecture 2. ∘

[2]Note that eq. (12.42) in Bishop's book is incorrect. In the original paper by Tipping and Bishop (1999) the correct distribution is stated.

Answer:

3.2 Variational autoencoder

Let us consider a more flexible nonlinear model that is given by

$$p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_{\boldsymbol{\theta}}(\mathbf{z}), \sigma^2_{\boldsymbol{\theta}} \mathbf{I}_D\right),$$
$$p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}_M),$$

where parameters $\boldsymbol{\theta}$ include variance parameter $\sigma^2_{\boldsymbol{\theta}} > 0$ and $\mu_{\boldsymbol{\theta}} \colon \mathbb{R}^M \to \mathbb{R}^D$ is a nonlinear model of the mean of the normal distribution. In this lab we will model $\mu_{\boldsymbol{\theta}}$ by a neural network and include its weights and biases in $\boldsymbol{\theta}$, but other models could be used equally well.

In the lab session we will discuss different training approaches for this model. One approach involves defining an encoding distribution $q_{\boldsymbol{\phi}}(\mathbf{z}; \mathbf{x})$ that may depend on

$\mathbf{x}$ and some parameters $\boldsymbol{\phi}$, and estimating and maximizing the so-called evidence lower bound (ELBO)

$$\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z};\mathbf{x})}\big[\log p_{\boldsymbol{\theta}}(\mathbf{x}\,|\,\mathbf{z})\big] - \mathrm{KL}\big(q_{\boldsymbol{\phi}}(\mathbf{z};\mathbf{x}) \,\big\|\, p_{\boldsymbol{\theta}}(\mathbf{z})\big).$$

A common choice is to define q as a multivariate normal distribution with diagonal covariance matrix, i.e., to set

$$q_{\boldsymbol{\phi}}(\mathbf{z};\mathbf{x}) = \mathcal{N}(\mathbf{z};\mu_{\boldsymbol{\phi}}(\mathbf{x}),\mathrm{diag}(\sigma^2_{\boldsymbol{\phi}}(\mathbf{x}))), \tag{1}$$

where $\mu_{\boldsymbol{\phi}}\colon \mathbb{R}^D \to \mathbb{R}^M$ defines the mean of the normal distribution and $\sigma^2_{\boldsymbol{\phi}}\colon \mathbb{R}^D \to \mathbb{R}^M_{\geq 0}$ the diagonal entries of the covariance matrix. We will model these functions with a neural network, but, of course, other choices are possible as well. For this particular choice of the encoding distribution q a closed-form expression of the KL divergence term exists.

Question 3.3: *Let $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z};\mathbf{0},\mathbf{I}_M)$ as in the nonlinear model above and choose the distribution $q_{\boldsymbol{\phi}}(\mathbf{z};\mathbf{x})$ in eq.* (1) *for the encoder.*

Show that

$$\begin{aligned}&\mathrm{KL}\big(q_{\boldsymbol{\phi}}(\mathbf{z};\mathbf{x}) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{z})\big)\\ &\qquad = \frac{1}{2}\Bigg(\sum_{i=1}^{M}\big(\sigma^2_{\boldsymbol{\phi}}(\mathbf{x})\big)_i + \big\|\mu_{\boldsymbol{\phi}}(\mathbf{x})\big\|_2^2 - M - \sum_{i=1}^{M}\log\big(\sigma^2_{\boldsymbol{\phi}}(\mathbf{x})\big)_i\Bigg)\end{aligned}$$

(cf. exercise 10.1).

Answer:

4 Laboratory session

This section contains instructions for the laboratory session. The main lab exercise is to build and train models that generate images that look like hand-written digits. The generation will be made possible by unsupervised learning of the most important features of such images from a data set presented in Section 4.1.

4.1 MNIST data set

We consider the so-called MNIST data set, which is one of the most well-studied data sets within machine learning and image processing.

The data set consists of 60 000 training data points and 10 000 test data points. Each data point consists of a grayscale image with 28×28 pixels of a hand-written digit. The digit has been size-normalized and centered within a fixed-sized image. Each image is also labeled with the digit (0, 1, . . . , 8, or 9) it is depicting. In Figure 1 a batch of 25 data points from this data set is displayed.

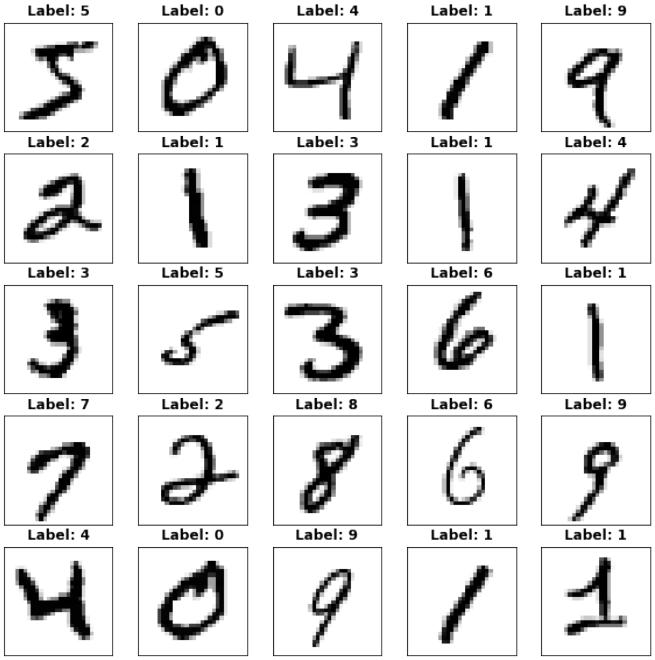

Figure 1: Samples from the MNIST data set.

We consider each image as a vector $\mathbf{x} = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^\mathsf{T}$, where each x_j corresponds to one of the $p = 28 \times 28 = 784$ pixels in the image. The value of each x_j represents the color of that pixel. The color value is within the interval $[0, 1]$, where $x_j = 0$ corresponds to a black pixel and $x_j = 1$ to a white pixel. Anything between 0 and 1 is a gray pixel with corresponding intensity.

Based on a set of training data $\{\mathbf{x}_i\}_{i=1}^N$ with images, the problem is to find a good model that allows us to sample from the distribution

$$p(\mathbf{x})$$

of the images. Unfortunately, it is unclear what the best way is to model this distribution of 784-dimensional vectors, since probably most of these vectors do not even represent images of hand-written digits.

We approach the problem by trying to extract all important features of the images and compress them to a low dimensional space, in such a way that we can reconstruct the images reasonably well. Instead of generating new images by sampling 784-dimensional vectors, we then try to sample a new set of low dimensional image features and to construct a proper image from it.

4.2 Principal component analysis

One of the most common tools for dimensionality reduction is PCA. In exercise 11.1 we performed a PCA on the MNIST data.

Task 4.1 Go back to exercise 11.1 and solve it if you have not done so yet. ◦

4.3 Probabilistic principal component analysis

The PCA performs dimensionality reduction and hence allows us to condense the high dimensional images to a two-dimensional representation. However, the PCA does not provide us with a generative model of the images and hence does not allow us to sample new MNIST-like images from the data distribution. Therefore we move to a probabilistic setting with a probabilistic description of the encodings and the mapping from the latent space to the space of images.

Task 4.2 Download the Jupyter notebook PPCA.ipynb and open it. Alternatively, you can open the notebook on Google Colab.

Work through the notebook. You can write down your answers to the questions in the notebook in the box below. ◦

Answer:

4.4 Variational autoencoder

The linearity assumptions of the probabilistic PCA model limit the quality of the generated images and intuitively are a bit questionable. Therefore we add a nonlinear mapping to our model.

Task 4.3 Download the Jupyter notebook VAE.ipynb and open it. Alternatively, you can open the notebook on Google Colab.

Work through the notebook. You can write down your answers to the questions in the notebook in the box below. ○

Answer:

4.5 Deep hierarchical variational autoencoder

The variational autoencoder (VAE) that we trained in the lab allows us to generate MNIST-like images. The quality of the images is clearly better than the quality of the samples from the probabilistic PCA model. However, it is still a quite simple model which, of course, also limits the quality of the samples.

The so-called nouveau variational autoencoder (NVAE) (Vahdat & Kautz, 2021) is a state-of-the-art VAE with an advanced hierarchical structure of the encoder and decoder. The authors applied it to multiple image data sets such as the MNIST data set and the CelebA (Liu et al., 2015) and CelebA HQ (Karras et al., 2018) data sets of celebrity images. Figure 2 shows samples of the NVAE for the CelebA HQ data set.

Figure 2: Samples from the NVAE (Vahdat & Kautz, 2021).

Task 4.4 Download the Jupyter notebook NVAE.ipynb and open it. Alternatively, you can open the notebook on Google Colab. We recommend that you run the notebook on Google Colab since it allows you to use GPUs which will speed up the computations a lot (you can enable GPU support in the "Runtime" $\rightarrow$ "Change runtime type" menu).

Run the notebook and play around with it. Try some of the suggestions in the task at the end of the notebook. ○

References

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2018). Progressive growing of gans for improved quality, stability, and variation.

Kingma, D. P., & Welling, M. (2019). *An introduction to variational autoencoders*. arXiv: 1906.02691 [cs.LG].

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep learning face attributes in the wild. *Proceedings of International Conference on Computer Vision (ICCV)*.

Tipping, M. E., & Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *61*(3), 611–622.

Vahdat, A., & Kautz, J. (2021). Nvae: A deep hierarchical variational autoencoder.