Homework 4: Auto-encoders

Abstract

In this homework, the *Auto-Encoder* architecture is employed in an image recognition task. Different numbers of encoding dimensions are employed, and their effect on both the recognition and the generation of images are probed. An implementation of a *Variational Auto-Encoder* is also used for the same task; differences in performance are then highlighted.

Code development

Auto-encoder

Data preprocessing: the data comes in `.mat` format, as it did in assignment 2. The samples do not require any particular attention in terms of preprocessing, except for the fact that, if one prints them, they will appear to be rotated and flipped. To avoid this inconvenience, a Pytorch Dataset object has been created, with transforms which rotate each sample and convert them to Tensors. As for validating and testing, a function has been created which splits the dataset into three parts, each of them having its own DataLoader, so that a validation set can be used during training in order to avoid overfitting (via an EarlyStopping procedure) and a training set can be used to evaluate the performance of the model. Of note is that the splitting can also return DataSamplers; this object directly refers to the indices in the dataset, so that the same samples can be drawn from different datasets (for instance, datasets with different transformations), and so make comparisons significant.

Architecture: the architecture of the auto-encoder does not differ from the provided one: it employs convolutional layers followed by linear layers for the encoder part, while the reverse happens for the decoder part, where linear layers are followed by transposed convolutional layers. While vanilla auto-encoders are usually purely composed of linear layers, an image oriented task makes convolutional layers the obvious choice.

Performance analysis: in order to get an idea of the effectiveness of the model, two functions were written that:

- find the centroid for a given digit. An auto-encoder creates low-dimensional encodings of the provided images, such that similar samples are clustered in the encoding space. As a result, one can in principle associate a cluster for each of the 10 digits in he MNIST dataset, and each of these clusters to their respective centroid, defined as

$$C_i = \frac{1}{|D_{train}|} \sum_{x \in D_{train}} \phi(x),$$

where D_{train} is the training set and $\phi : \{0, \cdots, 255\}^{28\times 28} \to \mathbb{R}^m$ is the encoding function (provided by the network) which maps all the possible 28×28 images to their m-dimensional encoding. Of course, $i \in \{0, 1, \cdots, 9\}$.

- sample around a centroid. Once a centroid for a given digit has been computed, one can then sample from the encoding space in order to generate new samples. Intuitively, sampling close to the centroid results in samples that resemble the chosen digit. Indeed, the robustness of the encoding can be probed by increasingly straying away from the centroid and then decode and plot the resulting image. This has been done by sampling from a m-variate gaussian distribution with 0 mean and fixed variance σ, adding it to the encoding and then decoding. This allows to probe the robustness of the encoding, as for larger values of σ one expects to observe worse results (that is, digits being confused or complete nonsensical results to appear).

Variational auto-encoder

The Variational Auto-Encoder (VAE) is a modification of the standard Auto-Encoder idea, where latent variables (the encoded values in standard AE) are given the role of parameters of a hidden generative process. Specifically, for VAEs, such hidden generative process is parameterized a multivariate gaussian with diagonal covariance matrix (i.e., latent parameters are uncorrelated). From a practical point of view, this means that the encoder part of a VAE outputs two sets of values, of equal dimension, being the mean μ and the variance σ. The decoder then uses these values to sample a value from the gaussian distribution, a process that is known as "reparameterization trick": if $\mathbf{z} \in \mathbb{R}^m$, where m is the latent space dimension, then, for a VAE, we can write

$$\mathbf{z} = \mu + \epsilon \star \sigma$$

where $\mu, \sigma \in \mathbb{R}^m$, $\epsilon_i \sim \mathcal{N}(0, \mathbf{I}), \quad i = 1, \cdots, m$ and $\star$ denotes an element-wise multiplication.

This architecture requires a specific loss that takes into account both the so-called *reconstruction error*, which can be thought of as the error that arises from the encoding-decoding procedure (in a sense, it is akin to the loss of information that one experiences from compressing and decompressing data), and a regularization term, which comes from the fact that we decided to model the (unknown and intractable) true generative process underlying the data with an uncorrelated multivariate gaussian. The loss itself for input $\mathbf{x^{(i)}}$ is computed as

$$\mathcal{L} = \frac{1}{2}\sum_{j=1}^{m}\left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) + \frac{1}{L}\sum_{l=1}^{L}\log\left(p(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})\right)$$

where the first term on the right hand side is the KL divergence of the approximate prior from the actual one, while the second term is the usual reconstruction error.

Results

Auto-Encoder results

In order to probe the auto-encoder architecture and its effectiveness as a function of the encoding dimension, a loop over different values was implemented. For each of them, the training relied on early stopping. The results are shown in figure 1, comparing different architectures with each other. The plot clearly shows how increasing the number

of encoding features helps the network in creating more effective abstract representations. This should not come as a surprise: the higher the encoding dimension, the lower the amount of information that has to be discarded. Indeed, one may argue that the increase in performance is not the ultimate aim of an auto-encoder. Dimensionality reduction seems a much more fitting task for such an architecture; indeed, some sort of tradeoff between reconstruction capabilities and encoding dimension should be taken into account.

Regardless, the best performing architecture (which proved to be the one with 6 latent variables) was used to perform all subsequent tests. First, the network is run on unmodified test samples, then on occluded samples, and then on noisy samples (with different noise levels). The results are as follows:

- Unmodified: 0.0208
- Occluded: 0.0271
- Noisy (0 mean, 1 variance): 0.0316

Further attention was devoted to the effects of varying noise levels on the performance, which are collected in Fig. 2. Clearly, as noise increases, the network experiences an increase in the loss, which seems exponential.

On a similar note, the generating performance of the model was tested, by sampling around the centroids for each digit with varying degrees of gaussian noise. This simple test shows how, with some noise, the generated samples deviate strongly from the intended digit. However, effects only appear when high enough values of variance are reached: the encoding dimension being 6, the model shows a somewhat high robustness (see Fig. 3).

Variational Auto-Encoder

As for the VAE, the network was mainly tested for its generating capabilities, and was compared to a standard AE of equal latent dimension. In order for the encoding space to be as easy to interpret, it was set to be 2-dimensional. For both the VAE and the AE, a set of points in the encoded space was drawn (see Figs. 4 and 6), and the corresponding images plotted. The results are shown in Figs. 5 and 7.

The standard AE seems to encode points in a more sparse way, with encodings covering a larger area than the VAE counterpart. The VAE shows its superiority in the regions between the clusters: here, the AE produces gibberish results, while the VAE produces a much smoother transition. This is even truer if one looks at the regions that the VAE samples from, since it crosses more clusters than the AE.

Comments

The AE architecture performs as expected; for a high enough latent dimension (in this case, 6), the AE also shows appreciable robustness to data corruption. The VAE architecture offers an improvement in the generation task; however, the vanilla AE is also valid. Additional time may be devoted to training the network with augmented data, including corrupted samples.

The end results are satisfying, confirming the expectations. I found this assignment to be very interesting, particularly for what concerns the generating capabilities of neural networks.

Plots

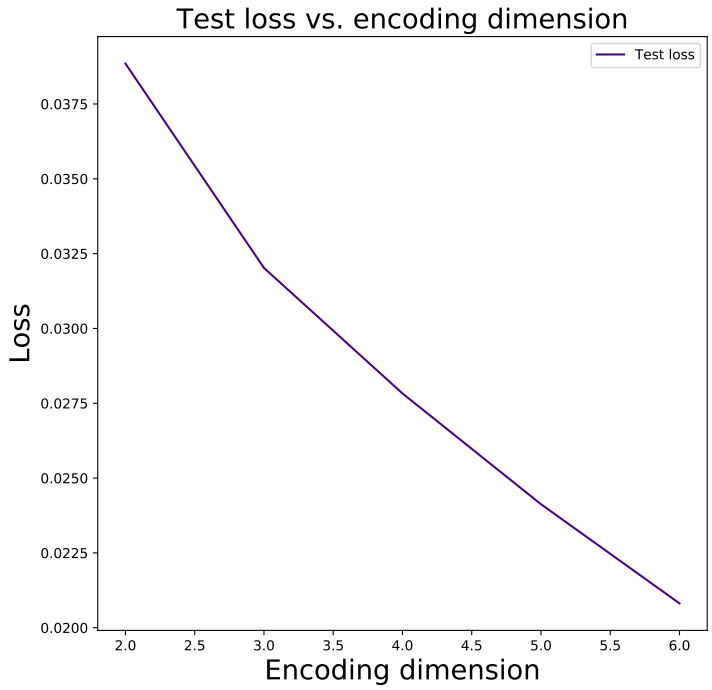


Figure 1: Behavior of the test loss as a function of the number of latent variables in the auto-encoder (ranging from 2 to 6).

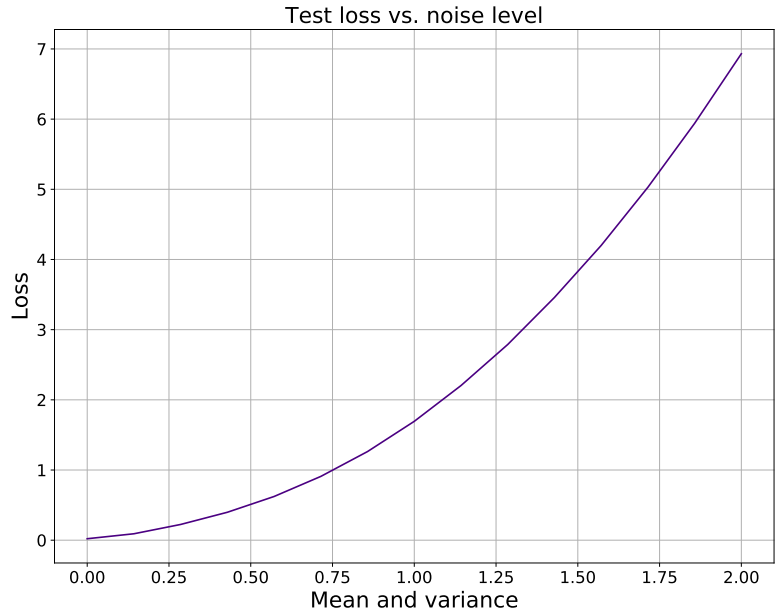
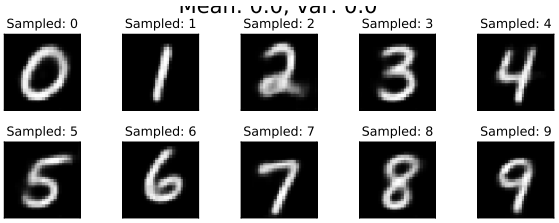
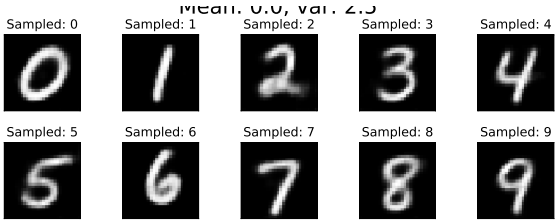


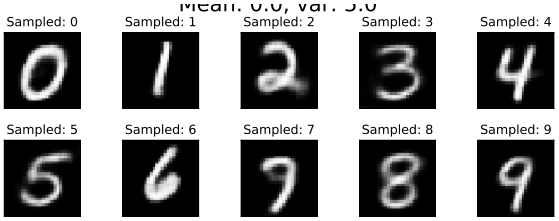
Figure 2: Loss on the same test samples; the noise is gaussian, with equal mean and variance.



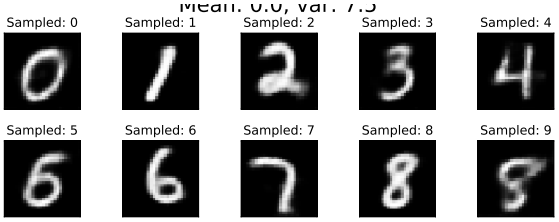
(a)



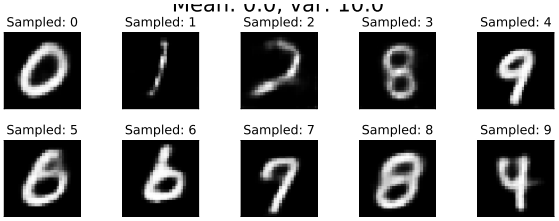
(b)



(c)



(d)



(e)

Figure 3: Samples generated around the centroids for varying levels of noise. Significant distortions appear at higher levels, where similar digits are confused by the network. For all the plots, the mean was set to 0, while the variance went from 0 to 10 in steps of 2.5.

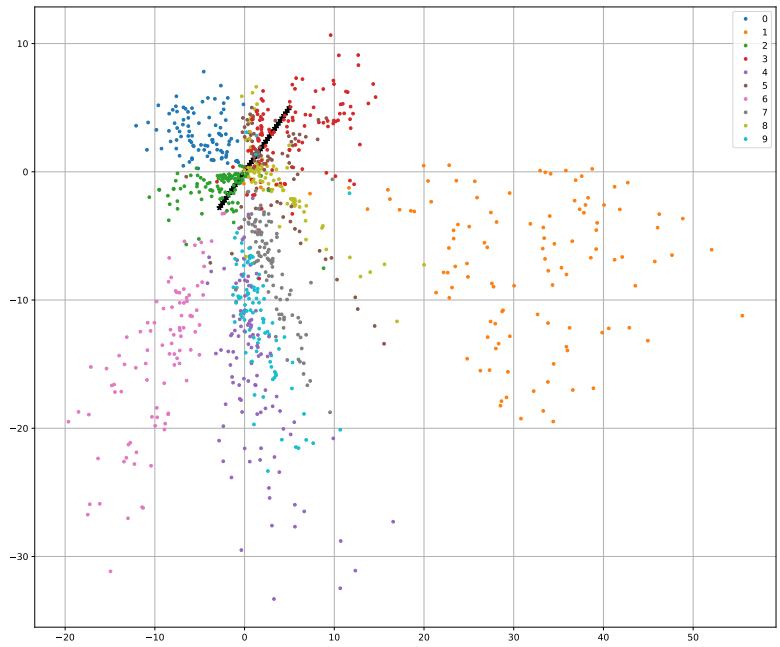


Figure 4: Encoding space of dimension 2 for an AE. The sampled points are shown as black stars.

Figure 5: Samples drawn from the encoding space in Fig. 4. Samples go from the upper-right corner to the lower-left one. Samples are blurred in transition regions, but are clear elsewhere.

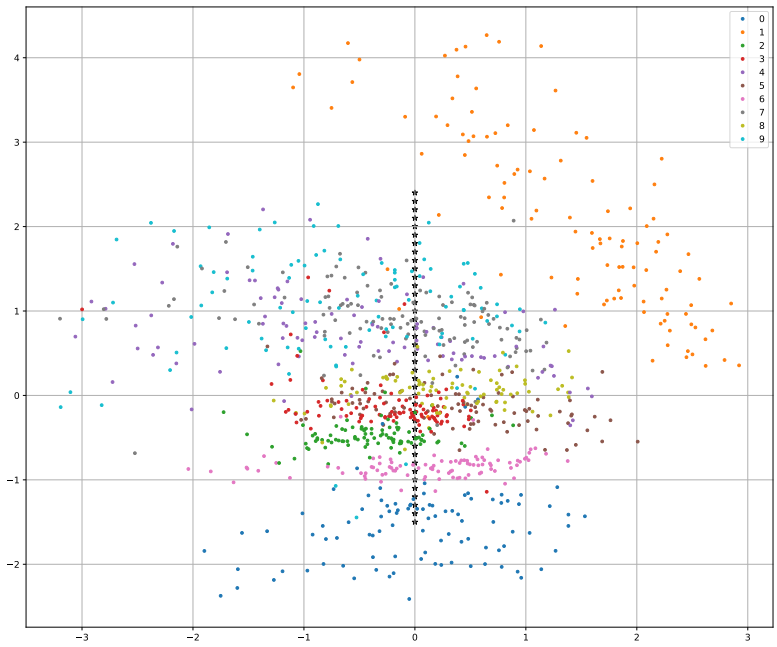


Figure 6: Encoding space of dimension 2 for a VAE. The sampled points are shown as black stars.

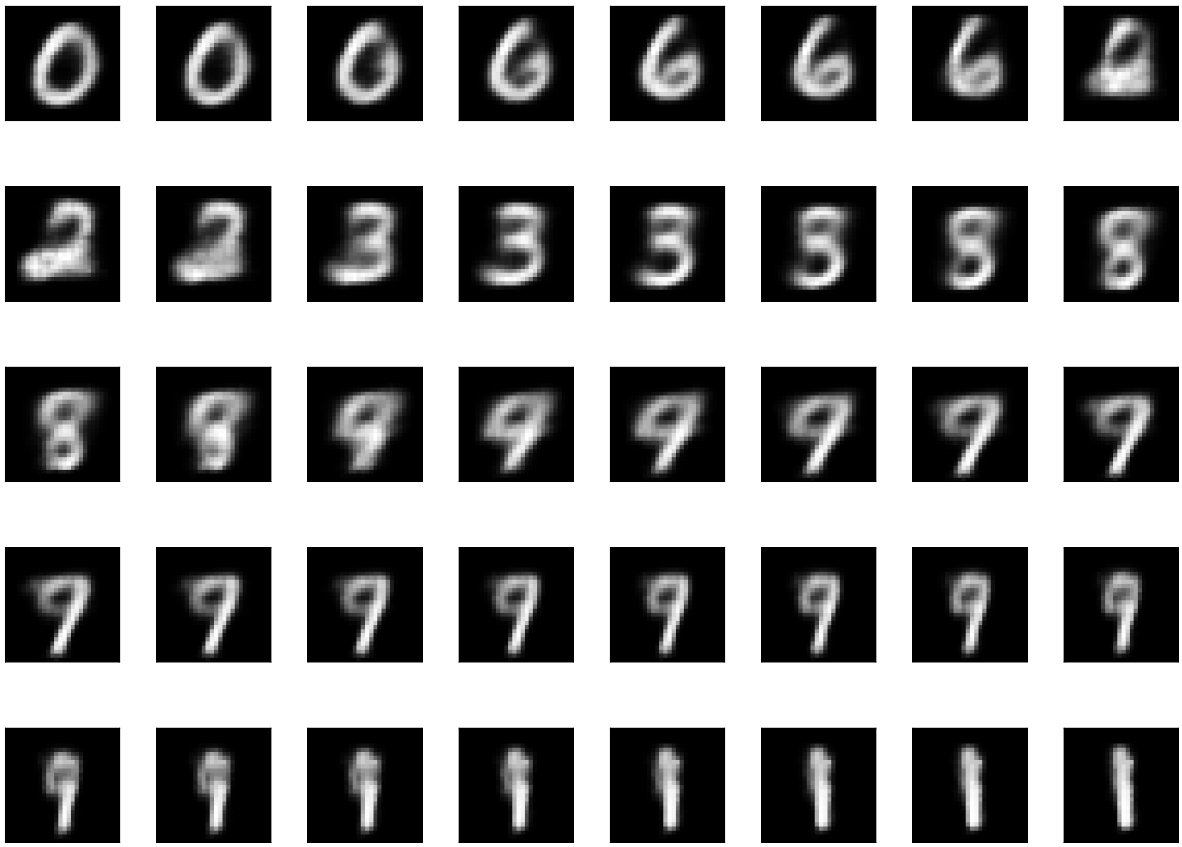


Figure 7: Samples drawn from the encoding space in Fig. 6. Samples go from the lower to the upper extreme. Samples cross various clusters, but are still recognizable despite the clusters being very narrow.