BARF: Bundle-Adjusting Neural Radiance Fields Development Track

Hyunjin Kim Geon Park

Abstract

In this report, we summarize and implement BARF: Bundle-Adjusting Neural Radiance Fields, elaborate our approach and the challenges we encountered during the process, and show the results, comparing them to the results reported in the original paper. BARF, which allows training of NeRF without accurate camera pose information, is motivated by extending the 2D image alignment problem into 3d using Neural Radience Fields, and the key idea is to incorporate coarse-to-fine positional encoding. We encountered some challenges when trying to implement our version of BARF, such as implementing NeRF correctly, and calculating the pose distance. The experimental results confirm the claims in the original paper. Code is available at: https://github.com/mujjingun/BARF

1. Introduction

Motivation The problem with NeRF [3] is that accurate camera pose information must be provided in order to train the model. However, this assumption is unrealistic, since obtaining accurate camera pose information is impossible in most realistic scenarios. To this end, BARF [1] was proposed, which can recover accurate camera pose information only from a set of images and noisy camera pose information, or even from no camera pose information at all.

Problem Setup The input of BARF is a set of images, and the output of BARF is a neural radiance field, which takes 3D coordinates, and predicts a color value. During the training phase, BARF minimizes the sum of the MSEs between each input image and the scene representation rendered with the trainable pose parameter, with respect to both the camera poses and the NeRF parameters. However, naively applying the above framework, the authors run into the problem of the pose parameters falling into poor local minima. So the authors propose coarse-to-fine positional encoding. Positional encoding helps NeRF take into account finer details in the object. Using coarse (less components) positional encoding makes the scene representation smoother, but also lack detail, whereas using fine (more components) positional encoding makes the scene sharper and able to express more detail. BARF gradually increases the number of positional encoding components during training, so that the initial smooth signal can guide the camera poses into roughly the correct spots, before learning high-fidelity scene representation, which is prone to bad local minima.

Key Challenges Implementing BARF is challenging in several ways:

- Implementing NeRF from scratch correctly
- Implementing the differentiable rigid transformation function $\mathcal{W}$
- Implementing the coarse-to-fine positional encoding

Implementation Details We implemented the following parts by ourselves:

- 2D planar alignment section
- NeRF model & training code
- Coarse-to-fine positional encoding code
- Differentiable rigid transformation

No external collaborators were utilized for our implementation. We used the PyTorch3D library for the $\mathfrak{se}(3)$ exponential map transformation.

Experimental Setup We reproduce all of the experiments reported in the main text of the paper, namely:

- 2D planar alignment experiment
- Blender scene experiment (w/o pos enc, full pos enc, BARF)
- LLFF real-world dataset experiment (w/o pos enc, full pos enc, BARF)

Summary of Achievements In summary, our achievements are:

- Implementing BARF correctly
- Reproducing the experiment results reported in the original paper.

2. Method Summary

NeRF To construct the 3D scene, NeRF uses the concept that how the camera renders the 2D image from the 3D scene. For the given training image, first, they select the pixel and render the pixel color using the rendering procedure along the ray which connects the camera viewpoint and selected pixel. The rendering procedure consists of 3 steps; 1. Sample points along the ray, 2. Compute the color and density of each point using MLP, 3. Render color using rendering equation. After rendering the color of the pixel, they optimize the model using the L2 distance between ground truth color and rendered color as a loss function. Moreover, to improve the result quality, they apply positional encoding that map input into higher dimensional space and hierarchical sampling that samples points following the distribution which is produced by network output density.

BARF BARF uses the coarse-to-fine positional encoding in order to prevent the trainable pose parameters from getting stuck in poor local minima, while jointly minimizing the sum of each image's MSE loss with respect to the NeRF parameters and the camera poses.

Problem Setup The input of BARF is a set of images $\mathcal{I}_1,\dots,\mathcal{I}_M$, and the output of BARF is a neural radiance field $f(\mathbf{x};\Theta)$ parameterized by Θ, which takes 3D coordinates $\mathbf{x}$, and predicts a color value. During the training phase, BARF minimizes the following objective with respect to both the camera poses $\mathbf{p}_i$'s and the NeRF parameters Θ:

$$\min_{\mathbf{p}_1,\dots,\mathbf{p}_M,\Theta}\sum_{i=1}^{M}\sum_{\mathbf{u}}\left\|\hat{\mathcal{I}}(\mathbf{u};\mathbf{p}_i,\Theta)-\mathcal{I}_i(\mathbf{u})\right\|_2^2, \tag{1}$$

where

$$\hat{\mathcal{I}}(\mathbf{u};\mathbf{p})=g\left(f(\mathcal{W}(z_1\bar{\mathbf{u}};\mathbf{p});\Theta),\dots,f(\mathcal{W}(z_N\bar{\mathbf{u}};\mathbf{p});\Theta)\right)$$

denotes the voluemtric rendering of the radiance field represented by f at pixel position $\mathbf{u}\in\mathbb{R}^2$ and pose $\mathbf{p}$. The pose parameter $\mathbf{p}\in\mathbb{R}^6$ is parameterized with the $\mathfrak{se}(3)$ exponential map. $\mathcal{W}(z\bar{\mathbf{u}},\mathbf{p})$ is the rigid transformation function, which must be differentiable and transforms a 3D point $z\bar{\mathbf{u}}$ in camera coordinates into world coordinates using the camera pose $\mathbf{p}$. Inside f, the positional encoding is applied. The coarse-to-fine positional encoding of a coordinate x is defined as follows:

$$\text{PosEnc}(x,\alpha)=\left[x,\gamma_0(x,\alpha),\dots,\gamma_{L-1}(x,\alpha)\right], \tag{2}$$

where the k-th frequency component γ_k is defined as

$$\gamma_k(x,\alpha)=w_k(\alpha)\cdot[\cos(2^k\pi x),\sin(2^k\pi x)], \tag{3}$$

where the weight w is defined similar to the Hanning window from Signal Processing:

$$w_k(\alpha)=\begin{cases}0 & \text{if } \alpha<k\\ \frac{1-\cos((\alpha-k)\pi)}{2} & \text{if } 0\le\alpha-k<1\\ 1 & \text{if } \alpha-k\ge 1.\end{cases} \tag{4}$$

3. Implementation Details

We use ***pytorch*** to implement all of the methods.

3.1. 2D Planar Image Alignment

We use ***F.grid_sample*** from pytorch for extracting patches using a grid of coordinates, and matplotlib library for visualizing the warps. The coordinate transformation is implemented using the $\mathfrak{sl}(3)$ homography map with 8 parameters, consulting [2]. For the objective, we use the following MSE formulation found in the original paper.

$$\min_{\mathbf{p}_{2,3,4,5},\Theta}\sum_{i=1}^{5}\sum_{\mathbf{x}}\|f(\mathcal{W}(\mathbf{x};\mathbf{p}_i);\Theta)-\mathcal{I}_i(\mathbf{x})\|_2^2,$$

where $\mathbf{p}_i$ is the learned warp of the i-th patch. $\mathbf{p}_1$ is fixed to the identity $[0,0,0,0,0,0,0,0]$, as in the original paper.

3.2. NeRF

To implement NeRF correctly, we split overall NeRF code as the 5 parts such that **(1)** Constructing Network, **(2)** Data Loading, **(3)** Sampling Points, **(4)** Rendering Color, and **(5)** Positional Encoding. First of all, for constructing the network, we use ***torch.nn*** module and define the model using our implementation. Next, we implement dataset loading part by referring *nerf-pytorch* [4]. Especially, we use *nerf-pytorch* code for loading the LLFF dataset because of its complex structure. The sampling part is implemented based on uniform sampling along the ray, and the rendering part is implemented based on the rendering equation $\hat{C}(r)$:

$$\begin{aligned}\hat{C}(r)&=\sum_{i=1}^{N}T_i(1-\exp(-\sigma_i\delta_i))\mathbf{c_i},\\ \text{where } T_i&=\exp(-\sum_{j=1}^{i-1}\sigma_j\delta_j)\end{aligned} \tag{5}$$

which is proposed in the NeRF paper. Especially, alpha compositing and depth rendering is not mentioned in the main paper, we borrow those parts from *nerf-pytorch* code. Lastly, we implement positional encoding based on Equation 6. We do not implement hierarchical sampling since it is not used in the whole BARF experiments.

$$\begin{aligned}&\text{PosEnc}(x)=\left[x,\gamma_0(x),\dots,\gamma_{L-1}(x)\right],\\ &\text{where } \gamma_k(x)=[\cos(2^k\pi x),\sin(2^k\pi x)]\end{aligned} \tag{6}$$

3.3. BARF

The main difference between NeRF and BARF is that BARF updates camera poses in the optimization step. Moreover, to make camera pose alignment well, BARF adopts coarse-to-fine positional encoding which enables both avoiding local minima of camera poses and representing high fidelity results. Except for these two differences, remain training procedure is almost the same as NeRF. Thus, we use scratch NeRF that we implemented as the baseline and add coarse-to-fine positional encoding and trainable camera pose parameters to implement BARF.

Coarse-to-fine positional encoding The only difference between the original positional encoding and coarse-to-fine positional encoding is multiplying weight function $w_k(\alpha)$ (Equation 4) to each frequency component. Thus, we append weight function to original positional encoding code that we implemented for NeRF.

Differentiable rigid transformation The differentiable rigid transformation is implemented as a matrix multiplication between the original world-to-camera pose matrix and the transformation matrix given by the $\mathfrak{se}(3)$ exponential map, parameterized by 6 values. We used the `transforms.se3_exp_map` library function from ***pytorch3d*** for the computation of the $\mathfrak{se}(3)$ exponential map.

4. Experimental Results

We perform 2 types of experiments such that *2D planar image alignment* and *BARF*. For 2D planar image alignment, we use the same cat image from the original paper. Since we couldn't find the exact values for the random warps for the patches, we used a different set of warps for our experiment. For BARF, we use NeRF Synthetic dataset (called Blender dataset) and the LLFF Real-World dataset. Both datasets contain 8 objects. For quantitative comparison, we use PSNR, SSIM, and LPIPS as metrics that are used in both NeRF and BARF original papers. Every experiment settings are the same as the original paper except that we use 512 sampled points per ray in inference time (In training time, we use 128 points per ray which is the same as the original paper). This difference makes the output quality a little bit better but not significant.

4.1. 2D Planar Image Alignment

Following the original paper's settings, we extract five random patches from the original image, and then optimize for the MSE between the patches and the reconstructed image with respect to both the pose parameters and the neural image representation. We show the quantitative results of the 2D planar alignment in Table 1, and the qualitative results in Figure 1. Unlike the original paper, the $\mathfrak{sl}(3)$ error of

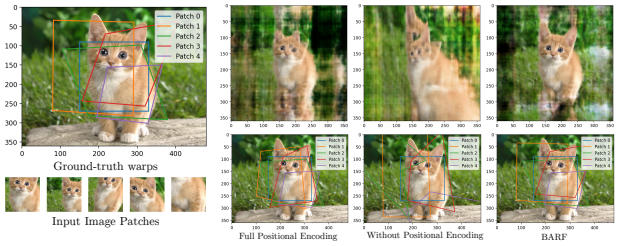

Figure 1. **Qualitative results** of the planar image alignment experiment. **Left:** The ground-truth warps and the extracted image patches. **Right:** Visualization of the trained image representation (top row), and the learned warps (bottom row).

positional encoding		$\mathfrak{sl}(3)$ **error**	patch PSNR
naive (full)	reported	0.2949	23.41
	ours	0.1917	24.16
without	reported	0.0641	24.72
	ours	1.7891	22.71
BARF	reported	0.0096	35.30
	ours	0.0155	29.64

Table 1. **Quantitative results** of the planar image alignment experiment. We denote the values reported in the original paper as "reported", and the values obtained by our implementation by "ours". Note that our values cannot be directly compared to the original paper's.

the version that does not use any positional encoding (without) is very high. The main reason for this is that the learned warp parameters are very sensitive to the input configuration. We had to use a different warp configuration from the original paper, since we couldn't find the warps parameter values used in the original paper. We can observe that only BARF is capable of reconstructing the original warp parameters, thus reconstructing the original image in the area covered by the patches correctly.

4.2. BARF: 3D Synthetic Objects

We use the blender synthetic dataset for the NeRF experiment. Our experiment setting for synthetic objects is totally the same as the original paper. In detail, we first convert camera poses (extrinsic) into $\mathbf{p} \in \mathbb{R}^6$ which is the element of $\mathfrak{se}(3)$ Lie algebra, and then give pose perturbs following Gaussian distribution with mean 0 and standard deviation 0.15. We show the quantitative results in Table 2, and show the qualitative results in Figure 2. Unlike the original paper, the result using full-positional encoding is pretty good. It has much less camera pose error compared to the original paper. Even for ***Chair*** scene, using full-positional encoding achieves the best performance. Thus, we can think that there could be some wrong implementation in the original paper. Also, we can notice that the quality is similar to or even better than BARF results. The main reason that

Scene		Camera pose registration						View synthesis quality											
		Rotation (°)			Translation			PSNR				SSIM				LPIPS			
		full	w/o	BARF	full	w/o	BARF	full	w/o	BARF	NeRF	full	w/o	BARF	NeRF	full	w/o	BARF	NeRF
Chair	rep	7.186	0.110	0.096	16.638	0.555	0.428	19.02	30.22	31.16	31.91	0.804	0.942	0.954	0.961	0.223	0.065	0.044	0.036
	ours	0.098	0.318	**0.090**	2.16	0.68	**0.62**	**29.79**	28.52	29.47	32.31	**0.963**	0.949	0.961	0.977	**0.052**	0.082	0.053	0.036
Drums	rep	3.208	0.057	0.043	7.322	0.255	0.225	20.83	23.56	23.91	23.96	0.840	0.893	0.900	0.902	0.166	0.116	0.099	0.095
	ours	0.528	0.060	**0.052**	3.93	0.43	**0.43**	22.98	23.33	**23.89**	24.21	0.912	0.915	**0.923**	0.929	0.119	0.127	**0.104**	0.091
Ficus	rep	9.368	0.095	0.085	10.135	0.430	0.474	19.75	25.58	26.26	26.68	0.836	0.922	0.934	0.941	0.182	0.070	0.058	0.051
	ours	0.088	0.090	**0.083**	0.74	**0.73**	0.76	26.68	25.81	**26.94**	27.50	0.953	0.940	**0.953**	0.961	**0.059**	0.079	0.061	0.047
Hotdog	rep	3.290	0.225	0.248	6.344	1.122	1.308	28.15	34.00	34.54	34.91	0.923	0.967	0.970	0.973	0.083	0.040	0.032	0.029
	ours	0.831	**0.256**	0.273	4.76	1.08	**1.05**	22.87	**23.58**	23.22	34.82	0.904	**0.915**	0.910	0.985	0.048	0.059	**0.047**	0.023
Lego	rep	3.252	0.108	0.082	4.841	0.391	0.291	24.23	26.35	28.33	29.28	0.876	0.880	0.927	0.942	0.102	0.112	0.050	0.037
	ours	0.179	0.094	**0.072**	1.18	0.60	**0.47**	**28.55**	25.98	28.29	30.40	**0.954**	0.909	0.951	0.968	**0.052**	0.130	0.061	0.038
Materials	rep	6.971	0.845	0.844	15.188	2.678	2.692	16.51	26.86	27.84	28.48	0.747	0.926	0.936	0.944	0.294	0.068	0.058	0.049
	ours	0.789	0.940	**0.290**	9.19	6.36	**2.29**	17.60	25.52	**27.13**	28.80	0.817	0.931	0.949	**0.965**	0.097	0.068	**0.049**	0.031
Mic	rep	10.554	0.081	0.071	22.724	0.356	0.301	15.10	30.93	31.18	31.98	0.788	0.968	0.969	0.971	0.334	0.050	0.048	0.044
	ours	0.223	0.068	**0.057**	1.27	0.48	**0.48**	29.03	28.98	**29.69**	31.35	0.971	0.970	**0.973**	0.977	0.045	0.048	**0.043**	0.040
Ship	rep	5.506	0.095	0.075	7.232	0.354	0.326	22.12	26.78	27.50	28.00	0.755	0.833	0.849	0.858	0.255	0.175	0.132	0.118
	ours	1.068	**0.121**	0.311	7.74	**0.65**	2.16	22.71	23.59	**24.04**	28.00	0.860	0.867	**0.879**	0.918	0.142	0.172	**0.133**	0.106
Mean	rep	6.167	0.202	0.193	11.303	0.768	0.756	22.12	26.78	27.50	29.40	0.821	0.917	0.930	0.936	0.205	0.087	0.065	0.057
	ours	0.476	0.243	**0.154**	3.87	1.38	**1.03**	25.03	25.66	**26.59**	29.68	0.917	0.925	**0.937**	0.960	0.077	0.096	**0.069**	0.051

Table 2. **BARF experiment results (Blender dataset)**. We denote the values reported in the original paper as "rep", and the values obtained by our implementation by "ours". Translation error is scaled by 100.

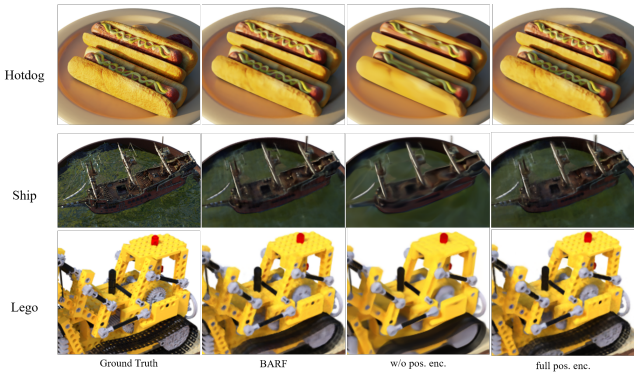

Figure 2. **Blender qualitative results**. Unlike original paper, full-positional encoding produces fine results in our implementation

full-positional encoding has worse metric results is because of Procrustes alignment which is based on the mean rotation and translation of all poses (Also this is the reason why quantitative result of ***Hotdog*** scene is worse).

4.3. BARF: 3D Real-World Scenes

We use the LLFF real-world dataset for the real-world NeRF experiment. Same as other experiments, all setting is totally the same as the original paper. However, unlike 3D synthetic dataset, in this experiment we initialize all camera poses as $\mathbf{p} = [0, 0, 0, 0, 0, 0]$ (represented as $\mathfrak{se}(3)$ vector). The qualitative and quantitative of LLFF dataset and its detail is in **Supplementary Material**.

5. Conclusion

We implemented BARF and confirmed the results reported in the original paper. The first difference between the main paper's result and our result is that ours work well for not only coarse-to-fine positional encoding (BARF) but also full-positional encoding. Also, our result of the LLFF dataset is poor which cannot represent the background well, and we think the possible reason is that we do not use a normalized coordinate system. However, except for this problem, we successfully reproduce the results of the original paper and even improve it (full-positional encoding).

Limitations One limitation of the paper is that the model is sensitive to the initial camera pose.

6. Acknowledgments

Our team members' roles are as follows:

- Geon Park - 2D planar alignment, Differentiable rigid transformation.
- Hyunjin Kim - Implementing NeRF, Coarse-to-Fine positional encoding.

This project was done without any external collaborators.

6.1. Code Reference

NeRF-pytorch code [4] (with background, depth, llff dataset loading).

References

[1] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. BARF: Bundle-adjusting neural radiance fields. *CVPR*, 2021. 1

[2] Christopher Mei, Selim Benhimane, Ezio Malis, and Patrick Rives. Homography-based tracking for central catadioptric cameras. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006. 2

[3] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. *ECCV*, 2020. 1

[4] Lin Yen-Chen. Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020. 2, 4