Notion project page
Working group discord thread
Intro slides
Todo list

MedARC Brain-inspired Transformers

The intention for weekly meetings is to update everyone as to what is being worked on, get everyone on the same page regarding big picture goals, help anyone get unstuck on their problems, brainstorm next things to work on / new ideas for the project, etc. If you are new, the weekly meeting is a great place to introduce yourself and see how you can contribute to the project. If you cannot attend weeklies, you can still contribute to the project – you can watch our recordings of all meetings on the notion and you should still add updates to this Google doc. You can find tasks to work on by checking our todo list. Message Connor or post in the working group if you ever want to contribute and don't know how.

Sep 20

Connor

- Nothing new. Hoping to have some time to get back to this next week.

Alisa

- Ran a few visualization experiments and analysis here detailed here:
  Visualizations for topomoe_tiny_3s_patch16_128

Aug 23

Connor

- Setting up some training runs on full imagenet. Models will still be either small or base scale. But full resolution images, full dataset, longer training schedule so should hopefully see more interesting learned structure.
- Planning on training models from three families: vit, soft moe, and topo moe.

Aug 16

- Talked about approach for extracting activations and how we want to analyze. Alisa is working on this. Notes taken here.

Aug 9

Connor:

- Started paper draft overleaf with initial description of model architecture https://www.overleaf.com/read/hstkgswkdjmq#33bdab

Aug 2

Connor:

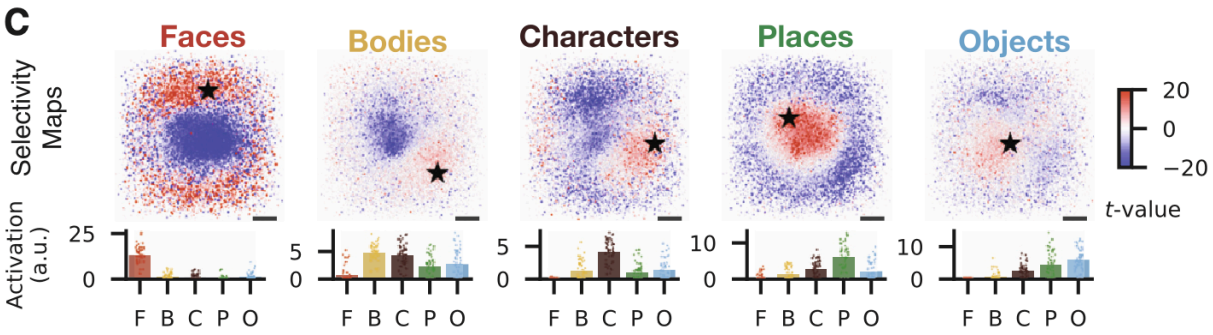
- Implemented Soft MoE: https://github.com/clane9/columnformers/blob/soft_moe/topomoe/models/softmoe.py
- Soft MoE block:
    1. standard self attention
    2. route tokens to slots (dispatch)
    3. MoE MLP (fixed slot assignment)
    4. route slots to tokens (combine)
- Topo MoE block:
    1. route tokens to tokens ("pool")
    2. map expert weights (query, mlp) to tokens
    3. MoE attention (expert query weights, learned token assignment)
    4. MoE MLP (learned token assignment)
- Soft MoE applies experts in slot space using a fixed expert-slot assignment; Topo MoE applies experts in token space using a learned expert-token assignment.

- Thinking about how to analyze expert selectivity. First some review of other works approaches:
- TDANN:

- Used stimuli for faces, bodies, characters, places, objects from fLoc (Nb, grayscale on phase scrambled background).

- Measured activation of each unit to each image
- Computed selectivity t score (on=target categories, off=other categories)

$$t = \frac{\mu_{on} - \mu_{off}}{\sqrt{\frac{\sigma_{on}^2}{N_{on}} + \frac{\sigma_{off}^2}{N_{off}}}},$$



- I don't get why they see selectivity for faces/places? The model was trained on ImageNet, which does not have many faces or places?
- I'm a little uneasy about the use of fLoc stimuli, since they are out of distribution wrt to the training data.

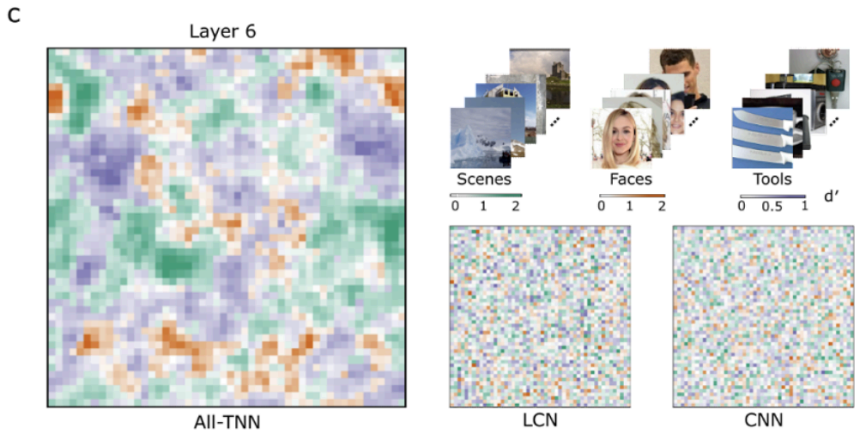
- All-TNN
  - Used stimuli for faces, places, tools from VGG-Face/ecoset:

    To determine high-level object selectivity in the last layer of the models, we selected 500 images for each superclass used to test selectivity (faces, places and tools). The places and tools images are selected from the 10 most common classes for the respective superclass found in ecoset validation set. The faces are taken from the VGG-Face dataset75. Places: ‘House’, ‘City’, ‘Kitchen’, ‘Mountain’, ‘Road’, ‘River’, ‘Jail’, ‘Castle’, ‘Lake’, ‘Iceberg’ Tools: ‘Phone’, ‘Gun’,

‘Book’, ‘Table’, ‘Clock’, ‘Camera’, ‘Cup’, ‘Key’, ‘Computer’, ‘Knife’ Faces: 10 identities taken from the VGG-Face dataset75.

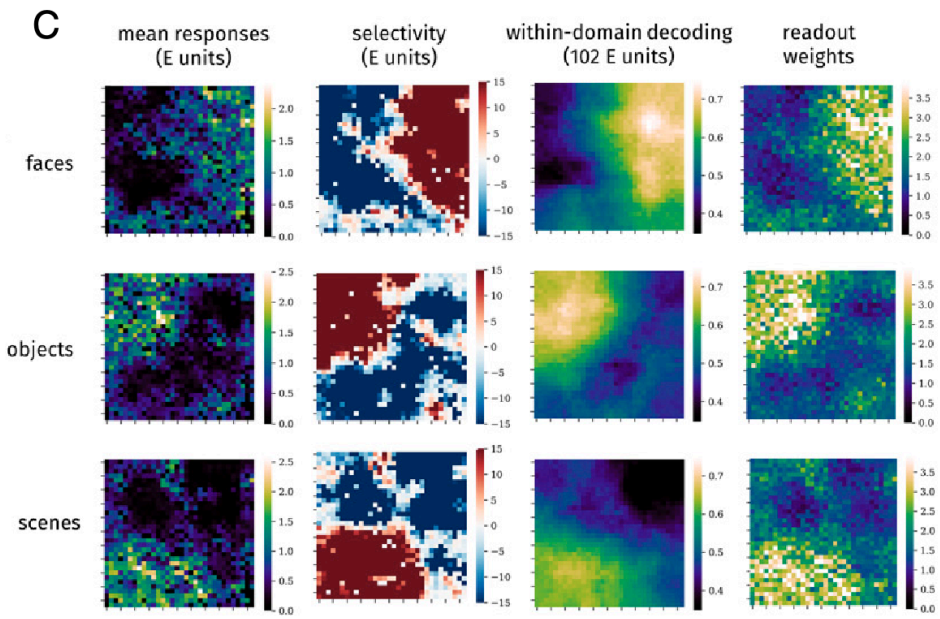
- Similar selectivity metric (Nb typo):

$$d' = \frac{|\mu_{in} \mu_{out}|}{mean(\sigma_{in}, \sigma_{out})}$$



- ITN:
  - Used stimuli for faces, objects, scenes from VGG-Face, ImageNet, Places365
  - Similar t-test based selectivity analysis:

    each unit is analyzed separately for its mean response to each stimulus domain (objects, faces, scenes), using untrained validation images from the same categories used in training. In addition to computing the mean response to each domain, we compute selectivity, a ubiquitous metric used in neuroscience, to analyze how responsive a unit is to one domain compared to all others. We compare the responses of each domain versus the others using a two-tailed t test



- I think we could start with an exploratory approach
    - **Figure 1**: large grid of (num_patches * 3 x num_patches * 3) validation images. Each 3x3 block corresponds to the top 9 most strongly activating images for the corresponding patch. (Can also look at least strongly activating for comparison.) The goal is to explore what images drive activation in each region of the topographic map. (Note that map region correlates with expert composition.)

      Inspiration from here:

- **Figure 2:** grid of (num_patches x num_patches) plots. Each plot is a bar plot showing the mean activations for each of the 100 imagenet-100 categories. The goal is to see if there are any patterns in the category selectivity profiles for the different patches.

  This figure might be hard to interpret with 100 categories in arbitrary order. To reduce the density, perhaps we could do some combo of sorting according e.g. dendrogram cluster ordering and/or looking at meta-category clusters (dogs, small objects, etc). To cluster, can use CLIP features or sth.

- We might then try a hypothesis driven approach with the expected categories
  - **Figure 3:** same as Figure 2 but using fLoc images/categories

    I am concerned that the images are OOD, esp since we have trained only on ImageNet-100 rather than full ImageNet. But we can still try, for comparison to TDANN
  - **Figure 4:** same as Figure 2 but using images from ITN domains.

    Since there are only 3 domains, can also plot as 3 maps similar to the figure above.

    I think this suffers less from the OOD issue as Figure 3. But faces/scenes are still quite OOD.

- Questions:
    - How to deal with the embed dim in each patch when computing patch responses?
        - I think just reduce over the embed dim with e.g. l1 norm
        - Can also think about mining for selective units within each patch somehow?
    - Where should features be extracted from?
        - I think immediately after the nonlinearity in each MLP. This shows which features the layer is specifically computing, in expanded dimension, as opposed to the cumulative residual features, in lower dimension, after each block.
    - Is it ok to do the analyses at the patch level given that patches are not independent?
        - Patches are linked via the expert mapping. Only the experts are independent. But the patches are a convenient representation to do the analysis on. I think it's fine. It should be no worse than TDANN, where there is also implicit weight sharing between separate regions of the map.
        - we can do both, patch level and expert level

- different analyses:
    - patch level selectivity
    - expert maps/ individual expert selectivity
    - decoding
    - lesioning
- important, topomoe does not necessarily need to be a transformer, could also have a convnet version

July 26

Helpful discussion with Nick to try to clarify parts of the TopoMoE architecture.

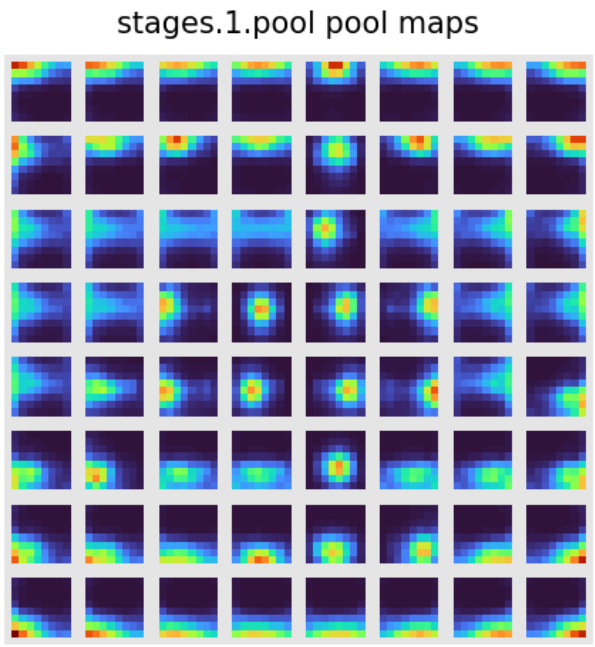
July 19

TopoMoE overview presentation. Recording here.

July 12

Connor:

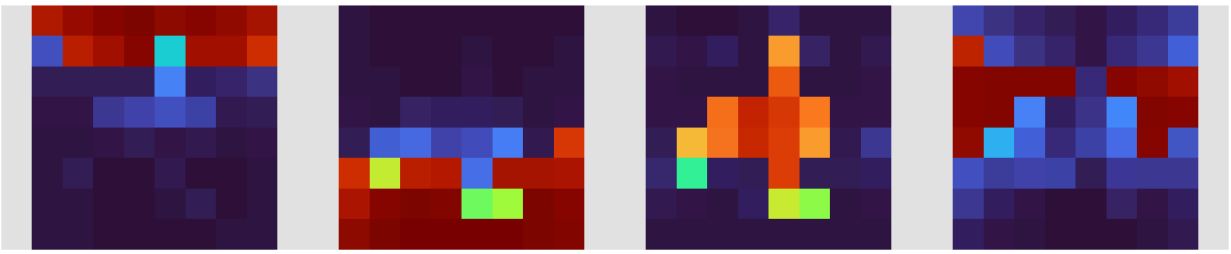
- Did some training runs with new topo MoE.
- Architecture design:
    - Multiple stages
    - Each stage contains one or more ViT blocks
    - Each stage has a topographic representation map, the grid of tokens
    - To route between stages, we use position based associative pooling (similar to soft-MoE)
    - Within a stage, we use topographic mixture of experts. Each expert has a learned position embedding. Experts are assigned to positions in the map by association.
    - Same number of experts and expert layout for all blocks in a stage
    - Each expert has independent query and mlp weights. key, value, norm weights are shared across experts.
    - First block of each stage uses cross attention to pull in high-res information from the previous stage output, using the pooled output for computing queries.
    - Cross-entropy wiring cost
- Two stage tiny ViT:
    - Pooling maps from stage 0 -> stage 1



- Cross-attention maps for stage 0 -> stage 1

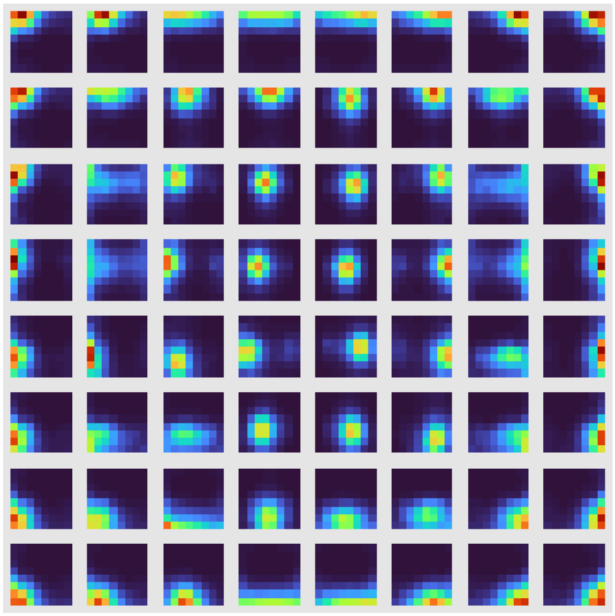
- Expert maps for stage 1 (stage 0 has one expert)

stages.1.maps expert maps

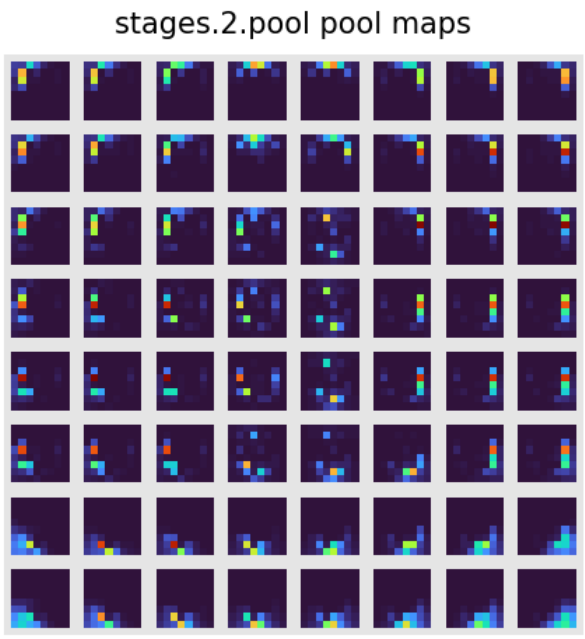


- Three stage tiny ViT:
    - Pooling maps stages 0 -> stages 1

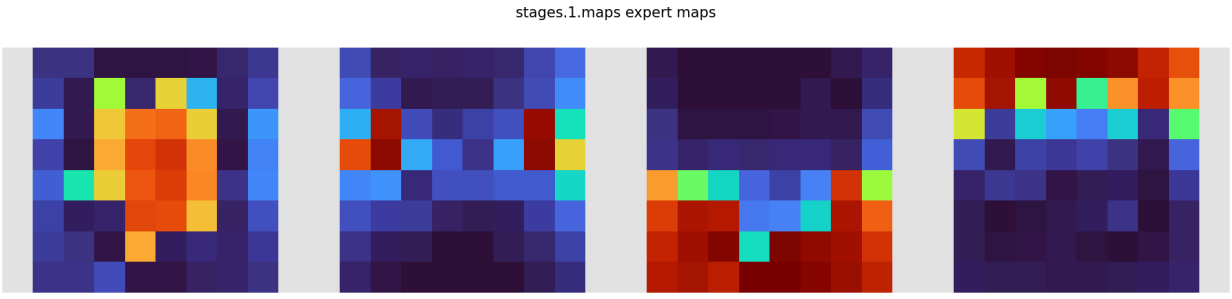
stages.1.pool pool maps



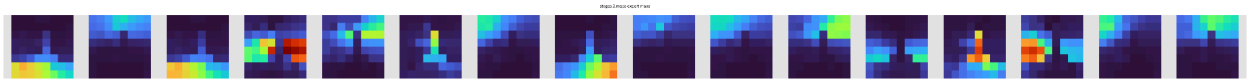
    - Pooling maps stages 1 -> stages 2



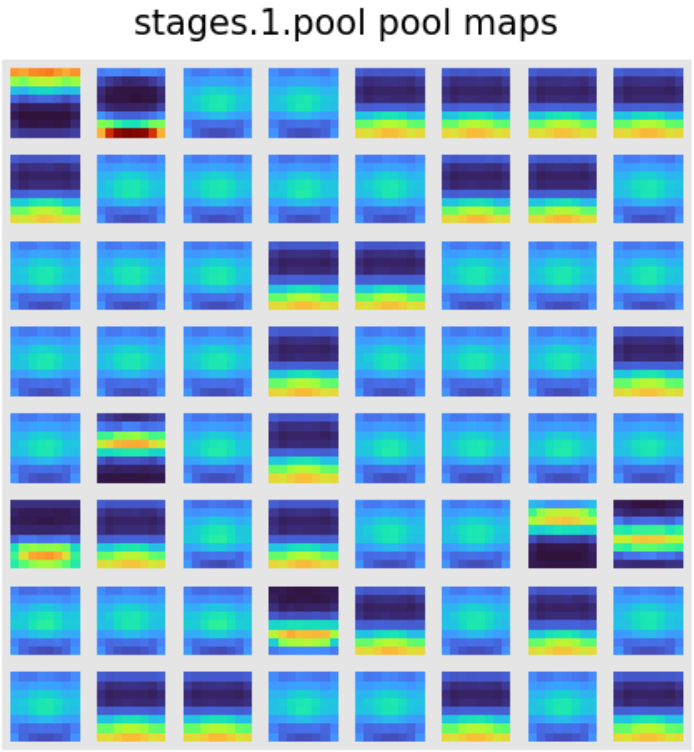
- Expert maps stage 1



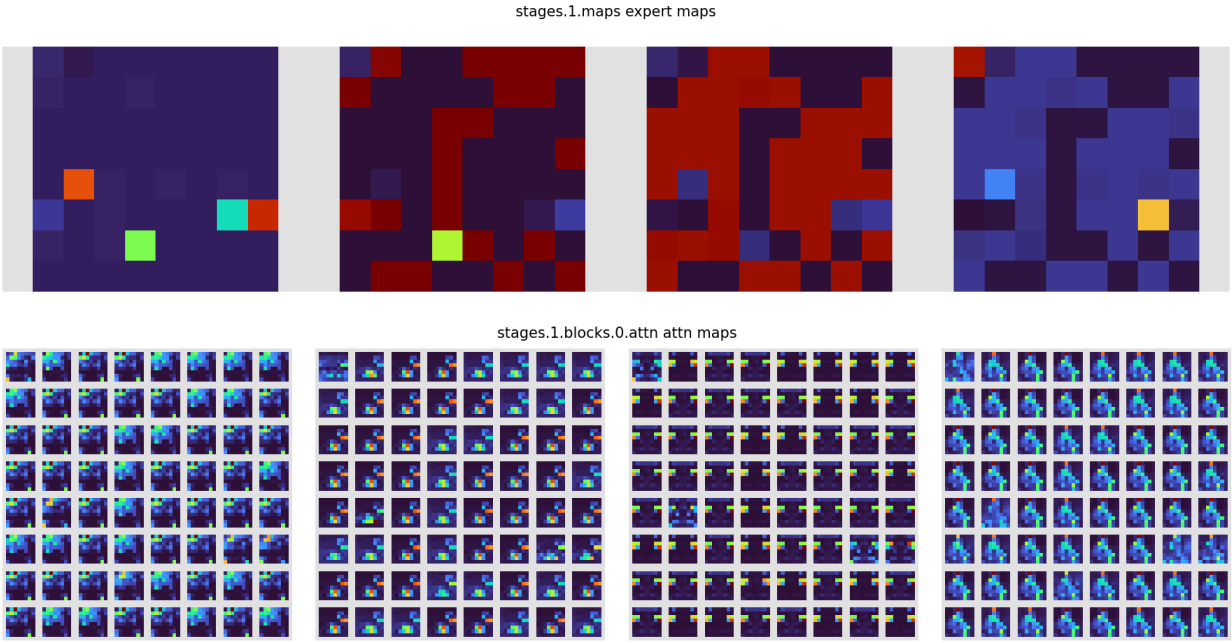
- Expert maps stage 2



- Two stage **no wiring cost**
  - pool maps stage 0 -> stage 1


stages.1.pool pool maps

- expert maps


stages.1.maps expert maps
stages.1.blocks.0.attn attn maps

July 5

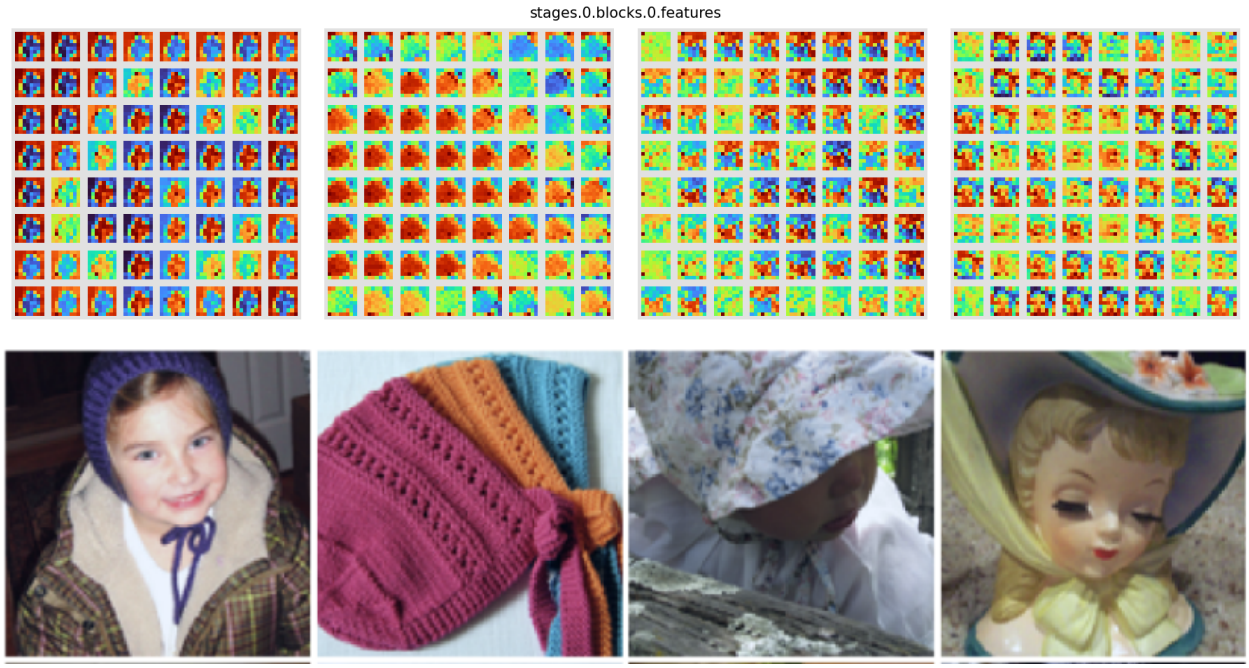
Connor:

- Back from a few weeks off for conference and travel
- Added tests and training script for the topo MoE and quadformer model. You can find the code in the topomoe folder in the "branching" branch. I'm ready to launch some training runs.
- The only major change between the columnformers and topomoe training scripts is I've switched to using timm for dataset creation/data loading. This will make it easy to run experiments on the full imagenet, which I want to do pretty soon.
- Implemented a cross-entropy based wiring cost. I feel like it could be nice since it will have the same gradient/scale as the main classification loss. It is basically the same as the l1 wiring cost, just with a different distance-based weighting computed as

```
weight = -0.5 * dist**2 / sigma**2
weight = -F.log_softmax(weight, dim=-1)
```

- Alisa and I have been working on the blocksparse locally connected layer. See the open draft PR here. Currently the test is failing because backward through the blocksparse matmul doesn't work.

  RuntimeError: addmm: computation on CUDA is not implemented for Strided + Strided @ SparseBsr
- I'd like to submit an abstract for this work to the NAISys meeting. The meeting seems like a perfect fit with a lot of great invited speakers. The deadline is July 12 but it's just a single page text abstract.
- Feature correlation maps for untrained (randomly initialized) ViT on ImageNet-100 images.
    - Nb these maps look a lot like our earlier topographic expert maps.
    - This gives some evidence for how image statistics can drive correlated responses which in turn can drive topographic specialization without regularization

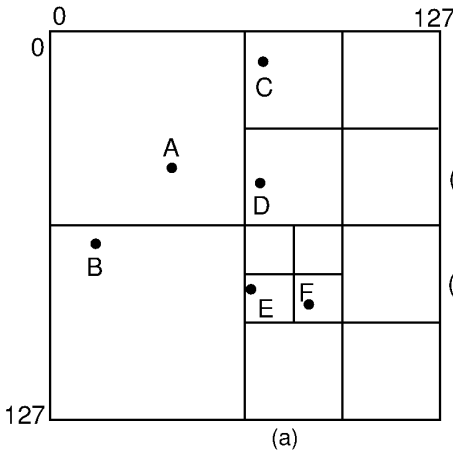


May 31

Connor:

- I implemented the topo MoE model with fixed expert assignment maps.

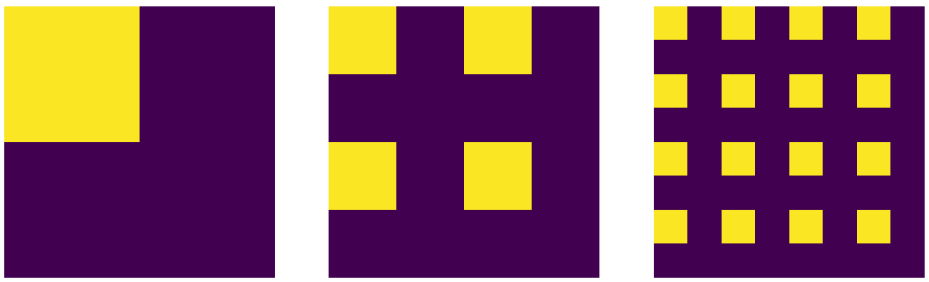
- The model consists of a sequence of stages. At the start of each stage, the number of experts is increased by 4x (like a quad tree).



(a)

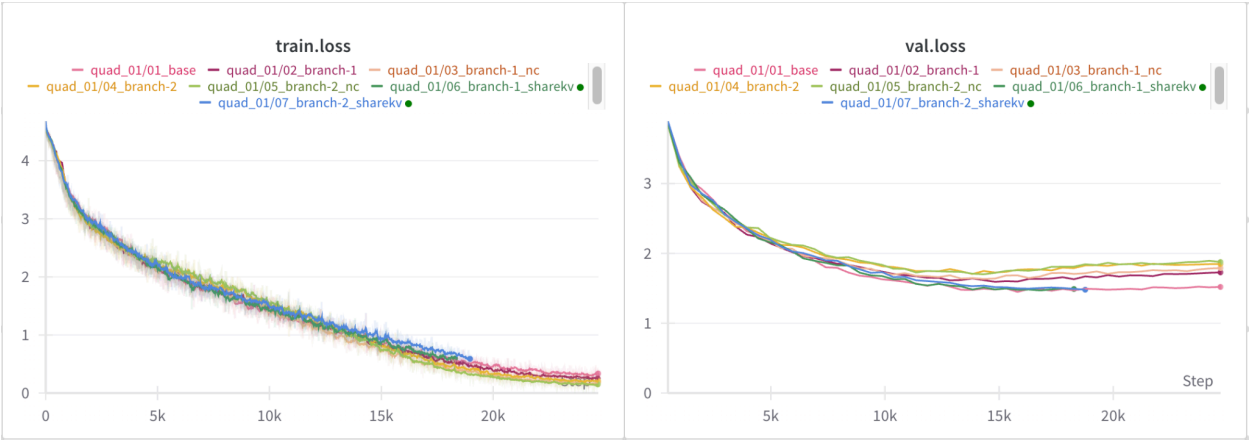
Each expert has independent mlp weights, query weights, and possibly independent key/value/proj weights as well. The independent mlp weights compute specialized features, the independent queries learn specialized "attention" selection.

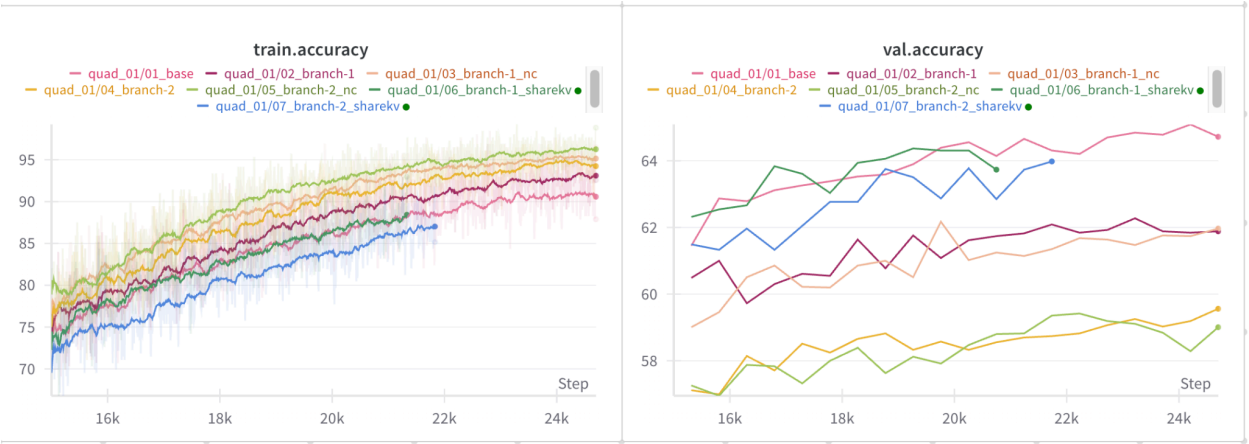
- To help with routing to these somewhat awkwardly placed experts, I implemented a "pooling + branching" strategy. basically, the input is 2x2 average pooled and then tiled as a 2x2 grid. This way each expert sees a (downsampled) copy of the input (much like classic convnets with pooling in between stages). For later stages, this is done recursively in every expert block.



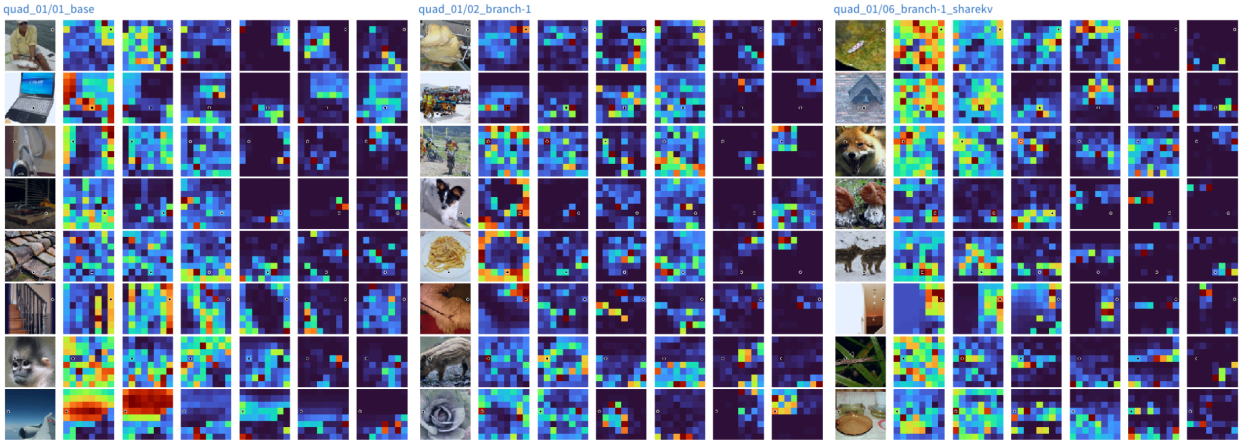
- Training runs:
  - **01_base**: classic ViT for reference [1, 1, 1, 1, 1, 1]
  - **02_branch-1**: two stages. num experts: [1, 1, 1, 4, 4, 4]. one branch point. conserved params (i.e. fixed total params)
  - **03_branch-1_nc**: two stages. num experts: [1, 1, 1, 4, 4, 4]. one branch point. scaled params (i.e. fixed per expert params; same flops as 01_base).
  - **04_branch-2**: 3 stages. num experts: [1, 1, 4, 4, 16, 16]. two branch points. conserved params.
  - **05_branch-2_nc**: 3 stages. num experts: [1, 1, 4, 4, 16, 16]. two branch points. scaled params.
  - **06_branch-1_sharekv**: same as 02 but key/value/proj weights shared.
  - **07_branch-2_sharekv**: same as 04 but key/value/proj weights shared.

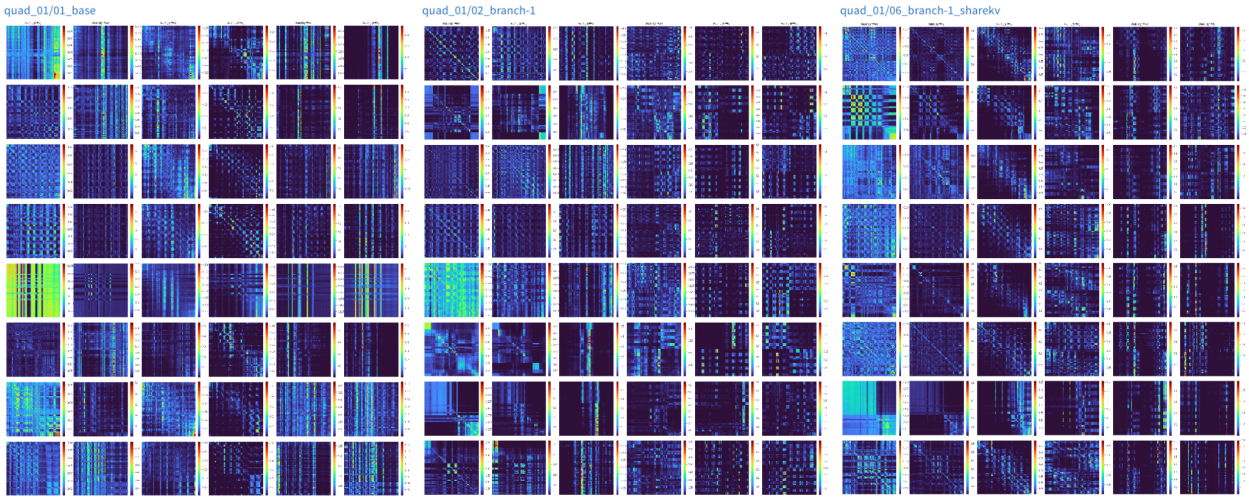
- The models train fine, similar to earlier topo MoE (with learned maps). The larger networks overfit a bit more (branch-2_nc). Perhaps this shows the model is trying to use the extra capacity?



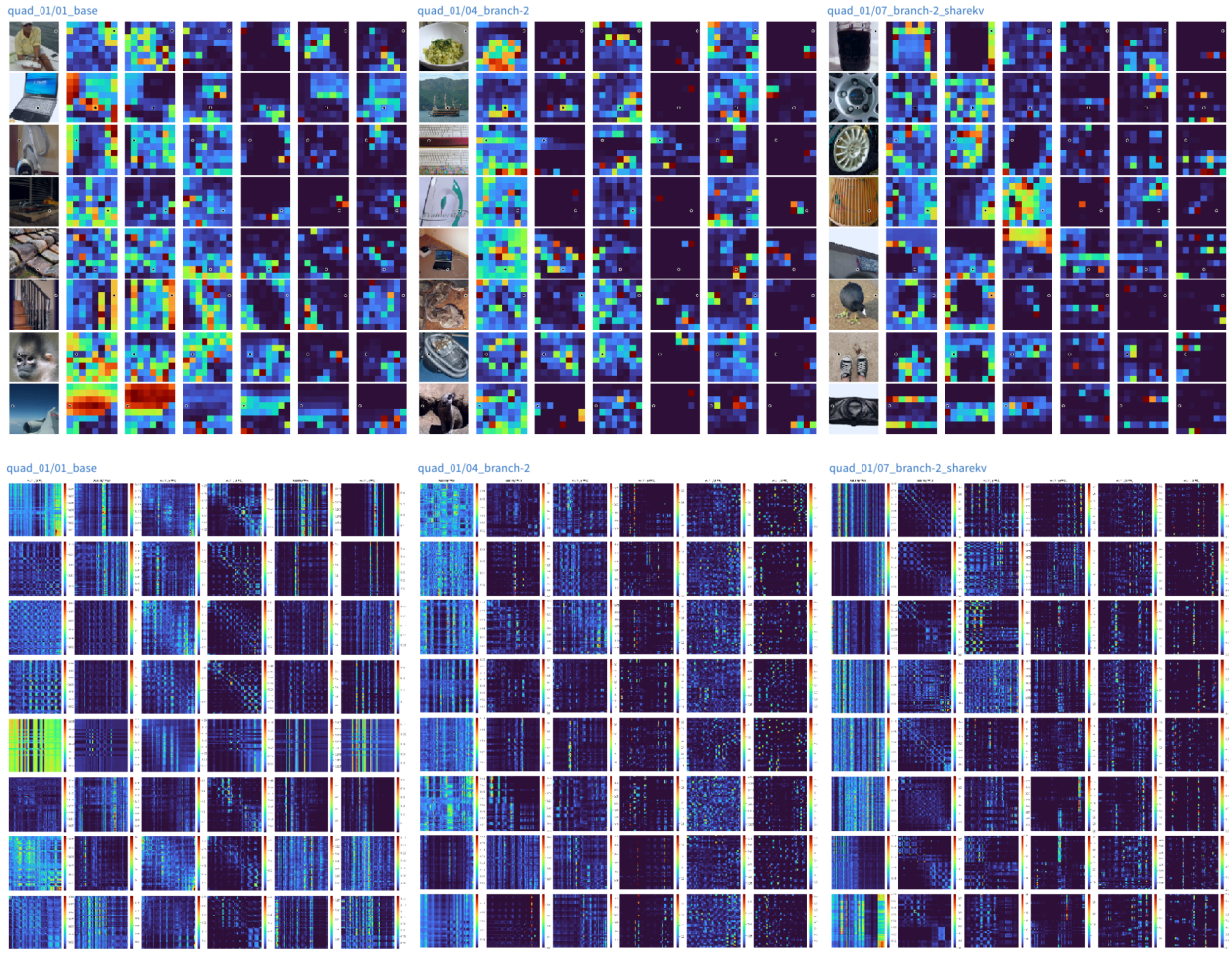


- branch-1 attention maps and matrices (base, branch-1, branch-1_sharekv)





- branch-2 attention maps and matrices (base, branch-2, branch-2_sharekv)



- hard to see clear differences. but perhaps the middle and right columns are sparser than the base in the later layers.
- with decoupled key/value/proj weights, you see blockier attention maps.
- **todo**:

- ablate the branch/pooling; see what happens if experts are fed quadrants of the input. (shouldn't work)
- analyze selectivity of each expert
- scale up. imagenet-100 128px is too tiny.

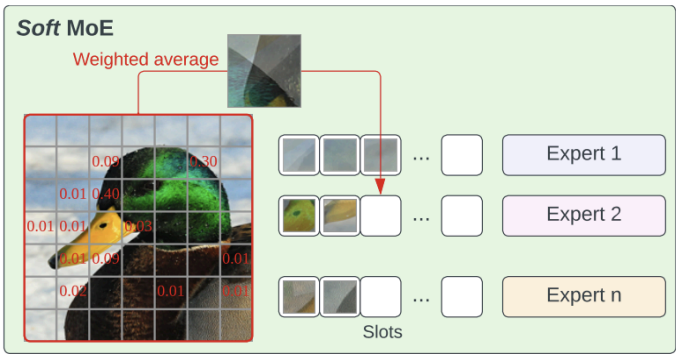
May 24

Connor:

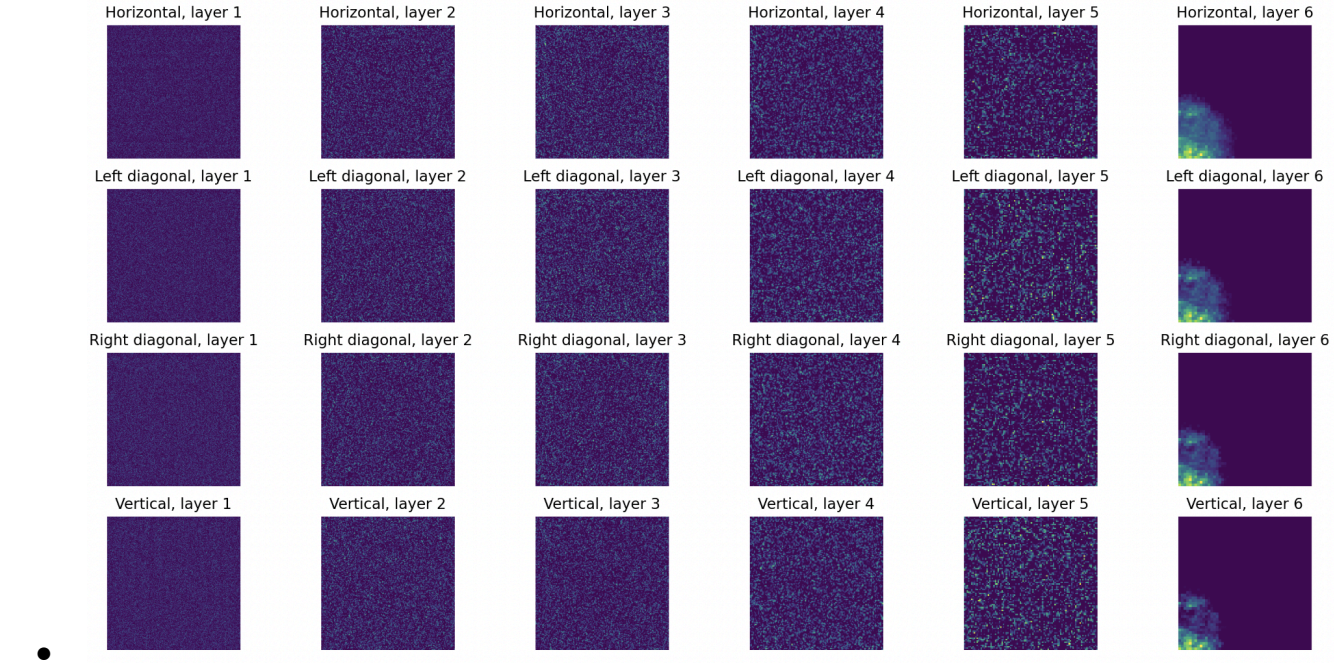
- I put together some slides summarizing our recent results.
- Something I want to try next to better understand the topo MoE results is a bit of a simpler experiment using fixed expert assignment maps.

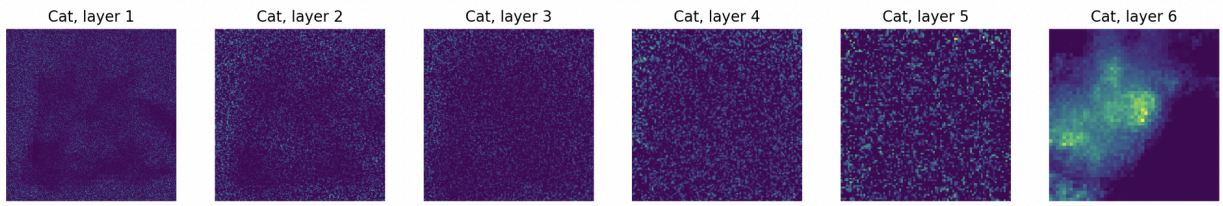
I think it should be easier to analyze the expert specialization and figure out how to get the position based expert routing to work in this simpler setting.

I also like that it creates a bridge between our topo MoE and soft MoE.



In soft MoE, each expert has a fixed number of slots. And slots are filled with weighted combinations of patches via an attention-like mechanism. What I'm proposing amounts to letting slots = patch positions, and doing away with the separate Soft MoE routing module. Tl;dr "attention is the only router you need".

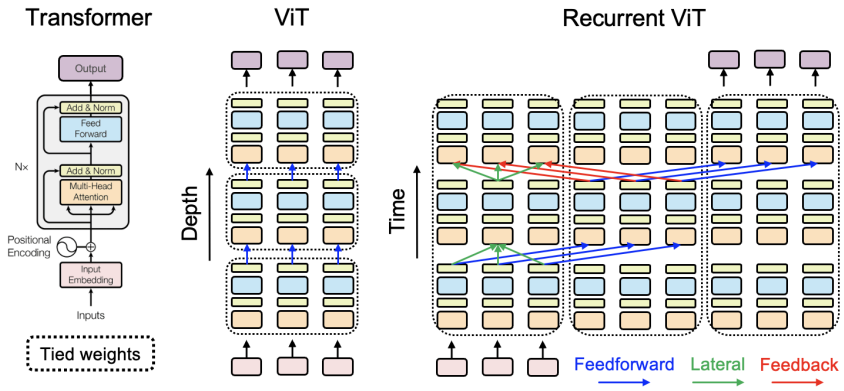
- @alisa: I will try to look into the blocksparse errors on my cluster over the weekend and see if I get the same thing
- Alisa - trained the naive all-tnns implementation for 16 epochs
- 

- 

May 17 2024

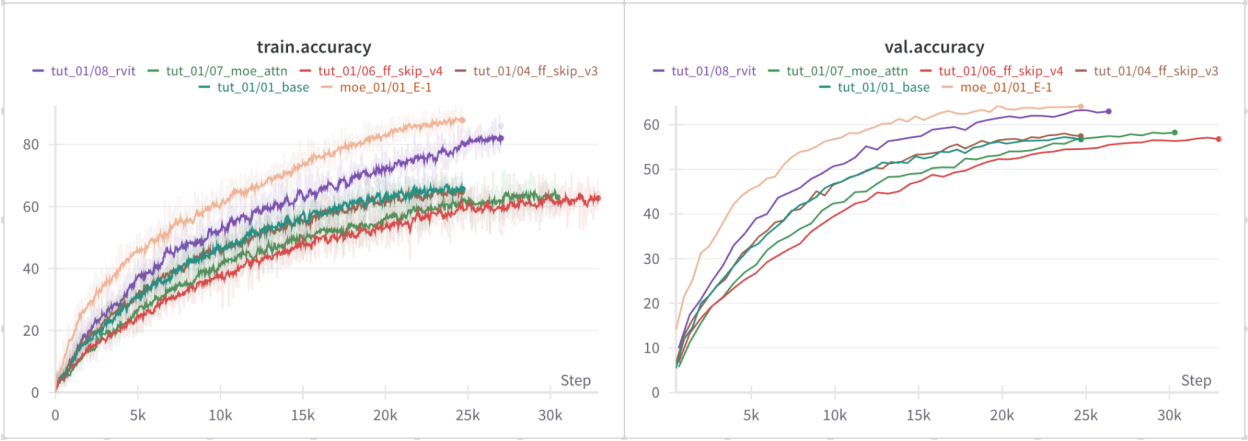
Connor:

- I made a few changes to how recurrence is implemented to try to address the redundant attention issue we've been seeing
    - Applying direct input from the previous layer into attention (commit). This means queries are computed based on the output of the previous layer, rather than the output of current layer itself at the previous timestep. I feel this is more consistent with classic ViT where self-attention is applied to the output of the previous layer. But in our case we can still cross-attend to other layer outputs, unlike classic ViT.
    - Add option for a mixture of experts Attention (commit). Previously we only allowed MoE MLP. This is fine in the feedforward case. But in the recurrent case, it has the possibly unwanted effect of sharing attention parameters across all tokens and timesteps.
    - Add a pure recurrent ViT (no topography) (commit).

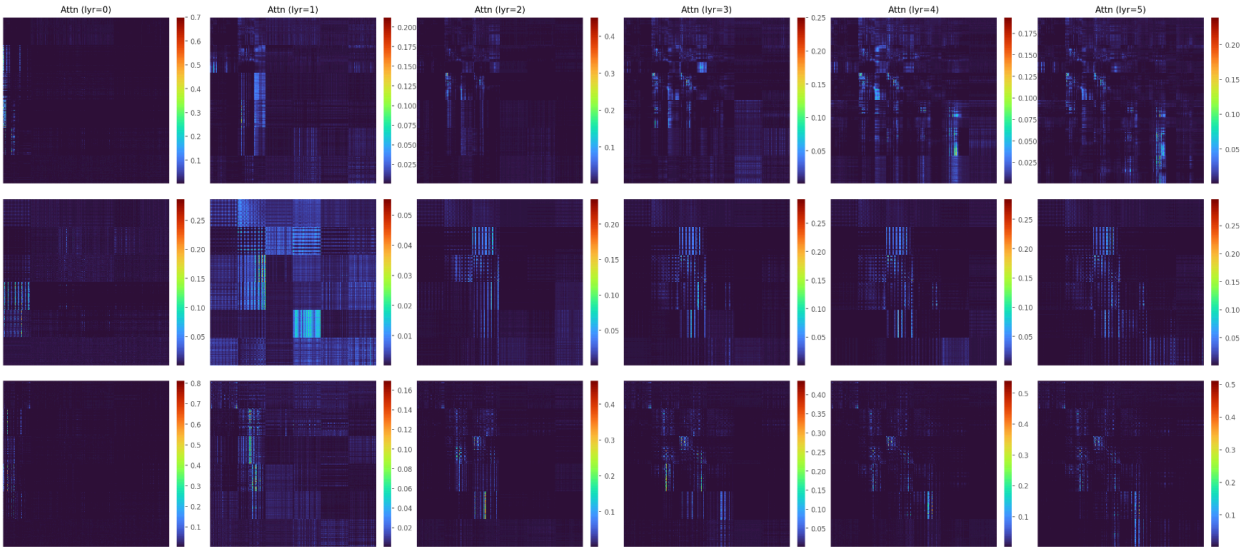


This just directly extends the classic ViT by computing all layers at each time step and allowing layers to cross-attend to each other. (Nb afaik no one has done this and it's pretty interesting in its own right imo.)

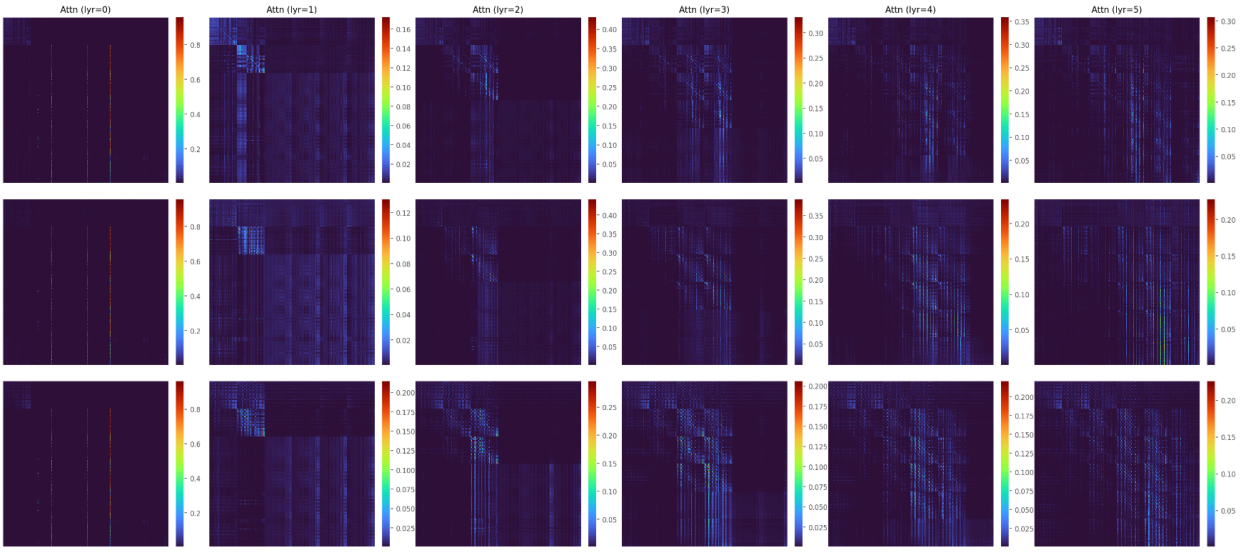
- Accuracy curves:
  - Adding the direct connections into attention doesn't clearly help (06_ff_skip_v4)
  - Adding the MoE attention does improve performance a tiny bit vs previous (07_moe_attn). (Nb, these runs needed a lower batch size and lr to fit in memory.)
  - Recurrent ViT works pretty nice (08_rvit). The training recipe is def not optimized, but it reaches about the same val accuracy as the classic ViT (moe_01/01_E-1).

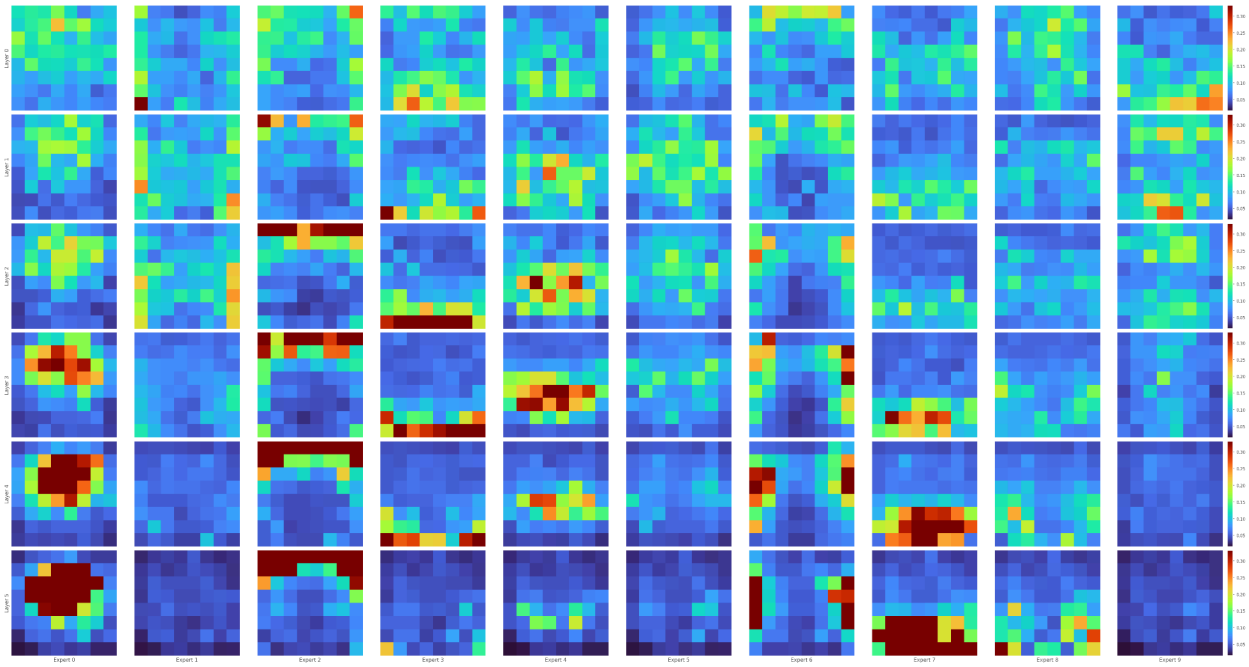


- Attention patterns for recurrent ViT. Looks kinda interesting! Definitely some cross-layer (off-diagonal) communication happening.



- By contrast, the recurrent topo MoE model still has more redundant looking attention. So just adding the MoE attention did not really fix things.



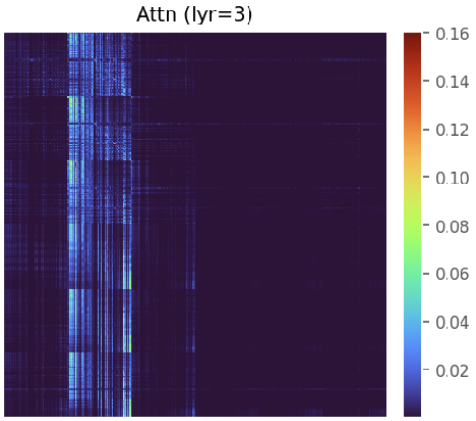


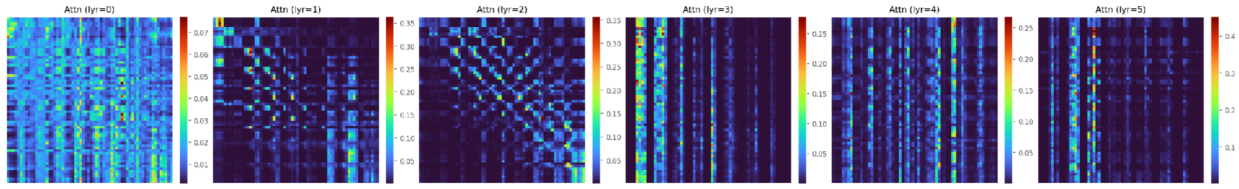
- literature search for papers that already did rViT
- maybe focus on one publishable unit at a time
-

May 10 2024

Connor:

- The main thing bugging me about the current TUT results is the redundancy in the attention weights. Each layer has basically the same attention weights at every time step. Whereas in the classic ViT, you see a lot of interesting variation in the attention weights across layers.
- I think the reason is probably that we are sharing the parameters of the attention module across all layers.
- I want to do a few baseline experiments exploring this with ViT:
    - Recurrent ViT with tied Attention and MLP (basically UT)
    - Recurrent ViT with tied Attention and untied MLP
    - Recurrent ViT with untied Attention and untied MLP

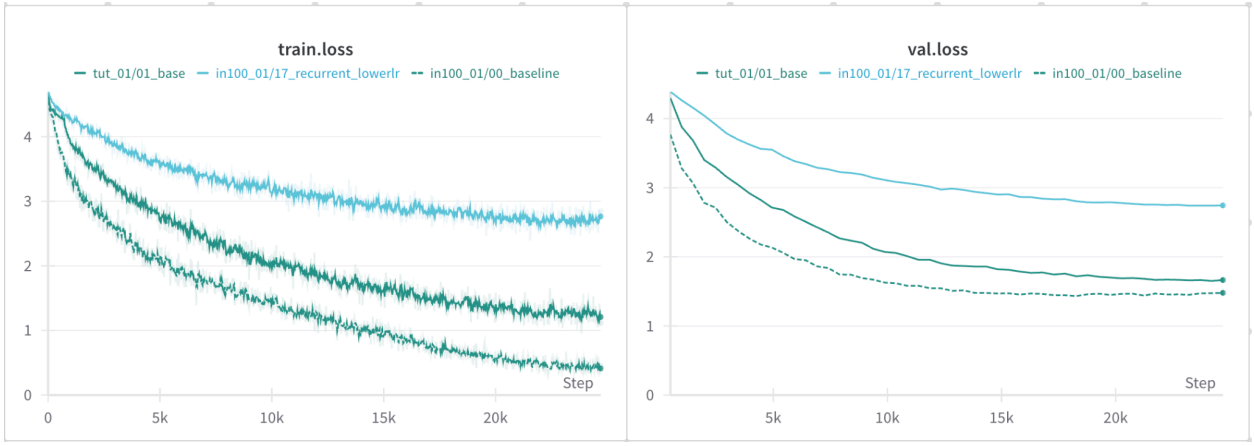




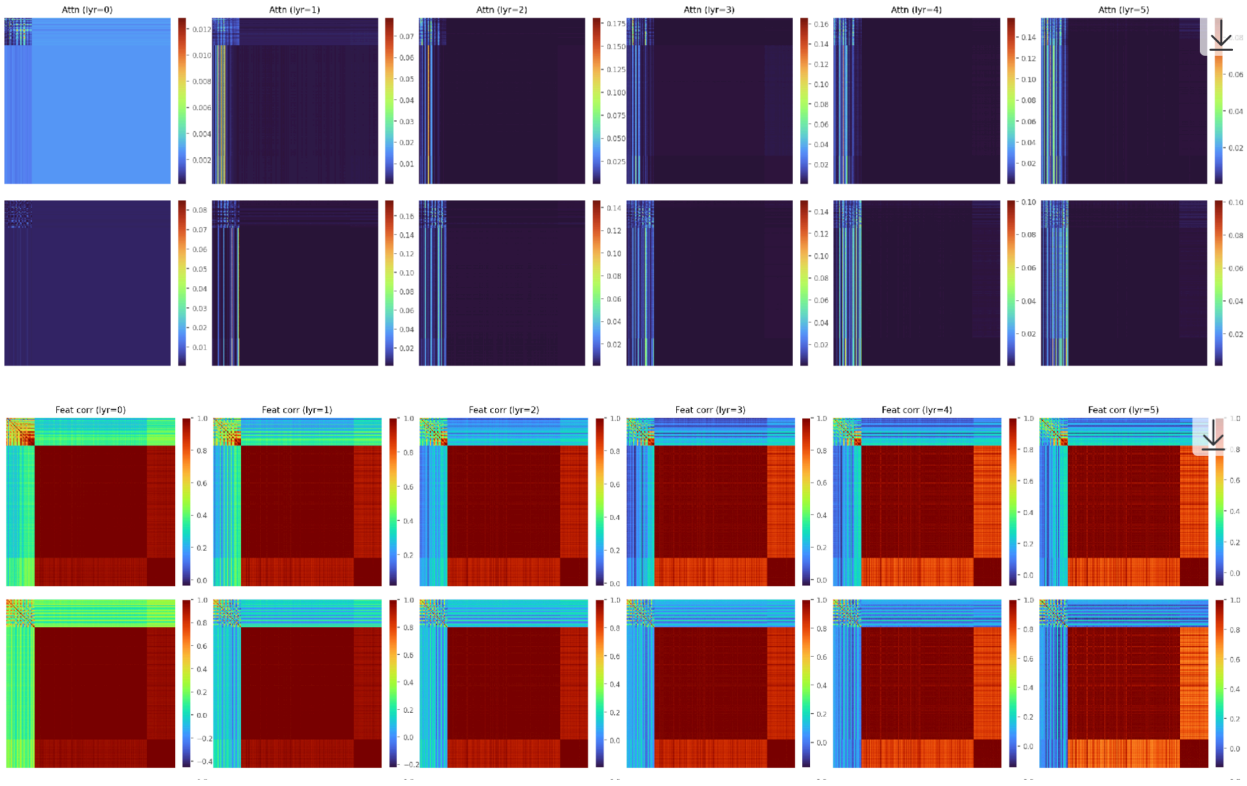
May 3 2024

Connor:

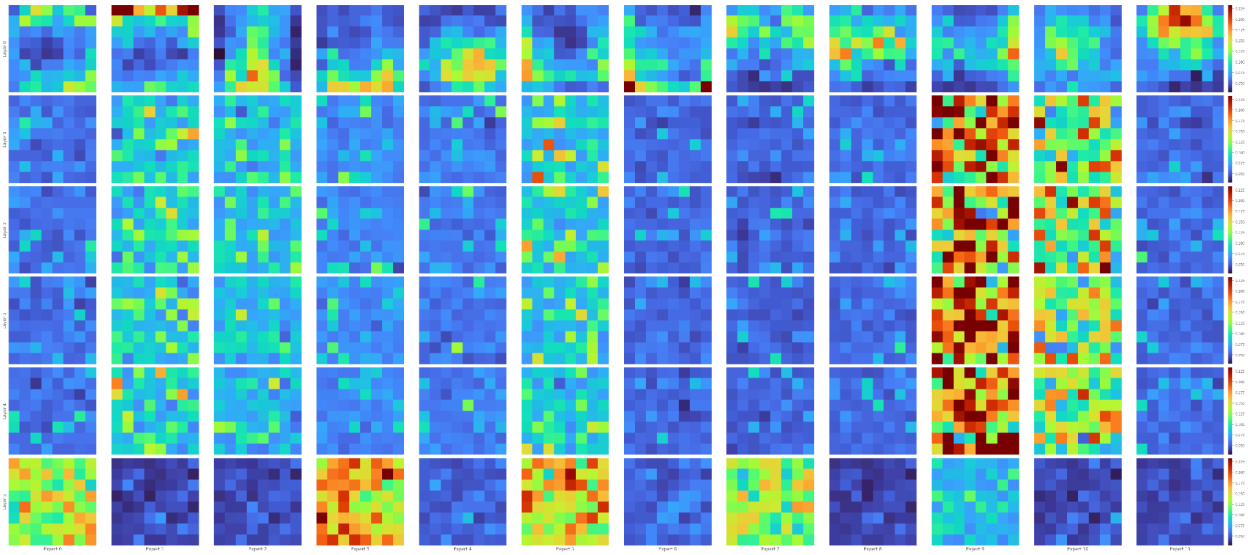
- Did some initial experiments combining recurrence and topographic MoE. I'm calling the architecture a Topographic Univeral Transformer (TUT), following the UT and SUT.
- See a draft PR here with some more discussion
- Initial run performs much better than our previous attempt at a recurrent architecture:
    - **TODO**: a pure UT baseline that adds recurrence to ViT and nothing else. This would actually be pretty interesting to see if feedback connections naturally emerge and if it improves performance. I don't think anyone's done this (?)



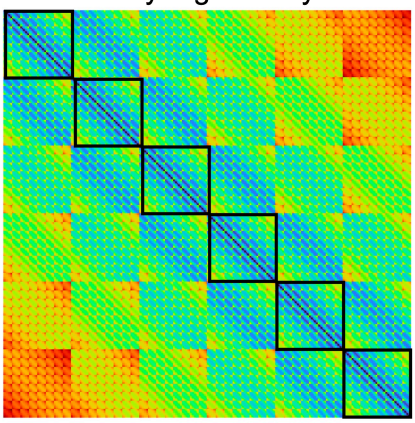
- The attention matrices look degenerate though. Only the first "layer" is really active in the attention weights. Similarly with the feature correlations
    - **Q:** how can performance be ok with only the first layer 64 tokens being used? Are 64 tokens enough space?



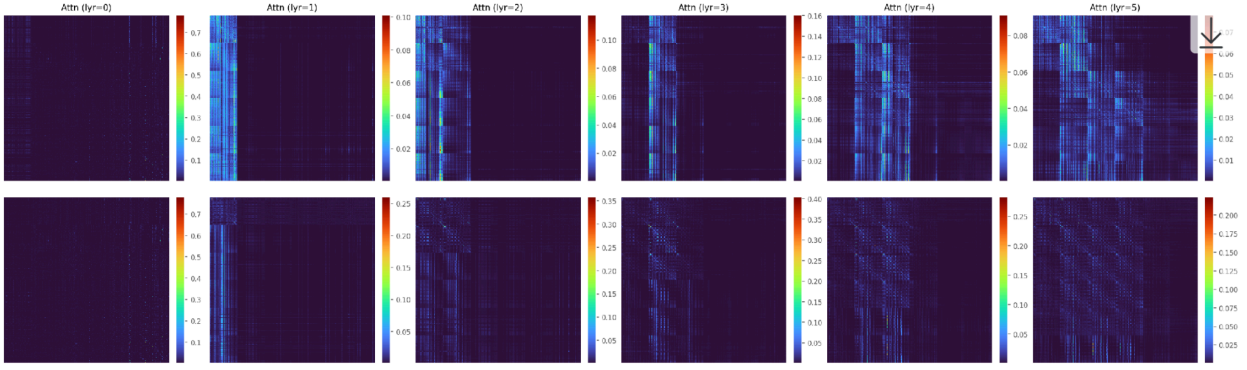
- The coefficient maps for the first layer are very interesting, but for later layers they are degenerate

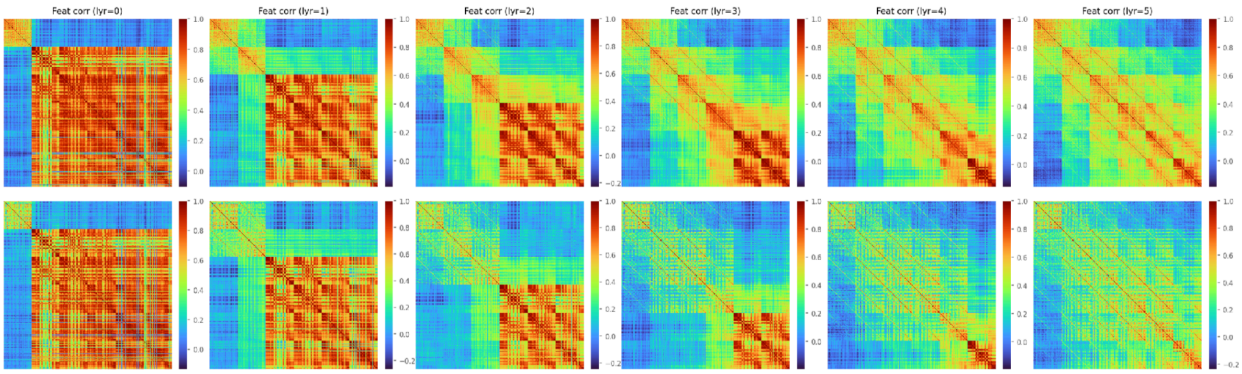


6 layer geometry

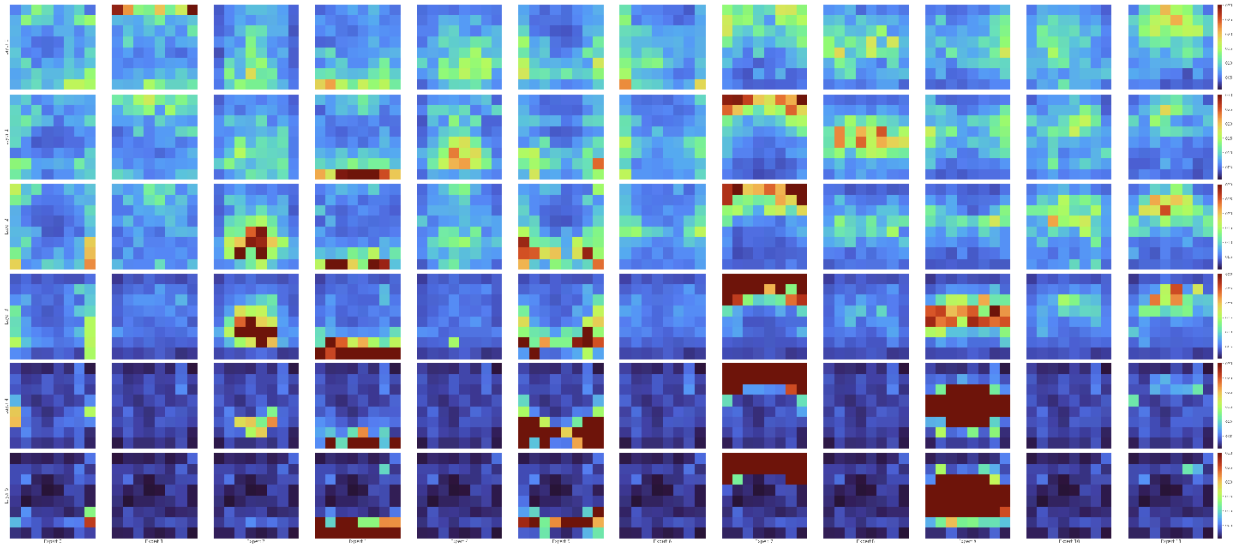


- I added direct skip connections between layers, which made things look better. Note the communication both above and below the block diagonal!

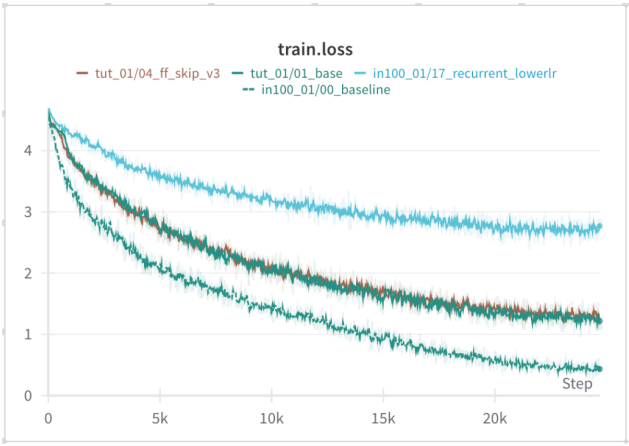


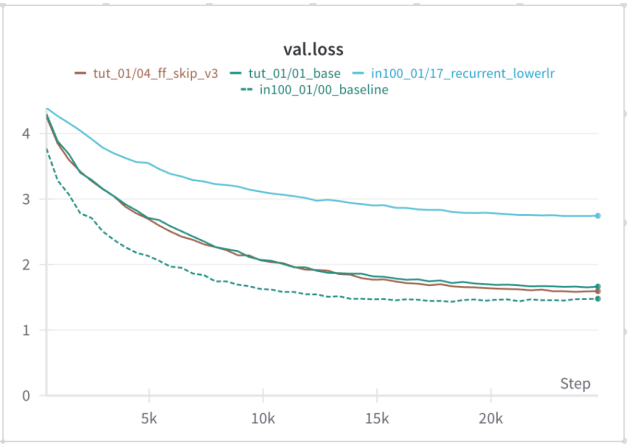


- But still it feels there is something not perfect about it. Maybe too much redundancy in the attention values between blocks.
- The expert coefficient maps (depth x experts) are quite interesting! spatial localization, consistent receptive field across depth, some depth specialization.



- But note that the direct skip connection did not really improve performance. Basically the extra tokens are not really being used.
    - **Q:** why aren't the extra tokens useful at all? Shouldn't the model be able to make use of the extra representational space?
    - **Q:** You see some experts localized to later layers (eg 3rd from the right, last 2-3 layers). But "activating" these layers with direct skip connections doesn't really seem to contribute to performance. What's going on? Does it just not matter whether experts are distributed across 64 vs 384 tokens?

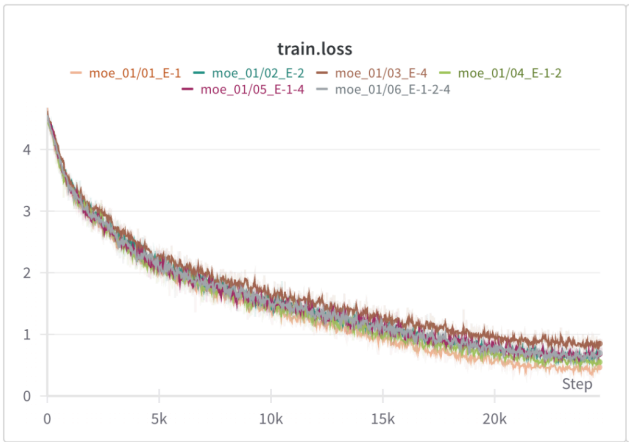


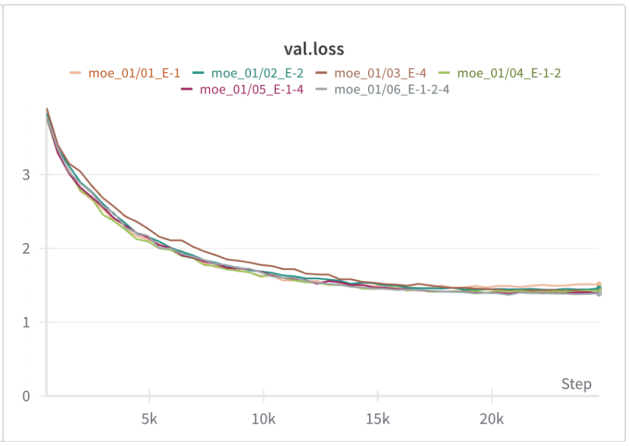


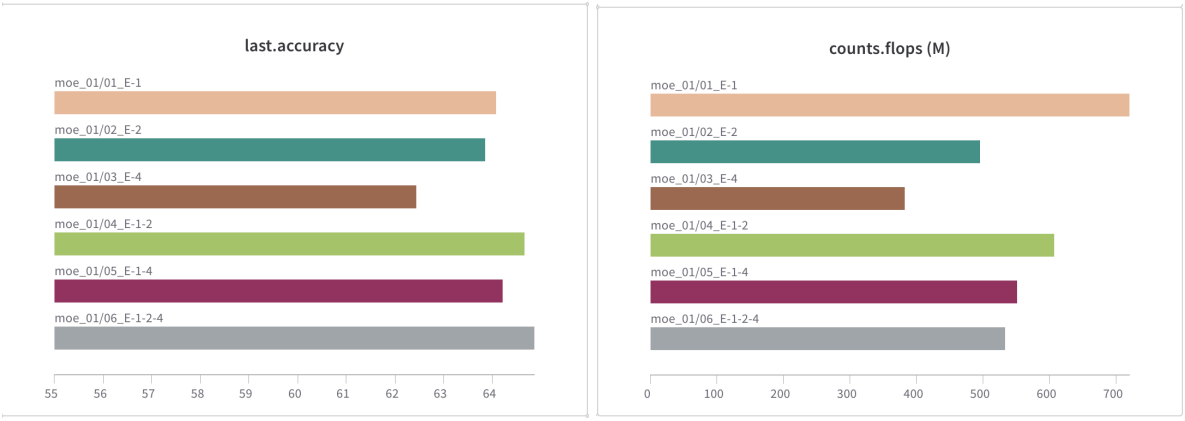
Apr 26 2024

Connor:

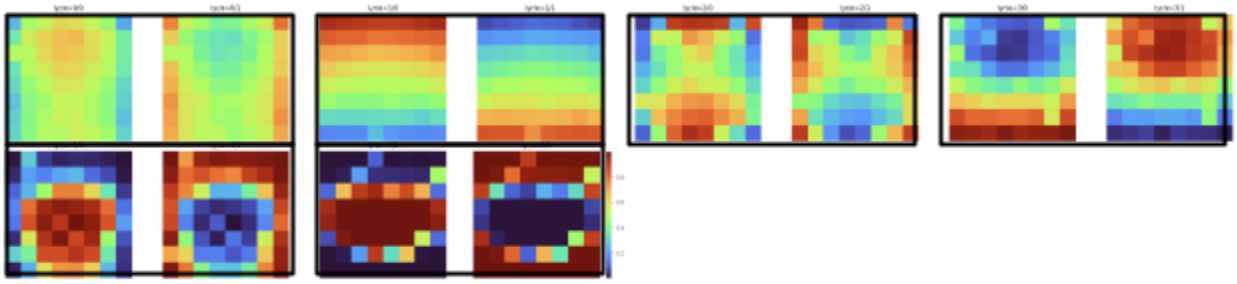
- Finished implementation of "topographic mixture of experts" (proposed new name, subspace mixture of experts was non specific)
- Ran a sweep of training runs on ImageNet-100 (vit tiny 6 layer, 50 epochs) with varying expert configs:
    - 1 expert (i.e. ViT baseline)
    - 2 experts per block
    - 4 experts
    - 1-2 experts: [1, 1, 1, 2, 2, 2]
    - 1-4 experts: [1, 1, 1, 4, 4, 4]
    - 1-2-4 experts: [1, 1, 2, 2, 4, 4]
- Crucially, equal number of params in every case.
- Wandb report here.



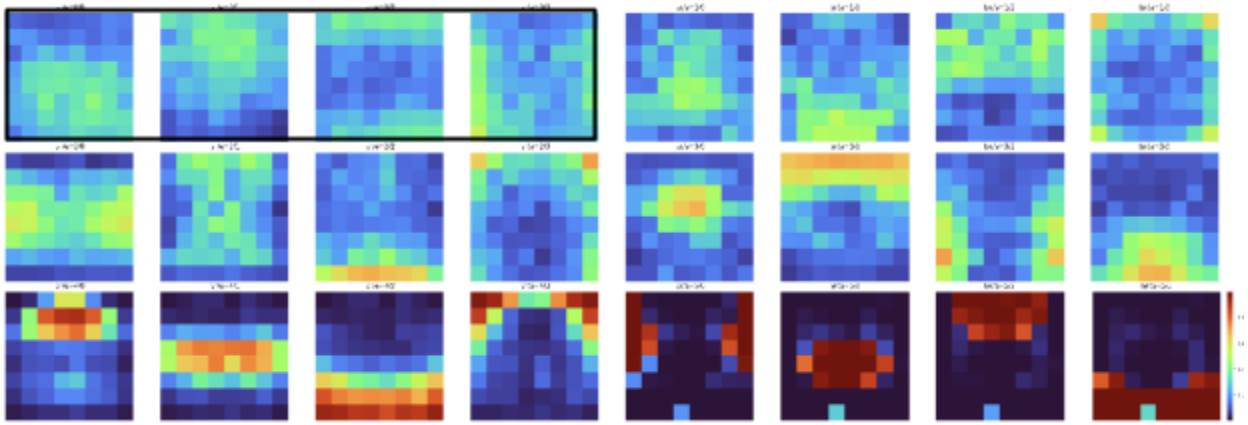




- 2 experts per block:
    - experts are allocated in a spatially topographic way! note that no regularization was used, this emerged naturally!
    - you see experts for center/surround, top/bottom, etc
    - assignment distributions also appear sharper in deeper layers

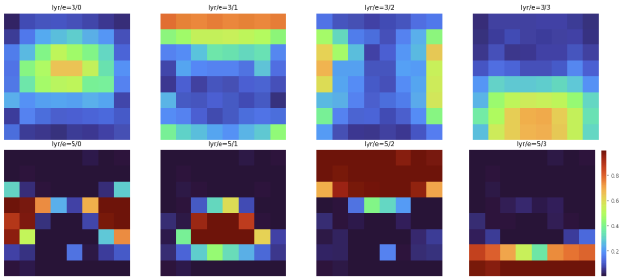


- 4 experts per block:
    - Similar. distributions in early layers are even softer. Maybe it doesn't want to have to specialize here??
    - Across images, no consensus on what expert to pick in early layers.
    - at later layers, you've done more routing, you have wider fov. all of these make it easier to choose a consistent expert for each position.
        - maybe you don't see this in the classic sparse moe. something to test

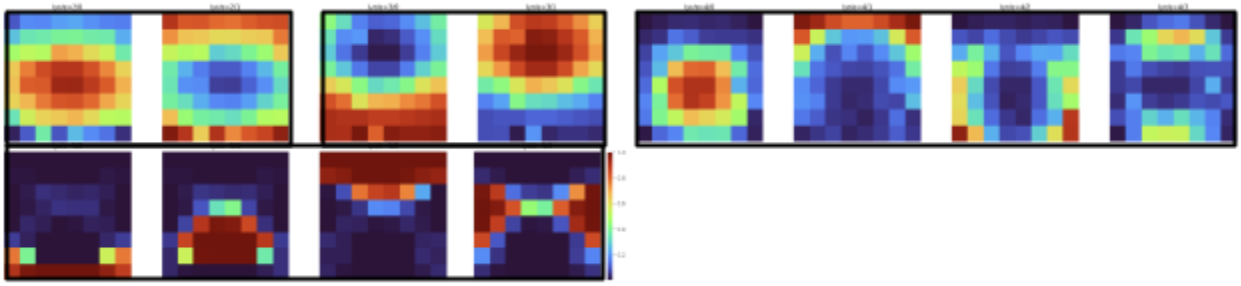


- [1, 1, 1, 4, 4, 4]

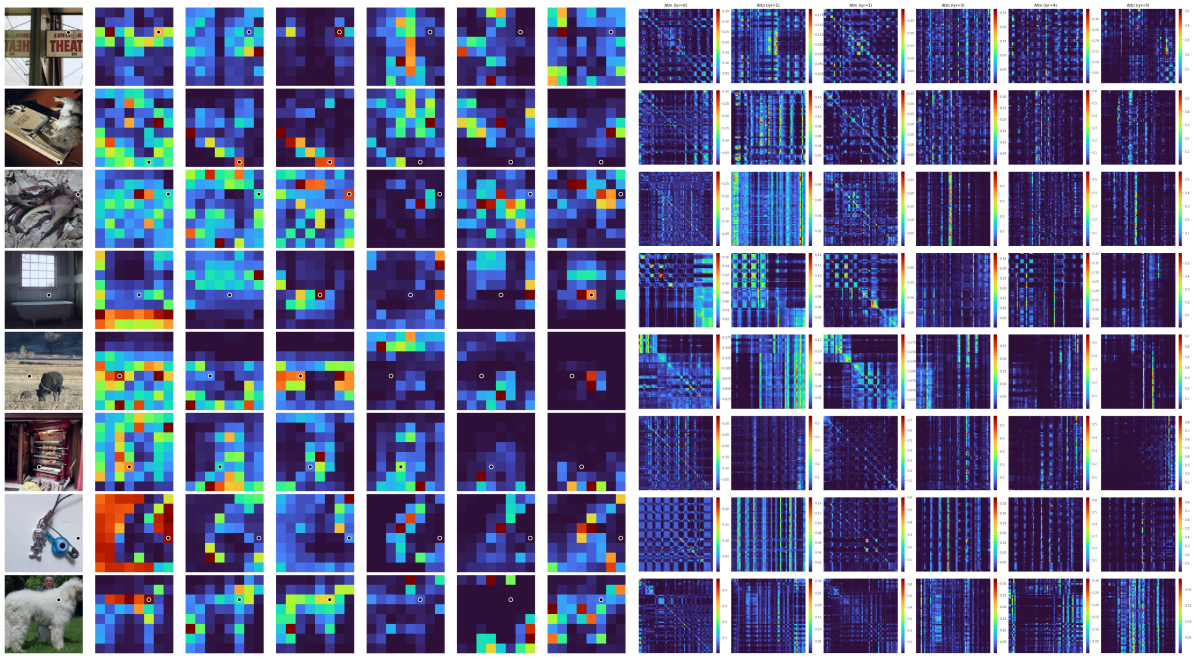
- Now we use 4 experts, but only for the last 3/6 layers
- Coefficient maps look quite similar to the 4 experts per block case

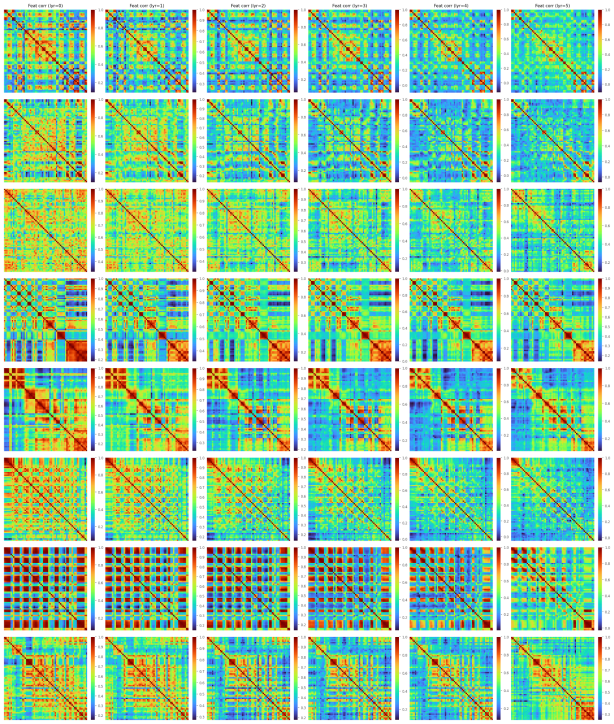


- [1, 1, 2, 2, 4, 4]



- attention maps and attention grid for E-1-4 ([1, 1, 1, 4, 4, 4] case)
  - not sure if there's anything to take away here. but kinda interesting to stare at





- next steps:
    - Run a [4, 4, 4, 1, 1, 1] case to further check hypothesis that experts are better used in later layers
    - Compare closely with methods/results of other MoE papers
    - Investigate what the different experts are specialized for
        - try classification with a spatial pooling head and see if different categories attend to different experts
        - investigate which stimuli maximally/minimally drive activity in each expert
    - Understand why spatial topography naturally emerged without regularization
    - Scale up experiment
    - Figure out the inference time flops for the MoE network
    - Paul: would you get topographic routing maps in the typical sparse routing MoE?
        - Could be something good for someone to implement
- Alisa:
    - Question about spatial similarity loss and alpha values in All-TNNs paper
    - Model is implemented but needs speeding up (currently backprop takes ~16s)
    - Some issues training in colab

Apr 12 2024

Alisa:

- Made a start on implementing the All-TNNs paper in this repository. There are three versions of the topographic layer, one that I came across here, it's basically just conv2d but with untied weights. I also created two other versions, one is pretty much the same

as above and another one only assumes one in and out channel and creates a sparse matrix of all the weights first (which is input_height*input_length x num_kernels) and multiplies the input (flattened into a vector) by it.

- The authors are very vague on how they work out the dimensions of their layers in Figure 1, they say that they use tensorflow's LocallyConnected2D layer and unfold each channel dimension to create a sqrt(channels)*height x sqrt(channels)*width 2D layer.
    - CL: I also don't really understand their description. Btw I emailed them to see if they'd be willing to share code. They're open to it but probably have to finish their revisions first.
- Some questions/thoughts:
    - Having several channels, they pass each input patch through several filters, then I guess when they 'unfold' the layer, the outputs for the same patch are near each other?
    - I think it makes more sense to keep the output channels since otherwise the layer dimensions drop off too quickly.
    - If we assume they use some padding P, stride S and C out channels, with square input of dimension D, and kernel dimension K, there are many combinations of P, S and C that satisfy sqrt(C) * (floor((D - K + 2 * P) / S) + 1) = out_dimension.
- Next steps:
    - Add functionality to unfold the 3D layer into 2D
    - Work out how to achieve their dimensions
    - Implement the spatial similarity loss
    - Create the training loop

Mar 29 2024

Connor:

- Working on MoE implementation that more closely follows the existing literature
    - Vision MoE: classic sparse MoE in a vision setting. (lucidrains impl)
    - Soft MoE is supposed to be a better approach to MoE which avoids non-differentiable top-k selection leading to training instability.
    - There is also soft merging of experts, which combines expert weights similarly to us. The paper seems a bit rushed though and was rejected from ICLR.
    - One challenge with sparse top-k routing is getting gradients through to the bottom e-k experts. With a router layer, you don't have this problem. But in our case we have static routing coefficients (num_tokens, num_experts)
- We have mostly been thinking about wiring cost as a way to promote smooth topography. We should also consider other (more direct) approaches
    - smoothness regularizer applied to expert routing coefficients. E.g. total variation.
    - spatial weight similarity loss from All-TNN. (Actually would be quite similar to ^^)

- Would like to see someone work on implementing the All-TNN from (https://arxiv.org/abs/2308.09431).
    - This is the most closely related work. But their code is not released yet. If we can replicate their topography, it should give insight for how to achieve our own kind of topography.
- Results from soft MoE on expert allocation. later layers work better

Table 5: Expert placing ablation with a Soft MoE S/16 with 12 layers (indexed from 0 to 11).

Sparse Layers	Experts per Layer	Total Experts	IN/10shot	JFT prec1
11	512	512	70.0%	51.5%
10	512	512	70.1%	52.0%
10, 11	256	512	71.7%	52.2%
5, 11	256	512	70.4%	52.1%
8, 9, 10, 11	128	512	**72.8%**	**53.2%**
2, 5, 8, 11	128	512	71.1%	52.5%
4:11	64	512	**72.1%**	**53.1%**
1:4, 8:11	64	512	70.5%	52.1%

Ihab :

- Initial results of increasing the number of experts gradually
- Working on a figure that shows the distributions of coefficients per layers and tokens
    - loading checkpoint state could also provide quick answer
- Exploring performance with high number of experts but same number of params as ViT

Alisa:
Interesting paper to discuss: https://arxiv.org/pdf/2112.04035.pdf
Some questions from reading the SUT paper and how we could implement it for our use case:

- Do we want to use a router? (Will require us to introduce an auxiliary loss, also mutual information maximisation?)
- Mixture of Multi-head Attention?
- Absolute vs relative position embeddings?
- When we tried recursion before, did we implement some sort of halting mechanism?

Mar 22 2024

Connor :

- Current/possible directions:
    - SuMoE: subspace mixture of experts as a way to adapt the degree of weight untying
        - What is the best way to formulate the MoE combination? Linear combination, softmax?
            - **In particular what about coefficients that depend on input?**
            - The brain does kind of route inputs to different weights depending on content. How do we model this?
        - What is a good way to initialize the expert weights/coefficients?

        - Can we match ViT performance with matched number of parameters and num experts > 1?
        - Should we test out classic sparse MoE in our setting.
        - What is the best number of experts per layer
            - num experts = 1 for layers 1-4,
            - num experts = 2 layers 4-8,
            - = 4 layers 8-12,
        - Would keeping some memory of what experts were used in the previous layer and/or their individual outputs (in addition to summing them as done in sparse MoE) lead to more specialization within experts and better routing of information?
    - Wiring cost regularization.
        - Does wiring cost regularization promote topographic structure?
        - What is an experimental setting where we can study wiring cost in relative isolation? All-TNNs?
        - What are good ways of formulating wiring cost?
        - What is the state of literature regarding wiring cost?
        - Nb, you should only be able to get nice topographic structure in the expert layout by using wiring cost. otherwise the layout should end up being random.
        - premise: weight untying + wiring cost = topography
    - Recurrence.
        - Can we get depthwise recurrent architecture to perform reasonably in a vision setting?
        - E.g. can we get the sparse universal transformer to work in a vision setting? Have any papers done this?
    - Communication.
        - We considered various formulations for inter-column communication (self-attention, fixed query "selection", linear mixing).
        - What is the performance of each communication strategy, in both tied and untied weights settings?
        - Are there small adjustments to make to the formulation/initialization with the alternative communication strategies to make them easier to train with?
    - Architecture optimization
        - In basically all experiments, we've noticed that architecture changes tend to make training slower, require lower learning rate, gradient spikes.
        - How can we make these novel architectures easier to train?
        - Do we need to think through initialization/weight conditioning?
        - Do we need to modify normalization or add stabilizing layers like LayerScale?
        - Do we need to be re-tuning hyper-parameters for each architecture change?

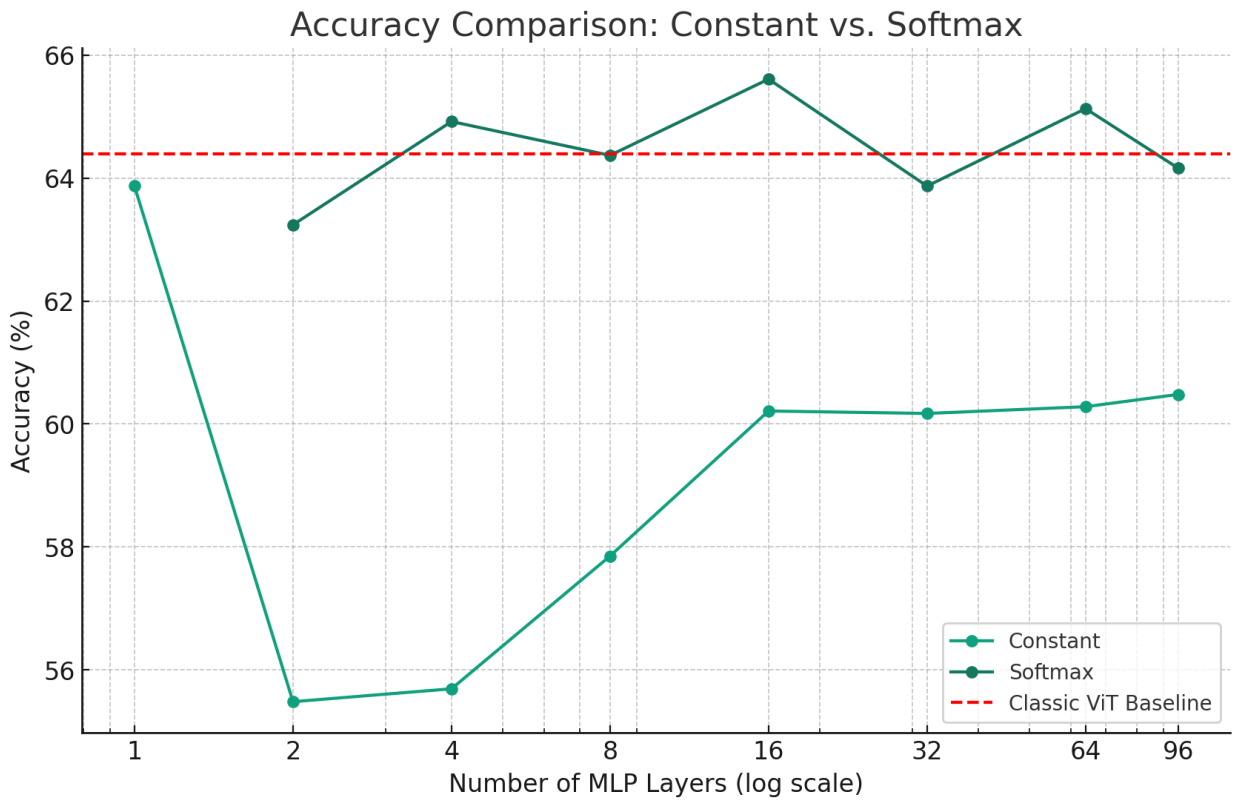
        - Are there tools (e.g. signal propagation) we can use to understand why changing architecture impedes training?
    - Replicating All-TNNs
        - As the closest prior work, maybe we should try to replicate some of their results. It might give some insight into the above issues. The code is also not available yet afaik.
    - Scaling up
        - It's possible that we are playing around at too small of scale. We might need to take a few shots at a bit larger scale (~8 gpu x 24 hours).
    - Non-vision domains
        - Vision is the primary focus, since the primate visual cortex is the primary motivation. However we could consider trying out the architectures in other domains (eg speech, text).
    - Colab playground
        - Improve the colab playground notebook to make it more approachable and useful for small scale experiments.
        - Think about what experiments can be done effectively at small scale.
    - Repo maintenance
        - Keeping README up to date
        - Literature review and bib updates
        - Information for new contributors
        - Code refactoring

- Thoughts on shooting for a CCN 2024 2 page extended abstract? (deadline 4/12)

Mar 14 2024

Ihab :

- Explored scaling laws of numbers of experts (linear combination), uniformly allocated, from 1 to 96 expert (with 64 tokens)
    - Using linear combinations with a mixture of experts (>1) results into lower performance than classic vit (=1).
    - Using a higher number of experts improves performance, but doesn't close the gap to classic vit
- Implement & explore usage of softmax instead of linear combination

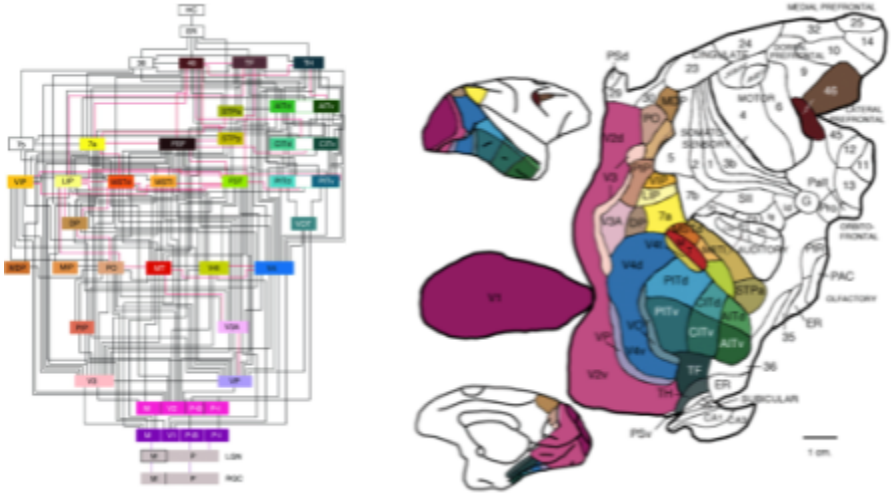


- Specific questions :
    - Softmax : What is the matrix of the learnt coefficients at the end of training for each token?
    - Can we couple the coefficients to the input image/token like for attention? Would we notice specific experts specializing in different classes?
    - Repeat experiments with different seeds
    - Progressive allocation effect
    - Wiring cost

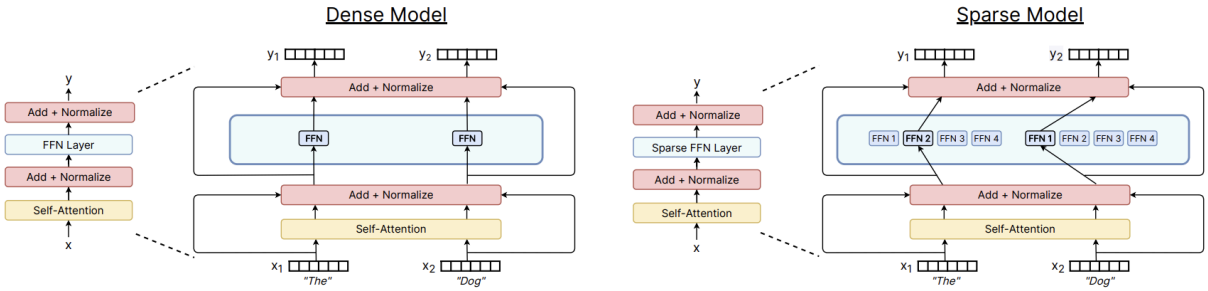
Mar 1 2024

Connor

- This project is interested in topographic functional specialization. Why does the brain have it?



- One idea is that specialization lets you compute *specialized* features for each input
- We see a similar motivation at play in sparse mixture of experts models
    - rather than compute all features for all inputs, only compute a subset of features for each input
    - maintain expressivity while reducing flops.



- Here we want to explore this connection between mixture of experts and functional specialization.
- We introduce a topographic subspace mixture of experts (SuMoE) layer
    - Each position in the layer grid learns a unique set of weights.
    - The weights are computed as linear combinations over a pool of M experts.
- Concrete questions:
    - Does having experts help?
        - performance / flops
    - What is the optimal allocation of experts to each layer?
        - Uniformly allocated?
        - Progressively allocated (1, 1, 2, 2, 4, 4)
    - Does penalizing wiring cost promote topographic arrangement of experts?
        - SuMoE creates opportunity for topography by assigning experts positions
        - Wiring cost will promote topography

[Link to github]

```python
class MixtureMlp(nn.Module):
    def __init__(
        self,
        in_features: int,
        hidden_features: Optional[int] = None,
        out_features: Optional[int] = None,
        seq_len: Optional[int] = None,
        rank: int = 16,
        act_layer: Optional[Layer] = nn.GELU,
        bias: Union[bool, Tuple[bool, bool]] = True,
        drop: Union[float, Tuple[float, float]] = 0.0,
    ):
        super().__init__()
        assert seq_len is not None, "seq_len required for MixtureMLP"

        self.rank = rank
        hidden_features = hidden_features or in_features
        out_features = out_features or in_features
        biases = to_2tuple(bias)
        drop_probs = to_2tuple(drop)
        act_layer = nn.Identity if act_layer is None else act_layer

        self.fc1 = MixtureLinear(
            in_features, hidden_features, rank=rank, bias=biases[0]
        )
        self.act = act_layer()
        self.drop1 = nn.Dropout(drop_probs[0])
        self.fc2 = MixtureLinear(
            hidden_features, out_features, rank=rank, bias=biases[1]
        )
        self.drop2 = nn.Dropout(drop_probs[1])

        # learned static linear coefficients per position
        # each position learns a unique mlp in the subspace spanned by the experts
        self.coef = nn.Parameter(torch.empty(seq_len, rank))
        self.reset_parameters()
```

```python
    def reset_parameters(self) -> None:
        # init so that coef @ weight is ~ N(0, 0.02)
        std = (0.02 * self.rank**-0.5) ** 0.5
        trunc_normal_(self.coef, std=std)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        # standard mlp forward pass
        # maybe we should also try softmax coefficients?
        x = self.fc1(x, self.coef)
        x = self.act(x)
        x = self.drop1(x)
        x = self.fc2(x, self.coef)
        x = self.drop2(x)
        return x
```

```python
class MixtureLinear(nn.Module):
    def __init__(
        self, in_features: int, out_features: int, rank: int = 16, bias: bool = True,
    ):
        super().__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.rank = rank

        self.weight = nn.Parameter(torch.empty((out_features, in_features, rank)))
        if bias:
            self.bias = nn.Parameter(torch.empty(out_features, rank))
        else:
            self.register_parameter("bias", None)
```

```
        self.reset_parameters()

    def reset_parameters(self) -> None:
        # init so that coef @ weight is ~ N(0, 0.02)
        std = (0.02 * self.rank**-0.5) ** 0.5
        trunc_normal_(self.weight, std=std)
        if self.bias is not None:
            nn.init.zeros_(self.bias)

    def forward(self, input: torch.Tensor, coef: torch.Tensor) -> torch.Tensor:
        # B N D
        # N R
        # linear combination of weights at each position. cf model soups
        weight = (coef @ self.weight.transpose(1, 2)).transpose(0, 1)
        if self.bias is not None:
            bias = coef @ self.bias.t()
        output = torch.einsum("bnc,ndc->bnd", input, weight)
        if self.bias is not None:
            output = output + bias
        return output
```

Feb 16 2024

Connor

- I found some important related works to think about:
    - Universal transformers: Classic transformer but with only a single large block that is unrolled recursively for multiple steps. Basically the same as our idea for recursive unrolling.
    - Sparse mixture of experts: Most are probably familiar with this. The key point is that each block consists of several "expert" feedforward modules. Each token is processed by only a subset of them. The rationale is to increase model capacity (i.e. parameter count) without increasing flops (sound familiar?). Untying weights can be seen as an extreme case where each token is processed by its own expert.
    - Sparse universal transformer (SUT): The combination of the above two. I was disappointed to find this, bc I was gonna suggest we try it. Basically the architecture consists of a single block containing many experts, which is unrolled

recursively. This is very closely related to our recurrent columnformer architecture. The only difference is here experts are matched to tokens by a router, whereas in our case each column "expert" is associated 1-1 with a token.
    - A Review of Sparse Expert Models in Deep Learning
- Based on these works, and discussion from last week, I'm trying to sketch the main argument for this work (or at least one candidate)
    - Current vision models like CNNs and ViTs exploit the inductive bias that the same set of features are useful at every position.
    - This is motivated in part by the retinotopic, equivariant organization of early primate visual cortex.
    - However, this organizational principle does not apply to the entire visual cortex.
    - Middle and higher visual stages exhibit striking topographic regional specialization. For example, independent dorsal and ventral streams, independent streams for small, medium, and large objects, and category specialized areas for faces, bodies, objects, places, etc.
    - Similarly, feature specialization has recently become useful in deep networks. In Sparse Mixture of Experts (SMoE) models, each position of the representation map is selectively processed by a subset of features, optimally chosen by a router.
    - In both cases, the intuition is to increase representational capacity without increasing compute cost.
    - In this work, we relax strict position-wise weight sharing by introducing a *subspace mixture of experts* layer (SuMoE).
    - In the SuMoE layer, the weights at each position are computed as a linear combination of expert weights, using static learned coefficients per position.
    - By adapting the number of experts, we can control the degree of weight sharing from fully shared (num experts = 1) to fully untied (num experts = sequence length).
    - We replace the feedforward modules in the standard ViT with SuMoE feedforward modules.
    - In addition, we use a wiring cost penalty to promote smooth, topographically organized expert layouts within each layer.
    - We find that: *(focusing on feedforward, leaving recurrent and growth stuff for follow-up)*
        - Deeper layers benefit from more experts (Or maybe num experts = 1, i.e. classic ViT is optimal for each layer. Which would be interesting and somewhat surprising. Ultimately, it's an empirical question what the optimal number of experts is.)
        - The emergent topographic layout of experts mirrors primate visual cortex. In middle layers, we see the emergence of distinct streams, which further divide into category specialized patches in later layers.
        - Global, dynamic attention based communication is necessary for the emergence of topographic organization. Strict local communication does not allow the emergence of globally distinct streams.

- This more narrow, somewhat different scope might be of interest to a different subset of people. E.g. the MoE transformer architecture is broadly interesting and trendy in the pure ML space right now.
- Basically, we are trying to answer: Why does topographic functional specialization emerge in visual cortex? The hypothesized answer is that specialization increases expressivity while saving compute.

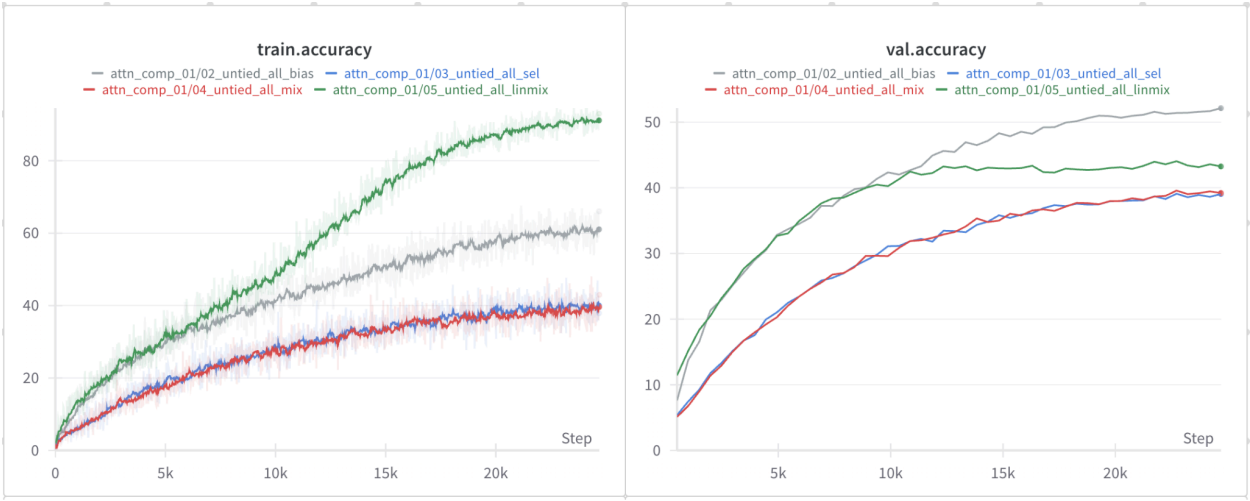
Feb 9 2024

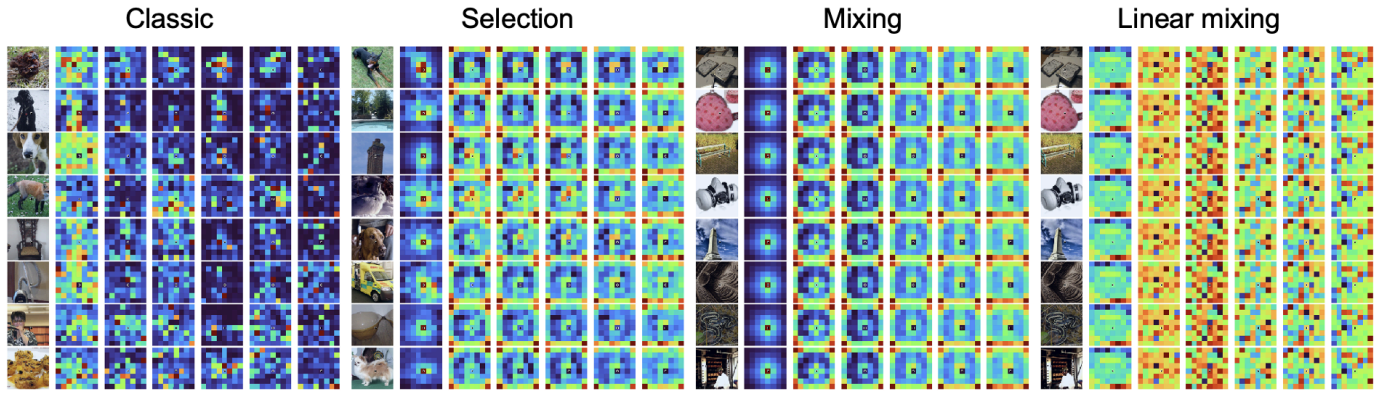
Ihab

- Ran incremental experiments on simplifying the transformer and adding locality.
- Attempt to finding a balance between good performance and model size before the untying.

Connor

- Experimented with different "attention" mechanisms
    - classic: classic multi-head self-attention
    - selection: simplified mechanism. the input is used as both key and value. the query is static and does not depend on the input. the idea is to model the fixed stimulus tuning we see in visual cortex.
    - mixing: even simpler mechanism. the attention matrix is altogether static and does not depend on the input. this is very close to what ConvNext and MLP-Mixer do.
    - linear mixing: mixing without softmax. This is even closer to ConvNext and MLP-Mixer. (ConvNext applies mixing depthwise, which is equivalent num heads = dim. Mixing matrix is convolutional (toeplitz). MLP-Mixer uses a single mixing matrix equivalent to num heads = 1, and applies a nonlinearity.
- Main message:
    - All-TNN is to AlexNet as columnformer is to ViT
        - Why untie weights? Because untying weights enables non-homogeneous layers, which in turn enables topography
        - Why do we need a transformer version of the untied-weight-network? Strict local receptive field vs unconstrained receptive field with locality bias.
    - Additional novel components
        - recurrence
        - geometry
        - scaling by growing
        - low-rank untied weights
        - globally connected information hubs
            - ViT registers are kind of like hubs? (https://arxiv.org/abs/2309.16588)





params : W_K, W_Q, W_V, B

$K = XW_K$

$Q = XW_Q$

$V = XW_V$

$Y = \text{softmax}(QK^\top + B)V$

params : Q, B

$Y = \text{softmax}(QX^\top + B)X$

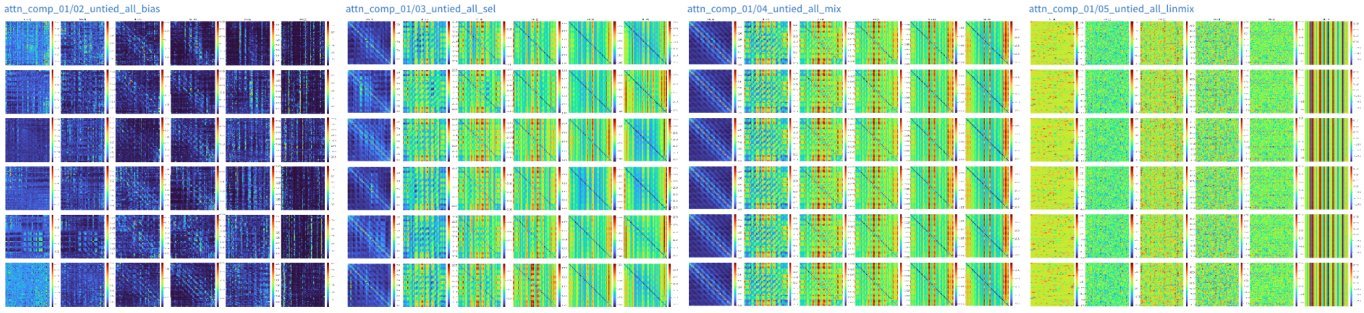
params : B

$Y = \text{softmax}(B)X$

params : B

$Y = BX$

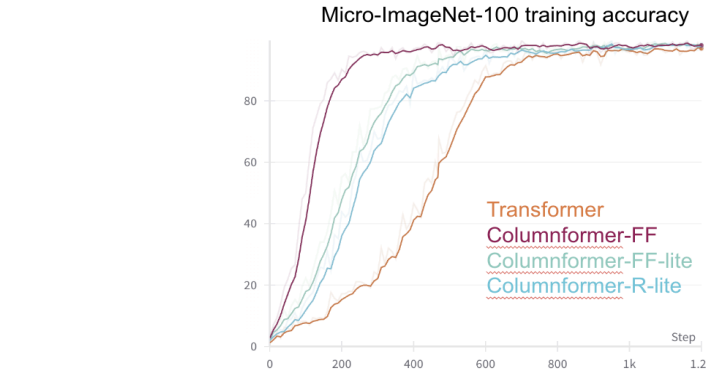
(Cf ConvNext, MLP-Mixer)

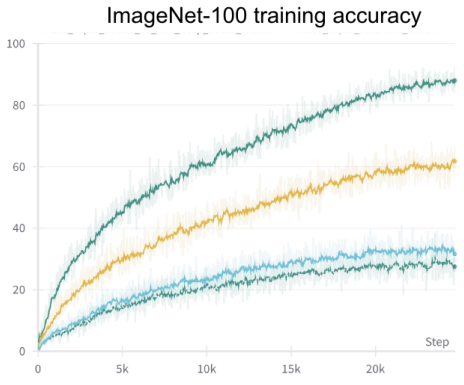


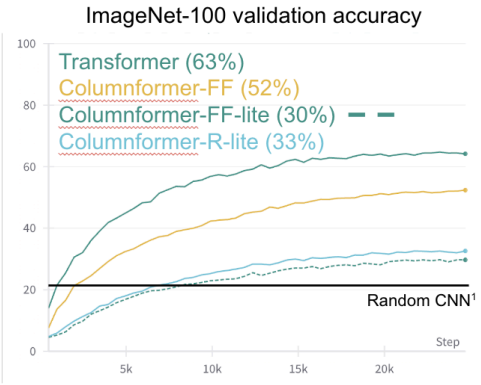
Feb 2 2024

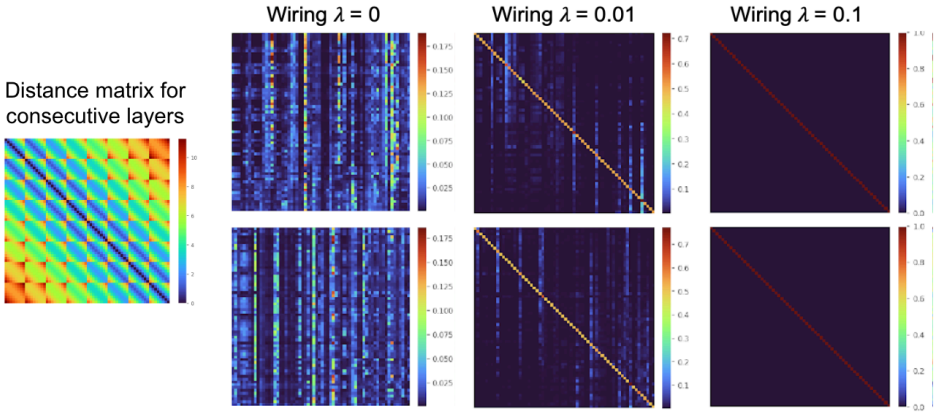
Connor

- See the slides we presented to the Kietzmann Lab last week. These have an updated overview of the model and some initial results.









- I'm interested in trying out a simpler form of "attention" following ConvNext and MLP-Mixer. Rather than dynamic key-query based attention, we use a static (but learned) "mixing" matrix

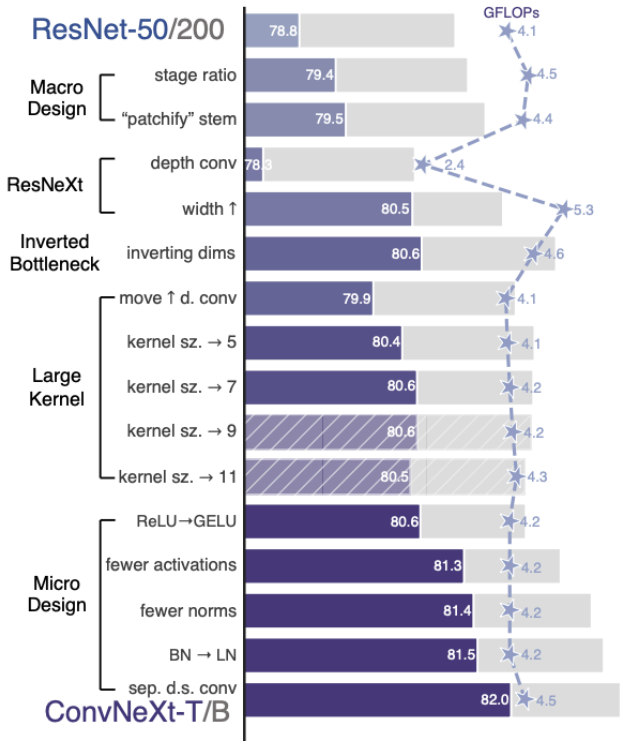
- Ihab: MLP was made for tied weights setting. Can we rethink the MLP and find something that works better in the untied setting? Nb every change to MLP we made hurt performance a lot.
- Paul: we can work towards visualizing the class-specific spatial pooling maps. We need this to show evidence that the basic topographic idea works.

```python
class Mixing(nn.Module):
    """
    Multi-head static feature "mixing" similar to ConvNext and MLP-Mixer.
    """

    def __init__(
        self,
        dim: int,
        seq_len: int,
        groups: int = 8,
        attn_drop: float = 0.0,
    ):
        super().__init__()
        assert dim % groups == 0, "dim should be divisible by groups"
        assert seq_len, "seq_len required for mixing"

        self.dim = dim
        self.seq_len = seq_len
        self.groups = groups
        self.group_dim = dim // groups

        self.weight = nn.Parameter(torch.empty(groups, seq_len, seq_len))
        self.bias = nn.Parameter(torch.empty(seq_len, seq_len))
        self.attn_drop = nn.Dropout(attn_drop)
        self.reset_parameters()

    def reset_parameters(self) -> None:
        trunc_normal_(self.bias, std=0.02)
        nn.init.zeros_(self.weight)

    def forward(self, x: torch.Tensor):
        B, N, C = x.shape
        x = x.reshape(B, N, self.groups, self.group_dim).transpose(1, 2)

        attn = self.weight + self.bias
        attn = attn.softmax(dim=-1)
        attn = self.attn_drop(attn)
        x = attn @ x

        x = x.transpose(1, 2).reshape(B, N, C)
        return x, attn
```

Jan 19 2024

Connor

- Rewrote model implementation to support a step-by-step ablation from the columnformer back to the transformer.
- Integrated with train script and debugged. Will be launching some runs.
- Proposed ablation order:
    - Simplifying the transformer (vision_transformer_tiny_patch16_128):
        - reduce mlp dim
        - single head attention
        - remove attention value and proj
    - Adding transformer locality:
        - add wiring cost promoting local attention
        - add attention bias
        - local attention bias initialization
        - which transformer variants do local attention?
    - Untie weights (vision_columnformer_ff_tiny_patch16_128):
        - untie mlp weights
        - untie attention weights
        - untie norm weights
    - Add recurrence (vision_columnformer_r_tiny_patch16_128):
        - flatten layers into sheet and unroll recurrently
- The plan is to train each of these tiny versions (6 layer, dim 384) of each of these variants on imagenet-100 (128px, 100 epochs). At the end, we want a figure like this:



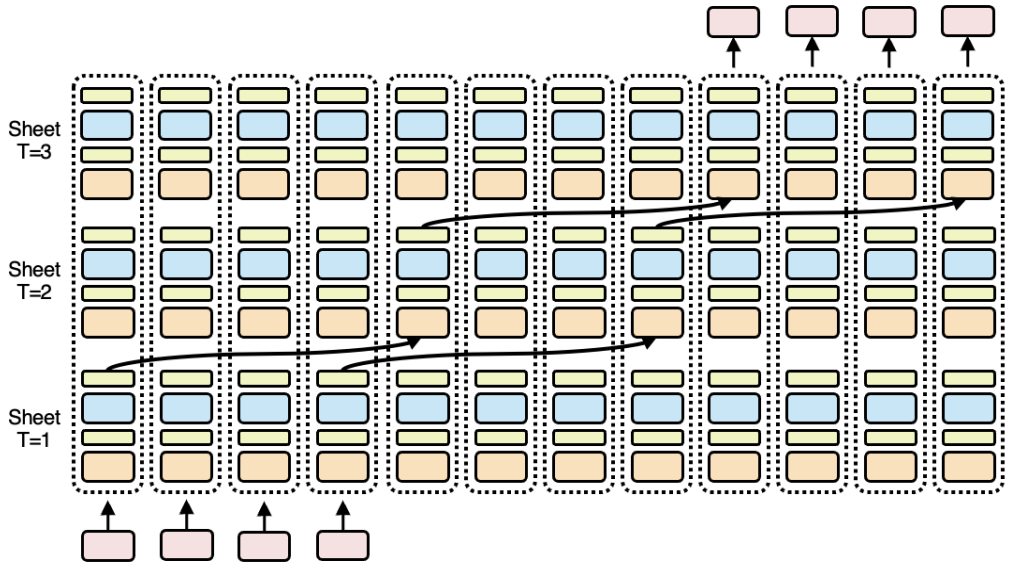
- would love volunteers to take on one of the groups of models in the ablation

- alternatively, experiment with all combinations in each block
- no need to overcommit to this order. exploring within a block to find the best order makes sense.
- Why do we even need to reduce the attention and mlp parameters?
    - untying weights increases parameter count by a factor of the sequence length
    - importantly, it doesn't the flop count. so models should be just as efficient to train
    - the challenge is that a massively over-parameterized model might overfit. but this doesn't mean we shouldn't try it.
- **2 versions of experiment: preserve flops or preserve parameters**
    - to test whether it's important to limit columnformer parameters, we can do two versions of the ablation experiment, one with the first 3 transformer simplification steps, one without.
    - simplifying transformer = matching parameters = reducing flops
    - matching transformer = increasing parameters = matching flops

Jan 12 2024

Connor

- **Announcement:** We are meeting with members of the Kietzmann lab to talk about the columnformers idea and topographic ANNs in general. This is the group who developed All-TNNs. The meeting will be on Wed 1/24 at 11am EST. Will post an official announcement in the channel.
- Development updates:
    - Added a training script for larger scale training. Supports distributed training, mixed precision, wandb logging, figures and metrics hooks, etc.
    - Possibly overkill and should instead use something like lightning. Lmk what people think.
- Question from Mohammed: Why don't we also untie weights depthwise?
    - In principle, you should not need to. The "diagonal" path through the network is precisely what you would have if you untied weights depthwise. The recurrent case is strictly more general.
    - Nonetheless, it will be a good idea to try. More general does not necessarily mean better.



Paul

- Has some questions after looking through the codebase

Jan 5 2024

Connor

- Added a related work section to the repo. This is a good place to find relevant background. Anyone please feel free to open PRs to add more papers!
- Started my own development branch where I'm working on some stuff.
    - datasets_factory.py: create imagenet-100, micro-imagenet-100 datasets with timm transforms.
    - Being able to observe what's going on during model training will probably be important to understand and diagnose problems. Started adding some figures and metrics. The idea is for these to follow a general interface to make it easy to register to a model. (**Q:** I feel like pytorch lightning must support this sort of thing, but I'm not too familiar with it. Does anyone feel like that's what we should be using?)
    - Added a ViT "baseline" classification model. Important to know what the ViT performance is for our specific task (imagenet-100 128x128 100 epochs). We expect this to be more like a ceiling than a baseline though. Goal is to get as close as we can.

Ihab

- Contributed imagenet-100 benchmark notebook
- Initial baseline performance. 23%.

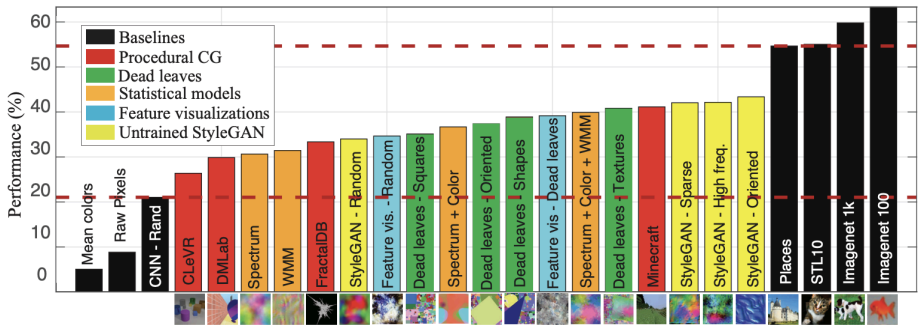
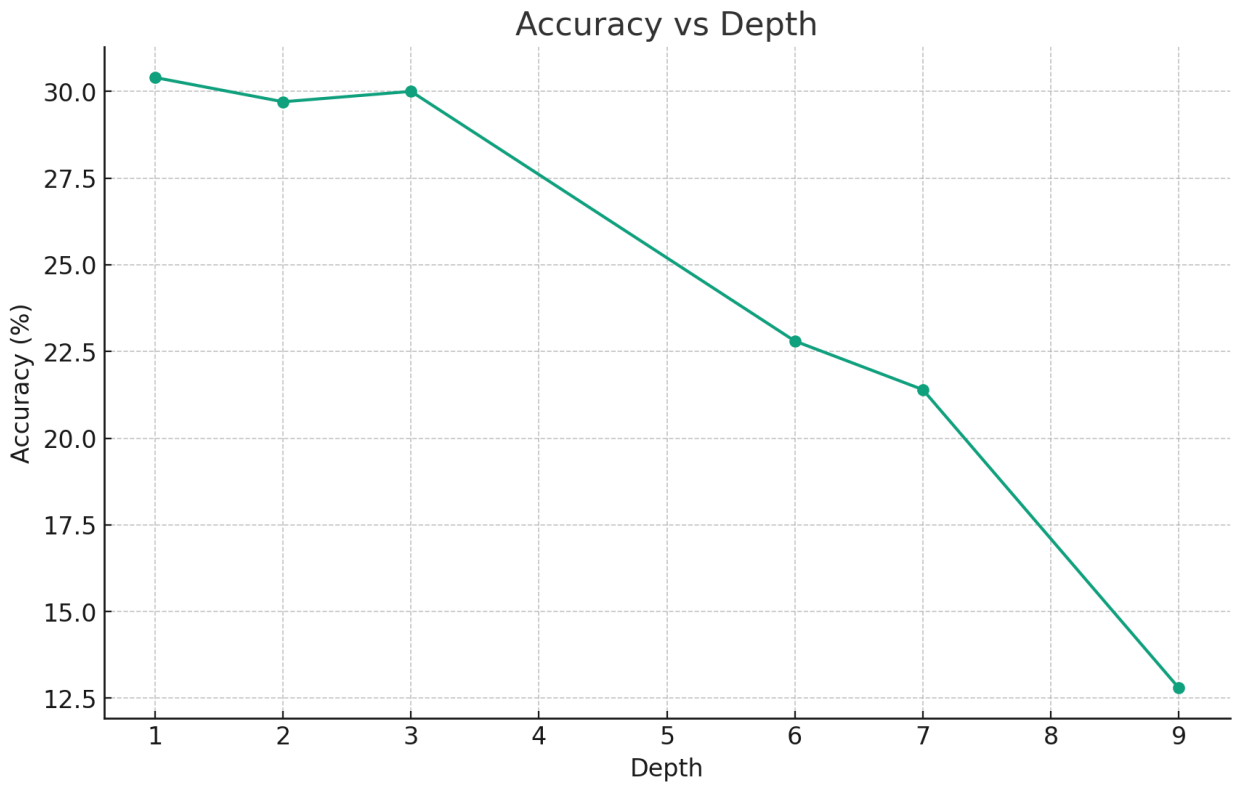


Figure 3: Top-1 accuracy for the different models proposed and baselines for Imagenet-100 [60]. The horizontal axis shows generative models sorted by performance. The two dashed lines represent approximated upper and lower bounds in performance that one can expect from a system trained from samples of a generic generative image model.


Accuracy vs Depth
Accuracy (%)
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
1
2
3
4
5
6
7
8
9
Depth