UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Split Activation Networks for Neural Fields

Author:
Mykhailo KILIANOVSKYI

Supervisor:
Oles PETRIV

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2023

Declaration of Authorship

I, Mykhailo KILIANOVSKYI, declare that this thesis titled, "Split Activation Networks for Neural Fields" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Study hard what interests you the most in the most undisciplined, irreverent and original manner possible."

Richard Feynman

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Split Activation Networks for Neural Fields

by Mykhailo KILIANOVSKYI

Abstract

Neural field modeling is a developing area that improves state-of-the-art results in tasks such as 3D scene reconstruction, image manipulation, generative modeling, and other aspects of deep learning. In this work, we present SplitNet, a novel neural network architecture for neural field modeling that combines multiple activation functions in a single layer. We try different techniques to improve performance, such as proper weight initialization, and benchmark its performance on image representation, 3D scene reconstruction, and image classification tasks. As a part of the work, we found a way to improve the performance of previous work on implicit neural networks with sinusoidal activations in a limited setting and study how well this improvement generalizes to other tasks and data.

Acknowledgements

Many thanks to those fighting for our freedom against tyranny and totalitarianism in the Russia-Ukraine war, making it possible to live our normal lives under their protection from evil. Thanks to Oles Petriv for his support and for sharing unique ideas. Thanks to Reface for donating computational resources for educational purposes. Many thanks to the organizers of a Master's Program on Data Science at UCU for creating such a lovely learning environment.

Contents

List of Figures

List of Tables

List of Abbreviations

MLP	**M**ulti **L**ayer **P**erceptron
NERF	**N**eural **R**adiance **F**ield
CNN	**C**onvolutional **N**eural **N**etwork
LSUV	**L**ayer-**S**equential **U**nit-**V**ariance
PSNR	**P**eak **S**ignal-to-**N**oise **R**atio
SIREN	**SI**nusoidal **RE**presentation **N**etworks
MSE	**M**ean **S**quared **E**rror
GPU	**G**raphics **P**rocessing **U**nit

Chapter 1

Introduction

1.1 Domain description

Deep learning has revolutionized many fields with its ability to learn complex patterns from data. One area where progress was recently made is the modeling of neural fields (Xie et al., 2022). This conceptual framework advanced the practical areas such as image representation (Sitzmann et al., 2020, Martel et al., 2021, Dupont et al., 2022), generative modeling (Skorokhodov, Ignatyev, and Elhoseiny, 2021) 3D scene reconstruction (Mildenhall et al., 2020, Gao et al., 2022) and beyond. The nature of neural field modeling challenges established deep learning tools and leads to researching and devising new paradigms and methods, such as positional encoding (Mildenhall et al., 2020) and periodic activation functions (Sitzmann et al., 2020). In this work we present the SplitNet, novel architecture for neural field modeling. Since previous work showed that activation functions are crucial in this area, SplitNet main novelty is usage of 4 activation functions in one layer, namely hyperbolic tangent, sigmoid, sine and cosine. The objectives of this work are to study the properties of SplitNet architecture and its components, evaluate and improve performance. The broader goal is to gain more understanding of processes that occur during neural field modeling and deep learning in general.

To achieve these objectives, we are aimed to establish a representative benchmark and compare performance to relevant baseline from previous work. Also, we are going to study the behavior of the network during training and apply methods to improve gradient flow health, such as proper weight initialization.

The structure of this work is the following.

This is the **Chapter 1** that makes the brief overview of the work.

Chapter 2 sets the stage by reviewing what is done in the field. First, it reviews neural field modeling and its application. Also, this chapter highlights the relevance of SplitNet research and references literature to support this claim. The last section reviews neural image representation task and motivates its usage for the initial benchmark.

Chapter 3 delves deeper into neural field modeling and sinusoidal activation functions. Sinusoidal networks are a subset of SplitNet, and this chapter describes experiments to improve the performance of the former.

Chapter 4 formally defines SplitNet architecture and describes initial research on improving its performance.

Chapter 5 delves deeper into designing SplitNet and devising a weight initialization strategy.

Chapter 6 presents early results in applying Split networks to other tasks, namely image classification and 3D scene reconstruction.

Chapter 7 concludes the work done in this work and proposes directions for future work.

Chapter 2

Related Work

In this chapter, we set up the stage for our work and define a context. This chapter is structured as follows:

1. Overview of work on neural fields and their broad applications
2. Review implicit neural networks with periodic activations and highlight their importance
3. Describe current attempts of using neural representations as primary data format
4. Overview the role of small multilayer networks in modern deep learning pipelines as the motivation of this research
5. Introduce image representation task and motivate its usage as a main task to do analysis on

2.1 Neural Fields

We refer to a field as a concept that describes how certain quantities, such as force or energy, are distributed in space and/or time. A field can be defined as follows:

A field is a quantity defined for all spatial and/or temporal coordinates. (Xie et al., 2022)

Examples of fields in physics include electric fields, magnetic fields, and gravitational fields. Many phenomena of interest, such as images or 3D scenes, can also be formulated as fields. Images, for instance, can be described as a field of pixel intensities. More specifically, it can be seen as a mapping from 2D pixel coordinates to the RGB color. 3D scenes can be thought as a mapping of spatial locations (xyz coordinates and/or viewing direction) to color or opacity.

Natural fields in most cases have complex underlying structure, and it is infeasible to model them analytically. Neural networks, on the other hand, can be thought of as universal function approximators, giving a practical tool to approximate and model different phenomenas formulated as a field.

Neural fields have gained a lot of attention with successful work on neural radiance fields (NERFs) Mildenhall et al., 2020. NERF achieved state-of-the-art results in modeling 3d scenes and novel view synthesis. A scene is modeled volumetrically with multi-layer perceptron (MLP). It takes 5 coordinates as an input (x, y, z spatial coordinates and viewing angles θ, ϕ) and returns RGB colors and density σ. Radiance along each camera ray is then accumulated with respect to density σ to get the final color of a pixel. Model is scene-specific (MLP is learned for a specific scene, and scene becomes "baked" into network).

Follow-up studies go on improving all aspects of NERFs, such as training speed, inference speed, memory footprints and multi-scene generalization. We refer

to the recent extensive survey on the application of neural fields in visual computing for more details Gao et al., 2022.

2.2 Implicit neural networks with periodic activation functions

The big stepping stone in developing neural fields is the research of implicit neural networks with periodic activation functions (SIREN) Sitzmann et al., 2020. First, authors investigated the spectral bias problem and found out that standard activation functions, such as ReLU or hyperbolic tangent (tanh) fail to reconstruct fine details. One of the possible solution to this is the usage of positional encoding Mildenhall et al., 2020, such that the frequency of the input signal is artificially boosted. But as authors of the SIREN showed, this is not enough and Relu networks with positional encoding still underperform in capturing high-frequency details. Moreover, such networks fail to capture the first and second derivatives of the signal. This prevents them from successful application in tasks where neural fields are operated as differential equations, such as in Poisson image reconstruction or solving the wave equation. To address this problem, the authors proposed the usage of sinusoidal activation functions. It superior performance was empirically verified and compared with other approaches on the number of tasks for neural fields modeling, such as neural image representation, audio representation, video representation, solving Eikonal equation, Poisson image reconstruction, image inpainting etc. One of the main components of the SIREN network is proper weight initialization. It allows to keep the distribution of the activations at initialization constant regardless of the number of layers. This eliminates the problem of vanishing and exploding gradients. Also, an additional constant factor ω_0 was introduced to the weight matrix of the linear layer: $\mathbf{W} = \hat{\mathbf{W}} * \omega_0$. It allows to boost the gradients to the weight matrix while preserving the original distribution of the activation. This work explains in detail the process of devising the new activation function for neural fields, deriving the proper weight initialization and gradient flow, and the procedure of empirical qualitative evaluation of the performance.

2.3 Neural representations as data format

Traditional signal representation include 3D tensors of RGB values for images, meshes, point clouds or voxel grids for 3D scenes (Martel et al., 2021). While these data formats are well studied, they may be suboptimal for modern applications and impose limitations. Emerging applications often include optimization or deep learning models on some stages, and properties of a way the data is represented in such pipelines are very important. Desired properties include end-to-end differentiability, low memory requirements and scalability in learning-based pipelines. Explicit approaches in most cases scale poorly and require preprocessing (that could be a bottleneck) before using in deep learning pipelines.

There is an upcoming trend in finding ways to use neural representations as primary data format. The work Davies, Nowrouzezahrai, and Jacobson, 2021 explores the neural networks as a first-class 3D shape representation. Authors note that weight-encoded signals previously was overlooked, and argue that neural representations are competitive with classic approaches to storing and manipulating 3D

objects, such as meshes or point clouds. This method can be viewed as lossy compression, trading off memory for computation, that can be useful in practical applications. A similar approach is described in work on functas Dupont et al., 2022 - doing the downstream tasks on implicit data representations. Here, authors decouple dataset creation and learning into separate stages. At the first stage, for each point of the dataset, they fit the implicit neural network. To reduce computational and storage costs, meta-learning is applied. On the second stage, downstream tasks are solved directly on implicit representations. The authors found out compelling properties of this approach, such as amortized memory scaling, removing the problems of different resolutions, solving problems with discretization and improving the optimization on downstream tasks. While this approach does not beat state-of-the-art, it paves out the way to the promising direction of implicit data representation.

Replacing traditional data representations with neural representations is a promising direction and already have practical and commercial applications in some areas such as 3D rendering and computer graphics. Implicit neural representations are also easier to integrate into existing deep learning pipelines. One of the main pros of implicit neural representation is plausible memory consumption. Coordinate-based networks can overcome unaffordable memory rate of growth, for example $O(n^3)$ for 3D scene. However, there are fundamental challenges of using them as a drop-in replacement building block in traditional pipelines. One of the main challenges is high computational cost that was traded for lower memory consuption. In a straightforward setting, implicit network should run forward pass for each input coordinate (that is, every point in a 3D space or every image pixel). With large networks, such pipelines could take a lot of time to train and inference. Therefore, more efficient, compact and fast approaches should be developed, and this is one of the main concerns of our research.

2.4 Small neural networks in modern pipelines

We claim that small Multilayer Perceptrons (roughly less than 5 layers, less than $200,000$ parameters) are an important part of modern state-of-the-art applications of different kinds. Therefore, understanding and improving the performance of such networks is a potential orthogonal improvement to many tasks.

Here are a couple of examples to support our claim.

TensoRF. (Chen et al., 2022a). This work presents a novel way to reconstruct 3D scenes via tensor decomposition, that makes training and rendering time much lower. A cruicial component of this pipeline is a small neural network to decode latent features to the final RGBσ values. Quote from the paper:

> *"For neural features, we use a small MLP with two FC layers (with 128-channel hidden layers) and ReLU activation."*

In further chapter, we make a more detailed overview and benchmark SplitNet agains MLP used in the paper.

MobileNeRF (Chen et al., 2022b). This work presents a way to integrate neural radiance fields into rasterization pipeline, making possible to run NERFs inside on even small mobile devices. Small MLP is used in final stages to model light effects and is implemented with shader. Quote from the paper on its importance:

> *"Without the small MLP, the model cannot handle reflections, as shown in (i)."*

2.5 Implicit neural image representation

Implicit neural image representation is an approach to modeling images in machine learning that uses neural networks to implicitly define the pixel values of an image, rather than storing the image as an explicit grid of pixels. With implicit representation, an image is not represented directly in memory. Instead, a function (in this case, a neural network) models it implicitly, taking as input the coordinates of a pixel and outputting the color of that pixel.

Connection of neural image representation to practical tasks. Despite the success of implicit neural representations in 3D computer vision, this approach has been under-explored in the domain of 2D imaging. Neural image representation is the first step in new architectures for downstream image tasks that may have more pleasing properties or even be competitive to state-of-the-art solutions. There are architectures that are using neural image representation for solving denoising, superresolution and inpainting problems (Czerkawski et al., 2021, Sitzmann et al., 2020, Skorokhodov, Ignatyev, and Elhoseiny, 2021) or image compression Martel et al., 2021.

Motivation to use neural image representation as a main task in this work. Besides direct downstream practical applications, neural implicit representation task offers a suitable benchmark to test the fundamental properties of neural networks and neural fields in a simple setting. Other tasks of neural field modeling can have computational requirements too high which makes certain experiments too costly or even impossible. For example, the first NERF model can take up to 12 hours to train (Mildenhall et al., 2020)) a representation of a single scene, modeling a short video can take 15 hours of neural network training (Sitzmann et al., 2020). However, as was shown in previous work, improvements to the architecture that was found to perform well in one task of neural field modeling (for example, image representation), can also improve performance on other tasks (for example, 3D scene representation). The architectures we test in this work are not task-specific. Therefore, this performance correlation is the main motivation to test most of the hypotheses since it makes it possible to run more experiments cheaper.

Metric. Metric the most works use to assess the quality of a reconstructed image is Peak Signal-to-Noise Ratio (**PSNR**) (Mildenhall et al., 2020, Chen et al., 2022a, Sitzmann et al., 2020, Martel et al., 2021). PSNR is calculated based on the mean squared error (MSE) between the pixel values of the original image and the reconstructed image. The "peak signal" refers to the maximum possible pixel value in the image (for instance, in an 8-bit image this would be 255, or if the image is normalized to the range [0, 1] this would be 1), and the "noise" refers to the difference between the original and the reconstructed image. Formula to calculate PSNR with units in dB (decibels) is in Equation 2.1 where MAX_{pixel} is the maximum possible signal value and MSE is the mean squared error of the reconstruction:

$$PSNR = 20 \times log_{10}(MAX_{pixel}) - 10 \times log_{10}(MSE) \tag{2.1}$$

This formula is derived from the definition of decibel. The decibel (dB) is a logarithmic unit used to express the ratio of two values of a physical quantity, often power or intensity. It's defined as 10 times the logarithm (base 10) of the ratio of the two power quantities. From this definition, derivation of Equation 2.1 is described in Equation 2.2

$$PSNR = 10 \times log_{10} \frac{MAX_{pixel}^2}{MSE} = 20 \times log_{10} \frac{MAX_{pixel}}{\sqrt{MSE}} \tag{2.2}$$

The main drawback of this metric is poor correlation with human perception. However, we are more interested in the amount of signal the model is able to reconstruct, and that aspect is captured by PSNR. Calculating this metric is computationally cheap and makes our results comparable with previous work. Also, this metric is not image-specific and has a normalization term with regard to signal scale, which makes this metric comparable across different modalities and tasks of signal reconstruction.

Chapter 3

Sinusoidal activations

3.1 Introduction

Neural networks with periodic activation functions are an important stepping stone in understanding and developing neural fields. This chapter overviews the relevant background and describes experiments and tools on improving SIREN performance, that later would be used for designing Split networks. Codes for this and further experiments is available on public GitHub repository [1].

The objectives of this chapter are the following:

- Overview previous work on sinusoidal activation functions (SIREN) as an important stepping stone in neural fields
- Highlight parts that are relevant to the designing of a new family of implicit neural networks, such as details of deriving a good initialization strategy
- Present our putative contribution, which is the improved performance of sinusoidal implicit neural networks via empirically tuned initialization
- Highlight limitations and possible future work in this direction, as well as formulate further research hypotheses and describe their relevance

3.2 Background

The key feature of SIREN is its use of sinusoidal activation functions, which allows it to model periodic patterns and oscillations that are common in many types of data, such as images, sound, and other types of signals. This is in contrast to most other neural network architectures, which use non-periodic activation functions like ReLU or sigmoid.

One of the main advantages of SIREN is its ability to model fine details and high-frequency components in data. This makes it particularly well-suited for tasks like image and audio synthesis, 3D shape modeling, and solving partial differential equations.

3.2.1 SIREN initialization scheme

Another important aspect of SIREN is its initialization scheme, which is designed to ensure proper convergence. It makes SIREN networks easier to train and less prone to issues like exploding or vanishing gradients.

[1] https://github.com/kilianovski/my-neural-fields

Authors of SIREN devise the initialization scheme and claim that it is crucial for network performance. Indeed, our experiments confirm that using default initialization schemes like standard normal or Kaiming (He et al., 2015b) lead not only to worse results but the complete halting of the network's training.

The key idea of the proposed principled initialization scheme is to preserve the distribution of the activations without dependence on the network's depth. When such property is satisfied, a network of arbitrarily chosen depth and width is supposed to avoid problems of vanishing/exploding gradients, at least at initialization. The main argument relies on the assumption that if the input to a neuron in a layer has the same distribution as its output, then the distribution will be preserved across the network.

Creating an initialization scheme that retains activations consistently across all layers requires a thorough understanding of both the activation functions that are used and the distribution of the network's input.

In the case of SIREN and neural fields in general, the input to the network is usually the normalized coordinates. The distribution of such coordinates is uniformly distributed between -1 and 1. After applying sine nonlinearity, authors come up with a conclusion that the output of a such neuron is arcsine distributed (a variant of U-shaped Beta distribution). With this fact and control over the distribution of the weights, authors come up with a sampling range for the weights, such that the property of distribution preservation is satisfied.

The derivation treats the first layer separately since it receives uniform input in the range $[-1, 1]$ and returns the output that is arcsine-distributed. The second and consequent layers map arcsine distribution to the arcsine distribution. Recursive argument can be applied here, if the input distribution of a neuron is the same as the output (both in terms of distribution family and concrete parameters).

In this setting, the task of finding an appropriate initialization can be formulated as finding parameter q that defines the sampling distribution for the weights: $U(-q, q)$[2]. Note, that q is usually a function of the number of the input parameters n (also called fan_{in}) $q(n)$. This is due to the nature of the dot product in the linear part of a neuron: the larger the number of input parameters, the larger the dot product grows. The parametric form of the desired distribution is the following: $U(\frac{-c}{\sqrt{n}}, \frac{c}{\sqrt{n}})$. With this scheme, authors show and prove that the dot product (preactivation value) follows such normal distribution: $N(0, \frac{c^2}{6})$. Authors decide to keep the preactivation value to be standard normally distributed, thus the final initialization looks the following: $U(-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}})$. First layer is initialized from another distribution: $U(-\frac{1}{n}, \frac{1}{n})$ We refer to the original work for more detailed and formal derivation. We empirically find that on specific families of tasks, such as image representation, other values of initialization parameter c lead to better performance and analyze the limitation of our finding.

[2]We do not consider sampling from the normal distribution for simplicity. The exact same line of reasoning can be applied to it, and in practice uniform and normal sampling strategies are used interchangeably

experiment name	PSNR first	PSNR 2999	position first	position 2999
initc=24__flic=2	6.3765	**32.8515**	5	**0**
initc=6__flic=2.0	6.6530	32.5486	8	1
initc=6__flic=4.0	6.6553	32.1855	10	2
initc=24__flic=1	6.3698	31.8647	13	3
initc=12__flic=1	6.5677	31.7728	1	4
initc=42__flic=1	6.3570	31.6754	9	5
initc=6__flic=1.0	6.6833	31.7276	4	8
initc=6__flic=1	6.6833	31.7276	2	7
initc=64__flic=1	6.2974	31.2727	12	6
initc=3__flic=1	6.5945	31.1234	11	9
initc=6__flic=16.0	6.6615	30.5229	3	10
initc=6__flic=0.5	6.6441	30.3218	7	11
initc=1__flic=1	6.4423	30.2109	6	12
initc=6__flic=0.1	**6.7710**	25.8888	**0**	13

TABLE 3.1: SIREN learning dynamics. Comparison of different initialization models. This data brings the evidence that performance on initialization of the same architecture cannot be directly used to compare performance after the full training.

3.3 Experiment setup

The goal of our experiment is to investigate initialization strategies for SIREN and their impact on network convergence. We follow the original SIREN form of initialization, and vary two scaling coefficients (gains): $flic$ (stands for *first layer initialization coefficient*) and $initc$ (stands for *initialization coefficient*). The final parametric form we optimize is presented in equations (3.1) and (3.2):

$$W_0 \sim U(-\frac{flic}{n}, \frac{flic}{n}) \tag{3.1}$$

$$W_i \sim U(-\sqrt{\frac{initc}{n}}, \sqrt{\frac{initc}{n}}) \tag{3.2}$$

Hyperparameters. We chose a lightweight setup in favor of speed and number of experiments we can run. Also, we are interested in finding out if results, found on simple task setting, overfit to this setting or can be generalized to broader network architectures and tasks. We describe these experiments in later sections. We train on a 256x256 grayscale image of a Cameraman [3] . Network has 3 layers, outermost layer is set to linear. Adam optimizer is used with learning rate 1×10^{-4} and no weight decay. We train for 300 steps and justify this number as a decent tradeoff between speed and accuracy with the prestudy described in the next section.

3.4 Prestudy

Before running a full-scale parameter search, we conducted an exploratory analysis of SIREN learning dynamics. To cover a vast space of parameters, we run a dozen of experiments with different initialization for 3000 steps. Results are summarized in Table 3.1. One of the findings is that performance on the initialization is a bad

[3] https://scikit-image.org/docs/stable/api/skimage.data.html#camera

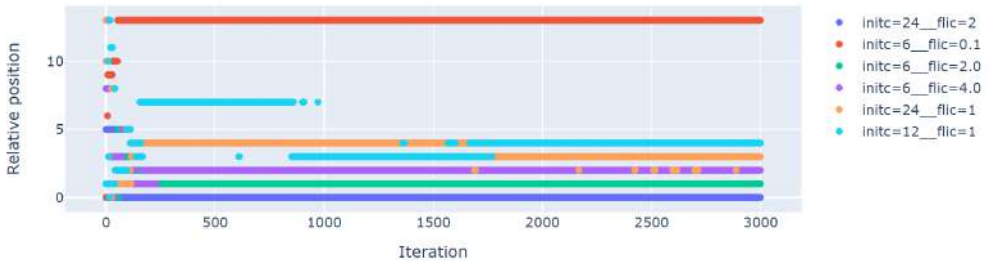

FIGURE 3.1: SIREN learning dynamics. The plot demonstrates relative training performance for different initialization models. Y-axis represents the position in comparison to other models (lower is the better). We selected the subset of models for visualization. It can be seen that after the 500 iterations, there is no large oscillation and models have found their place in terms of relative performance.

indicator of overall network performance. As we observe, networks with the most promising performance on the initialization end up being much worse than the networks with relatively bad PSNR scores. Therefore, we can't use too low number of training steps to empirically compare the effectiveness of initialization.

To select the optimal number of training steps, we visualize the relative position of networks during training, see Figure 3.1. The goal of this plot is to observe how the "leaderboard" is evolving during training. Surprisingly, the network with the best relative position after full training end up with the lowest score. However, this position transition happens very quickly, during the first 50 training steps. From this plot we observe, that after 1000 iterations there is no change in position except for local oscillations and after the first 200 iterations almost all large shifts in position are done. Therefore, we decide that 300 iterations for our hyperparameter search is a good tradeoff due to limited computational resources. This should allow promising initializations to unfold their potential, and, from the other side, to enable higher number of experiments.

3.5 Parameter search

In this study, we perform a hyperparameter serach to optimize the SIREN network's initial weights. We initialize the weights according to equations (3.2) and (3.1). Based on prestudy, parameter $flic$ was searched with the uniform distribution in range $[0.01, 10]$, parameter *initc* was searched with the uniform distribution in range $[1, 100]$. These ranges were chosed to cover a wide spectrum of possible values, from the obviously low to obviously high values, to ensure a comprehensive search. We use wandb platform (Biewald, 2020) for tracking experiments.

We search for parameters with a random search. In comparison to Bayesian search, we believe it is more robust for our setting of low number of hyperparameters and opportunity to do brute force efficiently. In comparision to grid search, it is more convenient since we can stop sweep any time and have approximately representative results.

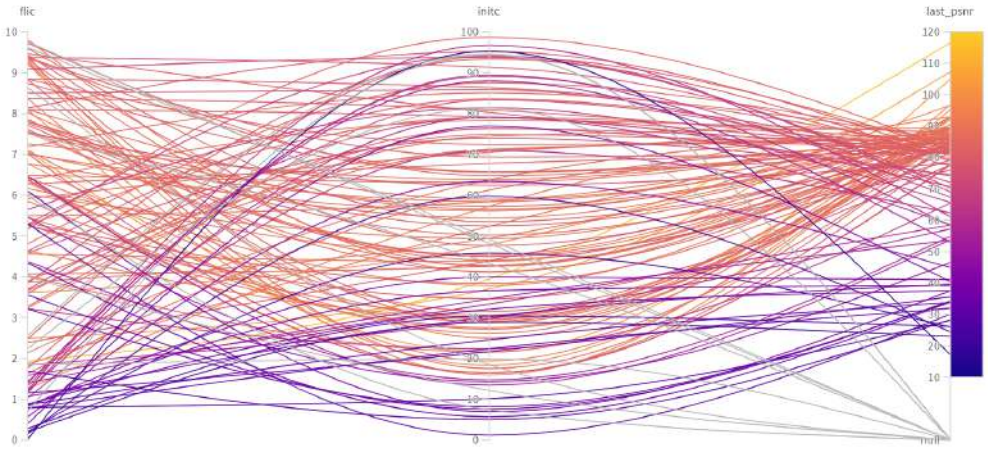

FIGURE 3.2: SIREN hyperparameter sweep visualization. This plot shows the coverage of parameters searched (first and second columns): both parameters are sampled from uniform distributions. Also, it shows the skewness of the distribution of the PSNR scores (last column). Grey arrows represent experiments that failed due to technical reasons.

.

3.6 Result

Distribution of the final scores is depicted on Figure 3.2. As we can see, the best initialization allows achieving PSNR score more of the 110, and the most of the probability mass, when sampling uniformly in the defined range, is concentrated between 80 and 90 PSNR. This visualization allows to quickly healthcheck if the software is running correctly in terms of uniformly covering the sampling interval.

Five best and five worst runs in terms of final PSNR score are presented in Table 3.2. We can make these conclusions from the data we have:

- First layer initialization coefficient ($flic$) is highly suboptimal in low values (probably below 1).

PSNR last	flic	initc
116.58	1.93	39.44
107.32	2.49	36.98
105.00	7.07	21.49
96.85	8.09	18.30
96.64	3.61	47.86
26.14	0.28	30.36
25.94	0.21	45.84
24.46	0.15	59.70
22.96	0.20	28.86
17.10	0.02	95.25

TABLE 3.2: SIREN initialization tunning results. Displaying 5 first and 5 last results, sorted by performance. $flic$ is the multiplier

Layer name	Baseline init distribution σ	Our init distribution σ
input	0.58	0.58
layer 0 preact	14.14	18.99
layer 0 act	0.71	0.71
layer 1 preact	1.51	2.79
layer 1 act	0.72	0.71
layer 2 preact	1.49	2.81
layer 2 act	0.72	0.71
layer 3 preact	1.46	2.81
layer 3 act	0.71	0.71
layer 4 out	0.03	0.09

TABLE 3.3: Comparison of standard deviations of activation distributions of the network we use for parameter search. Baseline distribution corresponds to the original initialization strategy used in Sitzmann et al., 2020. Our distribution corresponds to the best initialization strategy we found, namely with $flic = 2$ and $initc = 40$. Baseline initialization is equivalent to $flic = 1$ and $initc = 6$. Means are approximately zero for all cases and are not displayed there. Preactivation values are approximately Gaussian distributed.

- Upper limit looks very flexible though, with optimal values lying from values below 2, to values above 8.
- Making the same speculative conclusions on the optimal range of init coefficient for other layers (*initc*) looks harder. We can only say that it is likely that these two coefficients have nonmonotonic relations, and the model looks more sensitive to $flic$ parameter.
- This also is probably related to the fact that the initialization range of a layer roughly grows with $flic$ in $O(n)$ rate, while it grows with *initc* in $O(\sqrt{n})$ rate.

As a sanity check, we look at how these found parameters affect the distribution of the weights at initialization. To do this, we feed-forward the full dataset through the network at initialization and record intermediate activation at each layer. Comparison with baseline initialization is presented in Table 3.3. *preact* stands for values after the feedforward layer before the nonlinearity. As we can see, the main difference is in preactivation distributions: they are wider (have higher variance). Activation distributions are almost not affected. We hypothesize that this difference improves gradient flow to the weights and leave further investigations to future work.

Summary. It's clear from the results that the choice of the initialization parameters significantly affects the performance of the SIREN network. With empirically chosen parameters, we improved PSNR performance more than twofold. But the main caveat is in the simple setup: we used a small network fitting a small image in the hyperparameter search. And it is unclear if this finding is more general. Therefore, we formulate the research hypothesis as follows: "Initialization parameters $flic = 2, initc = 40$ of a SIREN network lead to better performance in comparison to baseline initialization $flic = 1, initc = 6$ on other tasks and other network architectures".

3.7 Generalization analysis

To test the hypotheses of generalization as formulated above, we conduct experiments to check if found initialization leads to better performance on other tasks with other network architectures. In particular, we conduct experiments in this settings:

- Image representation of different image with different network architectures
- Audio signal representation
- Video representation

3.7.1 Different image representations

FIGURE 3.3: SIREN image representation benchmark. Models with hidden sizes of 1024 and 512 fit the image almost perfectly. Artifacts in smaller networks differ between initialization. While the networks with baseline init produce blurry results, our initialization leads to white noise artifacts.

.

Model name	PSNR@3000	MAX(PSNR)	N of parameters
initc=6__flic=1__nh=3__h=1024__baseline	34.37	40.23	3154947
initc=40.0__flic=2.0__nh=3__h=1024	**46.22**	**108.91**	3154947
initc=6__flic=1__nh=3__h=512__baseline	30.66	30.97	791043
initc=40.0__flic=2.0__nh=3__h=512	**43.57**	**45.54**	791043
initc=6__flic=1__nh=3__h=256__baseline	24.82	24.82	198915
initc=40.0__flic=2.0__nh=3__h=256	**27.11**	**27.12**	198915
initc=6__flic=1__nh=3__h=128__baseline	**21.71**	**21.71**	50307
initc=40.0__flic=2.0__nh=3__h=128	21.62	21.62	50307

TABLE 3.4: Comparison of our initialization scheme with the original SIREN initialization across different network sizes. PSNR@3000 refers to the Peak Signal to Noise Ratio after 3000 training steps, while MAX(PSNR) refers to the highest PSNR achieved throughout training. N of parameters refers to the total number of trainable parameters (weights and biases) in the model.

.

Data We fit on a 512x512 crop of Gigapixel Tokyo image [4]. This image contains $\frac{512\times512\times3}{256\times256} = 786432$ times more training points than the grey image we tuned parameter on.

Architecture. We use an MLP with 3 hidden layers, and vary the width of a hidden layer, using values [128, 256, 512, 1024].

Hyperparameters. We use learning rate of 1×10^{-4} and train for 3000 epochs. We run each experiment twice with two different random seeds, to ensure the training is stable and the effect of randomness is negligible for score comparison.

In this setting, our results demonstrate that the tunning of the initialization scheme leads to substantial improvements in performance for different dataset and different network sizes. Table 3.4 illustrates that proposed initialization parameters outperform the baseline initialization for almost all network sizes. Noticeably, the proposed initialization provides significant performance improvements in networks with higher capacity (those with more parameters). The PSNR performance gap becomes smaller with the decreasing width. For the smaller networks performance is less prominent or even worse, but it still remains competitive.

Visual comparison for the experiment is demonstrated in Figure 3.3. Subjectively all images in the top row have better perceptual quality (proposed init scheme). Also, an interesting observation is the nature of noise. Underfit of networks with baseline initialization scheme reveals itself through the blurriness of a fitted image, while networks with proposed initialization parameters have white noise artifacts.

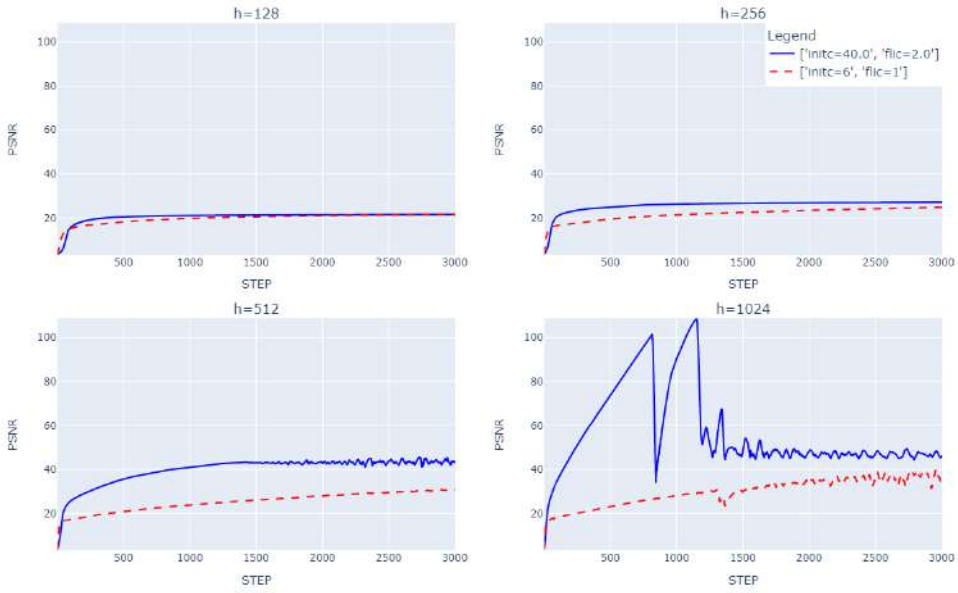

FIGURE 3.4: SIREN image fitting loss curves for different network sizes

.

Loss curves for this benchmark are visualized in Figure 3.5. We can observe, that the proposed initialization gives the most boost at the beginning of the training, and converges to a similar or better PSNR score at the end. Convergence of the largest network demonstrates instabilities. To add more evidence that final results are comparable and depend more on the initialization strategy and other hyperparameters

[4] 1.2 Gigapixel Panorama of Shibuya in Tokyo, Japan by Trevor Dobson.

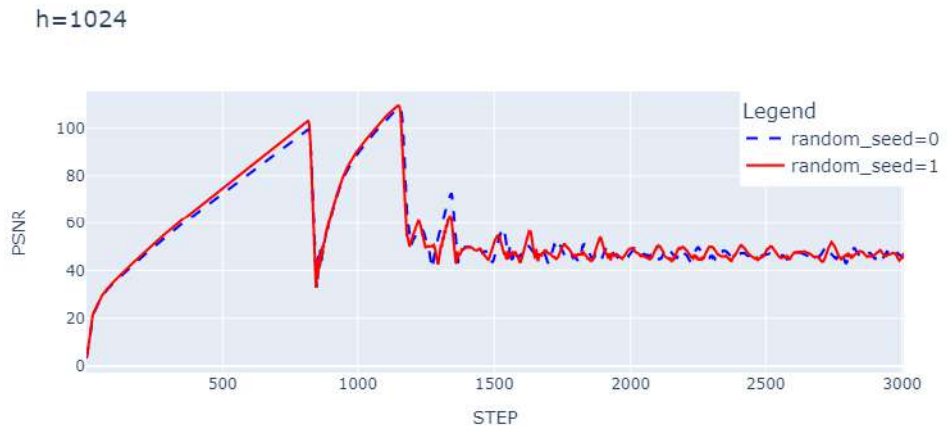

FIGURE 3.5: SIREN image fitting loss curves for network with hidden size 1024 for different random seeds. While the overall loss curves demonstrate instability, the trajectory remains stable across different random seeds.

.

than on randomness, we plotted the loss curves of the same model training with two random seeds in Figure 3.5. Loss curves of both runs are very close. Therefore, the effect of randomness does not influence our conclusions in a significant way.

3.7.2 Representing audio signal

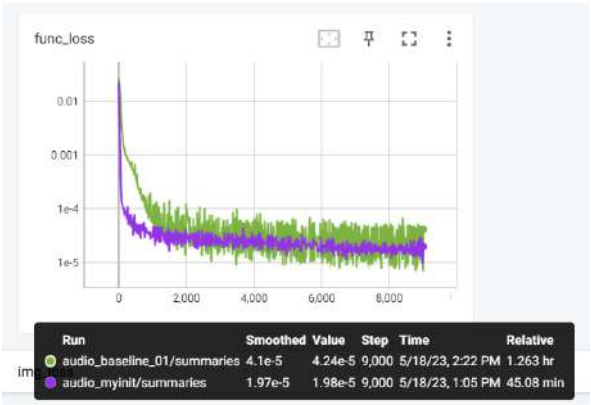

FIGURE 3.6: Loss curves of two networks fitting the audio signal. The network with the proposed initialization distribution converges faster at the first iterations and has less variance during the training. But in the end, both networks converge to the competitive values

.

In this experiment, we compare found initialization to the baseline on task of fitting the audio signal.

Reproducibility details

Data. We fit networks to the first 7 seconds of Bach's Cello Suite No. 1: Prelude (Bach), as was used in the original SIREN work. Sampling rate is 44100 samples per

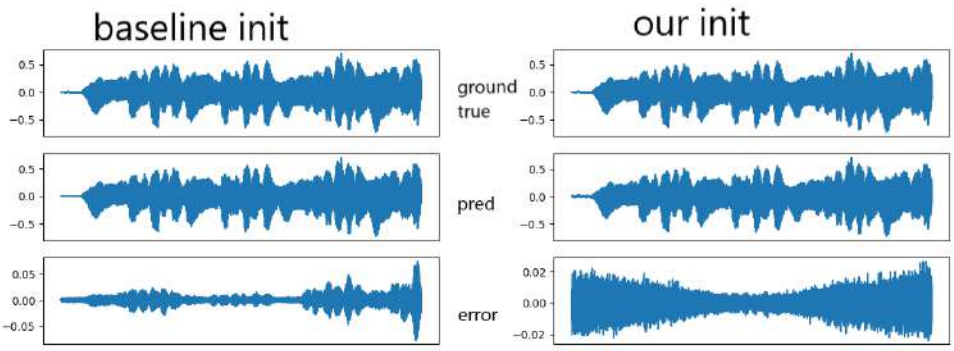

FIGURE 3.7: Waveform plot of audio signals fitted by neural networks. X axis represents time,Y axis represents amplitude. The last row demonstrates the prediction error, which is distributed differently for two initialization strategies

.

second, domain is scaled to [-1, 1] range. We used the SIREN implementation from the official GitHub repository (Sitzmann et al., 2020), and refer to the original work for other preprocessing details.

Architecture. We use a 5-layer MLP with a hidden size of 256.

Hyperparameters. We use a learning rate of 5×10^{-1}

Runtime. We train for 9,000 iterations

Final loss (Mean squared error) of the network with the proposed initialization converged to a slightly better value (see Figure 3.6). Also, it has better performance in the early stages of training.

An interesting observation was made about the distribution of the errors for two different initializations (Figure 3.7). The error of the baseline model has the error that putatively has correlation with the signal, while the error of our network is distributed independently from the signal. The latter distribution looks correlated with the time coordinate.

3.7.3 Representing video

Reproducibility details

Data. We fit networks to the "bikes sequence" video in the same way done in the original work (Sitzmann et al., 2020). Dataset contains of 250 frames of 272x640 resolution.

Architecture. Network is learning a mapping from 3 coordinates (time, x,y locations) to RGB values. The number of hidden units is set to 5, with number of hidden features equal to 1024. Outermost layer is set to a linear layer without activation function.

Hyperparameters. We use a learning rate of 1×10^{-4}

Runtime. We train for 3,000 iterations

The network with the proposed initialization on this task does not converge at all. Initial investigations do not give any results. The reasons may lie in the dataset specifics, or in network size. We make a conclusion that the proposed initialization may not work as an improvement that works out of the box and requires further work on resolving issues that impede convergence.

FIGURE 3.8: First row represents ground true frames, second represents baseline model prediction at iteration = 1505 with PSNR = 23.87 dB, second row represents predictions of the model with proposed initialization at same iteration with PSNR = 13.561 dB

.

3.8 Conclusions

3.8.1 Experiment results

Conducted experiments showed that performance of sinusoidal networks could be improved with empirically-driven initialization. We conducted a hyperparameter search of two parameters $flic$ and $initc$, that control the initialization distribution of the network's parameters in a way described in Equation 3.1 and Equation 3.2. The optimal combination of parameters we found is $flic = 2, initc = 40$ in comparison to baseline $flic = 1, initc = 6$.

The reserach hypothesis on generalization we formulated before the study was partially confirmed. Indeed, found initialization improved the performance of wider networks on harder image representation task. Also, we observed slight improvment in audio representation task compared to baseline. However, it spoiled the convergence on much harder video representation tasks.

3.8.2 Open questions

Error distribution. We observe the difference in the nature of the distribution of the errors. Errors in the proposed initialization scheme seem to be distributed independently (white noise), namely in image representation (Figure 3.3), or correlate with the input coordinates only, namely with the time coordinate in audio representation task (Figure 3.7). This difference may relate to the hypothesis, that neurons could learn in two different modes: "generalization mode" and "memorization mode" (Henighan et al., 2023). This observation could be formulated as a conjecture: these two distinct initialization strategies send the network on two principally different learning trajectories with distinct geometry of the optimization landscape. Further work in this direction requires heavier mathematical tools and a more strict experimental setup.

Video performance. It is unclear why networks fail to converge on video representation task. The first steps to do in this direction may include analysis of activation and gradient distributions. It may give insights onto reasons why the particular

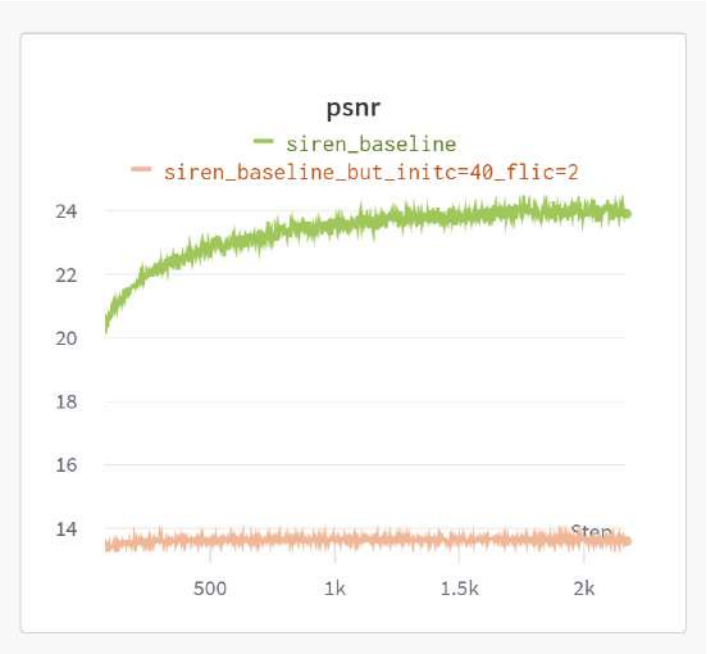

FIGURE 3.9: Learning curves of the video model. X axis represents training step,Y axis represents PSNR score. Model initialized with proposed parameters is stuck at the beginning of training.

.

initialization does not fit into this task, and highlights possible improvement points that may boost the performance on other tasks.

3.9 Summary

In this chapter, we made an overview of previous work on implicit neural networks with sinusoidal activations, namely SIREN (Sitzmann et al., 2020).

Also, we conducted an experiment on empirical tunning initialization parameters of a SIREN model. The found initialization boosted performance on image fitting task, compared to baseline initialization. We conducted further research and found that this performance improvement scales well on other datasets within the same task, and other network sizes. On the other hand, testing generalization on other tasks showed ambiguous results. While performance in audio representation was slightly better than baseline, the proposed initialization scheme prevented the network from learning any useful signal in the video representation task.

These results create a good starting point for the development of SplitNet architecture. Specifically, it is important to pay attention to the choice of initialization strategy for the sinusoidal part of the SplitNet, as it could significantly influence the model's performance.

Chapter 4

SplitNet formulation

4.1 Introduction

In this chapter we introduce the SplitNet, the neural network architecture that have 4 routes for the input signal with separate activation function each. We fix the architecture and particular combination of activation functions to reduce degrees of freedom in our research. This allows us to focus on experiments and make initial investigation of the topic. However, such particular combination of activation functions may be suboptimal and testing other combinations may be a topic of further work.

This chapter is structured as following:

1. We briefly review the relevant background of activation functions in deep learning
2. We formally formulate the SplitNet architecture
3. We make initial benchmark and analysis using neural image representation task as an example

4.2 Activation Functions review

Activation functions is a must-have component of modern neural networks. Activation functions make the neural networks capable of nonlinear morphing of the space. Without non-linearity, most of the deep neural network architectures become degenerate, since consequent linear mappings effectively can be reduced to single linear operation. In the context of neural fields, activation functions are of great interest. Choices that are considered to be a safe default in convolutional networks or transformers on more explored tasks such as object recognition or machine translation, showed to underperform on neural fields modeling (Sitzmann et al., 2020). Moreover, processing neural fields often requires special properties to be satisfied (e.g. preserving first or second order derivative of the signal), that require careful activation function setup. Therefore, we present a brief history and background of activation functions research and refer to recent survey such as Dubey, Singh, and Chaudhuri, 2022 for more details.

Sigmoid and tanh. In the early days of neural networks, sigmoid and tanh nonlinearities were extensively used. Sigmoid function is defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.1}$$

It has plausible interpretations, that contribute to the motivation of using this function. Since it outputs the value in the range of $[0, 1]$, it can be thought of as

an activation of the biological neuron. However, the output is not zero-centered with a mean of 0.5, which creates problems when stacking multiple sigmoid layers together. Similar activation function, namely hyperbolic tangent, comes without this drawback. It outputs the values in the range $[-1, 1]$, centering at zero. Tanh is defined as follows:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.2}$$

However, despite pleasing probabilistic properties, these functions. For example, the derivative of *tanh* and sigmoid approaches zero very quickly on the tails. It is sometimes referred to as neuron saturation (a more detailed explanation will follow in future work). When this happens, the derivative becomes extremely low, making the output insensitive to weight perturbation. This leads to the problem of vanishing gradients and bad convergence.

Rectified Linear Unit. ReLU is a simple but very effective non-linearity function that is still very common today. It is formulated as follows: $ReLU(x) = max(0, x)$. It trade-offs probabilistic interpretability for much lower computational cost and more healthy gradient flow. Derivative on the positive region always equals to 1, that does not shrink gradients so much as sigmoid or tanh. But this simple formulation has drawbacks, such as the problem of dead neurons. This situation can happen when a neuron always outputs negative pre-activation values, which leads to zero gradients all the time, meaning not learning at all. Also, this gradient sparsity could make training of systems such as Adversarial Networks unstable. Solutions such as Leaky ReLU solve this problem by replacing zero region with a small slope, allowing the gradients to flow even in the case of negative neuron output.

Further work on *Exponensial Linear Unit* (ELU) Clevert, Unterthiner, and Hochreiter, 2016 improves Leaky ReLU and its variants, demonstrating more robustness to noise. The output range is $[-1, \infty)$. Extension with a scaling parameter leads to Scaled ELU (SELU) Klambauer et al., 2017. This allows network to perform self-normalization automatically via gradient descent.

Automatic search of activation functions is a promising direction. For example, applying Neural Architecture search helped to discover SWISH activation function (Ramachandran, Zoph, and Le, 2017) and beat state-of-the-art results on computer vision tasks. This function is similar to ReLU, better in terms of convergence, and usually can replace the latter without major architectural changes.

(Formulas and detailed explanations of advanced activation functions will be included in future work).

To sum up, the main rough requirements to the activation function are as follows: a) it should add non-linearity b) it should have low computational cost c) it should be able to model different distributions of the training data d) it should have healthy gradient flows.

4.3 SplitNet Formulation

SplitNet is a neural network that consists of layers with so called "split activations". We define output of such layer as an elementwise multiple of 4 different activation functions:

$$layer(x) = tanh(W_{tanh}x) * \sigma(W_\sigma x) * sin(W_{\sin}x) * cos(W_{\cos}x.)$$

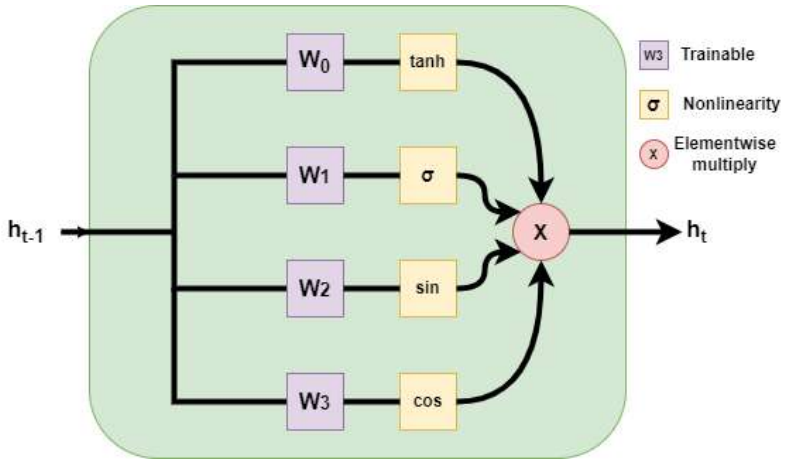

FIGURE 4.1: Schematic representation of SplitNet layer.

The input vector $x \in R^N$ is mapped via 4 different linear transformations $N \rightarrow M$, followed by 4 different nonlinearities and elementwise multiplication (see Figure 4.1). In our experiments we found almost no difference between bias and biasless versions of linear transforms, and we stick to default option of using bias.

4.4 Motivation

In this section, we try to describe the motivation of studying SplitNet architecture. It is important to note that while these theoretical reasons may sound plausible, the effectiveness of this approach should be validated through empirical experimentation.

Understudy of activation functions. Activation functions is an understudied topic in deep learning. The recent discovery of effectiveness of periodic activation functions for neural field modeling demonstrates that established defaults, such as ReLU nonlinearity, may be highly suboptimal. Moreover, most of the research is focused on the single activation functions and not on their interactions (Dubey, Singh, and Chaudhuri, 2022). We believe that there is a lot of undiscovered potential in this direction that can boost deep learning models in terms of performance or other plausible properties such as explainability.

Favorable properties.

- Handling periodic data. The usage of sinusoidal activation functions in neural field modeling may be an important or even crucial component, as demonstrated by Sitzmann et al., 2020. Including two periodic activations with different phase shifts (sine and cosine) adds more flexibility to the network, in theory.

- Gating mechanism similar to attention. Sigmoid and tanh parts of SplitNet can be viewed as a gating mechanism, since they can either pass feature fully (when output is 1), or block feature completely in the case of sigmoid and flip feature in the case of tanh. Each activation function can be seen as applying a different form of gating or weighting to the input features, and these are then combined through multiplication. This could allow the model to learn to emphasize or de-emphasize certain features depending on their relevance to the task, which is similar to what attention mechanisms do.

- Higher capacity with lower computational cost. We formulate the hypothesis, that multiplying different activation functions produce complex non-linear behavior, and a single layer can model more complex functions than a layer with a single activation function and same number of parameters. We expect the network to utilize this capacity.

Different architectures and combinations of activation functions can have very different performance characteristics on different tasks and datasets, so it's crucial to test this thoroughly in empirical real-world setting.

4.5 Initial performance analysis

We start the analysis of SplitNet with a simple experiment setup and comparison of the performance to the baseline.

4.5.1 Experiment setup

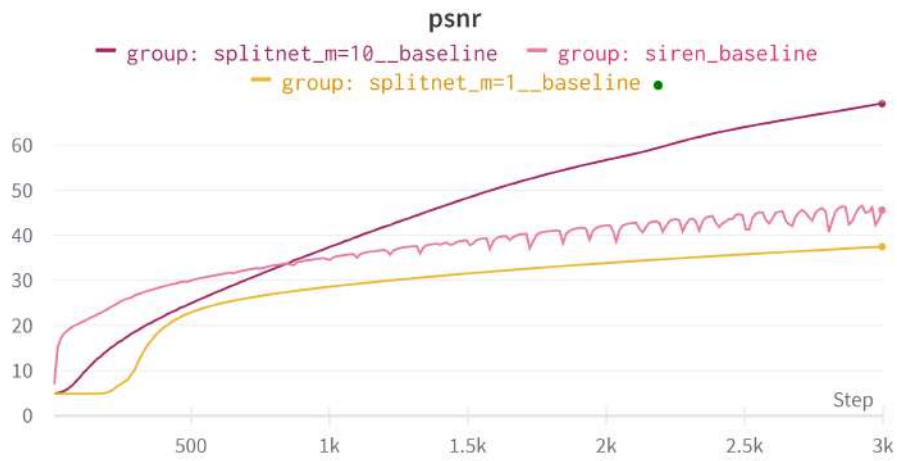

FIGURE 4.2: PSNR curves for the SplitNet and baseline SIREN training.

We chose the image representation task for its simplicity. We use grayscale Cameraman image used in previous chapter. We use network with 3 layers and hidden size of 128. As a baseline, we use SIREN with the same number of parameters.

Performance of SplitNet compared to baseline is depicted in Figure 4.2. We can see that while the performance of SplitNet (experiment *splitnet_m=1__baseline*) is comparable to SIREN baseline it is still consistently underperforming. In the next subsection we describe analysis of activation distributions and simple hyperparameter that can and allow to improve over baseline in this setting.

4.5.2 Activation multiplier

We analyse activation distribution and observe that the output range of every layer is very narrow. This phenomena can be explained by the shrinking properties of activation functions used. Each activation has values that are less than 1 in absolute values, and their product produce even smaller values. Activation distribution is visualised on Figure 4.3. We can see that the range of the SplitNet in the second

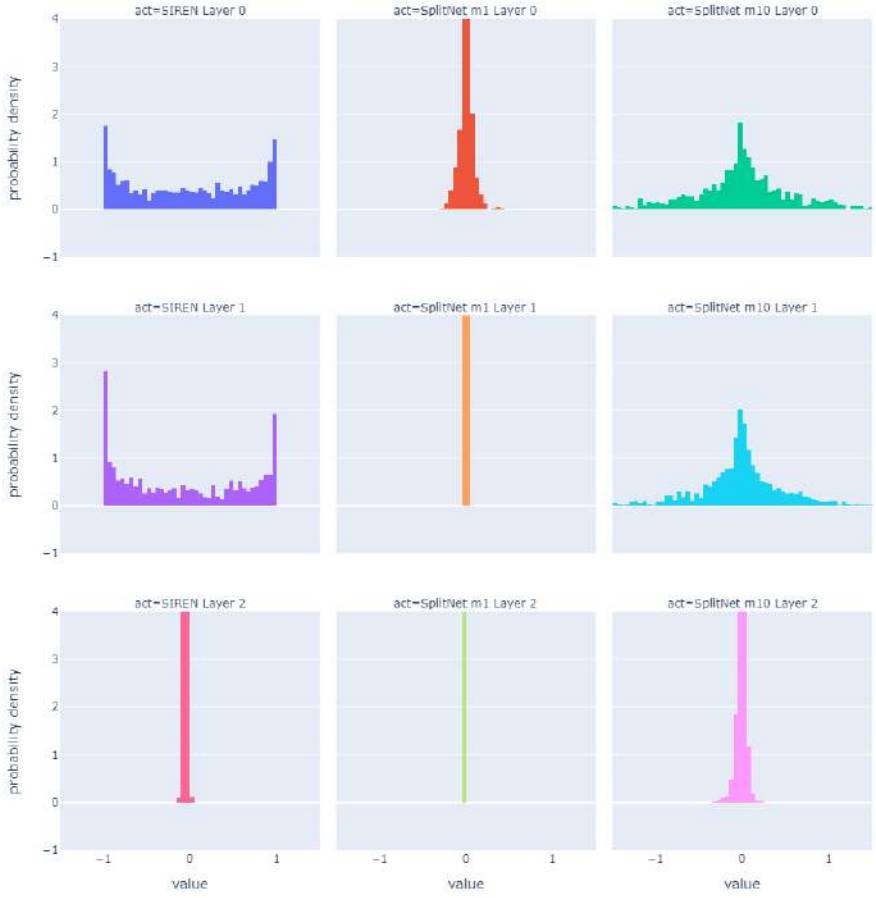

FIGURE 4.3: Histogram of intermediate activation distributions. First row represents SIREN network, second row represents SplitNet without multiplier (m=1), the third row represents SplitNet with activation multiplier $m = 10$. Activations in the third row looks healthier than in the second.

column is degenerate compared to SIREN. To increase the possible effective range of a network, we decide to introduce a hyperparameter m for the network, that boosts the activation value via multiplication:

$$SplitLayer_m(x, m) = m \times SplitLayer(x) \tag{4.3}$$

Throught the experiments we validated that boosting activations to the factor of 10 indeed improves the performance of SplitNet (Figure 4.2, experiment *splitnet_m=10__baseline*). We use the multiplier m for all layers except the last one.

4.6 Summary

In this chapter, we introduced the SplitNet architecture. We setup an image representation task to analyse performance of SplitNet. We found out that in such formulation, the effective range of the is smaller than $|1|$ and may limit the network performance. We come up with a simple trick of using multiplier m. We set this

value to 10 and empiricaly observed improvement of the performance. Further tunning of this parameter is required since choice of this particular value was not strict enough.

Chapter 5

Tunning SplitNet

5.1 Introduction

In this chapter, we explore the ways of improving performance of SplitNet via weight initialization strategies.

This chapter is structured as following:

1. We briefly review the relevant work on weight initialization
2. We describe experiment with using LSUV initialization for SplitNet
3. We describe experiment with hyperparameter tunning of SpitNet initialization

5.2 Weight initialization review

Training neural networks is a difficult task, and many work was put into making it work. Mathematically, this task is formulated as an optimization problem. Off-the-shelf algorithms for training neural networks include a lot of heuristics and assumptions, usually designed and tuned for specific setup. These assumptions could be suboptimal or even fatal if setup is different, for example, when using different activation functions. Also, early results hint to the optimization bottleneck in the proposed architecture. Thus, we describe initialization methods that make the neural network optimization feasible and help avoid common failure modes such as vanishing/exploding gradient problem.

One of the primary concerns when devising new activation functions is the issue of exploding and vanishing gradients. These phenomena can significantly impede the optimization process, thus limiting the performance of neural networks. To address these issues, we need to carefully consider weight initialization strategies that ensure a stable and robust training process.

5.2.1 Xavier init

The first serious progress in research of impact of weight initialization in deep learning was made in the work "Understanding the difficulty of training deep feedforward neural networks" by Glorot and Bengio, 2010. They investigated a similar problem of impediments in gradient optimization caused by activation functions.

Also, the authors analyzed the evolution of activations during training. First, excessive saturation is undesirable, because high saturation lead to small derivatives that prevents propagation of the learning signal. But on the other hand, pushing activations too far away from the saturation can also be detrimental. This can be due to the high similarity of the activation to the linear layer, meaning under-utilization of the network's capacity.

One historical subtlety is also interesting: authors emphasize the gains of using log-likelihood loss for classification in comparison to mean squared error. The former is considered more theoretically justified and well-suited for classification tasks, but the latter was extensively used prior to that work. Authors note that MSE loss causes more severe plateaus during training. This is a reminder that design choices that are taken for granted today may be suboptimal, and there is usually a place for improvement even on the fundamental level.

The work of Glorot and Bengio, 2010 introduced an innovative weight initialization technique known as Xavier initialization, which significantly improved the training process for deep neural networks.

The core idea behind Xavier initialization is to ensure that the variance of the input and output of each layer in the network remains roughly the same throughout the training process. By maintaining this balance, the technique helps prevent the gradients from becoming either too small (vanishing) or too large (exploding), thereby allowing for more stable and efficient training.

5.2.2 Kaiming init

The previously established Xavier initialization was not optimal for other initialization functions, such as ReLU. Work of He et al., 2015b dives deep into this problem. And, besides exploration of new activation function named Parametric ReLU (PRELU), design a sampling distribution that keeps the activation distribution constant across layers, respecting properties of ReLU activation function.

5.2.3 LSUV init

Derivation of proper initialization scheme usually requires knowledge of the architecture and activation functions. Work on Layer-sequential unit-variance (LSUV) initialization Mishkin and Matas, 2016 tries to circumvent this limitation. They propose iterative way of initialization, that does not make assumptions on particular activation functions or other design choices of the layer. Algorithm takes a couple iterations to converge on practice. The computational cost is not very high and comparable to couple forward passes. But it has the potential to remove the burden of analytical derivation of proper initialization scheme for new neural network modules or their combinations.

Main idea behind LSUV initialization is to calculate the standard deviation σ and mean μ of activations for each layer, after that divide weights by σ and subtract μ. In practice, this procedure is able to set activation distribution to be standard normal in couple of iterations. This procedure is iterative from layer to layer, starting from the first one. The main advantage is the ease of use with different activation functions, since no particular assumption about concrete non-linearity is made.

5.3 Applying LSUV initialization

Here we describe application of LSUV initialization to SplitNet. We apply LSUV initialization algorithm to SplitNet in the following way. We normalize pre-activation distribution to the standard normal of each non-linearity separately.

In the same setup as in previous chapter, LSUV initialization boosts the performance of SplitNet (Figure 5.1, experiment *splitnet_m=10__lsuv*).

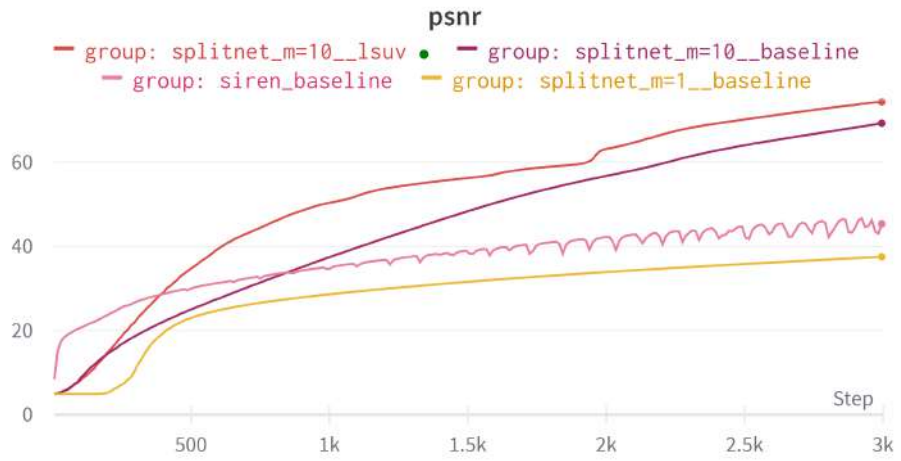

FIGURE 5.1: Training PSNR curves for SIREN baseline, SplitNet with default initialization and SplitNet with LSUV initialization.

5.4 Hyperparameter tunning

Experiment name	activation σ range	m range	Number of runs
Experiment 0	[0.1, 10]	[0.5, 100]	1917
Experiment 2	[0.5, 3]	[3, 15]	529
Experiment 3	[0.1, 2]	[12, 25]	778

TABLE 5.1: Configuration of hyperparameter tunning experiments

In this section we describe the hyperparameter tunning of SplitNet with the LSUV initialization.

Motivation. Using LSUV initialization showed to improve performance of a SplitNet. Therefore, we want to try to push this approach further and try to find optimal parameters, since default assumptions may be suboptimal.

Experiment setup We use the similar setting for tunning as in Chapter about SIREN. We use image representation task and fit network for 500 iterations. We use 5 layer MLP with a hidden size of 32. We tune 5 parameters: activation multiplier (see Equation 4.3) and 4 standard deviations for LSUV init. In particular we use custom formulation of LSUV init and allow each nonlinearity in SplitNet (tanh, sigmoid, sin, cos) have its own σ of pre-activation, instead of sticking to $\sigma = 1$ as in original work. We run 4 experiments (second one technically failed). Range of parameters for each experiment is described in Table 5.1

Experiment results. We run hyperparameter tunning sessions sequantially, manually adjusting hyperparameter ranges to improve performance. Histograms can be seen on Figure 5.2. As we can see, with each iteration spread of the values increases, as do the maximum value of PSNR, the metric we are most interested in.

Generalization analysis. To test how found hyperparameter generalizes, we train networks with these hyperparameters in slightly different setting: bigger image and different number of hidden layers and hidden size. Experiments show that performance was almost the same or even worse with hyperparameters found during initialization.

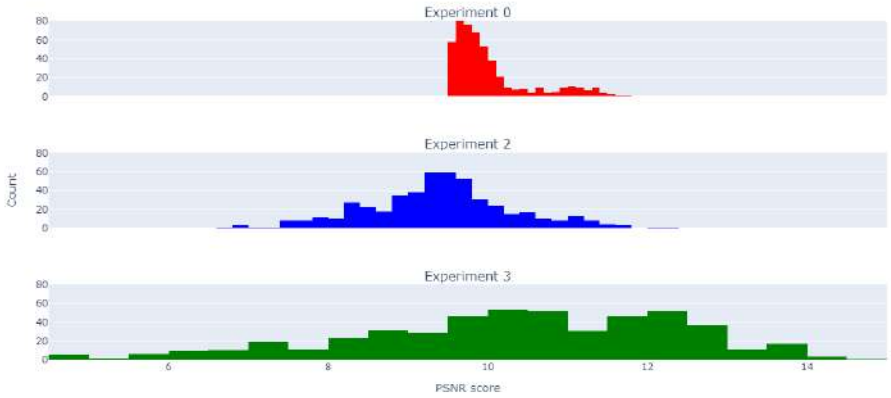

FIGURE 5.2: Histogram of PSNR scores of hyperparameter tunning experiments. After each run sampling configuration was adjusted manually with an objective to achieve the highest score

Conclusions. From the first experiment with LSUV initialization, we can conclude that this algorithm improves the performance of SplitNet and makes it competitive with SIREN. Attempts to furhter improve performance of initialization with hyperparameter tunning doesn't gave much results. One explanation to this could be impossibility to simply extrapolate hyperparameters to other settings due to complex and unexplored nature of multiple activations mulitplied and deep learning in general.

5.5 Summary

In this chapter, we explored ways of improving performance of SplitNet, mainly through weight initialization. LSUV initialization improved performance. Experiment with further tunning hyperparameters of initialization didn't help improving performance. Possible reasons may be the challenge in extrapolating hyperparameters to other settings.

Chapter 6

Experiments on other tasks

In this section, we present experiments with SplitNet on other tasks, namely image classification with convolutional neural networks and 3d scene modeling with neural radiance fields. This allows to compare performance of SplitNet in wider setting and find directions that works further investigation.

6.1 Image classification with CNNs

Model name	Test accuracy, %	N of parameters
CNN hsize=16	43.28	3418
CNN hsize=32	44.45	6170
CNN split hsize=64	**49.89**	**5122**
CNN split hsize=64 with LSUV	45.79	5122
CNN hsize=64	46.48	11674
CNN split hsize=128	**50.56**	**9218**
CNN split hsize=128 with LSUV	49.12	9218
CNN hsize=128	50.83	22682
CNN split hsize=256	**53.71**	**17410**
CNN split hsize=256 with LSUV	52.09	17410
CNN 4 layer	**65.45**	292170
CNN split 4 layer	62.78	201086

TABLE 6.1: Image classification benchmark results on CIFAR10.

.

To understand how well Split activation networks perform in general, we implemented a convolutional neural network with split activations and compared its performance CNN with ReLU activation. Architecture is the same as in Figure 4.1, except that now we have feature layers with 4 different activations that are multiplied elementwise.

For comparison, we used image classification task on CIFAR10 dataset (Krizhevsky, Nair, and Hinton, 2009) as a benchmark. We constructed a 2 layer CNN with ReLU nonlinearity as a baseline and vary the number of feature maps (hidden layer). Models were evaluated vased on their classification accuracy on the test set and total number of parameters. For SplitNet, we used two flavors: with default pytorch initialization (Kaiming uniform He et al., 2015b) and with LSUV initialization. The results of the experiments are shown in Table 6.1.

The table demonstrates that, for each group of models with approximately similar complexity (i.e., number of parameters), SplitNet models outperformed the CNN

baseline. This is true both for models without and with LSUV initialization. Data shows that Kaiming initialization works better in this setting.

A bigger baseline CNN was also used for comparison, achieving over 65% accuracy. CNN with split activation and a similar number of parameters showed worse results. This may suggest that bigger split models struggle from unhealthy gradient flow, and further work may be required to setup architecture with better initialization or activation statistics.

6.2 Radiance Field modeling

To test how SplitNet architecture perform in 3D setting, we conduct experiments with modeling radiance field. We base this experiment on TensoRF Chen et al., 2022a, a simple and fast neural radiance field model that gives state-of-the-art results. We replace an MLP feature decoder in this architecture with SplitNet and show that better performance could be obtained with a smaller number of parameters.

First, we provide a brief description of radiance field modeling and TensoRF architecture and motivation of choosing this model for comparison. Then, we formulate an experimental setup. After that, we discuss the results and consider its limitations.

6.2.1 TensoRF formulation

Radiance fields Various 3D scene representation methods exists. Classical ones include meshes, implicit functions such as Signed Distance Function, voxels and point clouds. Xie et al., 2022. The emerging way of 3D scene representation is with radiance fields modeled with neural network Mildenhall et al., 2020. Radiance field modeling is formulated as a task of mapping 3D location and viewing direction to the volume density and view-depended color. Incorporating viewing angles into the model allows modeling non-Lambertian surfaces, for example, glossy materials and their non-equal light reflection, causing specular highlights. The original NERF and some of its derivative work have a pure MLP formulation. That is, the whole scene is backed into the Multilayer Perceptron weights. While this approach results in a very compact representation, it has some fundamental drawbacks, such as high training and inference time. For each pixel, MLP is fully evaluated multiple times, usually hundreds of times. This approach is redundant in practical cases, because all features are evaluated for each part of the scene. While in practice the underlying signal may have local patterns, and this locality may be utilized to save computation. Recent methods, such as Yu et al., 2021 try to leverage this locality and disentangle the information about the scene all over the volume using voxel grids. That is, each part of a volume is associated with its own feature, making rendering extremely fast.

Note on voxels. Voxel stands for "volume pixel", that is the three-dimensional equivalent of a pixel in a two-dimensional image. Voxels are arranged in regular pattern (small cubes), with each voxel having its own position and size. Size of voxels is usually the same for all voxels in a grid. Properties of interest are attached to each voxel, such as color or density, that are used for scene visualization (rendering) or other processing. A 3D voxel grid of features can be represented as a 4D tensor, with 3 spatial dimensions and one feature dimension. The main problem of the voxels in 3D processing is the memory consumption, that scales cubicaly with resolution.

While the voxel-based method gain in inference speed, this improvement comes with a huge memory cost. Size of a voxel grid scales cubically with the resolution, and tradeoffs classical MLP-based NERFs compactness. For example, Lego scene represente with Plenoxel model (Yu et al., 2021) has a size more then 700 Megabytes. The central idea of TensoRF is to effectively represent the scene as a 3D voxel grid of features via tensor decomposition, reducing the memory footprints from $O(N^3)$ to $O(N^2)$ (Figure 6.1).

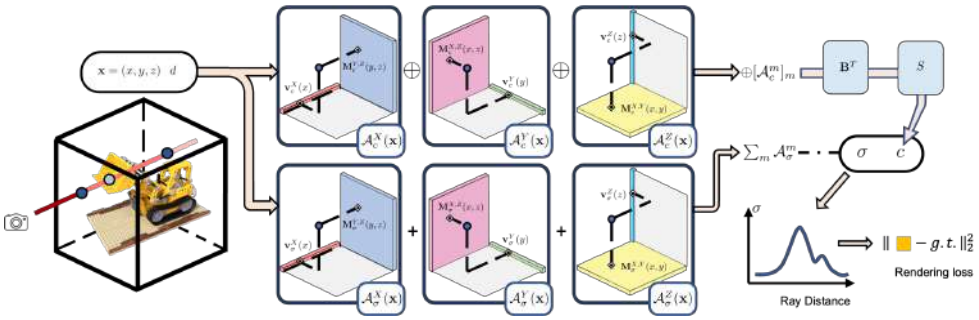

FIGURE 6.1: TensoRF architecture. Voxel grid is decomposed into a sum of outer products between vectors and matrices, reducing memory size from $O(N^3)$ to $O(N^2)$
. Image taken from Chen et al., 2022a.

Tensor decomposition. To reduce the model size TensoRF uses novel compact tensor decomposition to model the feature grid. 4D tensor is decomposed into a sum of lower-rank tensors, with each tensor modeled as an outer product between vector and matrix:

$$\mathbf{G} = \sum_{r}^{R} \mathbf{v}_r^X \otimes \mathbf{M}_r^{Y,Z} + \mathbf{v}_r^Y \otimes \mathbf{M}_r^{X,Z} + \mathbf{v}_r^Z \otimes \mathbf{M}_r^{X,Y} \tag{6.1}$$

Where R is the dimensionality of a feature grid, and X, Y, Z denote 3 planes that bound the rendering volume.

Authors call this novel decomposition Vector-Matrix (VM) decomposition and it lies in a core of TensoRF architecture. The pipeline is extended with a few non-critical components, such as separation of geometry and appearance features and additional vector in appearance grid decomposition (Figure 6.1). We refer to the original work of Chen et al., 2022a for more details.

The motivation of using TensoRF model for comparison:

- Fast convergence relative to alternatives with high-quality reconstruction
- Simple architecture formulation, leaving less space for bugs. Also, easy-to-run and reproducible code is available.
- Suitable for replacement with SplitNet small MLP architecture

While this particular radiance field model was chosen, no architecture-specific changes were made, and we suppose that results generalize to other architectures.

6.2.2 Experiment Setup

Original MLP architecture. After obtaining latent features from factorized voxel volume on a given coordinate, neural TensoRF uses a neural network to render the

final view-dependent color. The input to this neural network consists of grid features and angles of the viewing direction, fed through positional encoding, following the original NERF method Mildenhall et al., 2020. Architecture is simple and shallow:

$$Linear(150, 128) \rightarrow relu \rightarrow Linear(128, 128) \rightarrow relu \rightarrow Linear(128, 3) \tag{6.2}$$

Where $Linear(in\_dim, out\_dim)$ is a linear map from in_dim to out_dim dimensions, and $relu$ is a ReLU non-linearity. Output of this neural network are RGB colors. In Table 6.2 we refer to this baseline MLP as *MLP_Fea*. For comparison with a smaller version of SplitNet, we slightly decrease the number of parameters in this model via reducing the number of hidden features to 119, referring to this variation as *MLP_Fea_Smaller*.

SplitNet. We use a similar architecture (3 layers) for SplitNet. Number of hidden features is chosen to be 45, to make number of parameters comparable to original MLP decoder. To test that same quality could be obtained with smaller number of parameters, we decrease the number of hidden features to 42, resulting in *SplitNet_Smaller*.

SIREN. SIREN MLP is also used for comparison. The architecture is the same as in 6.2 except for the activation function, which is sinusoidal in this case. We apply the same initialization scheme as in the original work on SIREN Sitzmann et al., 2020

Training strategy. We use an original implementation of TensoRF, replacing the decoder. We test models on Lego dataset. We use 7000 iterations and report the final test PSNR score. Codes and instructions to reproduce results could be found at `https://github.com/kilianovski/TensoRF_splitnet`

6.2.3 Results

Experiment results a summarized in Table 6.2. Better test performance of 34.16 PSNR, compared to baseline 34.01 PSNR, is obtained with a smaller number of parameters. Moreover, a comparable to baseline performance could be obtained with almost 10% less parameters. To test if the size of the decoder has any impact on the overall performance, we decreased in a similar way the baseline decoder. This experiment showed that the smaller baseline decoder indeed decreases the overall score. SIREN architecture obtained lower scores.

Limitations. Experiment was conducted on a single scene, and using more scenes may increase confidence in a result. However, the Lego scene contains 100 train and 600 test images, each of the size 800x800 pixels. So the overall train dataset contains $800 \times 800 \times 100 = 6.4e^7$ training data points (rays) and $3.84e^8$ testing datapoints. We consider this enough for an initial evaluation and leave more thorough evaluation to the future work. We expect some metric variance from scene to scene. Also, results are obtained on a specific NERF architecture with specific shallow MLP decoder architecture, and results may differ for other setups.

Overall, these results add additional evidence to the potential of SplitNet and add reasons to research this architecture further.

Decoder	PSNR	Number of parameters
MLP_Fea	34.01	36227
MLP_Fea_Smaller	33.93	32609
SIREN	32.11	36227
SplitNet	34.16	35598
SplitNet_Smaller	34.00	32721

TABLE 6.2: TensoRF performance with different feature decoders on Lego dataset

Chapter 7

Conclusions

7.1 Summary

This work presented a novel SplitNet architecture for neural field modeling.

The first part of the research conducted a study on a subset of Split networks, namely Sinusoidal Neural Networks (SIRENs). Study of performance and activation distributions allowed to empirically find better initialization strategies and improve performance compared to the original work in the studied setting. Further research on a generalization of this finding confirmed that this initialization setting indeed improves performance on other image representation tasks, audio representation tasks, and with networks with varying width and depth. On the other hand, it was found that our improvement does not generalize perfectly since we found settings, such as the video representation task, where the proposed initialization strategy degraded the performance.

The second part of the study described the design process of SplitNet. Activation distributions were studied and problem of activation saturation was found. Introducing multiplication coefficient m to boost activation values was found to be a simple but working solution, that empirically proved its effectiveness. Also, weight initialization strategies were studied, and some of them, such as LSUV initialization (Mishkin and Matas, 2016) was found to be effective. Further work on hyperparameter tunning was conducted. In contrast to a more simple setting with sinusoidal activations, extrapolating found hyperparameters from one task setting to another was found challenging.

The third part of the work investigated performance of the proposed SplitNet architecture on more distinct tasks, such as 3D scene reconstruction and image classification. Early results showed that in these settings SplitNet gives competitive or better performance. This gives reasons and motivation to study split activation further.

7.2 Limitations

While the research conducted in this work provides valuable insights into the potential of SplitNet and neural field modeling, it also has several limitations that should be acknowledged. The experiments conducted in this thesis were limited to a few specific tasks and datasets, mainly an image representation task. To address this problem, initial research was conducted with other tasks, such as image classification, video and audio representation, 3D scene reconstruction. But their coverage remains limited. Also, the main baseline for comparison was SIREN, and comparison with other approaches can give a more objective view.

7.3 Future work

Here we propose some ideas that can be addressed in future work:

1. Incorporating other methods of improving gradient flow health besides weight initialization, such as training-time normalizations (such as BatchNorm Ioffe and Szegedy, 2015 and its variations) and skip connections (He et al., 2015a).

2. Setting up more advanced hyperparameter tunning or even neural architecture search

3. Exploring different tasks and datasets.

4. Incorporating SplitNet into existing models. Experiments with CNNs showed that using SplitNet as a plug-in deep learning building block has the potential to improve the performance of other downstream tasks. This can not only improve the performance of the pipelines but give insights into how SplitNet can be improved.

5. Exploring variations of SplitNet. In this work we studied only the particular combination of activation functions, namely hyperbolic tangent, sigmoid, sine and cosine. This combination may be suboptimal and other options can be studied. Also, more simple settings of activation interactions via multiplication (e.g. only two) can be studied to gain more insights.

Bibliography

Biewald, Lukas (2020). *Experiment Tracking with Weights and Biases*. Software available from wandb.com. URL: https://www.wandb.com/.

Chen, Anpei et al. (2022a). "TensoRF: Tensorial Radiance Fields". In: *European Conference on Computer Vision (ECCV)*.

Chen, Zhiqin et al. (2022b). "MobileNeRF: Polygon Rasterization Pipeline for Efficient Neural Field Rendering". In: *ArXiv* abs/2208.00277.

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2016). *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. arXiv: 1511.07289 [cs.LG].

Czerkawski, Mikolaj et al. (2021). *Neural Knitworks: Patched Neural Implicit Representation Networks*. arXiv: 2109.14406 [cs.CV].

Davies, Thomas, Derek Nowrouzezahrai, and Alec Jacobson (2021). *On the Effectiveness of Weight-Encoded Neural Implicit 3D Shapes*. arXiv: 2009.09808 [cs.GR].

Dubey, Shiv Ram, Satish Kumar Singh, and Bidyut Baran Chaudhuri (2022). *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. arXiv: 2109.14545 [cs.LG].

Dupont, Emilien et al. (2022). *From data to functa: Your data point is a function and you can treat it like one*. arXiv: 2201.12204 [cs.LG].

Gao, Kyle et al. (2022). *NeRF: Neural Radiance Field in 3D Vision, A Comprehensive Review*. DOI: 10.48550/ARXIV.2210.00379. URL: https://arxiv.org/abs/2210.00379.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *International Conference on Artificial Intelligence and Statistics*.

He, Kaiming et al. (2015a). *Deep Residual Learning for Image Recognition*. arXiv: 1512.03385 [cs.CV].

— (2015b). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. arXiv: 1502.01852 [cs.CV].

Henighan, Tom et al. (2023). *Superposition, Memorization, and Double Descent*. Anthropic. URL: https://transformer-circuits.pub/2023/toy-double-descent/index.html.

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv: 1502.03167 [cs.LG].

Klambauer, Günter et al. (2017). *Self-Normalizing Neural Networks*. arXiv: 1706.02515 [cs.LG].

Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton (2009). "CIFAR-10 (Canadian Institute for Advanced Research)". In: URL: http://www.cs.toronto.edu/~kriz/cifar.html.

Martel, Julien N. P. et al. (2021). *ACORN: Adaptive Coordinate Networks for Neural Scene Representation*. arXiv: 2105.02788 [cs.CV].

Mildenhall, Ben et al. (2020). *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. DOI: 10.48550/ARXIV.2003.08934. URL: https://arxiv.org/abs/2003.08934.

Mishkin, Dmytro and Jiri Matas (2016). *All you need is a good init*. arXiv: 1511.06422 [cs.LG].

Ramachandran, Prajit, Barret Zoph, and Quoc V. Le (2017). *Searching for Activation Functions*. arXiv: 1710.05941 [cs.NE].

Sitzmann, Vincent et al. (2020). *Implicit Neural Representations with Periodic Activation Functions*. DOI: 10.48550/ARXIV.2006.09661. URL: https://arxiv.org/abs/2006.09661.

Skorokhodov, Ivan, Savva Ignatyev, and Mohamed Elhoseiny (2021). *Adversarial Generation of Continuous Images*. arXiv: 2011.12026 [cs.CV].

Xie, Yiheng et al. (2022). "Neural Fields in Visual Computing and Beyond". In: *Computer Graphics Forum*. ISSN: 1467-8659. DOI: 10.1111/cgf.14505.

Yu, Alex et al. (2021). *Plenoxels: Radiance Fields without Neural Networks*. arXiv: 2112.05131 [cs.CV].