AI in the Sciences and Engineering 2024: Lecture 16

Siddhartha Mishra

Seminar for Applied Mathematics (SAM), D-MATH (and),
ETH AI Center,
ETH Zürich, Switzerland.

What you learnt so far

- Operator learning: Given Abstract PDE: $\mathcal{D}_a(u) = f$
- Learn Solution Operator: $\mathcal{G} : \mathcal{X} \mapsto \mathcal{Y}$ with $\mathcal{G}(a, f) = u$
- Approximate with Operator Learning Algorithms:
  - CNN/UNet
  - DeepONet
  - FNO
  - CNO
- We focus on approximating Time-dependent problems

Time-dependent PDEs

- Of the Abstract form:

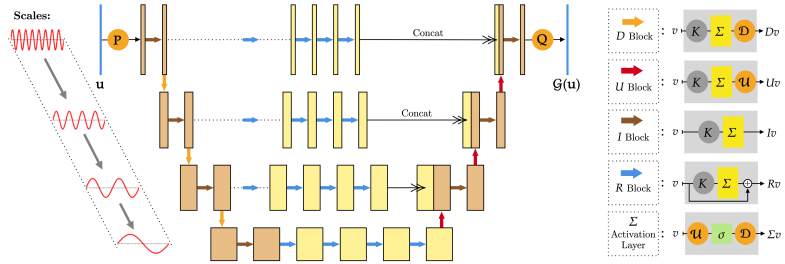
$$u_t + \mathcal{L}(t, x, u) = 0, \quad u(0) = \bar{u}.$$

- Solution operator: $\mathcal{S} : (0, T) \times \mathcal{X} \mapsto \mathcal{X}$; $\mathcal{S}(t, \bar{u}) = u(t)$
- Generated data is the form of Trajectories:

$$(u(0), u(t_1), u(t_2), \ldots, u(T)) = (\bar{u}, \mathcal{S}(t_1, \bar{u}), \mathcal{S}(t_2, \bar{u}), \ldots, u(T))$$

- Learning Task:
- Given $\bar{u}$ + BC: generate the solution trajectory $u(t)$, for all $t \in (0, T]$
- Direct or Autoregressive Evaluation of Neural operators is not efficient.

Lead Time Conditioning



- Lead Time as an Input Channel
- $\mathrm{CNO}\ (\bar{t}, u(t)) \approx \mathcal{S}(\bar{t}, u(t)) = u(t + \bar{t})$.
- Add Conditional Normalizations after each layer !!

$$\mathcal{N}(w) = g_N(t) \odot \frac{w - \mathbb{E}(w)}{\sqrt{\mathrm{Var}\ (w) + \epsilon}} + h_N(t),$$

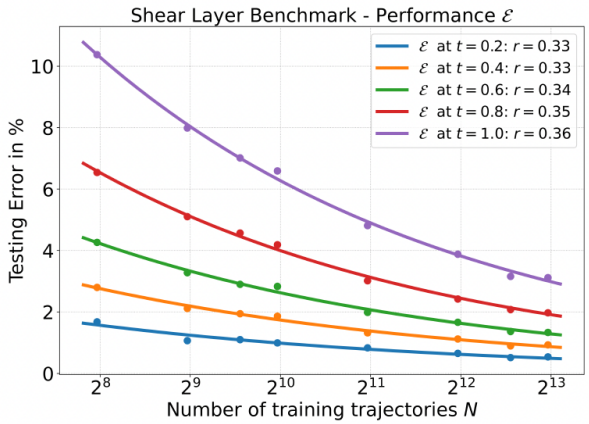
- g_N, h_N are MLPs in general.
- Instance,Batch,Layer Normalizations.

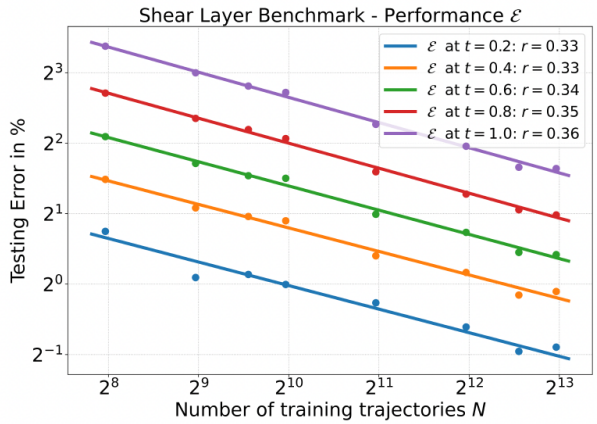
Training Strategies

- One at a Time training based on:
- Input-Target Pairs: $\bar{u}, \mathcal{S}(t_k, \bar{u}) = u(t_k)$
- For $t_K = T$, K training samples per trajectory.
- all2all training based on:
- Input-Target Pairs: $u(t_i), \mathcal{S}(t_j - t_i, u(t_i)) = u(t_j), \forall i < j$
- $\frac{K^2+K}{2}$ training samples per trajectory !!
- Inference is Direct or Autoregressive
- Multiple possibilities for Autoregressive Rollouts
- Evaluation at any time $t > 0$ including Out-of-distribution times.

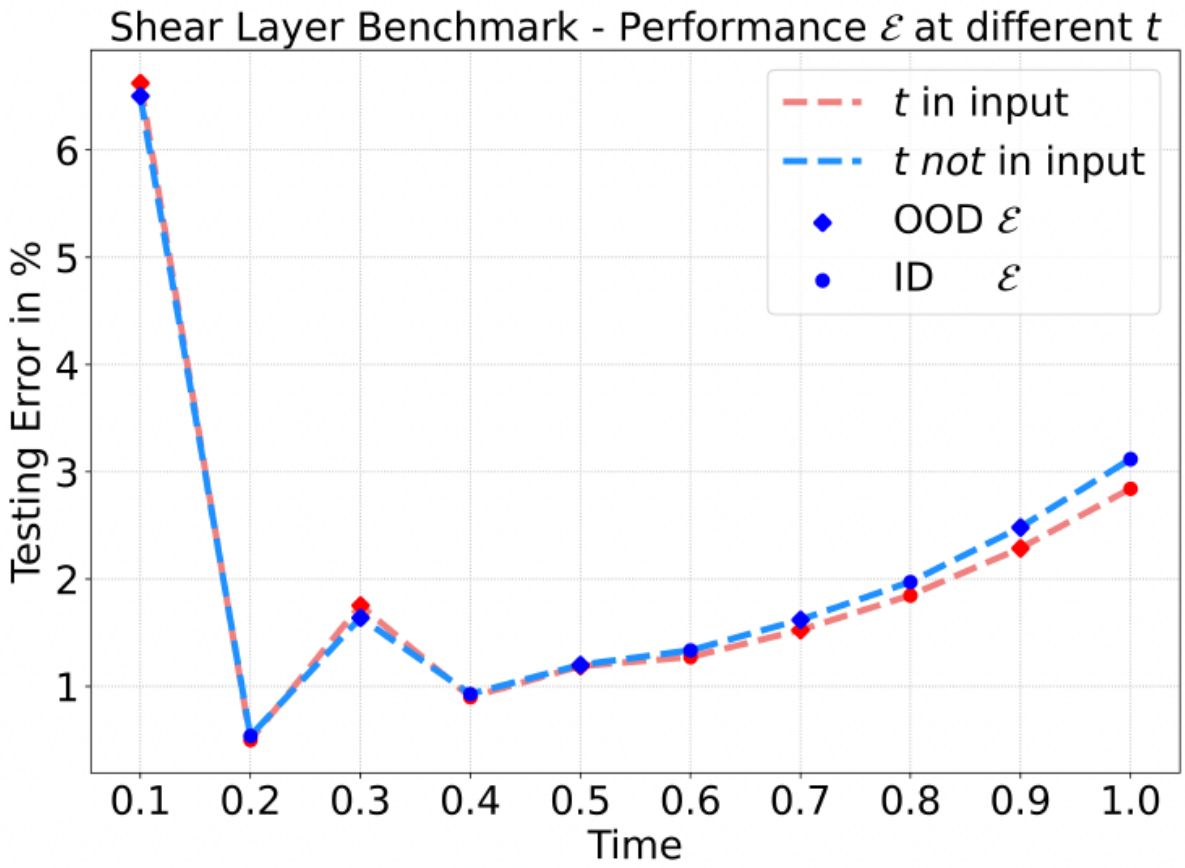
Results for Shear Layer

Error vs. Time

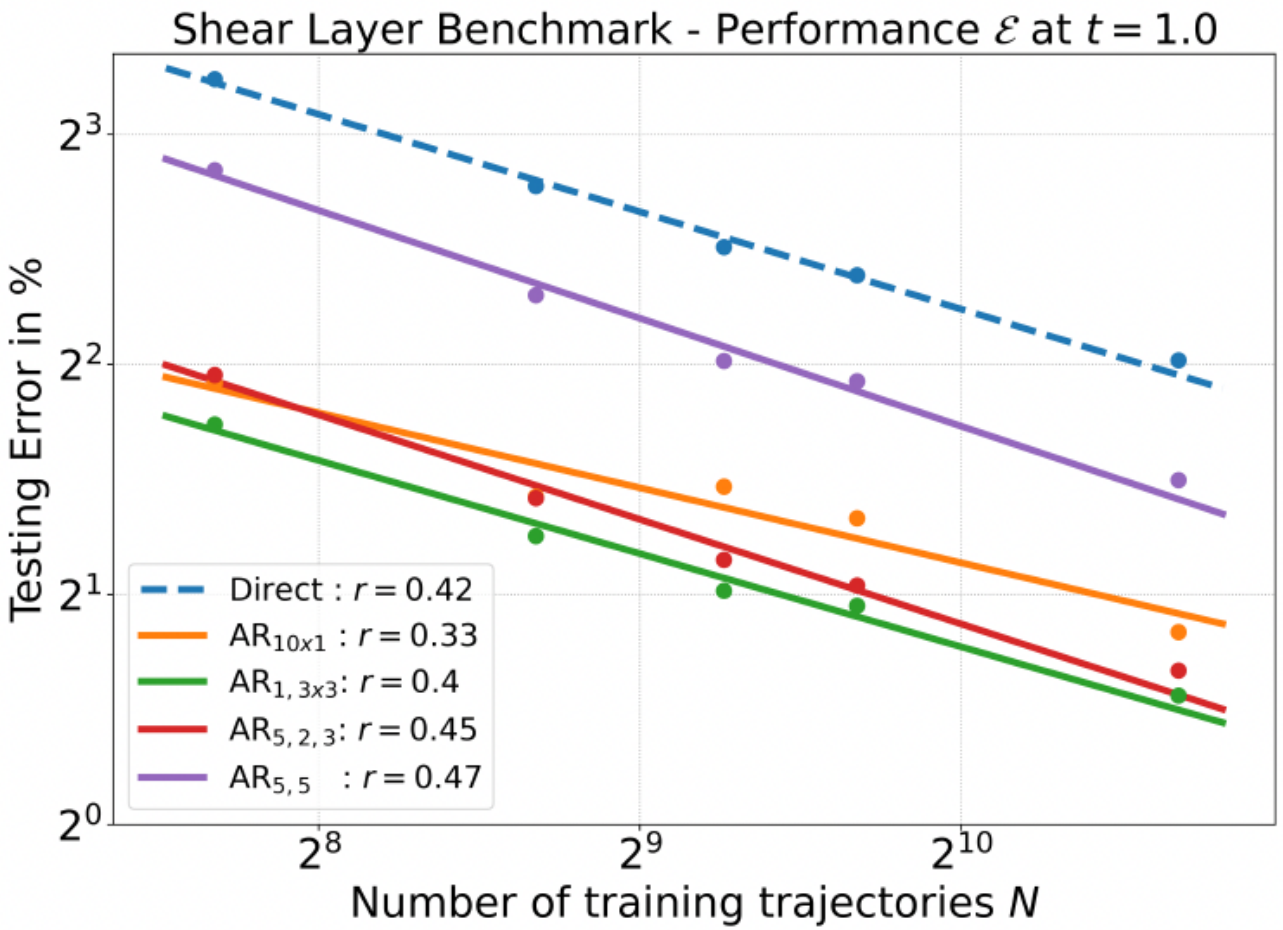




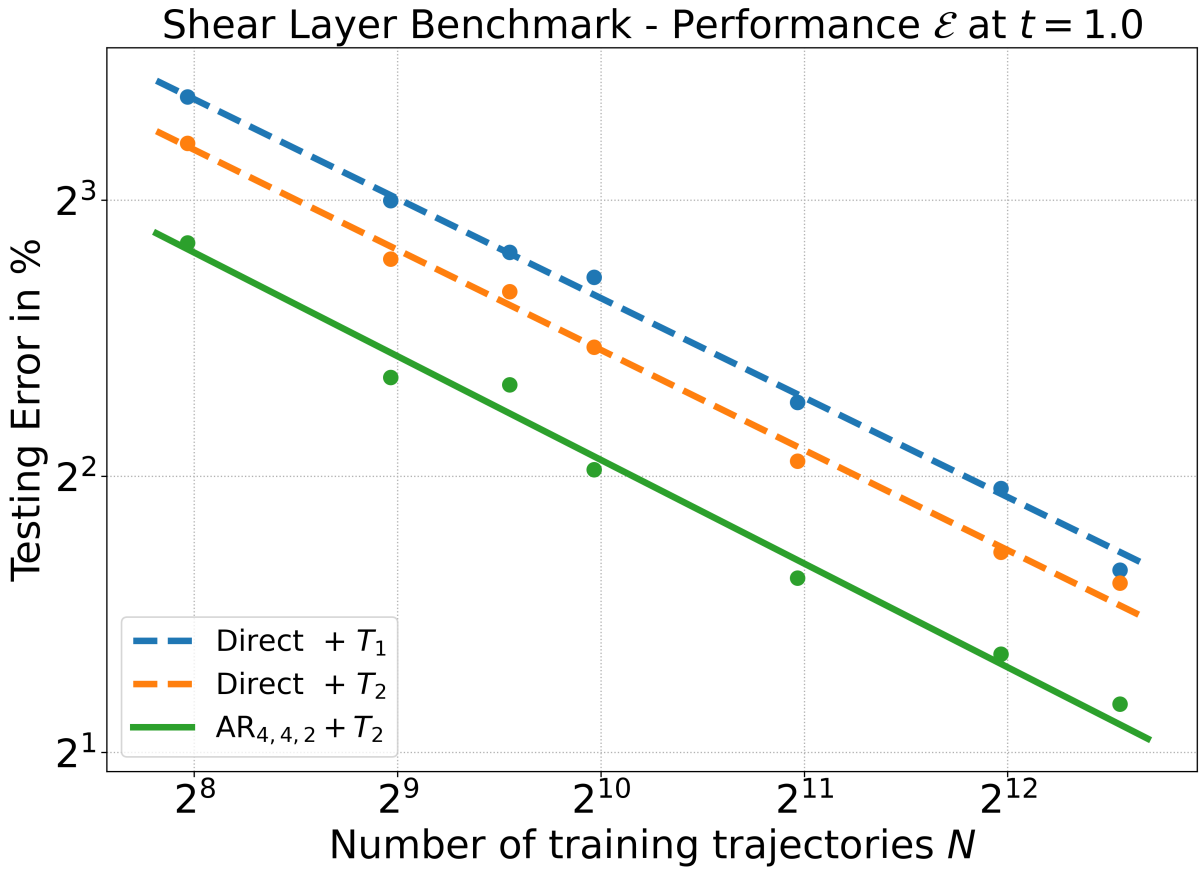
Results at OOD time levels.



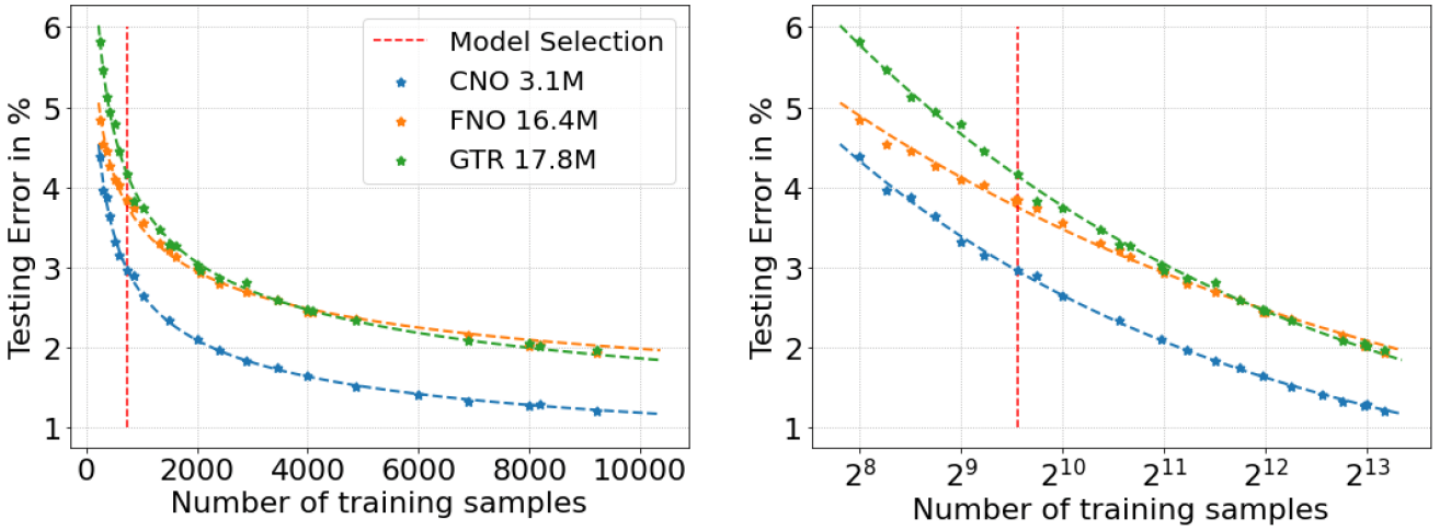
Results for Different Strategies I.



Results for Different Strategies II.



Key is Scaling



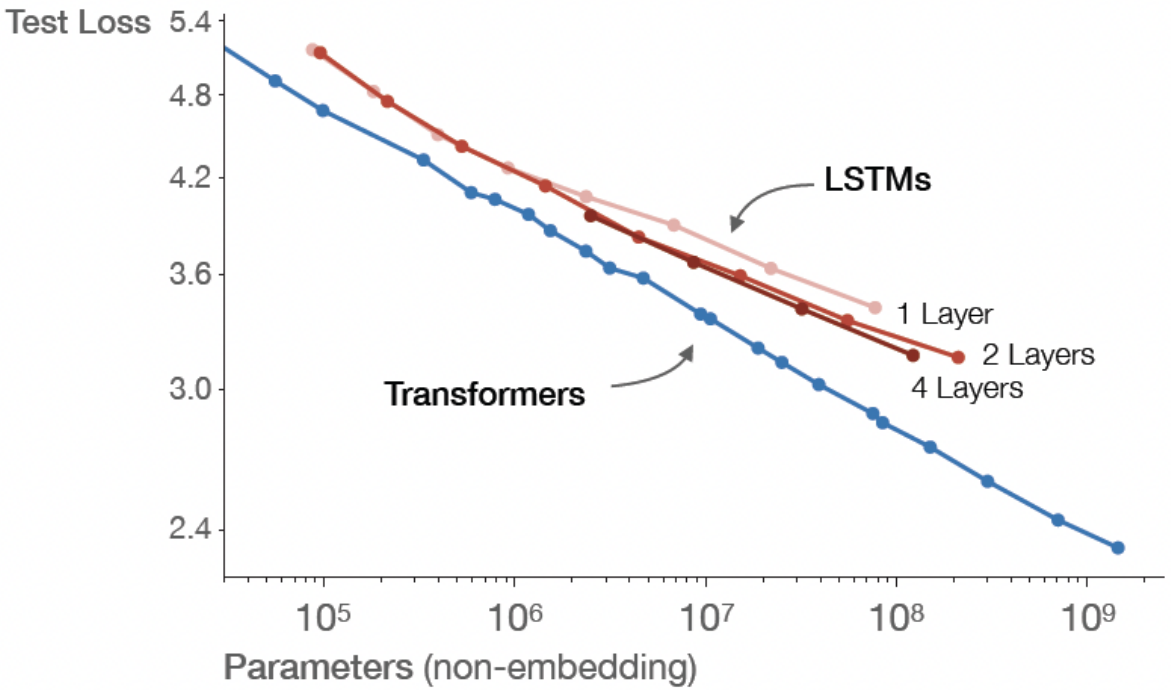
What Architectures Scale the best ?

What Architectures Scale the best ?

- What happens in Language Modeling ?

What Architectures Scale the best ?

- What happens in Language Modeling ?



Move Towards Transformers

Attention is all you need !!

- Introduced by Vaswani et. al, 2017
- Given K-inputs $\mathsf{v} \in \mathbb{R}^{K\times n}$ of n-features:
- Self-Attention: $\mathbb{A} : \mathbb{R}^{K\times n} \mapsto \mathbb{R}^{K\times n}$ such that $\mathsf{u} = \mathbb{A}\mathsf{v}$:

$$\mathsf{u}_k = \mathsf{W}\sum_{j=1}^{K} \mathrm{softmax}_k\left(\frac{\langle \mathsf{Q}\mathsf{v}_k, \mathsf{K}\mathsf{v}_j\rangle}{\sqrt{m}}\right)\mathsf{V}\mathsf{v}_j, (\mathrm{softmax}\ (w))_i = \frac{e^{w_i}}{\sum_{\ell=1}^{L} e^{w_\ell}}$$

- Query $\mathsf{Q} \in \mathbb{R}^{m\times n}$, Key $\mathsf{K} \in \mathbb{R}^{m\times n}$, Value $\mathsf{V} \in \mathbb{R}^{m\times n}$.
- Output Matrix $\mathsf{W} \in \mathbb{R}^{n\times m}$
- Permutation Invariance

Multi-Head Attention

- Multi-Head Self-Attention:

$$\mathsf{u}_k = \sum_{h=1}^{H} \mathsf{W}_h \sum_{j=1}^{K} \mathrm{softmax}_k \left(\frac{\langle \mathsf{Q}_h \mathsf{v}_k, \mathsf{K}_h \mathsf{v}_j \rangle}{\sqrt{m}} \right) \mathsf{V}_h \mathsf{v}_j$$

Attention as a Neural Operator: I

- Let $\mathsf{v} \in C(D, \mathbb{R}^n)$ be the input function.
- $x_k \in D$ be sampling points on a Regular Grid with size Δ
- Apply Self-Attention to Tokens $\mathsf{v}_k = \mathsf{v}(x_k)$:

$$\mathsf{u}_k = \mathsf{W} \sum_{j=1}^{K} \mathrm{softmax}_k \left(\frac{\langle \mathsf{Q}\mathsf{v}_k, \mathsf{K}\mathsf{v}_j \rangle}{\sqrt{m}} \right) \mathsf{V}\mathsf{v}_j, (\mathrm{softmax}\ (w))_i = \frac{e^{w_i}}{\sum_{\ell=1}^{L} e^{w_\ell}}$$

- Passing to the limit as $\Delta \to 0$ yields

$$\mathsf{u}(x) = \mathbb{A}(\mathsf{v})(x) = \mathsf{W} \int_D \frac{e^{\frac{\langle \mathsf{Q}\mathsf{v}(x), \mathsf{K}\mathsf{v}(y) \rangle}{\sqrt{m}}}}{\int_D e^{\frac{\langle \mathsf{Q}\mathsf{v}(z), \mathsf{K}\mathsf{v}(y) \rangle}{\sqrt{m}}} dz} \mathsf{V}\mathsf{v}(y) dy.$$

Attention as a Neural Operator: II

- Operator Attention $\mathbb{A} : C(D, \mathbb{R}^n) \mapsto C(D, \mathbb{R}^n)$ with

$$\mathrm{u}(x) = \mathbb{A}(\mathrm{v})(x) = \mathrm{W} \int_D \frac{e^{\frac{\langle \mathrm{Qv}(x), \mathrm{Kv}(y) \rangle}{\sqrt{m}}}}{\int_D e^{\frac{\langle \mathrm{Qv}(z), \mathrm{Kv}(y) \rangle}{\sqrt{m}}} dz} \mathrm{Vv}(y) dy.$$

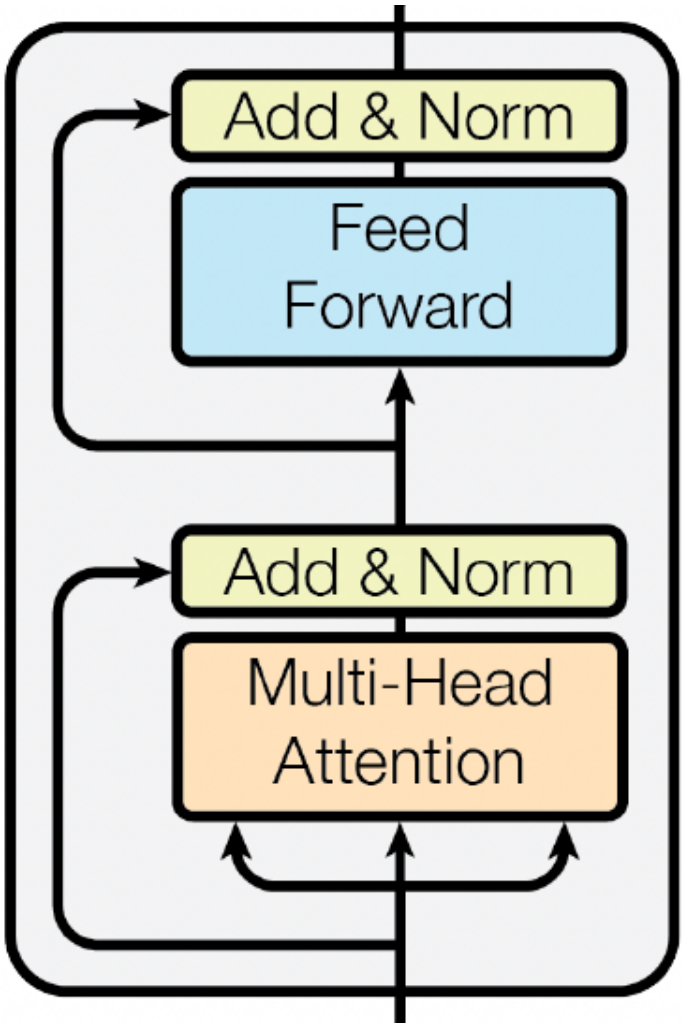
- Can be interpreted as as a Nonlinear Kernel Neural Operator:

$$\mathbb{A}(\mathrm{v})(x) = \int_D K(v(x), v(y)) v(y) dy.$$

- In contrast, FNO/CNO etc are Linear Kernel Neural Operators:

$$\mathbb{C}(\mathrm{v})(x) = \int_D K(x, y) v(y) dy = \int_D K(x - y) v(y) dy.$$

Additional Ingredients



Normalization ?

- Layer Normalization
- LayerNorm as a function:

$$LN : \mathbb{R}^n \mapsto \mathbb{R}^n, \quad LN(\mathsf{v}) = \alpha \odot \frac{\mathsf{v} - \mu_{\mathsf{v}}}{\sigma_{\mathsf{v}}} + \beta,$$

$$\mu_{\mathsf{v}} = \frac{1}{n}\sum_{j=1}^{n} \mathsf{v}_j, \quad \sigma_{bv}^2 = \frac{1}{n}\sum_{j=1}^{n} (\mathsf{v}_j - \mu_{bv})^2$$

- Straightforward to realize LayerNorm as a pointwise operator

$$(LN\mathsf{v})(x) = LN(\mathsf{v}(x))$$

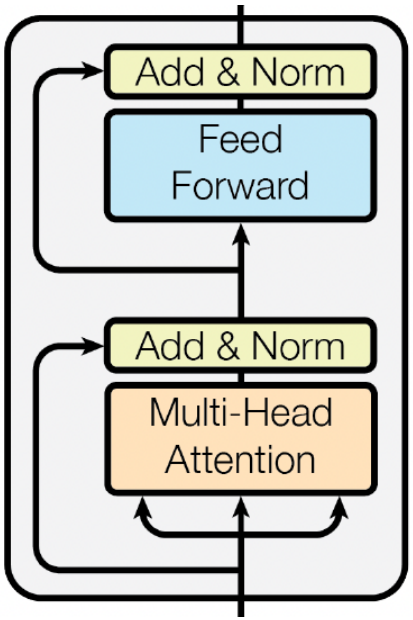
Feed Forward ?

- Use MLPs pointwise
- Operator interpretation is:

$$MLP(\mathrm{v})(x) = \overline{W}\sigma\left(W\mathrm{v}(x) + B\right), \quad \mathrm{v} \in \mathbb{R}^n$$

- with Weight Matrices $W \in \mathbb{R}^{d\times n}$, $\overline{W} \in \mathbb{R}^{n\times d}$
- and Bias vector $B \in \mathbb{R}^d$

An Operator Transformer Block



- A sequence of Operators of the form:
- Multi-head Self-Attention: $\bar{u} = MSA(v)$
- Residual + LayerNorm: $\hat{u} = LN(v + \bar{u}, \alpha_1, \beta_1)$
- MLP: $u' = MLP(\hat{u})$
- Residual + LayerNorm: $u = LN(u' + \bar{u}, \alpha_2, \beta_2)$

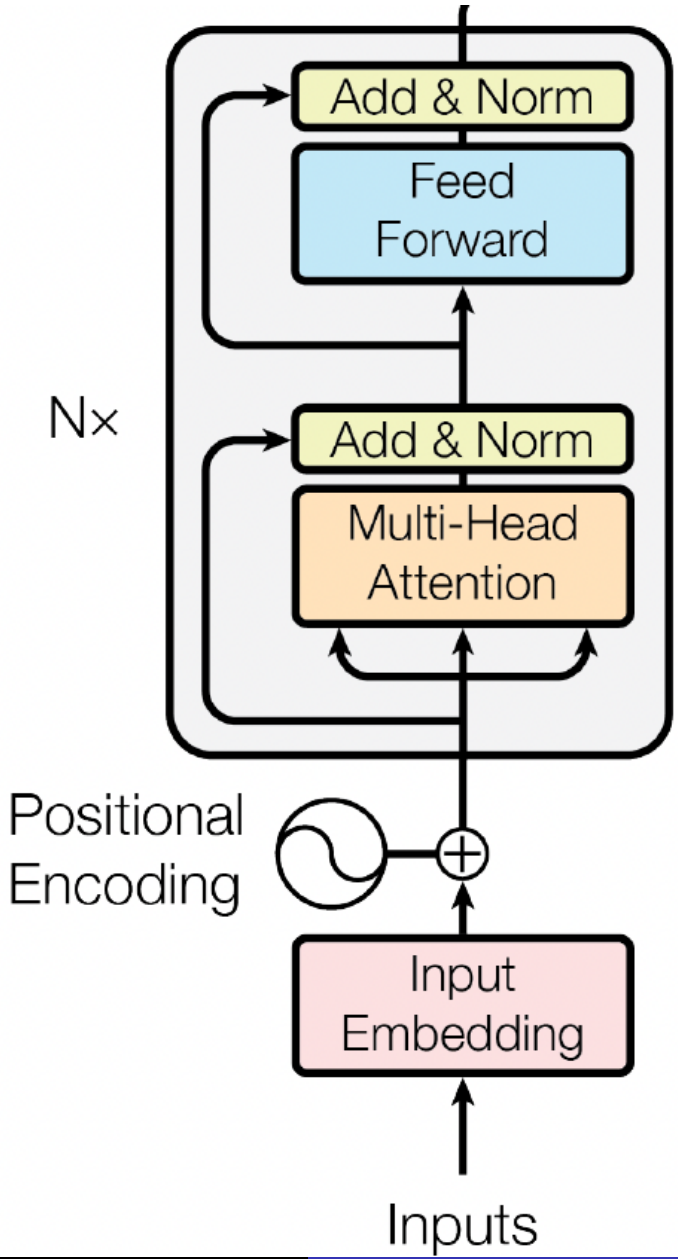
Issue: Where is Position ?

- All operators have Permutation Invariance.
- No sense of absolute or relative Position
- Have to include Positional encodings of form:

$$\hat{\mathrm{v}}_k = \overline{W}_v \sigma \left(W_v v_k \right) + \overline{W}_e \sigma \left(W_e e_k \right),$$

- With $e_k \in \mathbb{R}^K$ is the k-th unit vector
- Weight matrices
  $W_e \in \mathbb{R}^{d \times K}, \overline{W}_e \in \mathbb{R}^{n \times d}, W_v \in \mathbb{R}^{\bar{d} \times n}, \overline{W}_v \in \mathbb{R}^{n \times \bar{d}}$
- Fixed Sine-Cosine position encodings also work.

Final version of a Transformer Block



Caveat: Computational Complexity

- Computational Cost is Quadratic in # (Tokens) !!

$$\text{Compute} \sim \mathcal{O}(mnK^2)$$

- With K- Input Length, n Input features and m hidden dimension.
- But lots of possible Parallelism in Computation
- $\mathcal{O}(1)$ sequential operations.
- $\mathcal{O}(1)$ Path Length.
- Nevertheless, Infeasible for 2 or 3-d inputs.

Possible Solution

- Vision Transformers (ViT) of Dosovitskiy et. al.
- Key Ingredient: Patching (Patchification):
- Given Image on resolution $H \times W$, i.e. $\mathrm{v} \in \mathbb{R}^{H \times H \times C}$
- Divide into $N = \frac{H^2}{p^2}$ patches, each of size $p \times p$
- $\mathrm{v} \sim [\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_N]$ with $\mathrm{v}_k \in \mathbb{R}^{p^2 \times C}$, $1 \leq k \leq N$
- Introduce Patch Embeddings $\mathrm{E} \in \mathbb{R}^{n \times (p^2 \cdot C)}$
- Input is a Sequence of Patch Tokens:

$$\hat{\mathrm{v}} = [\mathrm{E}\mathrm{v}_1, \mathrm{E}\mathrm{v}_2, \ldots, \mathrm{E}\mathrm{v}_N]$$

- N Tokens, each with n features are fed into a transformer !!

Operator Version

- Patch Formation + Embeddings can be written as operator:

$$\hat{\mathrm{E}}(\mathrm{v})(x) = \sum_{k=1}^{N} \mathrm{F} \left(\int_{D_k} W(x)\mathrm{v}(x)dx \right) \mathbb{I}_{D_k}(x).$$

- With $D = \cup_{k=1}^{N} D_k$ non-overlapping and equal partition.
- With learnable $F \in \mathbb{R}^{n \times C}$
- Weight Function $W(x) = \sum_{1 \le i,j \le H} W_{ij}\delta_{z_{ij}}$
- With uniform grid points z_{ij} and discrete weights defined by,

$$\begin{aligned} W_{ij} &= \omega_{ij} \quad \text{if} \quad 1 \le i,j \le p \\ &= \omega_{i \bmod p, j \bmod p}, \quad \text{otherwise.} \end{aligned}$$

Issue: Where is Position ?

- Need to add Positional Encodings
- Input:
- Sequence of Patch Tokens + Learnable Positional Encodings

$$\hat{\mathrm{v}} = [\mathrm{Ev}_1, \mathrm{Ev}_2, \ldots, \mathrm{Ev}_N] + \mathrm{E}_{pos}$$

- With $E_{pos} \in \mathbb{R}^{n \times N}$
- Can be viewed as an operator:

$$\mathrm{E}_{pos}\mathrm{v}(x) = \sum_{k=1}^{N} w_k \mathbb{I}_{D_k}(x), \quad w_k \in \mathbb{R}^n, \ \forall 1 \leq k \leq N.$$

ViT operator Block

- For $D \subset \mathbb{R}^2$ + input $\mathsf{v} \in C(D, \mathbb{R}^C)$
- A sequence of Operators of the form:
- Patch Embeddings+ Positional Encoding: $\hat{\mathsf{v}} = \hat{\mathsf{E}}(\mathsf{v}) + \mathsf{E}_{pos}(\mathsf{v})$
- LayerNorm + MSA+ Residual: $\bar{\mathsf{u}} = \hat{\mathsf{v}} + MSA(LN(\mathsf{v}))$
- LayerNorm + MLP+ Residual: $\mathsf{u} = \bar{\mathsf{u}} + MLP(LN(\bar{\mathsf{u}}))$

Computational Complexity

- Given an Image at resolution $H \times W$
- Standard Transformer needs

$$\text{Compute} \sim \mathcal{O}((HW)^2)$$

- ViT needs

$$\text{Compute} \sim \mathcal{O}\left(\frac{(HW)^2}{p^4}\right)$$

- Still not scalable for small patch size p

Another Idea: Windowed Attention

- Introduced in Liu et. al.
- Use Windowed Attention:

- With M-Windows,

$$\text{Compute} \sim \mathcal{O}\left(\frac{HWM^2}{p^2}\right)$$

Operator version of Windowed Attention

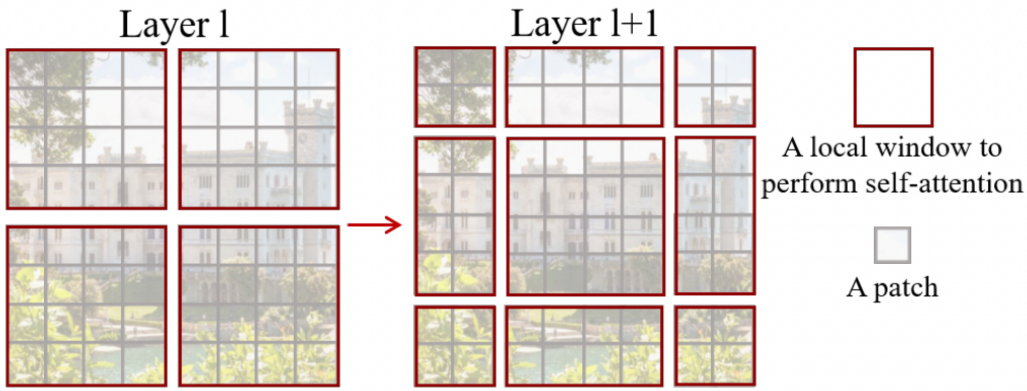
- For layer ℓ, Assume $D = \cup_{q=1}^{M} D_q^{w,\ell}$
- With non-overlapping Windows.
- Windowed Attention is instantiated as Operator:

$$\mathrm{u}(x) = \mathbb{A}_W(\mathrm{v})(x) = \mathrm{W} \int_{D_{q_x}^{w,\ell}} \frac{e^{\frac{\langle \mathrm{Qv}(x), \mathrm{Kv}(y) \rangle}{\sqrt{m}}}}{\int_{D_{q_x}^{w,\ell}} e^{\frac{\langle \mathrm{Qv}(z), \mathrm{Kv}(y) \rangle}{\sqrt{m}}} dz} \mathrm{Vv}(y) dy.$$

- Where $1 \leq q_x \leq M$ such that $x \in D_{q_x}^{w,\ell}$

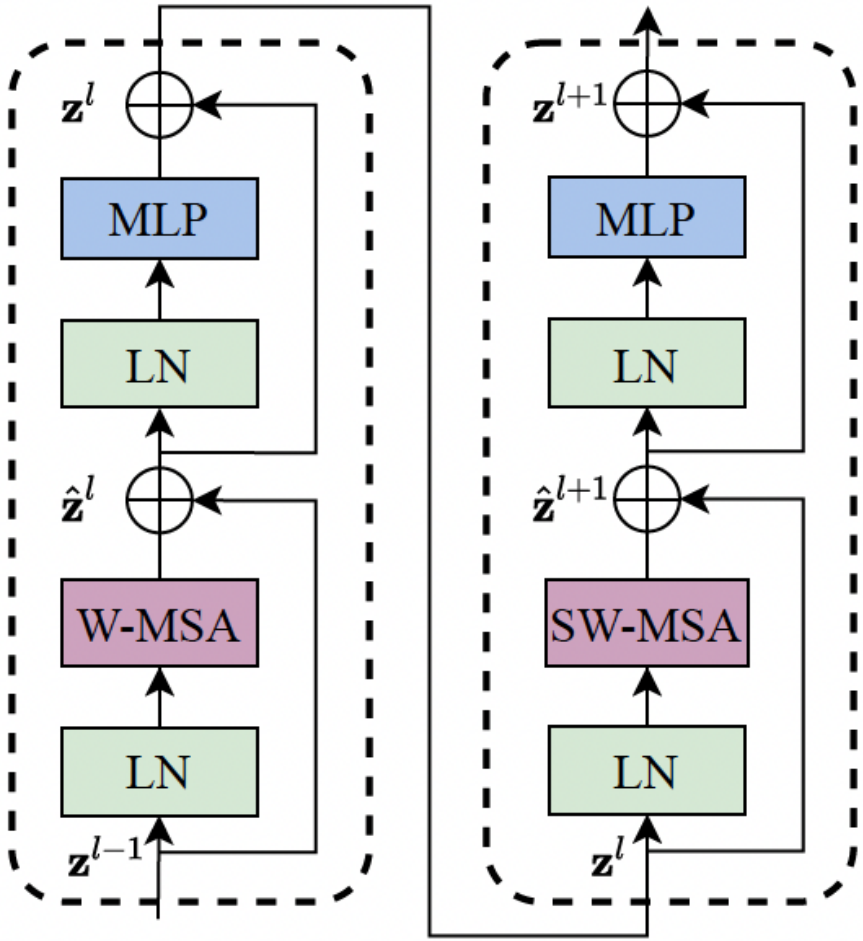
Shifting the Windows

- How to Tokens outside the Window ?
- Solution: Window shifts across Layers !!

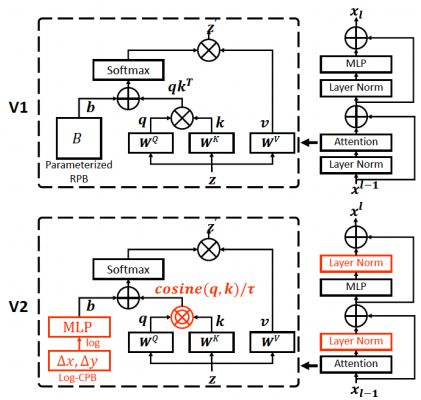


- Reduces Path Length across Tokens.

Swin Transformer Block



Modifications for Scalability



- Replace scaled dot product with Scaled Cosine
- Use MLPs on Relative Position Coordinates to generate positional encodings.

scOT: scalable Operator Transformer