Neural Networks and Deep Learning

Homework II, Simone Trevisan, 1238612

January 5, 2021

1 Denoising autoencoder

The aim of this exercise is to develop a convolutional denoising autoencoder, able to reconstruct the original image starting from a corrupted version. Then, the encoder is used to extract the features and a linear layer is added to classify the input.

1.1 Data preprocessing

The dataset was divided into a training set of 60.000 samples and a test set of 10.000 samples. To enhance the generalization of the algorithm and avoid the risk of overfitting, a data augmentation was performed: a copy of the train set was randomly rotated, translated and sheared. Before input the images into the network, they are corrupted with Additive White Gaussian Noise, generated randomly at each iteration.

1.2 Layout

The network is divided into an encoder and a decoder. The encoder extracts the features from the images using three convolutional layers with 8, 16, and 32 filters of size 3x3. At each convolutional layer the dimensions of the input are almost halved (stride = 2, padding = 1). These layers are followed by two

linear layers with 64 and 4 neurons to decrease further the dimension (288) of the features extracted. The decoder reverse the described process to reconstruct the denoised image. At each layer, a ReLU activation function is applied to add non-linearity to the model. Loss function is MSE.

The output has the same dimension of the input, but it is generated from a latent space of only 4 values. Thus, the network achieve a compression rate of $784/4 = 196$.

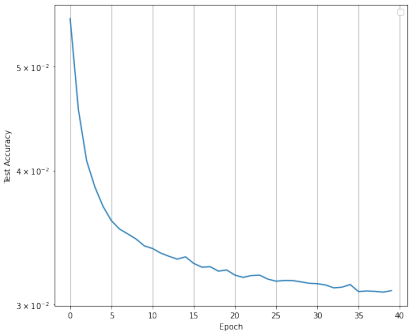
1.3 Optimization and generalization methods

Together with the previously describer data augmentation, other regularization methods are applied to avoid overfitting and increase the convergence speed. First, L2-penalization is applied to avoid the weights explosion. The amount of penalization is tuned together with other hyperparameters. Also, to improve generalization and speedup the training, input samples are arranged in mini-batches.

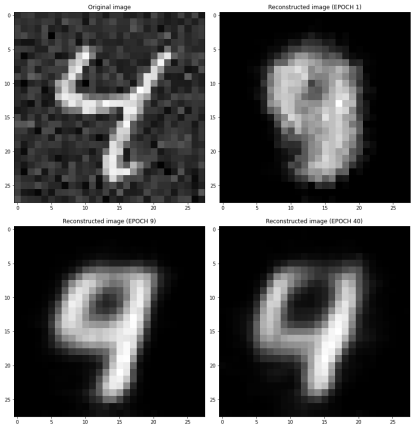
Hyperparameters (learning rate, weight decay and mini-batch size) are set using k-fold cross validation with k=4. The values taken into account are $lr = [1e-3, 1e-4]$, $wd = [1e-5, 1e-6]$, $bs = [256, 512]$. For each combination of hyperparameters, 4 neural network are trained (each with a different validation set). The minimum MSE reached by each network is averaged to give the configuration performance. The winning values are lr=1e-3, wd=1e-6, and bs=256.

1.4 Results

The final model is trained using the whole training dataset for 40 epochs. The trend of the MSE over the test set is shown in fig.**??**. After the first epoch, the network is already able to recognize the area where the digit is located and to reduce the noise in the surrounding area. At the end of the training, we can distinguish the digit 4, but since the outline is very smooth, it can be also considered to be a 9.



(a) MSE trend on the test set.



(b) Reconstruction trend of the corrupted input.

Figure 1

To deeply understand how the digits are encoded, the latent space is analysed in fig.2, where the first two principal components of the data PCA are plotted. Some digits are clustered in a defined area, while others are overlapping (i.e. 4 and 9).

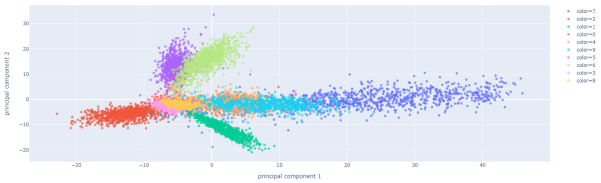


Figure 2: Plot of the two principal components of the latent space

1.5 Supervised classification

To classify the digits starting from the encoded features, the decoder is replaced with two linear layers of 256 and 10 neurons. The first (hidden layer) uses a ReLu activation function, while the second one (output layer) uses the log-softmax function. The loss is computed using negative log-likelihood. To avoid to impact the pretrained encoder weights too heavily, the learning rate is set to 1e-3 for the classifier parameters and to 1e-4 for the encoder parameters. Weight decay is set

to 1e-5 and mini-batch size to 256. The network is trained on the same dataset of the autoencoder for 10 epochs, achieving an accuracy of 0.97. The results is not directly comparable with the one developed in homework I, because of the different capacity and the corrupted input. Rather, it is interesting to compare the latent space with the one obtained before the supervised fine-tuning (3): now, the digits are much less overlapping, thus the encoder is able to better define the features that identify a digit.

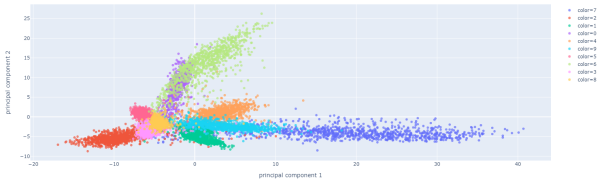


Figure 3: Plot of the two principal components of refined latent space

2 Generative Adversarial Network

The aim of this exercise is to develop a Generativa Adversarial Network (GAN) able to generate an axial MRI of Alzheimer affected brain. Architecture, data preprocessing, model, and results will be analyzed in the following sections.

2.1 Wasserstein GAN with Gradient Penalty

The GAN is an interesting architecture that models the training as a minimax, two-players, zero-sum game: a generator produce an image from a random input and the discriminator tries to recognize fake samples. Despite its impressive results, it is very hard to train and unstable. M. Arjovskyet et al., proposes in "Wasserstein Gan" to use the Wassertstein-1 distance, that is continuous everywhere and differentiable almost everywhere (under mild assumption). The WGAN value function is constructed using the Kantorovich-Rubinstein duality. To enforce the Lipschitz constraint on the discriminator, Ishaan Gulrajani et al. in "Improved Training of Wasserstein GANs", propose to exploit the fact that the optimal Lipschitz function has norm 1 almost everywhere under the two distribution (data and generator) adding a penalty to the loss function. Enforcing the unit gradient norm constraint everywhere is infeasible, so the authors propose to compute a random interpolation between a real and a fake

images assuming that if the norm of the gradient is 1 for all the interpolation, then Lipschitz is satisfied (details in the paper). The loss function is:

$$L = \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(||\nabla_{\hat{x}}D(\hat{x})||_2 - 1)^2]$$

with λ equal to 10.

2.2 Data preprocessing

The dataset, owned by Sarvesh Dubey (https://www.kaggle.com/tourist55/alzheimers-dataset-4-class-of-images), includes around 6000 MRI scans (5000/1000 trining/test) of Alzheimer patients, divided in 4 classes based on dementia severity. To reduce the computational and memory costs, images are downscaled to 64x64 pixels. Moreover, data are normalized to [-1,1].

2.3 Layout

The generator takes as input a random generated vector length 100. It is processed by 5 subsequent transpose convolutional layers with a decreasing number of features (1024, 512, 256, 128, 64), kernel size equal to 4 and stride=1. All the layers but the last one, are followed by batch normalization and ReLU activation function. The output of the fifth layer, a (1x64x64) image, is the input of a Tanh activation function to force the values into [-1,1] and match the training samples.

The structure of the discriminator (or critic as called by the authors) is specular to the generator with some differences. First, it uses a LeakyReLU with negative slope equal to 0.2 in place of a ReLU. Second, to keep the effectiveness of the penalty, it is needed to apply it with respect to each sample independently: batch normalization is then replaced with layer normalization in the paper. In this implementation, instance normalization is used because it is very similar to layer normalization but much more straightforward to apply. No activation function is used at the end of the last layer.

Ishaan Gulrajani et al., suggest in "Improved Training of Wasserstein GANs" to initialize the weights sampling from a normal distribution with zero mean and std=2.

Adam optimizers are used with learning rate lr=1e-4 and betas=(0.0,0.9) for both the generator and the critic. Since we want to train the critic heavier than the generator, at every epoch backpropagation is iterated 5 times before going on with the generator training.

2.4 Results

Results are obtained running the algorithm on a Google CoLab backend with GPU.

The network is trained for 80 epochs. The fig.4 shows the losses of the critic and the generator. The goal of the network is to maximize the (negative) loss shown in the left plot, while it has to minimize the loss of the generator loss. While it is clear that the network is still learning, the output (fig.5) is already good. The top right quadrant of the picture shows the results at the beginning of the training (only noise), while the bottom right shows the generated MRI at epoch 80, that is quite similar to actual MRI (top left). Finally, in fig.7 and in fig.6 a sample of generated image and its generator vector are shown. Even if the reconstruction is not perfect, it is possible to recognize the main neural structures that we expect to see in an MRI.

2.5 Further improvements

Proven that the model works, next steps will be to evaluate it with the test set, use higher resolution images and train for longer periods. Finally, implement a conditioned GAN to generate samples from a given class (demetia severity).

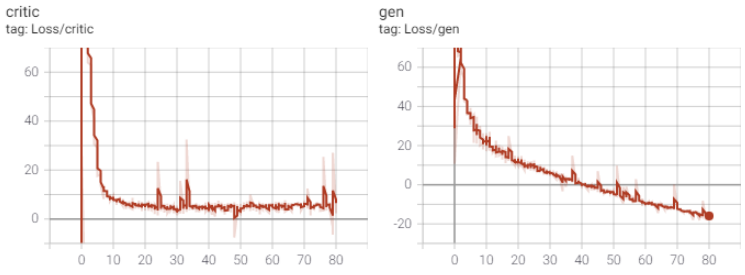


Figure 4: Critic loss (left) and generator loss (right)

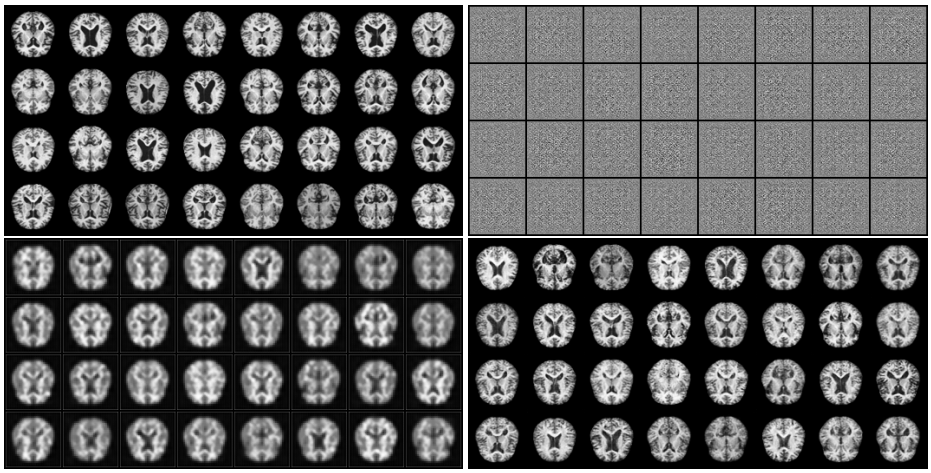


Figure 5: Real sample (top left), first epoch output (top right), 5th epoch output (bottom left), 80th epoch output (top right)

Figure 6: Random generator of the image in fig.7

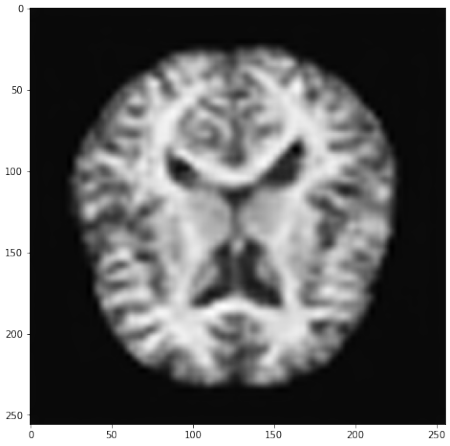


Figure 7: Image generated from vector in 6, (upscaled)