Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains[1]

July 8, 2020

[1] *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains* by Tancik et al., 2020

Overview if INRs

- Recently, there appeared a new paradigm of representing signals
- One can represent images/3D-objects/video/audio/etc as *implicit neural representations* (INR)
- INR is a neural network $\Phi(v)$ which takes a coordinates vector v and produces a pixel/voxel/frequency/etc value at that point:

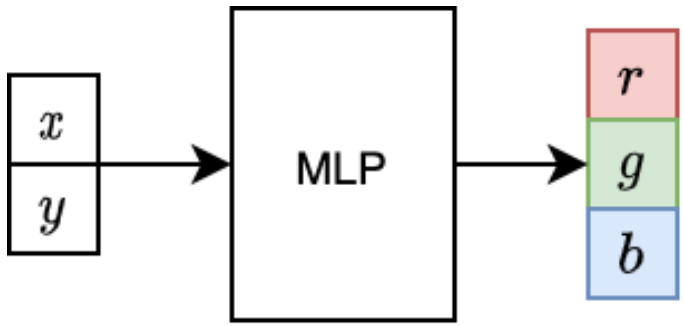


Figure: INR for 2D images: at each coordinate we predict a pixel value

- I.e. one neural network corresponds to a single image.

Overview of the paper

- Authors showed that standard MLPs fail to learn high frequency functions (both in theory and in practice)
- They showed that using "fourier features" overcomes the issue
- They test their ideas on a range of INR tasks: reconstructing images/3D-shapes/MRI-images/etc
- They used JAX to implement this all

Intuition. Part 1/3: the foreword

- Let's roughly divide images into low-frequency images and high-frequency images:
  - In a low-frequency image, things change slow from pixel to pixel
  - In a high-frequency image, things change fast

(a) A low-frequency image

(b) A high-frequency image

Intuition. Part 2/3: the problem

- Neural networks are bad at distinguishing small values between each other
- For example, if your input is 0.03 and you replace it with 0.05, your model will likely not to notice any difference
- And this is the main cause of why MLPs fail to model high-frequency images:
  - Our input coordinates for an INR lie in $[0, 1]$ domain
  - For high-frequency images, output changes a lot when an input changes slightly
  - Our model is incapable to detect small changes in the input
  - The result: MLPs "oversmooth" high-frequency images

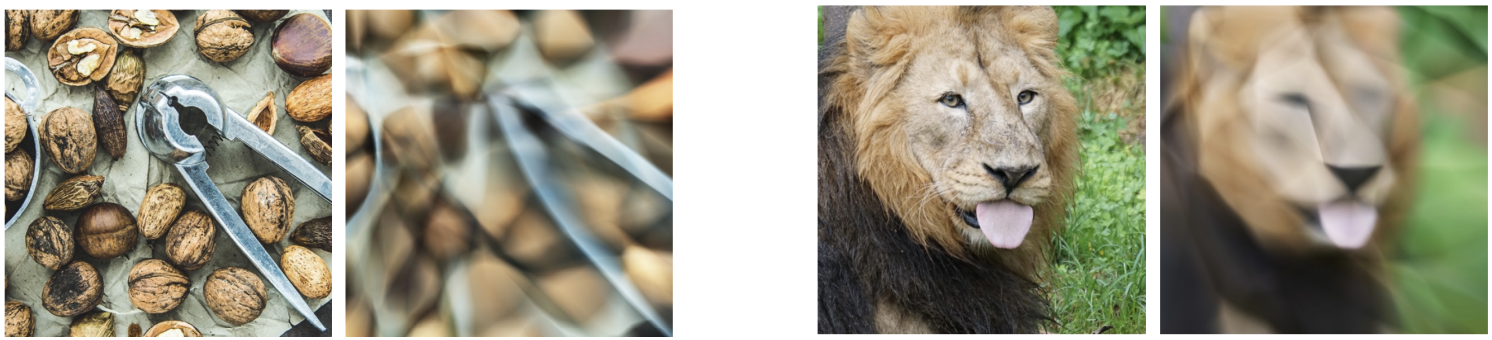


Figure: INR is trained to reconstruct an image: MLP oversmoothes high-frequency images

Intuition. Part 3/3: the solution

- How can we help the model to detect small changes in an input?
  - Step 1: multiply by a very large number
  - Step 2: apply $sin(x)$ or $cos(x)$

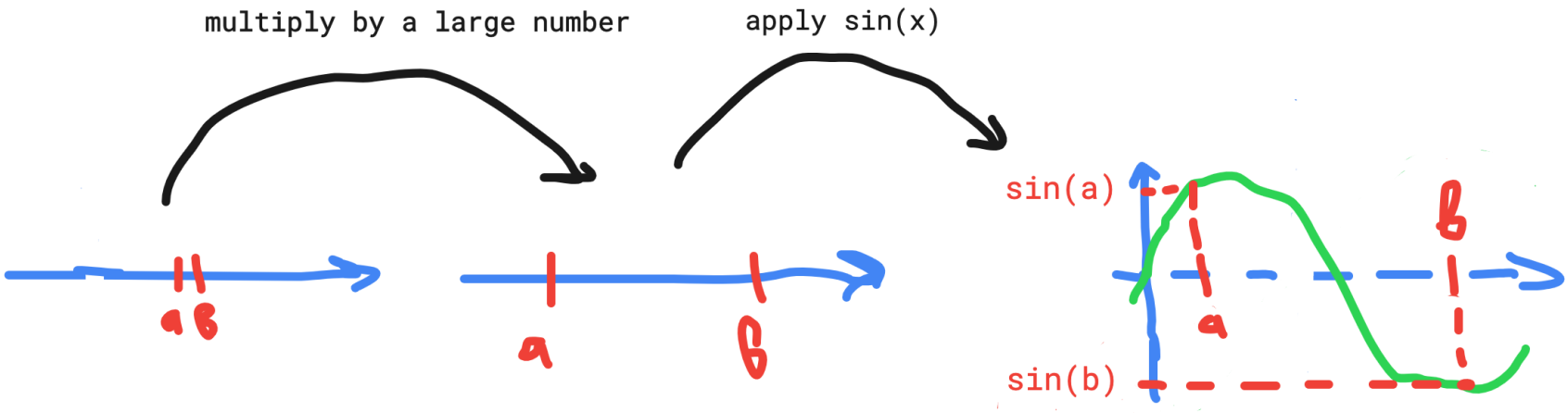


Figure: Intuition of the positional encoding trick

- Why do we need sines/cosines?
  - Neural networks do not like large-magnitude values (due to exploding variance)
  - Authors also show that this allows a model to have a shift-invariance property

Kernel regression

- Kernel regression is the following machine learning algorithm:
  - Obtain a training dataset $\boldsymbol{X}, \boldsymbol{y}$
  - Obtain a kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ (it's like a similarity function)
  - Compute kernel matrix $\boldsymbol{K}$ where $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  - Compute the value for a test point $\boldsymbol{x}$ by:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \left(\mathbf{K}^{-1}\mathbf{y}\right)_i k\left(\mathbf{x}_i, \mathbf{x}\right) \tag{1}$$

NTK theory

NTK (Neural Tangent Kernel) theory explores the behaviour of infinite-width MLPs:

- If the learning rate is small then the MLP simplifies to kernel regression with:

$$k_{\mathrm{NTK}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \mathbb{E}_{\theta \sim \mathcal{N}} \left\langle \frac{\partial f\left(\mathbf{x}_i; \theta\right)}{\partial \theta}, \frac{\partial f\left(\mathbf{x}_j; \theta\right)}{\partial \theta} \right\rangle$$

- Matrix $\boldsymbol{K}$ is poorly conditioned for a regular MLP, i.e. it has few large eigenvalues and others are small
- NTK shows that the components of the target function that correspond to large eigenvalues are learned much faster
- Components with large eigenvalues have lower frequency
- This makes the learning of high-frequency patterns too slow
- Authors propose to preprocess input coordinates with Fourier mapping:
  - This would make k_{NTK} be better conditioned
  - This would make k_{NTK} stationary[2] which gives invariance to shifts

[2] i.e. $k(\boldsymbol{x}, \boldsymbol{x}') = h(\boldsymbol{x} - \boldsymbol{x}')$ for some h

Fourier mapping for INRs

- Authors propose to embed the coordinates with:

$$\gamma(\mathbf{v}) = \left[\cos\left(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}\right), \sin\left(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}\right), \ldots, \cos\left(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}\right), \sin\left(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}\right)\right]^{\mathrm{T}}$$

  where $\boldsymbol{b}_i$ are random vectors of size d (for images, $d = 2$)
- This makes the NTK kernel stationary and have a better spectrum
- Authors sample $\boldsymbol{b}_i$ from $\mathcal{N}(0, \sigma^2)$ where σ^2 is a large number

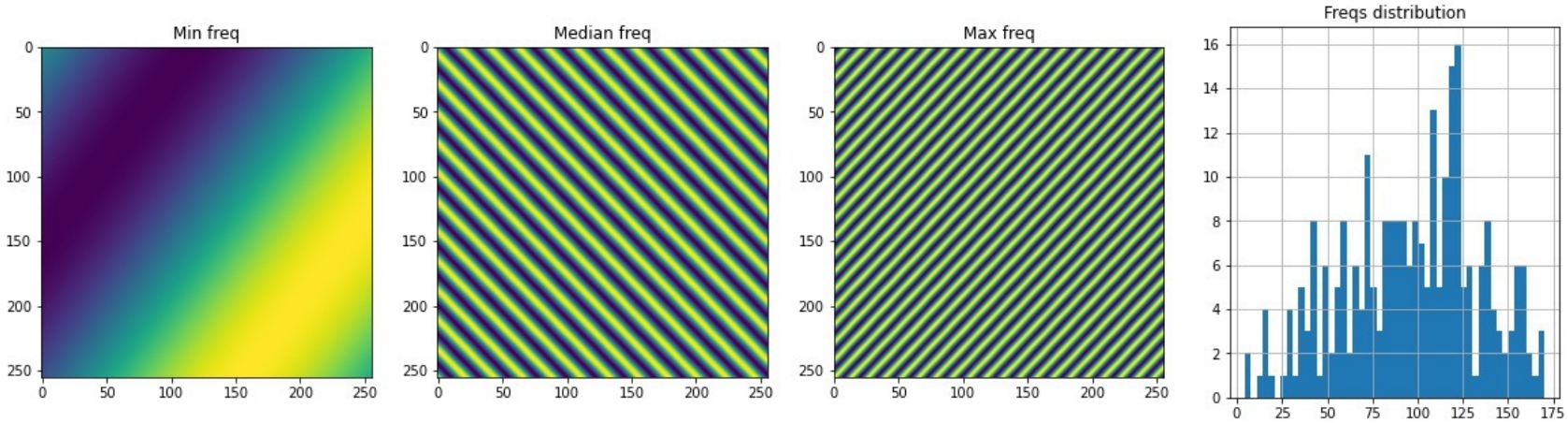
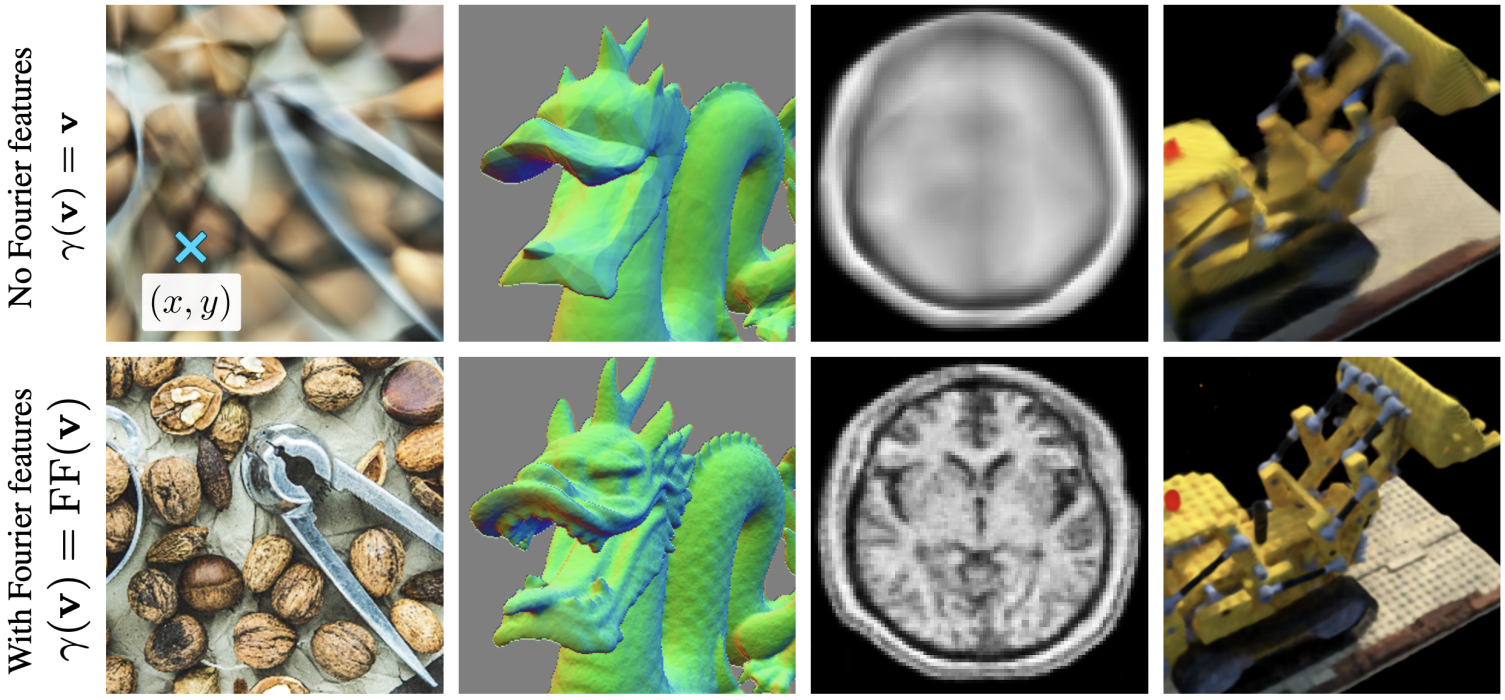


Figure: Visualization of the embeddings for some of the vectors $\boldsymbol{b}_i$ for 256×256 coordinates. Vectors $\boldsymbol{b}_i$ are sampled from uniform distribution.

Results



(b) Image regression
$(x,y) \rightarrow$ RGB

(c) 3D shape regression
$(x,y,z) \rightarrow$ occupancy

(d) MRI reconstruction
$(x,y,z) \rightarrow$ density

(e) Inverse rendering
$(x,y,z) \rightarrow$ RGB, density

Conclusion

- Authors provide an interesting (but not very transparent) theoretical analysis of INRs
- Their theoretical claims are confirmed by empirical evidence
- They achieve strong results on several domains
- One needs to search for optimal σ value when sampling $\boldsymbol{b}$ from $\mathcal{N}(0, \sigma^2)$