Homework 5 – Image Generation

I) warmup: Autoencoders

Autoencoders have a long tradition in machine learning. Here, we look at simple ones based on fully-connected networks.

I.1) linear autoencoder

For input vectors X and a given dimensionality k, implement a network consisting of a linear *encoder*

- flatten the input from 28x28 images to vectors of length 784
- 1 fully connected layer with 784 inputs and k outputs (no non-linearity)

that turns an input into a short code, and a linear *decoder* that from the code tries to reconstruct the input:

- 1 fully connected layer with 784 outputs (no non-linearity)
- reshape the image into images of size 28x28

I.2) training data

Use the MNIST data (download from **`https://cvml.ist.ac.at/courses/DLWT_W18/data/mnist.py`** or reuse from exercise 6 of homework 3). Scale the data to the range [0,1]. Use 90% for training and the rest for validation.

I.3) training

For $k = 2, 5, 10, 50, 100$ train the network using *mean squared loss*: $\ell(x, y) = \frac{1}{784} \sum_i (x[i] - y[i])^2$

I.4) visualization

Visualize the results by the following experiments:

- take 10 training image, encode them, decode them again and show the resulting images side-by-side
- take 10 validation image, encode them, decode them again and show the resulting images side-by-side
- for 10 validation and training images (can be the same as above):
  - compute their codes
  - interpolate linearly in 10 steps between the code of the training image and the code of the validation image (they might show different digits, that's okay)
  - decode the resulting codes and visualize the transition as an image sequence
- create 10 random codes by sampling randomly from a standard Gaussian, decode the codes and show the resulting images

What do you observe?

I.5) (optional) Principal Component Analysis (PCA)

The above autoencoder is in fact equivalent to just doing PCA to the given number of dimensions. Can you prove this?

I.6) (optional) non-linear autoencoders

Modify the example in 1) to have a non-linear activation of your choice after the encoding layer (e.g. ReLU), and a sigmoid after the decoder.

Repeat the training, now with *cross-entropy loss* (this is possible, as both inputs and outputs have [0,1] range).

Do the results improve compared to the linear case? Why?

I.7) (optional) stacked autoencoders

build a stacked autoencoder: instead of just one encoding and one decoding layer, use two or more of varying size. Try to get a better reconstruction than in the single layer situation.

Stacked autoencoders were considered very hard to train for a long time (leading to innovations such as per-layer pre-training [`http://science.sciencemag.org/content/313/5786/504`]). Do you observe the same?

II) Variational Autoencoders (VAEs)

We will make a few changes to the above architecture and procedure:

a) we'll estimate not just a latent code, but a *distribution over codes*
b) we'll train not (just) for reconstruction error, but using a variational objective

II.1) VAE architecture

To keep things simple, we'll parameterize the distribution over codes as an isotropic Gaussian with mean $\mu \in \mathbb{R}^k$ and diagonal covariance matrix with entries $\sigma^2 \in \mathbb{R}^k$. The encoder will consist of predicting μ and $s = \log \sigma^2$ from the input (the log is for numeric reasons). For his, build a network based on the following schematics:

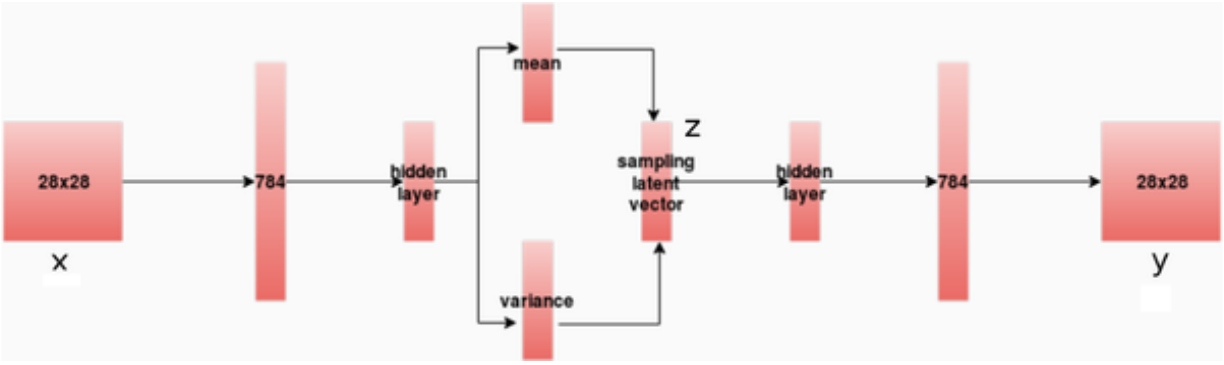

Figure 1: VAE architecture (from `https://github.com/chaitanya100100/VAE-for-Image-Generation`)

- for the 1st *hidden layer*, use 512 outputs and a nonlinearity of your choice
- for the *mean* (μ) and (log)*variance*(s), use k outputs each with no nonlinearities
- the *sampling latent vector* is computed as $z = \mu + \exp(s/2) * \epsilon$, where ϵ is a `tf.random_normal` of appropriate size
- for the 2nd *hidden layer*, use a *sigmoid* nonlinearity

II.2) VAE loss

As loss, compute the following expression, which is derived using the same reasoning as in the lecture, but with a binomial instead of Gaussian distribution for the images:

$$\ell(x, y, \mu, s) = \text{crossentropy}(x, y) + 0.5\Big(\|\mu\|^2 + \sum_i \exp(s[i]) - \sum_i s[i]\Big)$$

II.3) training

Train the VAE for $k = 2, 5, 10, 50, 100$ on the same MNIST data as above.

II.4) visualization

Visualize the results as in 4), using the mode of the predicted distribution (i.e. μ) as the code. What do you observe?

II.5) (optional) back to the autoencoder

Retrain the VAE with only the *crossentropy* part of the loss, i.e. without the second term. What do you observe?

III) (very optional) Generative adversarial networks (GANs)

Check out the git repository `https://github.com/tensorlayer/dcgan`. Read the code to understand what's going on. Try to get the examples to run. Modify some parameters, e.g. learning rates, see how the behaviour (objective, converngence...) changes.

Hand-in requirements

1. upload your code to I.1-3) and II.1-3), and the examples images from I.4) and II.4) to the IST *git* server.