Abstract Attention Robustness

Author

March 2023

1 Problem Statement

Find the boundary of attention robustness on a given perturbation by considering an abstraction over activation patterns.

2 Procedure

2.1 Pre-processing

To verify the attention robustness of model α^{org} on input image I, the first step is to load it and evaluate the output label l_p on image I. Next, two perturbation metrics are chosen to be used to obtain the perturbation layers that correspond to the input image I. These layers are then appended to the model α^{org} to create a new model α^{pert}, which takes the perturbation values as input and produces the output of α^{org} for the corresponding perturbed images.

To obtain the gradients G_0 of the input to model α^{org}, a neighbourhood around the input image I with the same activation pattern is selected, and the gradients are computed. A second set of perturbation layers, α^{gard}, is then computed, where zero input produces the gradient for the original image, and any perturbation input leads to the corresponding expected gradient.

2.2 Phase I

The proposed approach involves partitioning the input space into a 10*10 grid (the exact number is based on the specifications of the GPU), with the batch size set to accommodate all initial perturbations in a single batch. Executed in parallel on a GPU, the following steps are performed batch-wise:

1. For each partition, the algorithm considers the lower left point, denoted as LL, which contains two perturbation values. The algorithm then computes the concrete activation pattern, denoted as AP_C, for the model α^{pert} corresponding to the concrete point LL.

2. Next, the algorithm performs a forward pass through the network α^{org} to obtain the gradient G_{obt} with respect to the neighbourhood of input images corresponding to activation pattern AP_C. At any intermediate stage of the forward pass, we have the gradient with respect to the image layer on the current layer.

3. LL is then provided as an input to α^{grad} to obtain the expected gradient, denoted as G_{exp}. The algorithm computes the L_2 norm between G_{obt} and G_{exp}, denoted as δ_{LL}, and checks if it is within $(\gamma - \gamma_{curr}) * \lambda + c$ distance from δ_{accept}, where γ_{curr} is the current depth of the partition, γ is the depth at which we move to phase II, and λ and c are hyperparameters. The formula leads to lower values as current depth γ_{curr} reaches γ, as smaller a partition is less likely to have the boundary pass through it while still having a very different δ_{LL} from δ_{accept}. In each level of depth we further partition the input space, thus greater γ_{curr} has smaller partitions.

Partitions that satisfy the above condition are considered for further analysis and are further partitioned into four parts each and batched into a size of 120. If the current depth γ_{curr} is equal to γ, the algorithm executes phase II on the batches. Otherwise, the algorithm repeats the previous steps on the new batches with γ_{curr} incriminated by 1.

2.3 Phase II

In this phase, parallel execution of partitions (obtained from the previous phase) in batches utilizing GPU resources is employed to enhance computational efficiency.

For each partition, we calculate the δs at the four corner points, using methods similar to those in phase I. A partition is flagged if it's δs have different signs, indicating that the boundary passes through them.

Flagged partitions are then subjected to further partitioning into four parts each and batched with a size of 25, which is then directed to phase III.

2.4 Phase III

In this phase, we conduct batch-wise execution on partitions obtained from the previous step, leveraging parallelism on GPU architecture.

1. For each partition, we compute the abstract activation pattern AP_a for model α^{pert} using Deeppoly, which overapproximates all possible abstract patterns resulting from passing each concrete point in the partition through the model. Our notation labels a ReLU as 0 if it's always inactive, 1 if it's always active, and 2 if its status is unknown for the current partition.

2. To obtain the abstract obtained gradient G^a_{obt}, we utilize intervals instead of concrete values as in the previous phases. Whenever the status of a ReLU node in AP_a is unknown, we multiply by the interval [0,1]. Thus, G^a_{obt} becomes an array of intervals, each corresponding to a pixel position.

3. For the abstract expected gradient G^a_{exp}, we obtain the abstract output by passing the partition through α^{grad} using Deeppoly.

4. We compute the abstract delta δ^a for the partition, which overapproximates the L2 norm of all possible combinations of concrete values in G^a_{obt} and G^a_{exp}. If the interval δ^a is entirely greater than the threshold value δ_{accept}, we label the block as $\neg AR$ (Red). If it is completely less than δ_{accept}, we label the block as AR (Green). Otherwise, it is a boundary block (Black).

3 Toy Example

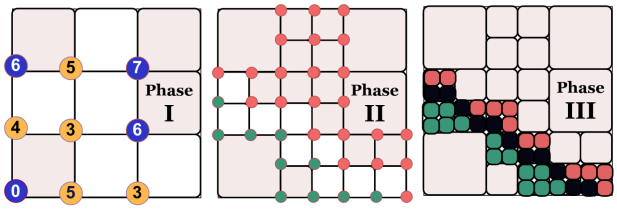

Figure 1: Toy Example

In this section, we shall present a comprehensive account of the algorithm using a toy example depicted in Figure 1. We set the hyper-parameters δ_{accept}, γ, λ, and c to 3, 0, 1, and 2, respectively.

The first step in the algorithm involves preprocessing, which results in the creation of three models α^{org}, α^{pert}, and α^{grad}.

In phase I, we divide the perturbation input space into 3*3 parts. For each partition, we consider the lower left point LL and compute δ_{LL} to values encircles in Figure 1. With the current depth γ_{curr} at 0, $(\gamma - \gamma_{curr})\lambda + c$ is evaluating to 2. Therefore, partitions with δ_{LL} in the range of $[1, 5]$ are marked in white (with LL in yellow) Conversely, the remaining partitions are coloured off-white (with LL in blue). Since $(\gamma == \gamma_{curr})$ is true, we further partition the white blocks into four parts and consider them for phase II.

In phase II, for each of the partitions obtained from phase I, we evaluate the four corner points and designate them red if δ is greater than δ_{accept}, otherwise green. If a partition has corner points of different colours, we designate it white, otherwise, we designate it off-white. The white partitions are further divided into four parts and sent to phase III.

In phase III, we use abstract interpretation to evaluate all points in a partition at once and obtain an over-approximation of the range of δ for all input points in the partition. If the range is entirely less than or greater than δ_{accept}, we designate the partition green (AR) or red ($\neg AR$), respectively. Otherwise, we designate it black and regard it as the boundary. The black partition can contain both AR and $\neg AR$ points. Therefore, conducting further depths

of partition in phase I would result in smaller partition sizes, stricter over-approximations, and a thinner boundary.

4 Limitation

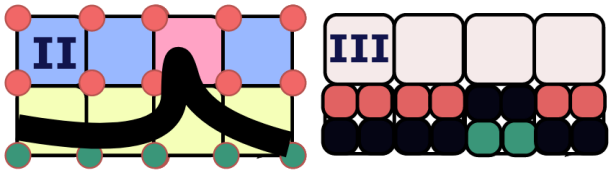

Figure 2: Edge case

During Phase II, in cases where the boundary enters and leaves the partition from the same side, $\delta's$ for all corner points will have the same sign, leading to the partition not being flagged as shown in Figure 2. This in effect is same as applying a mild smoothing to the true boundary. However, increasing the depth of partitions before Phase II would be comparable to a lower degree of smoothing, leading to the obtained boundary being closer to the true boundary.

5 Experimentation

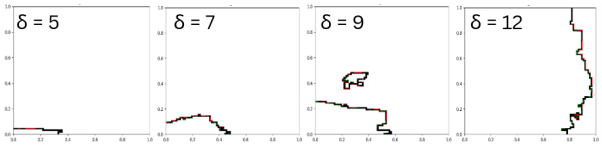

Figure 3: Comparison of results for different δ value.

Figure 3 illustrates the AR boundary for image 69994, classified using FNN-100 with hyperparameters γ, λ, and c, set to 2, 4, and 4, respectively, for different values of δ_{accept}. The boundary is demarcated by red (AR) and green ($\neg AR$) patches, indicating the status of either side. In general, the origin is always AR and crossing boundary swaps the status. The figure shows higher δ_{accept} values lead to a larger AR region. In our subsequent experiments, we selected δ_{accept} as 9.

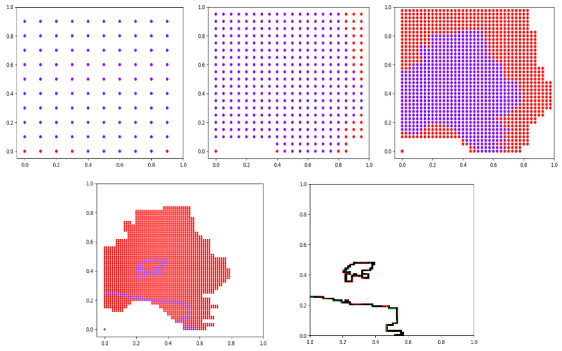

Figure 4: Phases on Image 69994 of FNN-100.

The results presented in Figure 4 is analogous to the toy example described in Section 3. The hyperparameters γ, λ, and c were set to 1, 2, and 2, respectively, to investigate attention robustness in relation to Image 69994 on the FNN-100 dataset.

The top row of Figure 4 illustrates the execution of phase I with depth increasing from left to right. At each stage, only partitions marked with a blue dot are sent to the next stage, whether in the same phase or the next. The second row depicts phase II and phase III output. The figure highlights that in the first iteration of phase I, we eliminate those partitions that clearly cannot contain the boundary. In subsequent iteration(s) of phase I, the level of obviousness is gradually reduced to narrow down the location of the boundary slowly. In phase II, only a small number of partitions are flagged for further analysis, as the selection criteria implemented in this phase guarantees that any flagged partition will necessarily contain the boundary.

The box plot in Figure 5 a) compares the time taken for different models, denoted as "MXYZ" where XYZ refers to the number of neurons in thousand in the FNN. To evaluate the performance, we considered a subset of images, namely 69991 to 69999, for each model. The values for hyperparameters λ and c were initially set to 3, 4, 5, and 9 for M100, M200, M400 and M800 respectively. In case a particular boundary was incomplete, we increased the values of λ and c by 2 and reevaluated until a complete boundary was obtained. However, only 9 out

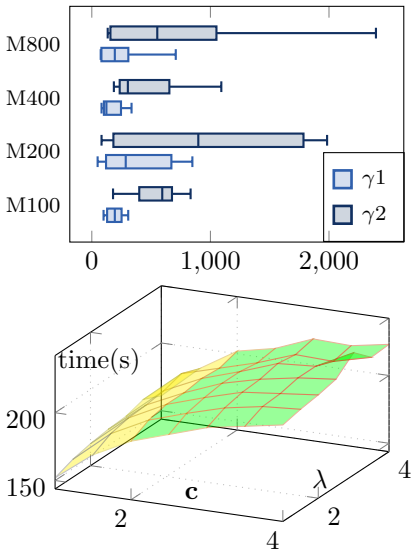

Figure 5: a)time across different model sizes b)hyperparameter with time

of 72 combinations of images, models, and γ required such adjustments, and 7 were needed a second re-evaluation, highlighting that finding suitable hyperparameters is not difficult.

The results demonstrate that the choice of γ significantly affects the computational time. Specifically, $\gamma = 2$ takes more time than $\gamma = 1$ as it involves iterating over phase-I three times instead of two. Moreover, the partition size in phase II and phase III is one-fourth for $\gamma = 2$ as compared to $\gamma = 1$, leading to a higher number of partitions for both phases, thus resulting in increased computation time. Although executing each batch for a phase takes more time for larger models, the number of batches is usually much less due to the small length of the boundary in the case of larger models. Thus, the length of the boundary, which was higher for smaller models, is a significant factor affecting the computational time.

The time taken and behaviour of our method for different values of λ and c for image number 69993 classified by FNN-100 are shown in Figure 5 b). As λ and c increase, more partitions pass through the filter in each phase, resulting in longer computation times. The yellow section of the plot highlights hyperparameter settings that produce overly strict filters, causing some boundary-containing partitions to not pass. In contrast, the green region corresponds to settings with λ and c above a certain threshold that ensures complete boundaries. Despite the same output in the green region, higher computation times are observed for higher λ and c as more unnecessary partitions pass at each stage. Increasing λ and c within the yellow region leads to increasingly complete boundaries.

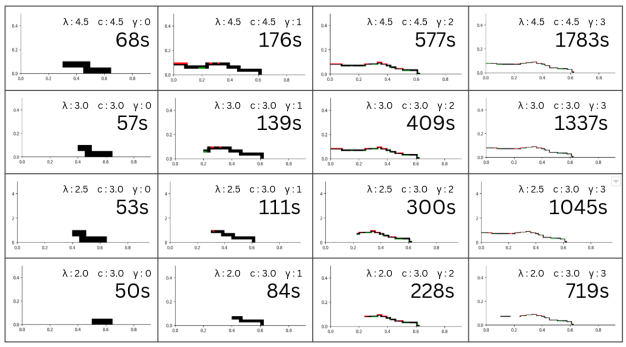

Figure 6: Compare Hyperparameter

The presented results in Figure 6 demonstrate a comparative analysis of output and execution time for image number 69996, classified using FNN-100 under different hyperparameters ($\delta_{accept} = 9$). The parameter γ is incremented from left to right, while λ and c are taken to be equal and incremented from bottom to top. It can be observed from the analysis that the execution time increases rapidly with the increase in the value of γ, whereas it increases gradually with the increase in λ and c. Additionally, the thickness of the partition is dependent on λ. The obtained results reveal that for low values of γ, λ and c, the boundaries are discontinuous or incomplete as the formula for acceptable distance from δ_{accept} is $(\gamma - \gamma_{curr}) * \lambda + c$ thus lower values lead to a few boundary containing partitions to not pass the filter. Now if γ, λ and c are higher than required it a few unnecessary partitions will pass the filter increasing the no of batched in the next phase.

Moreover, to illustrate the status of each side of the boundary, a few AR and $\neg AR$ blocks in green and red are drawn across the black boundary. Based

on the Figure 6, suitable choices of hyper-parameters could be ($\lambda = 4.5$, $c = 4.5$, $\gamma = 1$) taking 176 second or ($\lambda = 3.0$, $c = 3.0$, $\gamma = 2$) taking 409 seconds.

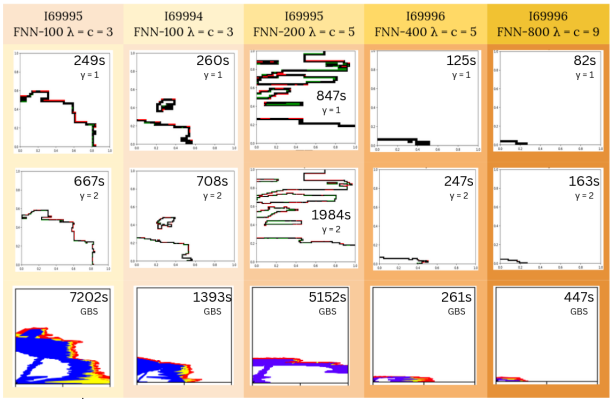

Figure 7: Comparison with GBS

Figure 7 showcases a comparison between our proposed method and GBS [**?**] over different images and models. For GBS we took the delta range to be 0.2 like in their paper. Notably, our method yields identical boundaries to GBS while also reporting secondary boundaries, providing valuable information on the AR status across the input space. Our approach showcases a substantial speed advantage over GBS for $\gamma = 1$ and even for $\gamma = 2$ it is several times faster, in large part due to our use of GPUs and abstracting activation patterns. Moreover, our method scales efficiently with larger models. For larger models the analysis usually quits with 100 per cent RAM usage, which is not the case for ours.

Figure 7 (Obi: This is a rough figure, will soon make a proper LaTeXversion) showcases a comparative analysis of the execution time taken by our proposed methodology and GBS, across various models and images. The raw data utilized to derive Figure 5 a) is presented in the table, with identical hyperparameters. The acronym 'TO' denotes out of Random Access Memory (RAM), and 'Inc.' implies that GBS could not generate the complete boundary but only a few points on it (Obi: Will rerun the search with greater delta range). We observe that our approach never resulted in a 'TO' scenario, and employing $\gamma = 2$ yielded only a marginal improvement in perfor-

Model	Image	Our (γ=1)	Our (γ=2)	GBS
FNN-100	69990	108.95	547.03	3162
FNN-100	69991	162.67	455.88	4788
FNN-100	69992	99.46	231.76	254
FNN-100	69993	211.26	647.35	TO
FNN-100	69994	248.61	666.21	4440
FNN-100	69995	259.67	708.3	7202
FNN-100	69996	177.01	471.15	149
FNN-100	69997	224.83	641.85	3521
FNN-100	69998	307.5	833.32	69991
FNN-100	69999	135.89	179.3	115
FNN-200	69990	48.62	179.79	1982
FNN-200	69991	666.74	1850.89	TO
FNN-200	69992	148.71	183.15	200
FNN-200	69993	684.39	1762.76	TO
FNN-200	69994	391.58	1246.98	1393
FNN-200	69995	847.58	1984.06	5152
FNN-200	69996	181.39	548.58	454
FNN-200	69997	142.06	407.91	284
FNN-200	69998	477.36	1438.21	TO
FNN-200	69999	52.57	82.33	29
FNN-400	69990	120.53	276.16	2281
FNN-400	69991	335.14	1092.37	TO
FNN-400	69992	81.92	185.79	326
FNN-400	69993	239.99	644.48	TO
FNN-400	69994	97.67	197.81	TO
FNN-400	69995	262.03	684.3	TO
FNN-400	69996	121.72	247.5	261
FNN-400	69997	125.02	332.07	303
FNN-400	69998	103.08	266.5	131
FNN-400	69999	154.71	460.66	397
FNN-800	69990	708.33	2395.78	TO
FNN-800	69991	431.02	1853.83	Inc
FNN-800	69992	148.2	459.23	563
FNN-800	69993	246.81	644.37	Inc
FNN-800	69994	79.47	134.02	Inc
FNN-800	69995	266.53	782.93	127
FNN-800	69996	82.07	163.64	447
FNN-800	69997	137.68	386.15	Inc
FNN-800	69998	75.05	137.44	162
FNN-800	69999	239.27	713.82	TO

Figure 8: Comparison with GBS on all

mance, whereas using $\gamma = 1$ significantly enhanced the speed of execution. The highlighted records correspond to those employed in generating Figure 7 (Obi: Please, suggest if we should present a different set of examples in Figure 7 based on figure 8)