EE-559 Mini-project 1: Noise2Noise Auto-encoder

Xianjie DAI
Dept. of Computer Science
EPFL
Lausanne, Switzerland
xianjie.dai@epfl.ch

Guanqun LIU
Dept. of Computer Science
EPFL
Lausanne, Switzerland
guanqun.liu@epfl.ch

***Abstract*—This report is for EE-559 Deep Learning, mini-project 1. Our group implemented the Noise2Noise (N2N) auto-encoder model for image denoising without clean images. The original paper can be found at [1]. In our project, we explored different data augmentation methods, loss functions, and optimization methods to achieve a better Peak Signal-to-Noise Ratio (PSNR) with limited training time. In the first section of this report, we provide our understanding of N2N based on Prof. François Fleuret's lecture and the original paper [2]. In the second section, we will provide the implementation of our project and other details about our project. The experiment outcome will be provided in the third section of this report.**

I. Introduction of N2N

When the deep learning method is used for image denoising, it usually requires a large number of training image sample pairs, images with noise, and clean images, but clean images are usually difficult to obtain. For example, in photography, long exposure is required to obtain the image without noise. In MRI images, it is more difficult to obtain noiseless images. The author [1] proposed a method that only requires images with noise as input pairs. Notice that noise has to be of zero means.

Based on my previous understanding of denoising, when noisy input pairs are trained in a neural network, the network will learn the transformation between two input noise patterns if only a small amount of training samples are given. Even if sufficient enough amount of training samples are given, due to the randomness of noise, the network is not likely to find the transformation between two noise pairs and minimize the loss and finally learn the clean image pattern.

To understand why noisy image pairs could be used to train a neural network, we must understand that the L2 loss

$$\arg\min_{\theta} E_y[(Y - f(X \mid \theta))^2]$$

converges to the expectation of the target $E[Y]$, where Y is the clean image, X is the noise image, and $f_\theta()$ is the network with parameter θ. To find the minimum, the first order condition must be satisfied:

$$\frac{\partial}{\partial\theta} E_y[(Y - f(X \mid \theta))^2] = 0$$

The differentiation is with respect to the parameters θ of model $f(x \mid \theta)$. Proceed with differentiating:

$$E_y[\frac{\partial}{\partial\theta}(Y - f(X \mid \theta))^2] = E_y[(Y - f(X \mid \theta))](-2\frac{\partial}{\partial\theta} f(X \mid \theta))$$

Notice that this always holds when:

$$E_y[(Y - f(X \mid \theta)] = 0$$

$$E_y[Y] = E[f(X \mid \theta)]$$

If X and Y are independent, the training task is as:

$$\arg\min_{\theta} E_{x,y}[L(Y, f(X \mid \theta))]$$

But usually X and Y are not independent, by the Law of Total Expectation, we have:

$$\arg\min_{\theta} E_x[E_{y|x}[L(Y, f(X \mid \theta))]]$$

Notice that L2 loss always converges to $E_y[Y]$ no matter what distribution Y is. This implies that the estimation remains the same for different Y, as long as different Y has the same expectation.

So far, we have figured out the theoretical basis of N2N. In the next section, we will explain more details of our project.

II. Implementation of N2N

A. N2N structure

In the origin paper of N2N [1], the author implemented a convolution auto-encoder, U-Net. The architecture is as the Fig. 1 shows.

B. Data augmentation

To finish the whole training process in 10 minutes and achieve a satisfying result, we cannot waste too much time on data augmentation. Since the expectation of the noisy image must be 0, we selected Normal distribution noise to augment the data and improve the robustness of our model. Also because the training images are only of size $[3, 32, 32]$, we denied other data augmentation methods, such as rotation and crop.

C. Initialization

There are several different initialization methods in deep learning: constant initialization, random initialization, Xavier initialization, sparse initialization, Kaiming initialization, and even more.

- We denied constant initialization because, by initializing all computing units to the same state, the outputs and backward to update gradient of computing units might be

NAME	N_{out}	FUNCTION
INPUT	n	
ENC_CONV0	48	Convolution 3×3
ENC_CONV1	48	Convolution 3×3
POOL1	48	Maxpool 2×2
ENC_CONV2	48	Convolution 3×3
POOL2	48	Maxpool 2×2
ENC_CONV3	48	Convolution 3×3
POOL3	48	Maxpool 2×2
ENC_CONV4	48	Convolution 3×3
POOL4	48	Maxpool 2×2
ENC_CONV5	48	Convolution 3×3
POOL5	48	Maxpool 2×2
ENC_CONV6	48	Convolution 3×3
UPSAMPLE5	48	Upsample 2×2
CONCAT5	96	Concatenate output of POOL4
DEC_CONV5A	96	Convolution 3×3
DEC_CONV5B	96	Convolution 3×3
UPSAMPLE4	96	Upsample 2×2
CONCAT4	144	Concatenate output of POOL3
DEC_CONV4A	96	Convolution 3×3
DEC_CONV4B	96	Convolution 3×3
UPSAMPLE3	96	Upsample 2×2
CONCAT3	144	Concatenate output of POOL2
DEC_CONV3A	96	Convolution 3×3
DEC_CONV3B	96	Convolution 3×3
UPSAMPLE2	96	Upsample 2×2
CONCAT2	144	Concatenate output of POOL1
DEC_CONV2A	96	Convolution 3×3
DEC_CONV2B	96	Convolution 3×3
UPSAMPLE1	96	Upsample 2×2
CONCAT1	$96+n$	Concatenate INPUT
DEC_CONV1A	64	Convolution 3×3
DEC_CONV1B	32	Convolution 3×3
DEV_CONV1C	m	Convolution 3×3, linear act.

Fig. 1: U-Net

symmetry or even the same, which lowers the flexibility of our network.

- We also denied random initialization because the hyper-parameters, such as the mean μand variance σof normal distribution, could be difficult to determine.
- Xavier initialization is a good initialization method because it makes sure the variance between input and output remains the same. Take the forward process of fully connected layer as example: $Y = WX + B$, with the assumptions that W, X, and B are independent of each other, W_{ij} B_i and X_j are independent and identically distributed, and $Var[B_i] = 0$ $E[W_{ij}] = 0$ and $E[X_j] = 0$.

$$Var(Y_i) = d * Var(W_{ij}) * Var(X_j)$$

To make sure the input and output have the same variance, $dVar(W_{ij}) = 1$ must be satisfied, where d is the input dimension.

For backward, $\nabla X = W^\intercal \nabla Y$, in order to make sure the input signal and output signal remain the same variance, $uVar(W_{ij}) = 1$ must be satisfied, where u is the number of computation units within this layer. Combining the forward process and backward process together, unless $d = u$, otherwise the variance cannot remain unchanged in forward and backward process. Finally, a harmonic average, $Var(W_{ij}) = 2/(d+u)$, is applied. If W_{ij} follows Normal distribution, then $W_{ij} \sim \mathcal{N}(0,\ 2/(d+u)$. If W_{ij} follows Uniform distribution, then $W_{ij} \sim \mathcal{U}(-\sqrt{6/(d+u)},\ \sqrt{6/(d+u)})$.
- However, Xavier initialization does not take activation function into consideration, and activation function will change the distribution in the forward and backward process. Luckily, Kaiming initialization has a solution to this problem. Take the forward process of ReLU and Fully Connected layer as example: $Y = Wf(X) + B$, where $f()$ is ReLU activation function, and X_j has a symmetric distribution around zero, which indicates that $Var(X_j) = 0.5 * Var(f(X_j))$. Following the previous analyse in Xavier initialization section, we have $dVar(W_{ij}) = 2$. The same analyse for backward process: $uVar(W_{ij}) = 2$.

Notice that there is a very strong assumption in the forward analysis of Xavier and Kaiming Initialization: X_j are independent and identically distributed. In some cases, this assumption might not be correct, because X_j might be the output of the previous Fully Connected layer, i.e. the same output passes different neurons, which will introduce correlation. After many iterations, the distribution of W_{ij} will also change. However, this method still works and accuracy could be improved using this method, which might indicate the influence of a change of distribution, and the introduced correlation might weak.

D. Optimizing method

There are many state-of-art optimizing methods in deep learning. We implemented several different optimizing methods in our projects, such as one of the most famous first-order optimizing methods, Stochastic Gradient Descent (SGD), and one of the most state-of-art and most popular adaptive optimizing methods in the industrial field, Adam.

- We chose SGD with momentum based on Nesterov's Accelerated Gradient described in [2] as the baseline. We followed what the experiment results in [2] suggest to adjust our learning rate and batch size. Since we are given a fixed time to train the model, the outcome of optimizing with SGD might not be satisfying. Therefore, we also implemented two other optimizing methods which converge more rapidly.
- Adam is compared with SGD and achieved the best performance.

E. Loss

As the output of an auto-encoder is still an image of the same size as the input image, L1 loss and L2 loss can be implemented.

III. EXPERIMENT OUTCOMES

The size of the training data is $[50000, 3, 32, 32]$, and pixel values are from $[0, 255]$. To prevent gradient explosion and achieve a better convergence result in ten minutes, we normalized the input to $[0, 1]$. Our project was conducted with a GPU accelerator, RTX-2070Super, with 8GB memory. The batch size and learning rate are designed under the instruction given in [2].

We provide a sample of our training results as Fig. 2 shows.

A. Comparison of different optimizing methods

In this experiment, we implemented MSE loss, Kaiming initialization, and no data augmentation as the baseline. SGD: different learning rates were tested with fixed momentum, and weight decay based on Nesterov's Accelerated Gradient.

(a) Noisy (b) Clean (c) Output of N2N

Fig. 2: A sample of our N2N output

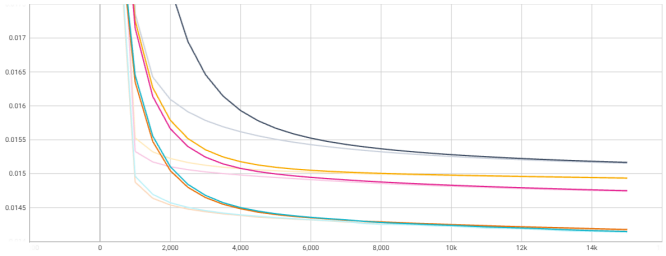

Fig. 3: SGD and Adam with different learning rate: from top to bottom, the top three curves are SGD with 0.001, 0.005, and 0.01 learning rate. The lower two curves are Adam with 0.0005, and 0.001 learning rate. The shallow curves are rough curves without smoothness.

We present the training-loss-versus-iterations figure in Fig. 3. Notice that we fixed batch size to 100 in all our experiments to finish the training in 10 minutes. We recorded training loss every 500 iteration, which leads to rough curves, therefore, we also provided smoothed curves in our figure.

Adam: different learning rates were tested with fixed batch size. We present the training-loss-versus-iterations figure in Fig. 3. We recorded training loss every 500 iteration, which leads to rough curves, therefore, we also provided smoothed curves in our figure. Finally, we present the testing PSNR of different optimizing methods with different learning rate in TABLE I.

Learning Rate	SGD	Adam
0.05	12.68dB	-266.26dB
0.01	25.08dB	24.91dB
0.005	24.85dB	25.33dB
0.0001	24.58dB	25.42dB
0.0005	24.46dB	25.42dB

TABLE I: Testing PSNR of different optimizing methods with different learning rate

B. Comparison of different data augmentation methods

In this experiment, we implemented MSE loss, Kaiming initialization, and the best optimizing method with the best hyper parameters as the baseline. The final testing PSNR of different data augmentation methods is as TABLE II shows.

C. Comparison of different initialization methods

Kaiming initialization considers the activation function, which is an improvement of Xavier. We were expecting a better result from Kaiming initialization. In this experiment, we implemented MSE loss, no data augmentation, and the best optimizing method with the best hyperparameters as the baseline. The final testing PSNR of different initialization methods is as TABLE III shows.

Data Augmentation Method	Final test PSNR
Original Training Images	25.42dB
Images with Gaussian Noise	24.51dB

TABLE II: Testing PSNR of different data augmentation methods

Data Initialization Method	Final test PSNR
Kaiming Initialization	25.42dB
Xavier Initialization	25.44dB

TABLE III: Testing PSNR of different initialization methods

D. Comparison of different loss functions

The theoretical basis of N2N is based on L2 loss, which always converges to $E(Y)$. However, BCE loss always converges to p, the label, which is a noisy verse of the true label. Therefore, we were not expecting a better result from the BCE loss. To be consistent with our other experiments, we finally abandoned our BCE experiment and only provide the comparison between L1 loss and L2 loss. In this experiment, we implemented Kaiming initialization, no data augmentation, and the best optimizing method with the best hyperparameters as baseline. The final testing PSNR of different loss functions is as TABLE IV shows.

Loss Function	Final test PSNR
MSE Loss	25.42dB
L1 Loss	24.59dB

TABLE IV: Testing PSNR of different loss functions

IV. Conclusion

To summarize, we implemented the famous N2N auto-encoder with the magic Pytorch framework. We also explored different data augmentation strategies, optimization methods, and loss functions. The best results in our project are 25.44dB with Xavier initialization, no data augmentation, L2 loss, and Adam optimizing with a learning rate of 0.001 and default betas and eps. We believe we can achieve a better result if we were given more training time.

REFERENCES

[1] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2Noise: Learning image restoration without clean data," arXiv.org, 29-Oct-2018. [Online]. Available: https://arxiv.org/abs/1803.04189.

[2] I. Sutskever, V. Profile, J. Martens, G. Dahl, G. Hinton, and O. M. V. A. Metrics, "On the importance of initialization and momentum in Deep Learning: Proceedings of the 30th International Conference on International Conference on Machine Learning - volume 28," Guide Proceedings, 01-Jun-2013. [Online]. Available: https://dl.acm.org/doi/10.5555/3042817.3043064.