Autoregressive Generation of Neural Field Weights

Using a transformer based architecture

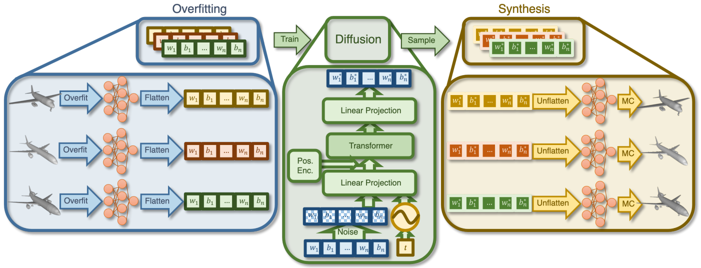
from Luis Muschal and Luca Fanselau

Table of Content

Related works

HyperDiffusion

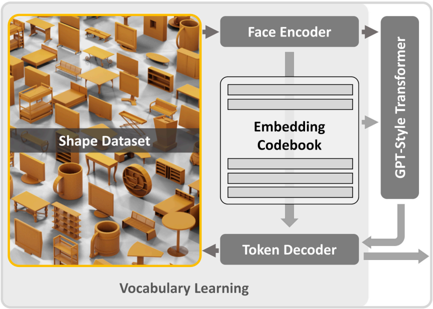
Operates on MLP weights directly to generates new neural implicit fields encoded by synthesized MLP parameters



ICCV'23 [Erkoç et al.]: Hyperdiffusion

MeshGPT

Autoregressively generate triangle meshes as sequences of triangles using a learned vocabulary of latent quantized embedding as tokens



CVPR'24 [Siddiqui et al.]: MeshGPT

Autoregressive Generation of Neural Field Weights

Neural Fields

Input coordinate location in n-dimensional space are mapped to target signal domain

Example:

With S being a surface in a 3-dimensional space $\mathbb{R}^3$. The Signed Distance Function $f : \mathbb{R}^3 \to \mathbb{R}$ is defined for a point $\mathbf{p} \in \mathbb{R}^3$ as:

$$f_\Theta(\mathbf{p}) = \begin{cases} \text{distance}(\mathbf{p}, S) & \text{if } \mathbf{p} \text{ is outside } S, \\ 0 & \text{if } \mathbf{p} \text{ is on } S, \\ -\text{distance}(\mathbf{p}, S) & \text{if } \mathbf{p} \text{ is inside } S, \end{cases}$$

Autoregressive Generation of Neural Field Weights

Autoregressive Process

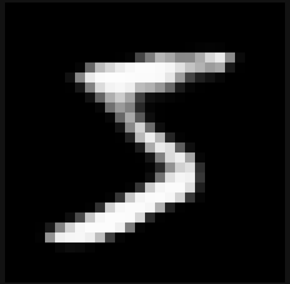
- Goal: generative modeling of neural fields $P(\theta_i \mid \theta_{i-1}, \theta_{i-2}, \ldots, \theta_0)$
- Using a generally available preset for GPT-like Architecture (like nanoGPT)
- Use Transformer to sample from the Probability, eg. $\theta_i = \mathrm{Transformer}(\theta_{i-1}, \theta_{i-2}, \ldots, \theta_0)$

Autoregressive Generation of Neural Field Weights

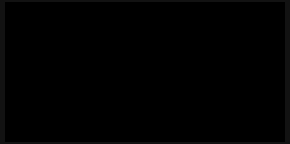
From nanoGPT to Regression Transformer

nanoGPT	vs.	Our Regression Transformer
Tokenizer		MLP Embedding on weight
Embedding + Positional Embedding		
N x Blocks (Causal Self Attention and MLP)		
Linear Transformation Embedding		
Cross-Entropy Loss		L1-norm as Loss

Ground Truth



N=1



Iteration: 0

Regression Transformer

Observing the Effects of Increasing N

- Transformer fails to capture the structure of the weights for larger N
- Why can't the sequence be remembered even for small values of N?

Iteration: 0

Challenges: Permutation Symmetries

The same signal can be represented by different weight matrices

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

permutated weight matrices are calculated using:

$$\tilde{W}_0 = PW_0$$
$$\tilde{W}_1 = W_1P^T$$

Overfitting Neural Fields

Finding a Solution

Minimize structural change by conditioning the training process using weight initialization

Approach:

Overfit single sample

Use weights for different sample (conditioned)

Train sample on randomly initialized weights (unconditioned)

Overfitting Neural Fields

Overfitting on one sample

Minimize structural change by conditioning the training process using weight initialization

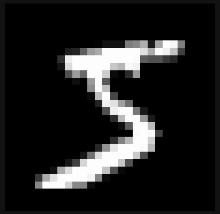
Approach:

Overfit single sample

Use weights for different sample (conditioned)

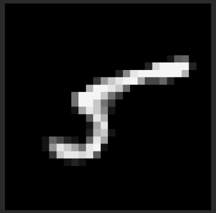
Train sample on randomly initialized weights (unconditioned)

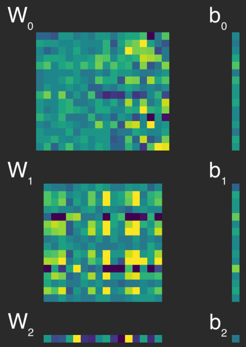
Ground Truth



First Sample

Legend





Epoch: 0

Overfitting Neural Fields

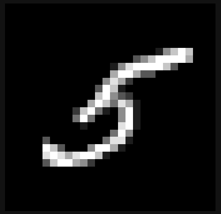
Overfitting on other sample

Minimize structural change by conditioning the training process using weight initialization

Approach:

- Overfit single sample
- Use weights for different sample (conditioned)
- Train sample on randomly initialized weights (unconditioned)

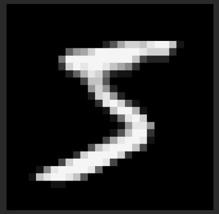
Ground Truth

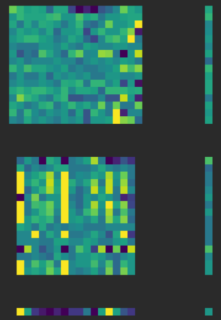


Unconditioned

Conditioned

Condition



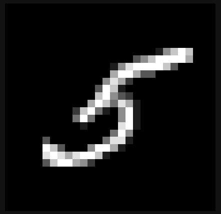
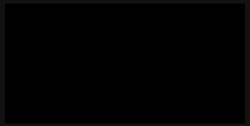
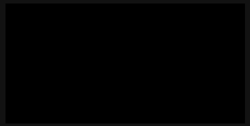
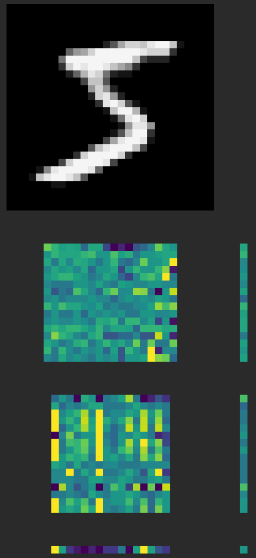


Epoch: 0

Overfitting Neural Fields

Visualizing the Difference:

$$\Delta(W) = W_{\text{pretrained}} - W$$
$$\Delta(b) = b_{\text{pretrained}} - b$$

Ground Truth	Unconditioned	Conditioned	Condition
			

Epoch: 0

Reminder: Regression Transformer

How far we got with unconditioned neural fields

- Transformer fails to capture the structure of the weights for larger N
- Why can't the sequence be remembered even for small values of N?

Iteration: 0

Regression Transformer

Using conditioned neural fields to verify the Hypothesis

- Training Regression Transformer using conditioned Neural Fields Weights
- Structural similarity of weights improve the performance of the Transformer

Ground Truth

N=4

N=32

Iteration: 0

Conclusion and Outlook: Tokenization

Predicting the next MLP weight as a token

Run into issues regarding special tokens:

$$\theta_i = \text{Transformer}(\theta_{i-1}, \theta_{i-2}, \ldots, \theta_0) \rightarrow \theta_0?$$

Solution: Find Tokens to encode the MLP weights and transfer from Regression Transformer to Classical Transformer Architectures

First Approach:

- Create Tokens using conditioned Neural Field Weights
- Naive Attempt: Use weight distribution for discretization
- Vector Quantization Attempt: Find optimal token representation using optimization techniques

Second Approach:

- Find layer representations of unconditioned neural fields that are permutation equivariant
- For example by using the graph structure of the neural fields and employing deep learning techniques suited for graphs

Thank you for your attention!

We hope you enjoyed our presentation and are looking forward to your questions.