Fast Training for Neural Radiance Fields via Voxel Representation

Orr Shalev
School of Computing
University of Georgia
Athens, Georgia, United States
orr.shalev@uga.edu

Porter Squires
Institute for Artificial Intelligence
University of Georgia
Athens, Georgia
jonathan.squires@uga.edu

Lihao Zhang
School of ECE
University of Georgia
Athens, Georgia, United States
lihao.zhang@uga,edu

***Abstract*—This project aims to implement and evaluate the TensoRF and DVGO methods to assess their performance in terms of synthetic image quality and training efficiency. First, we will identify a proper dataset that balances training time and effectiveness in evaluating performance. Then, we will implement these two projects on the dataset with the appropriate training settings. After training and testing, we will record both the training process and test results and conduct a comprehensive analysis of synthetic image quality and computational resource requirements. Also, we will explore the theoretical reasons behind the observed performance differences between TensoRF, DVGO, and NeRF. This exploration will include examining aspects such as the compactness of the models' representations, computational efficiency, and the fidelity of volumetric rendering.**

I. Introduction

A. NeRF

Synthesizing a 3D scene from 2D visual inputs is a complex challenge that involves capturing the geometric information, visual texture, and lighting conditions from the 2D images. Traditional 3D reconstruction methods requires extensive effort in predict the depth information, reconstruct the 3D models of the objects and environment, and assign the RGB value to the 3D meshes. However, in many applications, the primary goal is just to freely view the 3D scene from various angles – that is, to generate 2D "pictures" of the scene from different camera poses, focusing on the fidelity of these images rather than the accuracy of the intermediate processes. To address this kind of need, the concept of a neural radiance field network (NeRF) [3] was introduced.

Unlike previous approaches that only focused on replacing certain aspects of the 3D scene reconstruction workflow, NeRF utilizes a neural network to implicitly represent the radiance field of the original scene, i.e., the emitted anisotropic visible light at each voxel position. To "observe" a voxel in NeRF, a 5D vector is input, including the voxel's location (x, y, z) and the observation direction (θ, ϕ). The output is the color (r, g, b) and volume density σ.

Once NeRF is adequately trained for a given scene, any camera pose can be input into the workflow. Similar to classic volume rendering, a virtual image plane is determined based on the camera pose, consisting of uniformly distributed pixels. Rays are then cast from the camera to these pixels, and the NeRF network samples the voxels along each ray to calculate the RGB value of each pixel. Following the principles of classic volume rendering, the volume density represents the differential probability of a ray terminating at that voxel, with the pixel's RGB value being the integral of the transmittance multiplied by the RGB values of the voxels along the ray.

B. Training of NeRF

The success of NeRF lies not only in its novel concept but also in critical details of the training process. To enable NeRF to learn from training 2D images, the dataset should not only include the images but also the corresponding camera poses and parameters defining the virtual image plane in 3D space, and the rays for volume rendering. During training, the inputs consist of batches of rays from all pixels in the dataset, spanning various images. Since both the NeRF network and the render are differentiable, optimization of network parameters is achieved through gradient descent, minimizing the total squared error between the rendered and actual pixel color.

To ensure multi-view consistency, the volume density is constrained as a function solely dependent on location. Given that deep networks are more adept at learning low-frequency functions, the authors utilized the positional encoding, which will map the location inputs to a higher-dimensional space using a set of orthogonal sinusoidal functions similar to the Fourier Transform. This kind of encoding is also used in the popular Transformer architecture.

Furthermore, the traditional deterministic quadrature used in classical volume rendering, designed for discretized voxel grids, may limit NeRF's performance. To overcome this, a hierarchical sampling scheme is employed in the rendering process. This involves allocating samples along the ray based on their expected impact on the final pixel color, achieved by optimizing two networks: one coarse and one fine. The coarse network evaluates voxels from stratified sampling (uniform segmentation of each ray with random selection within each segment) to identify important voxels for sampling within the fine network, thereby focusing samples on more visible content in the 3D scene.

C. TensoRF

Despite the impressive capabilities of Neural Radiance Fields (NeRF) in synthesizing high-quality images from arbi-

trary camera poses with sparse training images, the complexity and computational intensity of the NeRF workflow pose significant challenges. These challenges include the requirement for computational resources and slow training times when training models for new scenes, even with the mentioned efficiency optimizations. This has spurred the development of methods aimed at accelerating the NeRF workflow, with TensoRF being a notable example.

TensoRF [2] introduces a novel approach to radiance field representation, distinguished by its compactness and rapid deployment capabilities for new scenes. In contrast to the coordinate-based MLP in NeRF, TensoRF employs voxel grids to explicitly represent the radiance field in terms of RGB color and volume density. To addressing the issue of large GPU memory requirements associated with storing voxels directly which escalates cubically with scene size, TensoRF utilizes a 4D tensor to represent this feature grid. Three dimensions of the tensor refer to the XYZ axes and an additional dimension for the feature channel. Such Tensor facilitates the application of classical tensor decomposition techniques such as CANDECOMP/PARAFAC (CP) and vector-matrix (VM) decomposition to reduce memory consumption. While this study focuses on CP and VM decompositions, the TensoRF framework is designed to be compatible with a wide range of tensor decomposition techniques for future explorations.

D. Direct Voxel Grid Optimization (DVGO)

Complementing the TensoRF approach, DVGO [1] offers a rapid method for reconstructing the radiance field from a set of images. DVGO aims to directly model 3D geometry and volume density using a dense voxel grid. For color, DVGO moves away from NeRF's view-dependent color representations. Instead, it adopts a hybrid representation that combines a feature grid with a shallow MLP for color encoding. Initially, the density voxel grid is initialized with opacities close to zero to minimize bias towards the camera's near plane. Additionally, to reduce the redundancy of voxels that are visible from only a few views, DVGO employs lower learning rates for these voxels. A key innovation in DVGO is its approach to scalability: it automatically identifies a bounding box to enclose the volume of interest, optimizing the allocation of the voxel grid. Furthermore, DVGO introduces a post-activation technique, applying activation functions after the trilinear interpolation of the density voxel grid, enabling the modeling of sharp linear surfaces with high precision. Notably, DVGO's use of 160^3 dense voxels has outperformed NeRF in their instances.

II. Research Plan

Our project will consist of implementing and comparing TensoRF and DVGO. Both architectures are described thoroughly in their associated publications and will be the main source used to guide implementation. We will evaluate our implementations on the Synthetic NeRF dataset and compare training time and averaged rendering PSNR and SSIM scores to the results obtained by the designers of TensoRF and DVGO.

A. TensorRF

Two major versions of TensoRF exist: TensoRF-CP which applies traditional CANDECOMP/PARAFAC decomposition, while TensoRF-VM uses a novel vector-matrix decomposition which was designed by the authors to improve performance. Thus, our first task for implementing TensoRF will involve implementing the two decomposition techiques.

*a) **TensoRF-CP**:* The simpler decomposition method metioned by the authors was TensoRF-CP. It consists of factoring a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ into

$$\mathcal{T} = \sum_{r=1}^{R} v_r^1 \circ v_r^2 \circ v_r^3$$

where $v_r^1 \circ v_r^2 \circ v_r^3$ is a rank-one component, $v_r^1 \in \mathbb{R}^I$, $v_r^2 \in \mathbb{R}^J$, and $v_r^3 \in \mathbb{R}^K$ are factorized vectors of the three modes for the rth component. Superscripts are the modes of each factor; $\circ$ represents the outer product. Each tensor element $\mathcal{T}_{ijk}$ is a sum of scalar products:

$$\mathcal{T}_{ijk} = \sum_{r=1}^{R} v_{r,i}^1 v_{r,j}^2 v_{r,k}^3$$

where i, j, k denote the indices of the three modes.

CP has been implemented by the TensorLy library for Python, and according to the authors minimal modification is necessary to use it for TensoRF-CP so we do not expect it to take much of development time.

*b) **TensoRF-VM**:* The authors of TensoRF designed a specialized decomposition inspired by block term decomposition (BTD) method that they have found to be more effective for radiance field reconstruction. For a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ the factorization decomposes it into multiple vectors and matrices:

$$\mathcal{T} = \sum_{r=1}^{R_1} v_r^1 \circ M_r^{2,3} + \sum_{r=1}^{R_2} v_r^2 \circ M_r^{1,3} + \sum_{r=1}^{R_3} v_r^3 \circ M_r^{1,2}$$

where $M_r^{2,3} \in \mathbb{R}^{J \times K}$, $M_r^{1,3} \in \mathbb{R}^{I \times K}$, and $M_r^{1,2} \in \mathbb{R}^{I \times J}$ are matrix factors for two of the three modes.

In the context of radiance fields, the three modes correspond to the XYZ axes, and $R_1 = R_2 = R_3$ as most scenes are equally complex along their three axes. Thus, decomposition can be rewritten as

$$\mathcal{T} = \sum_{r=1}^{R} v_r^X \circ M_r^{Y,Z} + v_r^Y \circ M_r^{X,Z} + v_r^Z \circ M_r^{X,Y}$$

The tensor element $\mathcal{T}_{ijk}$ can be described as:

$$\mathcal{T}_{ijk} = \sum_{r=1}^{R} \sum_{m} A_{r,ijk}^m$$

where $m \in XYZ$, $A^X_{r,ijk} = v^X_{r,i} M^{YZ}_{r,jk}$, $A^Y_{r,ijk} = v^Y_{r,j} M^{XZ}_{r,ik}$, and $A^Z_{r,ijk} = v^Z_{r,k} M^{XY}_{r,ij}$. We expect the implementation of this decomposition technique to take more effort than CP as customized implementation is required, including CUDA code as the calculation of $\mathcal{T}$ is trivially parallelizable.

*c) **Feature grids and radiance field**:* To represent the radiance field in TensoRF, interpolation is performed to match a 3D location x and viewing direction d to a 3D grid G with per-pixel multi-channel features, including $\mathcal{G}_\sigma$ and $\mathcal{G}_c$ to represent volume density and view-dependent color respectively. Thus, radiance fields in implementation will be modeled as

$$\sigma, c = \mathcal{G}_\sigma(x), S(\mathcal{G}_c(x), d)$$

where $\mathcal{G}_\sigma$ and $\mathcal{G}_c$ are represented as factorized tensors storing the trilinearly interpolated features from the grids at location x.

*d) **Factorized radiance fields**:* $\mathcal{G}_\sigma \in \mathbb{R}^{I \times J \times K}$ is a 3D tensor while $\mathcal{G}_c \in \mathbb{R}^{I \times J \times K \times P}$ is a 4D tensor, where I, J, K represent the X, Y, Z grid while P is the number of appearance feature channels. In this context, $\mathcal{G}_\sigma$ is factorized as follows:

Using CP-factorization:

$$G_\sigma = \sum_{r=1}^{R_\sigma} v^X_r \circ v^Y_r \circ v^Z_r$$

Using VM-factorization:

$$G_\sigma = \sum_{r=1}^{R_\sigma} \sum_{m \in XYZ} A^m_{\sigma,r}$$

Meanwhile, $\mathcal{G}_C$ requires an extra dimension for feature channels, so is factorized as follows:

Using CP-factorization:

$$G_\sigma = \sum_{r=1}^{R_\sigma} v^X_r \circ v^Y_r \circ v^Z_r \circ b_r$$

where there are R_c vectors b_r to match the total number of components.

Using VM-factorization:

$$\mathcal{G}_c = \sum_{r=1}^{R_c} A^X_{c,r} \circ b_{3r-2} + A^Y_{c,r} \circ b_{3r-1} + A^Z_{c,r} \circ b_{3r}$$

where there are $3 \times R_c$ vectors b_r to match the total number of components. The total factorization requires $3 \times R_\sigma + 3 \times R_c$ matrices and $3 \times R_\sigma + 6 \times R_c$ vectors, with $R_\sigma \ll X, Y, Z, R_c \ll X, Y, Z$. The appearance feature-mode vectors b_r can be represented as a single global matrix B of size $P \times 3 \times R_c$.

*e) **Feature evaluation**:* Via trilinear interpolation, we will extract continuous fields from the XYZ-mode vector/matrix factor. The interpolated value for a component tensor $A^X_r(x) = v^X_r(x) M^{Y,Z}_r(y, z)$ where $A^X_r(X)$ is A_r's trilinearly interpolated value at location $x = (x, y, z)$, $v^X_r(x)$ is v^X_r's linearly interpolated value at x along the X axis, and $M^{Y,Z}_r(y, z)$ is $M^{Y,Z}_r$'s bilnearly interpolated value at (x, y) in the YZ plane. Thus, trilinearly interpolating the two grids for VM-factorization is expressed as

$$\mathcal{G}_\sigma(x) = \sum_r \sum_m A^m_{\sigma,r}(x)$$

$$\mathcal{G}_c(x) = B(\otimes[A^m_{c,r}(x)]_{m,r})$$

*f) **Rendering and reconstruction**:* The factorized tensorial radiance field is expressed as

$$\sigma, c = \sum_r \sum_m A^m_{\sigma,r}(x), S(B(\otimes[A^m_{c,r}(x)]_{m,r}), d)$$

For volume rendering, the classic technique of volume rendering will be used. This involves marching along a ray for each pixel, sampling Q shading points along the ray and computing the pixel color by:

$$C = \sum_{q=1}^{Q} \tau_q (1 - exp(-\sigma_q \Delta_q)) c_q, \tau_q = exp(-\sum_{p=1}^{q-1} \sigma_p \Delta_p)$$

where σ_q, c_q are the density and color computer by the model at the sampled location x_q, Δ_q is the ray step size and τ_q is the transmittance.

To optimize the algorithm, gradient descent will be used while minimizing L2 rendering loss and including an L1 norm loss and total variation loss regularization terms.

B. DVGO

*a) **Post-activated density voxel grid**:* The authors of DVGO represent querying of the volume density for any 3D position x via a voxel-grid as a series of functions, beginning with an interpolation function:

$$interp(x, V) : (\mathbb{R}^3, \mathbb{R}^{N_x \times N_y \times N_z} \rightarrow \mathbb{R}^1)$$

where x is the queried 3D point, V is the voxel grid, C is the dimension of the modality, and $N_x \times N_y \times N_z$ is the number of voxels. The interpolated value then is used as input to a shifted softplus:

$$\sigma' = softplus(interp(x, V)) = \log(1+\exp(interp(x, V)+b))$$

where b is the shift hyperparemeter. Finally, post-activation is used:

$$\alpha = 1 - \exp(-\sigma' \delta)$$

where δ is the distance to the adjacent sampled point (step size when marching along ray).

*b) **Coarse Geometry Searching**:* In DVGO, the scene is represented using coarse geometry for areas dominated by free space. The coarse density voxel grid can be represented as:

$$V^{(density)(c)} \in R^{N_x^{(c)} \times N_y^{(c)} \times N_z^{(c)}}$$

where (c) superscript denotes coarse.

A query of any 3D point is represented as

$$\sigma'^c = interp(x, V^{(density)(c)})$$

$$c^{(c)} = interp(x, V^{(rgb)(c)})$$

A hyperparameter $M^{(c)}$ denotes the expected total number of voxels in the coarse stage is used to determine the size of the bounding box separating the coarse from fine regions; using it the voxel size can be calculated:

$$s^{(c)} = \sqrt{^3 L_x^{(c)} \times L_y^{(c)} \times L_z^{(c)} / M^{(c)}}$$

where $L_x^{(c)}, L_y^{(c)}, L_z^{(c)}$ are the lengths of the bounding box. From that, the number of voxels on each side of the bounding box can be calculated:

$$N_x^c, N_y^c, N_z^c = \lfloor L_x^{(c)}/s^{(c)} \rfloor, \lfloor L_y^{(c)}/s^{(c)} \rfloor, \lfloor L_z^{(c)}/s^{(c)} \rfloor$$

For coarse geometry, training begins with all grid values in $V^{(density)(c)}$ to 0 (to represent low importance) and set the shift hyperparameter b:

$$b = \log((1 - \alpha^{(init)(c)})^{-\frac{1}{s^{(c)}}} - 1)$$

where $\alpha^{(init)(c)}$ is a hyperparamater.

The learning rate for different points $v_i \in V^{(density)(c)}$ is determined by counting the number of training views n_i to which point v_i is visible, then scale its base learning rate by n_i/n_{max}, where n_{max} is the maximum view count over all grid points.

Reconstruction is done similarly to in TensoRF, with L2 loss being used with stochastic gradient descent and background entropy loss is used for regularization.

*c) **Fine Detail Reconstruction**:* Once the coarse geometry $V^{(density)(c)}$ is found, it is used as a component to finding the higher-resolution density voxel grid $V^{(density)(f)} \in \mathbb{R}^{N_x^{(f)} \times N_y^{(f)} \times N_z^{(f)}}$

where the (f) superscript is used to denote fine geometry. The volume density is found similarly to before:

$$\sigma'^{(f)} = interp(x, V^{(density)(f)})$$

However, an additional find voxel space for feature representation is added associated $V^{(feat)(f)} \in \mathbb{R}^{D \times N_x^{(f)} \times N_y^{(f)} \times N_z^{(f)}}$, where D is a hyperparameter representing feature-space dimension. Then, a shallow MLP parameterized by Θ which takes as input the interpolated volume, position, and viewing direction is used:

$$c^{(f)} = MLP_\Theta^{(rgb)}(interp(x, V^{(feat)(f)}), x, d)$$

To find which areas of $V^{(density)(c)}$ should be better represented in $V^{(density)(f)}$, the coarse geometry is queried to find the unknown space in points where the density found is greater than $\tau^{(c)}$. A bounding box for the unknown space with hyperparameter $M^{(f)}$ can be found in the same way as for the coarse stage.

Progressive scaling is applied to the voxel grids $V^{(density)(f)}$ and $V^{(density)(f)}$ by initially setting the number of voxels to be $\lfloor M^{(f)}/2^{|p|} \rfloor$ where p is the number of training checkpoints and doubling the number of voxels (via trilinear interpolation) at each checkpoint, such that after p steps the number of voxels is $M^{(f)}$.

Once the fine geometry has been calculated, some areas may still be free within; those can be skipped in both training and testing by ignoring points with density less than $\tau^{(f)}$

Training is performed in the same way as in the coarse stage except for a smaller regularization term.

C. Implementation Environment

The previously existing implementations of TensoRF and DVGO use the following python libraries: PyTorch, NumPy, OpenCV, ImageIO, SciPy, and miscellaneous utility libraries as viewable in their repositories. So, we will use these libraries for our implementation.

For experimentation, we will use the cuda environment made available by the school of computing at uga.

III. Schedule

The deadline for the project technical report is on May 1st. This leaves 7 weeks between March 12th and the technical report deadline.

To complete the project, we can break it into multiple steps: implementation, experimentation, and analysis. Implementation and experimentation will take much longer than analysis. Experimentation time can take anywhere around one or two weeks. So, to provide ourselves with two weeks to analyze our results and two weeks to experiment, we should complete the implementation by week 3 after submitting this proposal.

Orr will be mainly responsible for the implementation of DVGO, Porter will be mainly responsible for the implementation of TensoRF, and Lihao will be primarily responsible for experimentation and analysis, although we expect some degree of overlap in work. Work will be tracked in GitHub and Overleaf.

The following will be our schedule:

March 13th - April 3rd • Implementing DVGO and TensoRF.

April 3rd - April 17th • Experiment with DVGO and TensoRF on Synthetic-NeRF dataset.

April 17th - April 30th • Write technical report explaining analysis of results from experimentation.

May 1st • Ensure technical report is submitted.

IV. Work Done

GitHub for implementation at https://github.com/Pyerter/uga-csci8820-project-cv

References

[1] C. Sun, M. Sun, and H.-T. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2022. doi:10.1109/cvpr52688.2022.00538

[2] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "Tensorf: Tensorial radiance fields," Lecture Notes in Computer Science, pp. 333–350, Mar. 2022. doi:10.1007/978-3-031-19824-$3_2$0

[3] B. Mildenhall et al., "NeRF: Representing scenes as neural radiance fields for view synthesis," Computer Vision – ECCV 2020, pp. 405–421, 2020. doi:10.1007/978-3-030-58452-$8_2$4