IMPLICIT NEURAL REPRESENTATION WITH PERIODIC ACTIVATION FUNCTIONS

Marine VIEILLARD, Romain HÛ

March 13th, 2025
https://github.com/MV-13/ProjetDLM1.git

Contents

1 Introduction: Motivation for SIREN

Traditional neural architectures struggle to model signals with fine details and to effectively represent spatial and temporal derivatives. However, these elements are crucial for many physical problems, particularly for solving partial differential equations.

The introduction of periodic activation functions, particularly the Sinusoidal Representation Networks (SIREN) architecture, enables a better representation of complex natural signals as well as their derivatives. With these networks, it becomes possible to accurately model images, wave fields, videos, and sound while efficiently capturing their local variations.

Moreover, SIREN offers a promising approach for solving boundary value problems involving fundamental equations such as the Poisson equation, the Helmholtz equation, and the wave equation.

SIREN allows approximating a specific class of functions Φ that satisfy equations of the form:

$$F(x, \Phi, \nabla_x \Phi, \nabla_x^2 \Phi, \dots) = 0$$

The objective is to design a neural network capable of parameterizing Φ while respecting the conditions imposed by the equation F. These models have various applications, ranging from modeling discrete signals in images and videos to learning three-dimensional representations and solving complex physical equations.

Until now, such problems have mainly been addressed with multilayer perceptrons (MLPs) using the ReLU activation function. However, since the second derivative of ReLU is zero, these models struggle to capture details and even more so their derivatives. Even the tanh function, although possessing a nonzero second derivative, does not achieve the performance of SIREN in representing signal derivatives.

2 SIREN weights initialization

The efficiency of SIREN heavily relies on proper weight initialization. Poor initialization can lead to degraded performance in terms of both accuracy and convergence speed.

The key idea behind initialization is to preserve the distribution of activations throughout the network so that the final output does not depend on the number of layers. To illustrate this approach, consider a single neuron receiving an input x uniformly distributed over the interval $[-1, 1]$. Its activation is given by:

$$y = \sin(ax + b),$$

where a and b are learned parameters. It can be shown that for any $a > \frac{\pi}{2}$, covering at least half a period, the output follows an arcsin distribution, independent of the choice of b.

In a deeper network, each neuron receives a linear combination of n inputs $x \in \mathbb{R}^n$ with weights $w \in \mathbb{R}^n$, leading to the output:

$$y = \sin(w^T x + b).$$

If the weights w_i are sampled uniformly from $\left[-\frac{c}{\sqrt{n}}, \frac{c}{\sqrt{n}}\right]$, then as n approaches infinity, the dot product $w^T x$ follows a normal distribution with zero mean and variance $\frac{c^2}{6}$. When passed through the sine function, the output again follows an arcsin distribution for any $c > \sqrt{6}$, ensuring the preservation of statistical properties across layers.

In this framework, SIREN weights can be interpreted as angular frequencies, while biases act as phase shifts. Higher amplitude weights introduce higher frequencies in the network, whereas for $|w^T x| < \frac{\pi}{4}$, the sine function remains approximately linear, preserving the original frequencies.

Finally, to properly initialize SIREN, weights are drawn uniformly from:

$$\left[-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}\right]$$

to ensure that the input to each sine activation follows $\mathcal{N}(0, 1)$.

Additionally, the first layer is initialized with a frequency parameter ω_0 such that:

$$\sin(\omega_0 W x + b)$$

covers multiple periods over $[-1, 1]$. A typical value used in experiments is $\omega_0 = 30$.

3 Solving differential equations

To illustrate how one can solve a partial differential equation using a neural network, we will take as example the wave equation in 2 dimensions. The wave equation governs the behavior of a function $\phi(t, x, y)$ that must satisfy:

$$\frac{\partial^2 \phi}{\partial t^2} = c^2 \Delta \phi$$

with the following initial conditions:

- Neuman condition : $\frac{\partial \phi(0,x,y)}{\partial t} = 0$
- Dirichlet condition : $\phi(0, x, y) = f(x, y)$ with f being an initial distribution.

In the above equations, $\Delta \phi = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}$ is the Laplacian of ϕ, f is an initial wave distribution, typically a point source modeled by a dirac function, and we write ϕ instead of $\phi(t, x, y)$ when all arguments are involved, for the sake of simplicity. From this equation, we naturally deduct the following loss function for our neural network:

$$\| \underbrace{\frac{\partial^2 \phi}{\partial t^2} - c^2 \Delta \phi}_{\text{equation}} \|_2^2 + \lambda_1 \| \underbrace{\frac{\partial \phi(0, x, y)}{\partial t}}_{\text{neuman}} \|_2^2 + \lambda_2 \| \underbrace{\phi(0, x, y) - f(x, y)}_{\text{dirichlet}} \|_2^2$$

with hyperparameters λ_1, λ_2 to give an equivalent weight to each term of the loss function.

As we can see, deriving a loss function from the constraints of a partial differential equation is relatively straightforward. However, with a closer look, we see this supervision is indeed "implicit" for the equation constraint, meaning that it is not the neural network itself that is supervised, but its derivative. Although we know that ϕ itself has the universal-approximation property, nothing yet guarantees that its derivative has it.

3.1 The space of neural networks is stable by derivation

Let $\phi : \mathbb{R}^n \to \mathbb{R}^m$ be a SIREN model with L hidden layers and we note y_l the output of ϕ after the non-linearity of the l^{th} layer. Let us take a look at the derivative of ϕ w.r.t. x, using the chain rule:

$$\begin{aligned}
\nabla_x \phi &= (\frac{\partial y_n}{\partial y_{n-1}} \times ... \frac{\partial y_1}{\partial y_0} \times \frac{\partial y_0}{\partial x})^T \\
&= W_0^T sin'(y_0) \times ... W_{n-1}^T sin'(y_{n-1}) \times W_n^T sin'(y_n) \\
&= W_0^T cos(y_0) \times ... W_{n-1}^T cos(y_{n-1}) \times W_n^T cos(y_n) \\
&= W_0^T sin(y_0 + \frac{\pi}{2}) \times ... W_{n-1}^T sin(y_{n-1} + \frac{\pi}{2}) \times W_n^T sin(y_n + \frac{\pi}{2}) \\
&= \hat{W}_0^T sin(y_0) \times ... \hat{W}_{n-1}^T sin(y_{n-1}) \times \hat{W}_n^T sin(y_n)
\end{aligned}$$

where y_i stands for the output of the i^{th} layer of the network before its nonlinearity. By abuse of notation, we include the bias term in the W_i matrices and absorb the $\frac{\pi}{2}$ offset in the $\hat{W}_i$ matrices.

We can then point out that each term of the form $\hat{W}_i^T sin(y_i)$ is itself a SIREN network with i nonlinearities ($i - 1$ layers), meaning that each of these terms has the universal approximation property. The whole product thereby has the universal approximation property, meaning that it can be represented by another neural network.

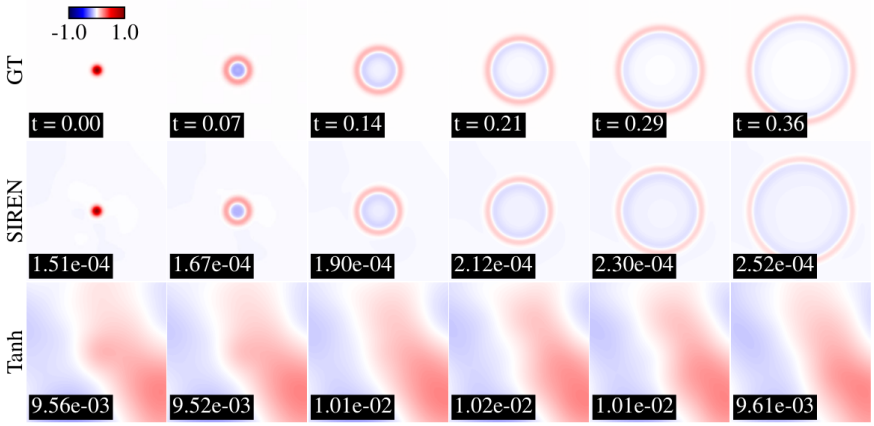

Figure 1: Resolution of the wave equation by a SIREN model and a tanh model. Results are compared over time with the ground truth computed by a principled solver.

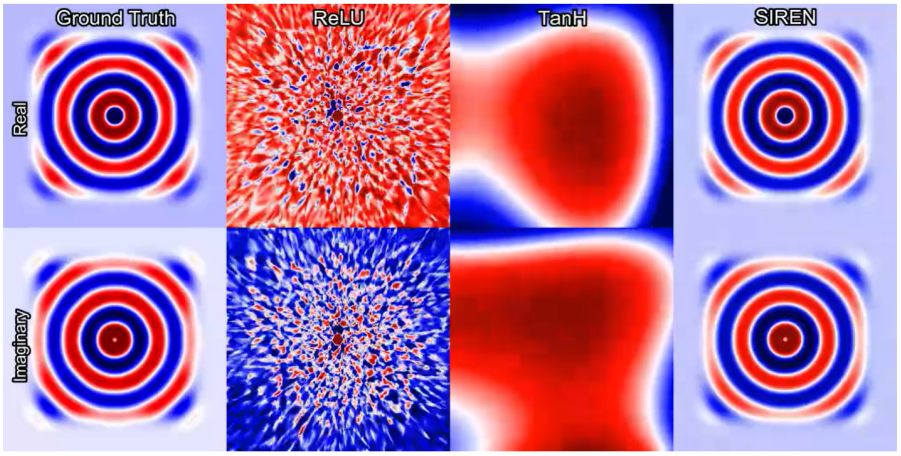

Figure 2: Resolution of the helmholtz equation by a SIREN tanh and ReLU model. Results are compared with the ground truth computed by a principled solver. The helmholtz equation is defined by $\Delta\phi + \frac{1}{c(x,y)^2}\phi = -f(x,y)$ and does not depend on time.

3.2 In practice

We sample points uniformly from the infinite-norm ball $\{x \in \mathbb{R}^2 \text{ where } \|x\|_\infty \leq 1\}$. We also approximate f with a narrow 2-dimensional gaussian function, as keeping a dirac function would require the algorithm to exactly sample the $(0,0)$ point multiple times in order to learn the dirichlet initial condition, which is of course infeasible. For the time component, we sample points with increasing time values, from 0 to, for example 0.4 : this allows the model to learn initial conditions first, and then propagate them to the future predictions. We also care to sample points closer to $(0,0)$ in space at time 0 to help the model learn the dirichlet initial condition. As we can see from figure 1 and figure 2, SIREN models easily reconstruct the solutions of the equations while tanh and ReLU models struggle to capture even the basic frequencies.

For the ReLU case, a possible explanation deducted from the demonstration realized in part 2.1 is the following. The derivative of a ReLU neural network is a neural network with a heaviside activation function ($\mathbb{1}_{x \geq 0}$). Although a heaviside network theoretically retains the universal approximation property, such a network cannot be trained using standard backpropagation, as the derivative of the heaviside activation function is 0 everywhere, leading to gradient vanishing. This could explain why ReLU networks fail to solve implicit representation problems based on their derivatives. The derivative of a SIREN, on the other hand, is not only trainable through backpropagation, but also retains all the good properties of SIRENs for representing smooth and periodic signals.

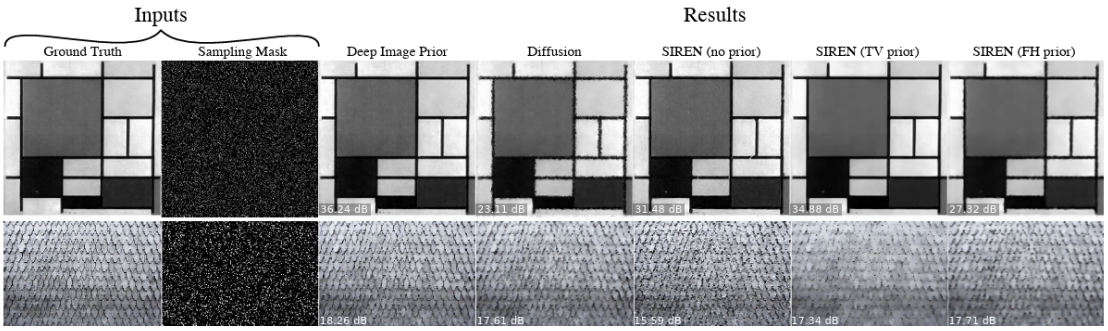

Figure 3: Inpainting results on two masked images using diverse priors, including TV, FH, diffusion, deep image and no prior.

4 Application to image processing: inpainting

We discuss in the following section the application of SIREN networks to inpainting tasks. Inpainting involves reconstructing the missing parts of a signal, often an image, based on fragments of the original signal. The image is parameterized by a SIREN $\phi(x, y)$, with x, y the pixel coordinates, but the training procedure is a bit special: instead of being supervised by a full image B, the network only has access to a partial image b as input. The RMSE of the output is calculated only on the ground-truth pixels present in b, ignoring all other pixels. In this way, the network learns to interpolate ground-truth pixels in a plausible way, without being supervised by the real image, hence the term "implicit" representation. Obviously, training such an implicit representation requires the masked image b to contain a substantial amount of context.

4.1 Continuous and smooth representations

In this context, SIREN networks possess a subtle advantage over classical computer vision architectures, such as CNNs. CNNs map images to a discrete representation that is entirely defined by their architecture. For instance, if the output of a CNN is defined as a 32×32 pixel grid, its resolution remains fixed at 32×32, and another CNN is required for a different resolution. An implicit representation model, on the other hand, learns a continuous representation of the image, meaning that its input (x, y) represents continuous coordinates of the image mapped to a pixel value. The final image reconstructed by the SIREN is thus theoretically considered as a discrete pixel grid, sampled from a continuous output. This allows SIREN models to capture fine-grained details and output images at any resolution.

Additionally, different priors can be enforced on the network in order to force it to interpolate missing pixels in a smooth way. This can be achieved by adding a prior to the RMSE loss. Examples of smoothing priors are:

- The Total Variation (TV) prior : $\frac{1}{N}\|\nabla\phi(x, y)\|_1$
- The hessian's Frobenius norm prior (FH) : $\frac{1}{N}\|hess(\phi(x, y))\|_F$

with N the number of sampled points from the original image and the Frobenius norm being the sum of squared components of the hessian matrix.

4.2 Handling aliasing

However, a precaution is to be taken when making smooth continuous representations of high resolution images: aliasing. As said earlier, implicit representations allow the production of smooth textures, but real images also contain abrupt transitions, for example a tree branch in a blue sky or a ship on the sea. These near-discrete transitions are dubbed "high frequency" transitions, as opposed to "low frequency" transitions, as the pixel values change very quickly over space. When one trains a SIREN model with a smoothing prior on high frequency transitions, the model is prone

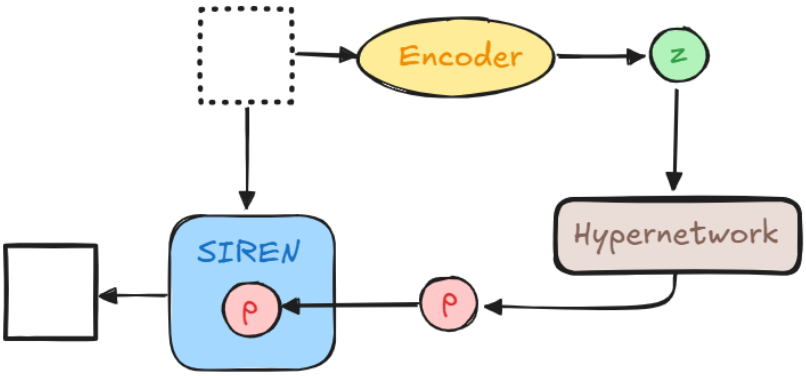

Figure 4: Model diagram for the hypernetwork architecture. When the model is presented with partial context (dashed square), the encoder maps that signal to a low dimensional latent vector z. The hypernetwork then decodes this latent vector into SIREN parameters, immediately creating a SIREN model suited to the context.

to overfitting the high frequencies, resulting in extremely sharp and unwanted patterns, especially in high-resolution renderings, known as aliasing. To counter this, post-processing treatment is often required with anti-aliasing techniques such as downsampling kernels. An example of such a kernel is:

$$\phi(x,y) \leftarrow h * \phi(x,y) = \frac{1}{N}\sum_{i=1}^{N} \phi(x+x_i, y+y_i)$$

where $x_i, y_i \sim \mathcal{P}$ of pdf $p(x_i, y_i) = \frac{1}{2}max(1-|x|)max(1-|y|)$. The convolution h is here written in a discrete and computationally implementable form, instead of the continuous integral formulation.

This kernel thus samples, for each pixel (x, y), random pixels based on their position relative to (x, y): closer pixels are more likely to be sampled than far ones. It updates the pixel value with the average value of ϕ on the sampled pixels. This probabilistic sampling prevents the kernel from enforcing a particular post-processing pattern on the image. If an anti-aliasing kernel only samples, say, on the left of the pixel to smooth, this would modify the original image with a left directed pattern. After all these adjustments, the final loss function for a SIREN network trained to reconstruct an image B based on a masked version b is:

$$RMSE(h * \phi(b), b) + prior$$

5 Inpainting on a whole dataset

This section takes the former topic further and expands a model able to inpaint a single image, to an architecture able to inpaint a whole dataset of varied images, in this case the *CelebA* dataset. This raises a number of challenges, especially about generalization of the model and memory usage. In this experiment, the model should be able, given a subsample of any image in the dataset, to reconstruct the corresponding image. However, using a single SIREN architecture may be inefficient for this task: if the model is too small (does not contain enough parameters), it will learn an "average" representation of the images in the dataset, due to its lack of memory. Memorizing the whole diversity of images in the dataset would require a huge SIREN model, which could be prone to overfitting as well as having very inefficient memory usage.

This problem can be addressed by using a hypernetwork architecture: instead of directly mapping a masked signal to its reconstitution, the model maps the signal to the parameters of a SIREN that will reconstruct the original image. This method hugely increases the generalization and memory efficiency of the model as SIREN parameters can easily be mapped to a low dimensional latent

Figure 5: Results of the hypernetwork architecture on the CelebA dataset with images center-cropped and downsampled to $32 \times 32 \times 3$ dimensions.

vector, as shown in figure 4. The loss function optimized by such an architecture is:

$$\frac{1}{H \times W} \underbrace{\|\mathcal{H} \circ \mathcal{C}(b) - B\|_2^2}_{\text{likelihood}} + \underbrace{\lambda_1 \frac{1}{k} \|z\|_2^2}_{\text{regularization of z}} + \underbrace{\lambda_2 \frac{1}{l} \|p\|_2^2}_{\text{regularization of p}} \quad \text{where :}$$

- H, W : height, width of the image ;
- $\mathcal{H}, \mathcal{C}$: hypernetwork, encoder. The encoder can be either a CNN or a MLP. By abuse of notation, $\mathcal{H} \circ \mathcal{C}$ here represents the image reconstructed by the SIREN model generated by the hypernetwork ;
- B, b : original, masked image. Note that this time, the model is supervised using the full image B. Otherwise, accurate learning would be impossible ;
- z, k : latent code and its dimension ;
- p, l : SIREN weight vector and its dimension ;
- the $\|z\|_2^2$ and $\|p\|_2^2$ components can be interpreted as regularization terms for p and z. In the case of p, as many SIREN networks can represent the same kind of images, this incites the model to choose the "lowest frequency" representation. For the case of z, this incites the model to choose the "smallest" latent code representation.

One important fact to note for this architecture is that SIREN weights are not updated from back-propagation, as they come from the output of the hypernetwork. Instead, it is the weights of the encoder and hypernetwork that are updated during training.

One can see from figure 4 that the architecture is able to reconstruct the main trends of an image using only 1% of its context (100 pixels out of 1024). However, even when provided with the full context, the network outputs an image that is not as detailed as one might expect. This is because the network is not trained to reproduce a single image : as such, it maps the input to a SIREN model

Number of Context Pixels	10	100	1000	512 (Half)	1024
CNP [16]	0.039	0.016	0.009	-	-
Sine Set Encoder + Hypernet.	0.035	0.013	0.009	0.022	0.009
ReLU Set Encoder + Hypernet.	0.040	0.018	0.012	0.026	0.012
PConv CNN Encoder + Hypernet.	0.046	0.020	0.018	0.060	0.019
CNN Encoder + Hypernet.	**0.033**	**0.009**	**0.008**	**0.020**	**0.008**

Figure 6: Performance of the hypernetwork architecture depending on the type of encoder (CNN, MLP, SIREN, etc.) The metric used here is the RMSE per pixel.

that is not necessarily the best one, even when given the full context. This however, shows that the architecture does not overfit images, and is testimony to its flexibility.

To conclude this section, it is important to stress that inpainting tasks are not solely about images. Many real world tasks involve reconstructing a continuous signal based on a sparse sample of it, including radar and sonar networks, seismic imaging, meteorological stations, medical imaging and even reconstruction of audio or video signals. SIREN networks, overall, offer a powerful and promising tool for all these domains. Combined with convolution or generative architectures, they could bring computer vision and implicit representations to another level of detail, fidelity and performance.

6 Beware of beguiling SIRENs...

SIREN networks are thus extremely powerful architectures for modeling continuous and smooth signals. They particularly excel at image, audio, video and even 3-dimensional object treatment, especially when compared with other more standard architectures such as ReLU, sigmoid or tanh. However, it is important to stress that SIRENs are not a panacea, as they struggle with non-smooth or non-continuous data. This means that SIRENs are not as good as other architectures for many important area, such as:

- Natural language processing, as textual data is inherently discrete ;
- Graph processing ;
- Classification tasks.

SIREN models also face extrapolation problems beyond their training domain. Their ability to interpolate is perfect for inpainting tasks, but there performance is still mediocre for outpainting tasks, which involves expanding the original domain, as shown in figure 7. The hypernetwork architecture, which is in fact a combination of SIREN networks and more standard CNN or MLP structures, might help these models to get the best of both worlds.

Finally, SIREN networks also suffer from possibly tedious hyperparameter tuning, including the ω_0 and weighting loss parameters discussed above, that must be adjusted specifically to each task. All these challenges show that research still has to be done in order to make SIREN architectures more versatile.

7 Conclusion

One of the main takeaways from this work on SIREN networks is that experimentation is an important aspect of deep learning : sometimes, even a small change can bring interesting results, and using sine activation is merely one of many possibilities. From splines to probability densities, truncated sine functions or exponential functions, many possibilities remain to be tested and will likely find strengths of their own.

But after all, why settle with activation functions only ? Another important part of neural networks is the dot product that is computed in each linear layer. It is well known that Support Vector Machines, which are very similar to neurons, can use non-linear kernels with interesting properties,

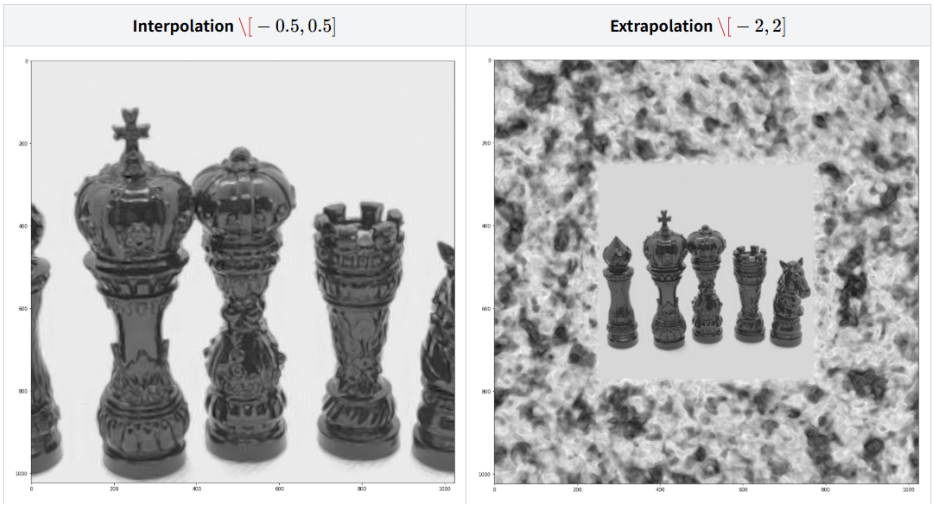

Figure 7: SIRENs are excellent at inpainting, but mediocre at outpainting. In this case, a SIREN model was trained to fit an image containing chess pieces. The coordinates were normalized in the $[-1, 1]^2$ interval as usual. In the left case, we ask the model to render the part of the image comprised in the $[-0.5, 0.5]^2$ interval. In the right case, the model is asked to extrapolate beyond its training domain in the $[-2, 2]^2$ interval.

such as power of a dot product or exponential of a norm. In the end, all these experiments could lead to a more general vision of deep learning and a better theoretical framework.