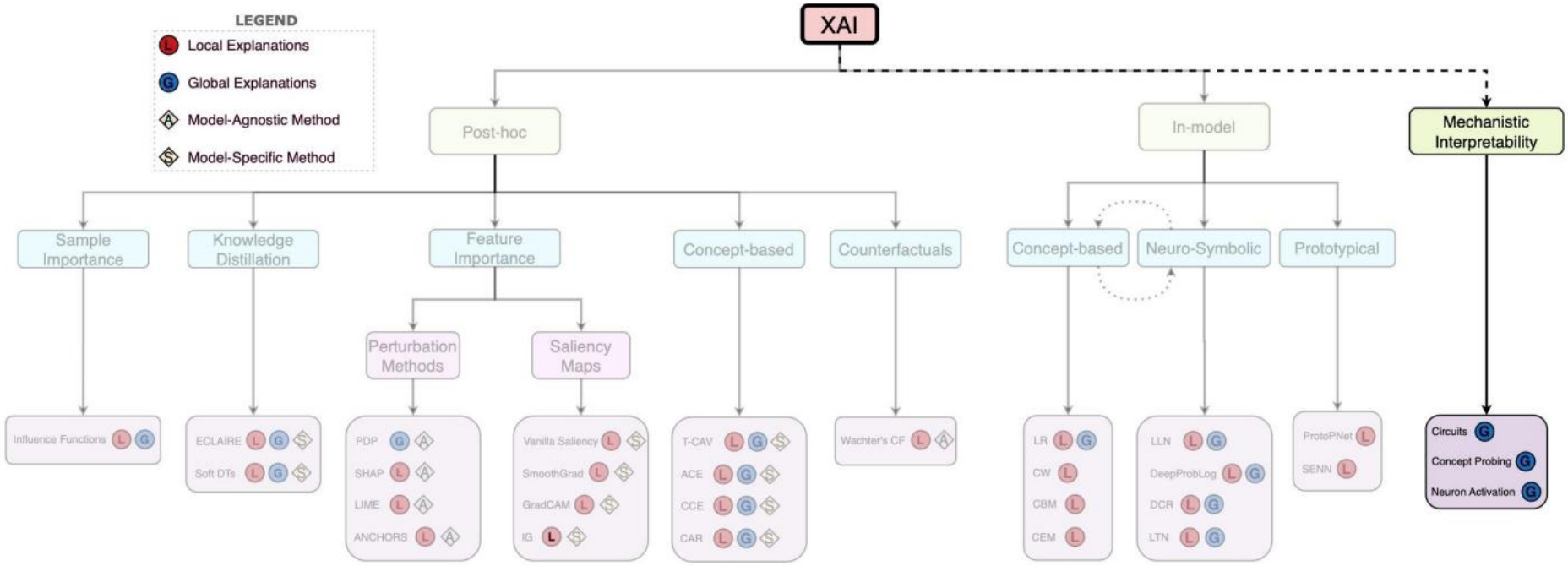
UNIVERSITY OF CAMBRIDGE

Finding Transformer Circuits with Edge Pruning

Adithya Bhaskar, Alexander Wettig, Dan Friedman, Danqi Chen

L193 Explainable Artificial Intelligence

Overview


LEGEND
Local Explanations
Global Explanations
Model-Agnostic Method
Model-Specific Method
XAI
Post-hoc
In-model
Mechanistic Interpretability
Sample Importance
Knowledge Distillation
Feature Importance
Concept-based
Counterfactuals
Concept-based
Neuro-Symbolic
Prototypical
Perturbation Methods
Saliency Maps
Influence Functions
ECLAIRE
Soft DTs
PDP
SHAP
LIME
ANCHORS
Vanilla Saliency
SmoothGrad
GradCAM
IG
T-CAV
ACE
CCE
CAR
Wachter's CF
LR
CW
CBM
CEM
LLN
DeepProbLog
DCR
LTN
ProtoPNet
SENN
Circuits
Concept Probing
Neuron Activation

Background

We already saw ACDC:

1. Build a dataset of a specific behaviour
2. Overwrite the value of an edge in the computational graph
3. If the output doesn’t change, then remove the edge from our circuit
4. Repeat until we have checked all edges

[1] Conmy, Arthur, et al. "Towards automated circuit discovery for mechanistic interpretability." *Advances in Neural Information Processing Systems* 36 (2023): 16318-16352.

Background

We already saw ACDC:

1. Build a dataset of a specific behaviour
2. Overwrite the value of an edge in the computational graph
3. If the output doesn’t change, then remove the edge from our circuit
4. Repeat until we have checked all edges

Edge Attribution Patching (EAP) linearly approximates activation patching by using gradients to assign importance to each edge.

[2] Syed, Aaquib, Can Rager, and Arthur Conmy. "Attribution patching outperforms automated circuit discovery." arXiv preprint arXiv:2310.10348 (2023).

Background

ACDC:
+ finds good circuits
- slow

EAP:
- finds worse circuits
+ fast

Background

ACDC:
+ finds good circuits
- slow

EAP:
- finds worse circuits
+ fast

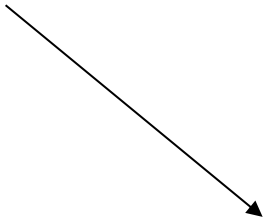
We want:
+ finds good circuits
+ fast

Background

ACDC:
+ finds good circuits
- slow

EAP:
- finds worse circuits
+ fast

We want:
+ finds good circuits
+ fast



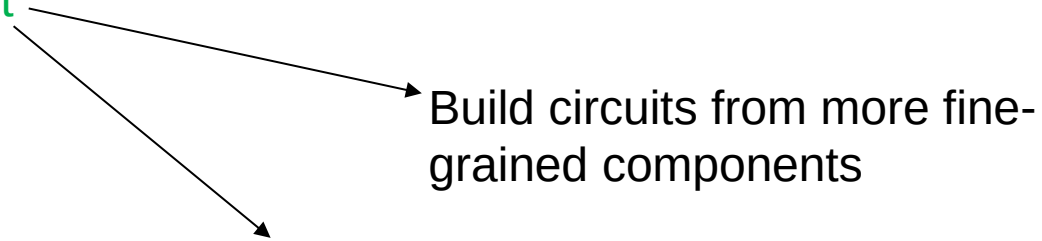
Scale to big datasets / models

Background

ACDC:
+ finds good circuits
- slow

EAP:
- finds worse circuits
+ fast

We want:
+ finds good circuits
+ fast

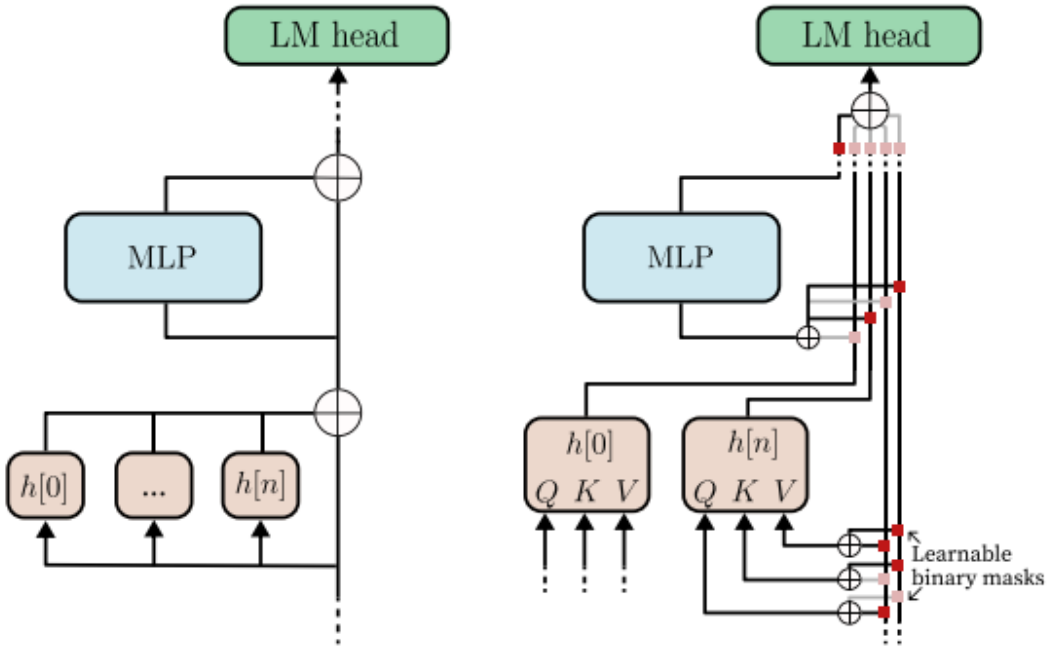


Background

Edge pruning aims to find useful circuits efficiently by using gradient descent to prune edges.

Method

1. Build computational graph with disentangled residual stream



[3] Bhaskar, Adithya, et al. "Finding transformer circuits with edge pruning." Advances in Neural Information Processing Systems 37 (2024): 18506-18534.

Method

1. Build computational graph with disentangled residual stream
2. On edge $j \to i$, associate a parameter z_{ji}, so that the node at the end of the edge takes value

$$y_i = f_i\left(\sum_{j\ upstream\ of\ i} z_{ji}y_j + (1 - z_{ji})\tilde{y}_j\right)$$

[3] Bhaskar, Adithya, et al. "Finding transformer circuits with edge pruning." Advances in Neural Information Processing Systems 37 (2024): 18506-18534.

Method

1. Build computational graph with disentangled residual stream
2. On edge $j \to i$, associate a parameter z_{ji}, so that the node at the end of the edge takes value

$$y_i = f_i\left(\sum_{j\ upstream\ of\ i} z_{ji}y_j + (1 - z_{ji})\tilde{y}_j\right)$$

layer i

output of node j

counterfactual output of node j

[3] Bhaskar, Adithya, et al. "Finding transformer circuits with edge pruning." Advances in Neural Information Processing Systems 37 (2024): 18506-18534.

Method

1. Build computational graph with disentangled residual stream
2. On edge $j \to i$, associate a parameter z_{ji}, so that the node at the end of the edge takes value

$$y_i = f_i\left(\sum_{j\ upstream\ of\ i} z_{ji} y_j + (1 - z_{ji})\tilde{y}_j\right)$$

layer i

output of node j

counterfactual output of node j

3. Learn z_{ji} by gradient descent, minimising: *

$$\mathcal{L} = \mathcal{L}_{KL} + \mathcal{L}_{sparsity}$$

* Some details omitted for clarity (see section on *Details of the Edge Pruning process* in the paper)

Method

1. Build computational graph with disentangled residual stream
2. On edge $j \to i$, associate a parameter z_{ji}, so that the node at the end of the edge takes value

$$y_i = f_i\left(\sum_{j\ upstream\ of\ i} z_{ji} y_j + (1 - z_{ji})\tilde{y}_j\right)$$

layer i

output of node j

counterfactual output of node j

3. Learn z_{ji} by gradient descent, minimising: *

$$\mathcal{L} = \mathcal{L}_{KL} + \mathcal{L}_{sparsity}$$

change in outputs compared to full network

Sparsity constraint on z

* Some details omitted for clarity (see section on *Details of the Edge Pruning process* in the paper)

Method

1. Build computational graph with disentangled residual stream
2. On edge $j \to i$, associate a parameter z_{ji}, so that the node at the end of the edge takes value

$$y_i = f_i\left(\sum_{j\ upstream\ of\ i} z_{ji}y_j + (1 - z_{ji})\tilde{y}_j\right)$$

3. Learn z_{ji} by gradient descent, minimising: *

$$\mathcal{L} = \mathcal{L}_{KL} + \mathcal{L}_{sparsity}$$

* Some details omitted for clarity (see section on *Details of the Edge Pruning process* in the paper)

Method

1. Build computational graph with disentangled residual stream
2. On edge $j \to i$, associate a parameter z_{ji}, so that the node at the end of the edge takes value

$$y_i = f_i\left(\sum_{j\ upstream\ of\ i} z_{ji}y_j + (1 - z_{ji})\tilde{y}_j\right)$$

3. Learn z_{ji} by gradient descent, minimising: *

$$\mathcal{L} = \mathcal{L}_{KL} + \mathcal{L}_{sparsity}$$

4. Round each z_{ji}

* Some details omitted for clarity (see section on *Details of the Edge Pruning process* in the paper)

Method

1. Build computational graph with disentangled residual stream
2. On edge $j \to i$, associate a parameter z_{ji}, so that the node at the end of the edge takes value

$$y_i = f_i\left(\sum_{j\ upstream\ of\ i} z_{ji}y_j + (1 - z_{ji})\tilde{y}_j\right)$$

3. Learn z_{ji} by gradient descent, minimising: *

$$\mathcal{L} = \mathcal{L}_{KL} + \mathcal{L}_{sparsity}$$

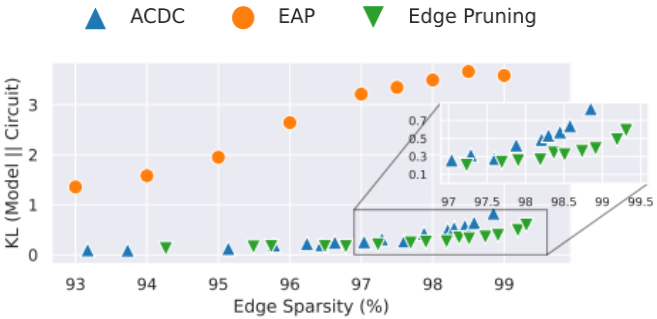
4. Round each z_{ji}

Similar to previous pruning methods [4] but using edges for granularity

[4] Louizos, Christos, Max Welling, and Diederik P. Kingma. "Learning sparse neural networks through $ L_0 $ regularization." arXiv preprint arXiv:1712.01312 (2017).

Results

Circuit performance:



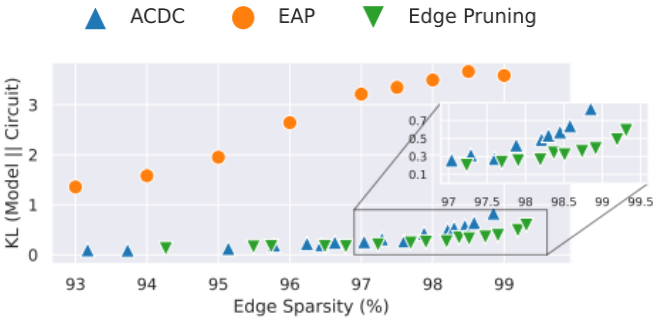
(a) IOI-t1 (IOI, 1 template)

Friends Juana and Kristi found a mango at the bar. Kristi gave it to → Juana

[5] Wang, Kevin, et al. "Interpretability in the wild: a circuit for indirect object identification in gpt-2 small." arXiv preprint arXiv:2211.00593 (2022).

Results

Circuit performance:



(a) IOI-t1 (IOI, 1 template)

Friends Juana and Kristi found a mango at the bar. Kristi gave it to → Juana

Performs better than ACDC and EAP on experiments presented

[5] Wang, Kevin, et al. "Interpretability in the wild: a circuit for indirect object identification in gpt-2 small." arXiv preprint arXiv:2211.00593 (2022).

Results

Efficiency:

Method	Sparsity (%) ↑	*200 examples*		*400 examples*		*100K examples*	
		KL ↓	Time (s) ↓	KL ↓	Time (s) ↓	KL ↓	Time (s) ↓
ACDC	96.6 ± 0.1	0.92	18,783	0.88	40,759	-	-
EAP	96.6 ± 0.1	3.47	**21**	3.66	**43**	3.78	12,260
Edge Pruning	96.6 ± 0.1	**0.25**	2,756	**0.22**	2,931	**0.20**	**3,042**

[3] Bhaskar, Adithya, et al. "Finding transformer circuits with edge pruning." Advances in Neural Information Processing Systems 37 (2024): 18506-18534.

Results

Efficiency:

Method	Sparsity (%) ↑	*200 examples*		*400 examples*		*100K examples*	
		KL ↓	Time (s) ↓	KL ↓	Time (s) ↓	KL ↓	Time (s) ↓
ACDC	96.6 ± 0.1	0.92	18,783	0.88	40,759	-	-
EAP	96.6 ± 0.1	3.47	**21**	3.66	**43**	3.78	12,260
Edge Pruning	96.6 ± 0.1	**0.25**	2,756	**0.22**	2,931	**0.20**	**3,042**

Much faster than ACDC in all tests

Faster than EAP for large datasets

[3] Bhaskar, Adithya, et al. "Finding transformer circuits with edge pruning." Advances in Neural Information Processing Systems 37 (2024): 18506-18534.

Summary of contributions

Goal: edge pruning aims to find useful circuits efficiently by using gradient descent to prune edges

Summary of contributions

Explicitly optimises circuit performance (unlike EAP)

Goal: edge pruning aims to find useful circuits efficiently by using gradient descent to prune edges

Performs gradient descent to evaluate all edges in parallel

Summary of contributions

Explicitly optimises circuit performance (unlike EAP)

Goal: edge pruning aims to find useful circuits efficiently by using gradient descent to prune edges

Performs gradient descent to evaluate all edges in parallel

- Results suggest improved upon previous works
- Further analysis on circuits found, to indicate that they are interpretable [6]

[6] Makelov, Aleksandar, Georg Lange, and Neel Nanda. "Is this the subspace you are looking for? an interpretability illusion for subspace activation patching." arXiv preprint arXiv:2311.17030 (2023).

Limitations

- Does not prune individual weights

- Lack of ablation of the hyperparameters e.g. mask initialisation strategy

- Little comparison with channel pruning [7], which is very similar

- No comparison to manually identified circuits for the same problems.

- Requires a lot of of memory: 32 H100s

- No discussion on why KL divergence loss is used rather than logit difference ($\log P(correct) - \log P(misleading)$), when previous methods they cited used logit difference

[7] He, Yihui, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks." *Proceedings of the IEEE international conference on computer vision*. 2017.

Questions?

Details of the Edge Pruning process Our formulation of pruning is based on that used by CoFi Pruning [Xia et al., 2022]. Specifically, we model the masks $\boldsymbol{z}$ based on the hard concrete distribution as done by Louizos et al. [2018]:

$$\mathbf{u} \sim \text{Uniform}(\epsilon, 1-\epsilon)$$
$$\mathbf{s} = \sigma\left(\frac{1}{\beta} \cdot \frac{\mathbf{u}}{1-\mathbf{u}} + \log \boldsymbol{\alpha}\right)$$
$$\tilde{\mathbf{s}} = \mathbf{s} \times (r-l) + l$$
$$\mathbf{z} = \min(1, \max(0, \tilde{\mathbf{s}}))$$

where σ refers to the sigmoid function, $\epsilon = 10^{-6}$, and $\log \boldsymbol{\alpha}$ indicates that the logarithm is applied element-wise. We fix the temperature $\frac{1}{\beta} = \frac{2}{3}$. The last two lines stretch the distribution to $[l, r] = [-0.1, 1.1]$ and accumulate the "excess" probability on either side to 0 and 1, respectively. The log alphas $\log \boldsymbol{\alpha}$ are the learnable parameters in this formulation.

Following, Wang et al. [2020], a target sparsity is enforced via a Lagrangian term [Louizos et al., 2018]. If the current sparsity is s, the term, parametrized by a reference value t is

$$\mathcal{L}_s = \lambda_1 \cdot (t-s) + \lambda_2 \cdot (t-s)^2$$

λ_1 and λ_2 are also updated during training via gradient *ascent* to keep the regularization tight. We vary the value of t throughout training, linearly increasing it from 0 to a target value, as outlined in Appendix A. Although it may be useful to think of t as a "target" sparsity, it is only a number. The runs usually converge to a value slightly below t, so it is prudent to set it to a value *greater than* 1—although s can then never reach the target value, it will be pushed to higher sparsities.

We have two sets of masks z. The first set associates a $0-1$ value z_e with each edge $e \equiv (n_1, n_2)$ in the computational graph. The second set tags each *node* of the graph n with a $0-1$ value z_n. The latter specifies whether a node is "active", i.e., producing output. In effect, the presence of an edge $e \equiv (n_1, n_2)$ is determined by the binary mask

$$\tilde{z}_{(n_1,n_2)} = z_{(n_1,n_2)} \times z_{n_1}$$

We initially only used edge masks but found that the method would have difficulty converging to high sparsities (i.e., end up at low sparsities). Introducing a second set of masks allows the process to eliminate many edges quickly, accelerating the removal of unimportant components. However, the lagrangian above only applies to the edge masks. This is fine since the node masks can only remove further edges, not introduce new ones on top of those chosen by the edge masks. The final loss is

$$\mathcal{L} = \mathcal{L}_{\text{KL}} + \mathcal{L}_{\text{edge},s}$$

Transform z to encourage $z \in \{0,1\}$

Details of the sparsity loss

Apply masks to *both* the edges and the nodes to achieve better sparsity