LOGICAL ACTIVATION FUNCTIONS: LOGIT-SPACE EQUIVALENTS OF BOOLEAN OPERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neuronal representations within artificial neural networks are commonly understood as logits, representing the log-odds score of presence (versus absence) of features within the stimulus. Under this interpretation, we can derive the probability $P(x_0 \wedge x_1)$ that a pair of independent features are both present in the stimulus from their logits. By converting the resulting probability back into a logit, we obtain a logit-space equivalent of the AND operation. However, since this function involves taking multiple exponents and logarithms, it is not well suited to be directly used within neural networks. We thus constructed an efficient approximation named AND_{AIL} (the AND operator Approximate for Independent Logits) utilizing only comparison and addition operations, which can be deployed as an activation function in neural networks. Like MaxOut, AND_{AIL} is a generalization of ReLU to two-dimensions. Additionally, we constructed efficient approximations of the logit-space equivalents to the OR and XNOR operators. We deployed these new activation functions, both in isolation and in conjunction, and demonstrated their effectiveness on a variety of tasks including image classification, transfer learning, abstract reasoning, and compositional zero-shot learning.

1 INTRODUCTION

An activation function is a non-linearity which is interlaced between linear layers within an artificial neural network. The non-linearity is essential in order for higher-order representations to form, since otherwise the network would be degeneratively equivalent to a single linear layer. Early artificial neural networks were inspired by biological neural networks, which featured either logistic-sigmoid or tanh as their activation function. These activation functions were motivated by the idea that each layer of the network is building another layer of abstraction of the stimulus space from the last layer. Each neuron in a layer identifies whether certain properties or features are present within the stimulus, and the pre-activation (potentiation) value of the neuron indicates a score or logit for the presence of that feature. The sigmoid function, $\sigma(x) = 1/(1+e^{-x})$, was hence a natural choice of activation function, since as with logistic regression, this will convert the logits of features into probabilities.

There is some evidence that this interpretation has merit. Previous work has been done to identify which features neurons are tuned to (examples include LSTM neurons tracking quotation marks, line length, and brackets (Karpathy et al., 2015); LSTM neurons tracking sentiment (Radford et al., 2017); methods for projecting features back to the input space to view them (Olah et al., 2017); and interpretable combinations of neural activities (Olah et al., 2020)). Analogously, within biological neural networks, neurons are tuned to respond more strongly to certain stimuli, and less strongly to others. At high representation levels, concept cells respond predominantly to certain concepts, such one's grandmother, or Jennifer Aniston (Gross, 2002; Quian Quiroga et al., 2005).

Sigmoidal activation functions are no longer commonly used within machine learning between layers of representations, though sigmoid is still widely used for gating operations which scale the magnitude of other features in an attention-like manner. The primary disadvantage of the sigmoid activation function is its vanishing gradient — as the potentiation rises, activity converges to a plateau, and hence the gradient goes to zero. This prevents feedback information propagating back through the network from the loss function to the early layers of the network, which consequently prevents it from learning to complete the task.

The Rectified Linear Unit, $\mathrm{ReLU}(x) = \max(0, x)$, activation function does not have this problem, since in its linear regime it has a gradient of 1. Another advantage of ReLU is it has very low computational demands. Since it is both effective and efficient, it has proven to be a highly choice of popular activation function. The chief drawback to ReLU is it has no sensitivity to changes across half of its input domain, and on average passes no gradient back to its inputs. This can lead to neurons dying[1] if their weights make them never reach their activation threshold.

If sigmoid is equivalent to converting our feature logits into probabilities, then the ReLU activation function is equivalent to truncating our logits and denying any evidence for the absence of a feature. Variants of ReLU have emerged, aiming to smooth out its transition between domains and provide a gradient its inactive regime (Clevert et al., 2015; Barron, 2017; Klambauer et al., 2017; Hendrycks & Gimpel, 2016; Elfwing et al., 2017; Ramachandran et al., 2017; Misra, 2019) However, all these activation functions still bear the general shape of ReLU.

In this work we set out to develop activation functions based on the principle that neurons encode logits, scores that represent the presence of features in the log-odds space. In Section 2 we derive and define these functions in detail for different logical operators, and then consider their performance on numerous task types including parity (Section 3.1), image classification (Sections 3.4 and 3.5), transfer learning (Section 3.6), abstract reasoning (Section 3.7), soft-rule guided classification as exemplified by the Bach chorale dataset (Section 3.3), and compositional zero-shot learning (Section 3.8).

2 Derivation

Manipulation of probabilities in logit-space is known to be more efficient for some calculations. For instance, the log-odds form of Bayes' Rule states that the posterior logit equals the prior logit plus the the log of the likelihood ratio for the new evidence (the log of the Bayes factor).Thus, working in logit-space allows us to perform Bayesian updates on many sources of evidence simultaneously, merely by summing together the log-likelihood ratios for the evidence. A weighted sum may be used if the amount of credence given to the sources differs — and this is precisely the operation performed by a linear layer in a neural network.

When considering sets of probabilities, a natural choice of operation to perform is measuring the joint probability of two events both occurring — the AND operation for probabilities. If we suppose our input space is $x \in [0, 1]^2$, and the goal is to output $y > 0$ if $x_i = 1 \, \forall \, i$, and $y < 0$ otherwise, using model with a weight vector w and bias term b, such that $y = w^T \dot{x} + b$. This can be trivially solved with the weight matrix $w = [1, 1]$ and bias term $b = -1.5$. However, since this is only a linear separator, the solution can not generalise to the case $y > 0$ iff $x_i > 0 \, \forall \, i$.

Similarly, let us consider how the OR function solved with a linear layer. Our goal is to output $y > 0$ if $\exists \, x_i = 1$, and $y < 0$ otherwise. The binary case can be trivially solved with the weight matrix $w = [1, 1]$ and bias term $b = -0.5$. The difference between this and the solution for AND is only an offset to our bias term. In each case, if the input space is expanded beyond binary to $\mathbb{R}^2$, the output can be violated by changing only one of the arguments.

2.1 AND

Suppose we are given x and y as the logits for the presence (vs absence) of two events, X and Y. These logits have equivalent probability values, which can be obtained using the sigmoid function, $\sigma(u) = (1 + e^{-u})^{-1}$. Let us assume that the events X and Y are independent of each other. In this case, the probability of both events occurring (the joint probability) is $\mathrm{P}(X, Y) = \mathrm{P}(X \wedge Y) = \mathrm{P}(X) \; \mathrm{P}(Y) = \sigma(x) \, \sigma(y)$.

[1] Though this problem is very rare when using BatchNorm to stabilise feature distributions.

However, we wish to remain in logit-space, and must determine the logit of the joint probability, $\text{logit}(\text{P}(X, Y))$. This is given by

$$\begin{aligned}\text{AND}_{\text{IL}} := \text{logit}(\text{P}(X \wedge Y)_{x \perp\!\!\!\perp y}) &= \log\left(\frac{p}{1-p}\right), \text{ where } p = \sigma(x)\,\sigma(y), \\ &= \log\left(\frac{\sigma(x)\,\sigma(y)}{1-\sigma(x)\,\sigma(y)}\right), \end{aligned} \tag{1}$$

which we coin as AND_{IL}, the AND operator for independent logits. This 2d function is illustrated as a contour plot (Figure 1, left). Across the plane, the order of magnitude of the output is the same as at least one of the two inputs, scaling approximately linearly.

The approximately linear behaviour of the function is suitable for use as an activation function (no vanishing gradient), however taking exponents and logs scales poorly from a computational perspective. Hence, we developed a computationally efficient approximation as follows. Observe that we can loosely approximate AND_{IL} with the minimum function (Figure 1, right panel). This is equivalent to assuming the probability of **both** X and Y being true equals the probability of the **least likely** of X and Y being true — a naïve approximation which holds well in three quadrants of the plane, but overestimates the probability when both X and Y are unlikely. In this quadrant, when both X and Y are both unlikely, a better approximation for AND_{IL} is the sum of their logits.

We thus propose AND_{AIL}, a linear-approximate AND function for independent logits.

$$\text{AND}_{\text{AIL}}(x, y) := \begin{cases} x + y, & x < 0,\ y < 0 \\ \min(x, y), & \text{otherwise} \end{cases} \tag{2}$$

As shown in Figure 1 (left, middle), we observe that their output values and shape are very similar.

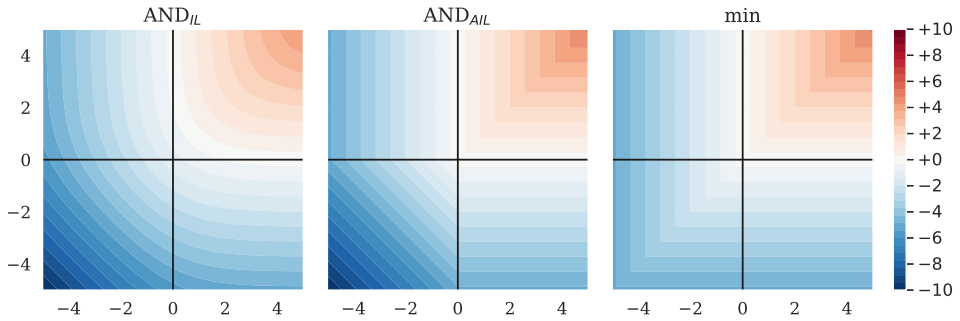


Figure 1: Heatmap comparing the outputs for the exact logit-space probabilistic-and function for independent logits, $\text{AND}_{\text{IL}}(x, y)$; our constructed approximation, $\text{AND}_{\text{AIL}}(x, y)$; and $\max(x, y)$.

2.2 OR

Similarly, we can construct the logit-space OR function, for independent logits. For a pair of logits x and y, the probability that either of the corresponding events is true is given by $p = 1 - \sigma(-x)\,\sigma(-y)$. This can be converted into a logit as

$$\text{OR}_{\text{IL}}(x, y) := \text{logit}(\text{P}(X \vee Y)_{x \perp\!\!\!\perp y}) = \log\left(\frac{p}{1-p}\right), \text{ where } p = 1 - \sigma(-x)\,\sigma(-y) \tag{3}$$

which can be roughly approximated by the max function. This is equivalent to setting the probability of **either** of event X or Y occurring to be equal to the probability of the **most likely** event. This underestimates the upper-right quadrant (below), which we can approximate better as the sum of the input logits, yielding

$$\text{OR}_{\text{AIL}}(x, y) := \begin{cases} x + y, & x > 0,\ y > 0 \\ \max(x, y), & \text{otherwise} \end{cases} \tag{4}$$

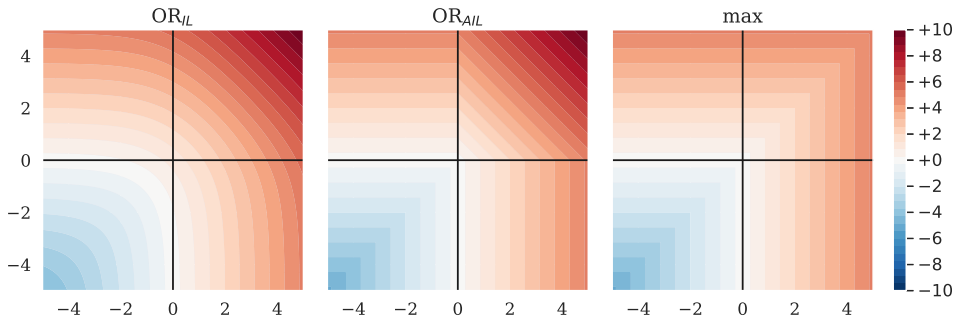


Figure 2: Comparison of the exact logit-space probabilistic-or function for independent logits, $\mathrm{OR}_{\mathrm{IL}}(x, y)$; our constructed approximation, $\mathrm{OR}_{\mathrm{AIL}}(x, y)$; and $\max(x, y)$.

2.3 XNOR

We also consider the construction of a logit-space XNOR operator. This is the probability that X and Y occur either together, or not at all, given by

$$\mathrm{XNOR}_{\mathrm{IL}}(x, y) := \mathrm{logit}(\mathrm{P}(X \bar{\oplus} Y)_{x \perp\!\!\!\perp y}) = \log\left(\frac{p}{1-p}\right), \tag{5}$$

where $p = \sigma(x)\,\sigma(y) + \sigma(-x)\,\sigma(-y)$. We can approximate this with

$$\mathrm{XNOR}_{\mathrm{AIL}}(x, y) := \mathrm{sgn}(xy)\min(|x|, |y|), \tag{6}$$

which focuses on the logit of the feature **most likely** to **flip** the expected **parity** (Figure 3).

We could use other approximations, such as the sign-preserving geometric mean,

$$\mathrm{signed_geomean}(x, y) := \mathrm{sgn}(xy)\sqrt{|xy|}, \tag{7}$$

but note that the gradient of this is divergent, both along $x = 0$ and along $y = 0$.

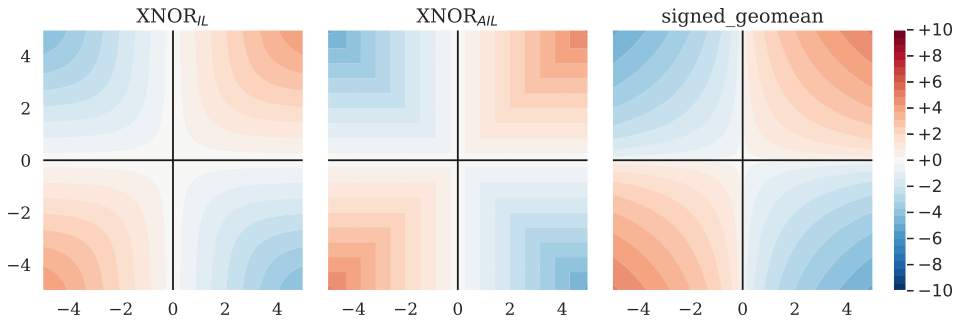


Figure 3: Comparison of the exact logit-space probabilistic-xnor function for independent logits, $\mathrm{XNOR}_{\mathrm{IL}}(x, y)$; our constructed approximation, $\mathrm{XNOR}_{\mathrm{AIL}}(x, y)$; and $\mathrm{signed_geomean}(x, y)$.

2.4 Discussion

By working via probabilities, and assuming inputs encode independent events, we have derived logit-space equivalents of the boolean logic functions, AND, OR, and XNOR. Since these are computationally demanding to compute repeatedly within a neural network, we have constructed approximations of them: $\mathrm{AND}_{\mathrm{AIL}}$, $\mathrm{OR}_{\mathrm{AIL}}$, and $\mathrm{XNOR}_{\mathrm{AIL}}$. Like ReLU, these involve only comparison, addition, and multiplication operations which are cheap to perform. In fact, $\mathrm{AND}_{\mathrm{AIL}}$ and $\mathrm{OR}_{\mathrm{AIL}}$ are a generalisation of ReLU to an extra dimension, since $\mathrm{OR}_{\mathrm{AIL}}(x, y = 0) = max(x, 0)$.

The majority of activation functions are one-dimensional, $f : \mathbb{R} \to \mathbb{R}$. In contrast to this, our proposed activation functions are all two-dimensional, $f : \mathbb{R}^2 \to \mathbb{R}$. They must be applied to pairs of features

from the embedding space, and will reduce the dimensionality of the input space by a factor of 2. This behaviour is the same as seen in MaxOut networks (Goodfellow et al., 2013) which use $\max$ as their activation function, $\mathrm{MaxOut}(x, y; k) := \max(x, y)$. Similar to MaxOut, our activation functions could be generalised to higher dimensional inputs, $f : \mathbb{R}^k \to \mathbb{R}$, by considering the behaviour of the logit-space AND, OR, XNOR operations with regard to more inputs. For simplicity, we restrict ourselves to considering only $k = 2$, but note that these activation functions are generalisable to higher dimensions. On the understanding that Goodfellow et al. (2013) found $k = 2$ yielded the best results for MaxOut networks, we anticipate the same will hold for the AIL activation functions.

2.5 Ensembling

By using multiple logit-boolean activation functions **simultaneously** alongside each other, we permit the network multiple options of how to relate features together. When combining the activation functions, we considered two strategies.

In the partition (p) strategy, we split the the n_c dimensional preactivation embedding equally into m partitions, apply different activation functions on each partition, and concatenate the results together. Using AIL activation functions under this strategy, the output dimension will always half that of the input, as it is for each AIL activation function individually.

In the duplication (d) strategy, we apply m different activation functions in parallel to the same n_c elements. The output is consequently larger, with dimension $m\, n_c$. If desired, we can counteract the $2 \to 1$ reduction of AIL activation functions by using two of them together under this strategy.

Utilising $\mathrm{AND}_{\mathrm{AIL}}$, $\mathrm{OR}_{\mathrm{AIL}}$ and $\mathrm{XNOR}_{\mathrm{AIL}}$ simultaneously allows our networks to access logit-space equivalent of 12 of the 16 boolean logical operations with only a single sign inversion (in either one of the inputs or the output). Including the bias term and skip connections, the network has easy access to logit-space equivalents of all 16 boolean logical operations.

3 Experiments

We evaluated the performance of our AIL activation functions, both individually and together in an ensemble, on a range of benchmarking tasks. Since $\mathrm{AND}_{\mathrm{AIL}}$ and $\mathrm{OR}_{\mathrm{AIL}}$ are equivalent when the sign of operands and outputs can be freely chosen, we evaluate on $\mathrm{OR}_{\mathrm{AIL}}$.

We compared the AIL activation functions against three baselines: (1) ReLU, (2) $\max(x, y) = \mathrm{MaxOut}([x, y]; k = 2)$, and (3) the concatenation of $\max(x, y)$ and $\min(x, y)$, which is analogous to the concatenation of $\mathrm{OR}_{\mathrm{AIL}}$ and $\mathrm{AND}_{\mathrm{AIL}}$ under our duplication strategy. This paired activation function is equivalent to $\mathrm{GroupSort}$ with a group size of 2 (Anil et al., 2019; Chernodub & Nowicki, 2017), and is sometimes referred to as the MaxMin operator.

3.1 Parity

In a simple initial experiment, we constructed a synthetic dataset whose labels could be derived directly using the logical operation XNOR. Each sample in this dataset consisted of four input logit values, with a label that was derived by converting each logits to probability space, rounding to the nearest integer, and taking the parity over this set of binary digits (i.e. true when we have an even number of activated bits, false otherwise).

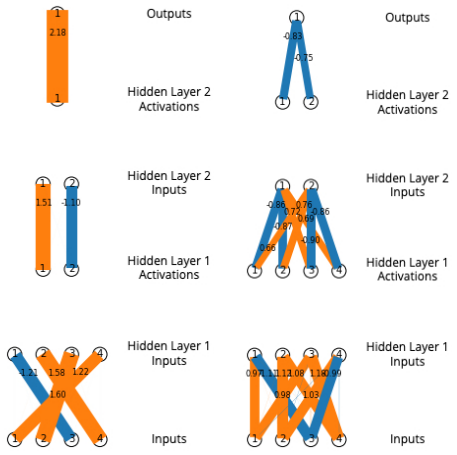
A very small MLP model with two hidden layers (the first with four neurons, the second with two neurons) should be capable of perfect classification accuracy on this dataset with a sparse weight matrix by learning to extract pairwise binary relationships between inputs using $\mathrm{XNOR}_{\mathrm{AIL}}$.

Indeed, after training for 100 epochs using Adam optimizer with default hyper-parameters, 1×10^{-4} weight decay, and one-cycle learning rate schedule with maximum learning rate of 0.01, our simple MLP in conjunction with $\mathrm{XNOR}_{\mathrm{AIL}}$ learned a sparse weight matrix able to perfectly classify any input combination. In comparison, an identical network setup using ReLU was only able to produce 60% classification accuracy (though this accuracy could be improved by increasing the width or depth of the network). This experiment provides an example situation where $\mathrm{XNOR}_{\mathrm{AIL}}$ is utilized by a network to directly extract information about the relationships between network inputs.

3.2 MLP ON COVERTYPE

We trained small one, two, and three-layer MLP models on the Covertype dataset (UCI Machine Learning Repository, 1998; Blackard & Dean, 1998). The Covertype dataset is a classification task consisting of 581 012 samples of forest coverage across 7 classes. Each sample has 54 attributes. We used a fixed random 80:20 train:test split; for training details and accuracy curves, see subsection A.7.

For each activation function, we varied the number of hidden units per layer to investigate how the activation functions affected the performance of the networks as its capacity changed. We did not use weight decay or data augmentation for this experiment, and so the network exhibits clear overfitting with larger architectures. $\mathrm{XNOR}_{\mathrm{AIL}}$ performs significantly best on Covertype ($p < 10^{-5}$) for MLP with 2 or 3 layers, followed by networks which include $\mathrm{XNOR}_{\mathrm{AIL}}$ alongside with other activations, and signed_geomean. The duplication strategy outperforms partition.



(a) $\mathrm{XNOR}_{\mathrm{AIL}}$ (b) ReLU

Figure 4: Training results, binary classification task with parity labels

3.3 MLP ON BACH CHORALES AND LOGIT INDEPENDENCE

The Bach Chorale dataset (Boulanger-Lewandowski et al., 2012) consists of 382 chorales composed by JS Bach, each ∼12 measures long, totalling approximately 83,000 notes. Represented as discrete sequences of tokens, it has served as a benchmark for music processing for decades, from heuristic methods to HMMs, RNNs, and CNNs (Mozer, 1990; Hild et al., 1992; Allan & Williams, 2005; Liang, 2016; Hadjeres et al., 2017; Huang et al., 2019). The chorales are comprised of 4 voices (melodic lines) whose behaviour is guided by soft musical rules that depend on the prior movement of that voice as well as the movement of the other voices. An example of one such rule is that two voices a fifth apart ought not to move in parallel with one another. The task we choose here is to determine whether a given short four-part musical excerpt is taken from a Bach chorale or not. During training, we stochastically corrupt chorale excerpts to provide negative examples (see Appendix A.8). We trained 2-layer MLP discriminators with a single sigmoid output and exchanged all other activation functions (summarized in Figure 5). We found that $\{\mathrm{OR}_{\mathrm{AIL}}, \mathrm{AND}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)$\}$ performed best, but that overall the results were comparable ($p<0.1$ between best and worst).

We furthermore investigated the independence between logits in the trained preactivation embeddings. We would expect that an MLP which is optimally engaging its neurons would maintain independence between features in order to maximize information. To capture the existence of correlations, we considered taking the cosine similarity between rows of the weight matrix, which corresponds to the similarity between the 2 preactivations in the same layer. We did two experiments: one where we measured correlations between all pairwise combinations, and another where we just took correlations between adjacent preactivations that would be paired for the logical activation functions. For these experiments we used 2 hidden layers and a width that showed maximal performance for each activation function. The results are summarized in Appendix A.9.

3.4 CNN AND MLP ON MNIST

We trained 2-layer MLP and 6-layer CNN models on MNIST with Adam (Kingma & Ba, 2017), 1-cycle schedule (Smith & Topin, 2017; Smith, 2018), and using hyperparameters tuned through a random search against a validation set comprising of the last 10k images of the training partition.

The MLP used two hidden layers, the widths of which were varied together to evaluate the performance for a range of model sizes. The CNN used six layers of 3x3 convolution layers, with 2x2 max

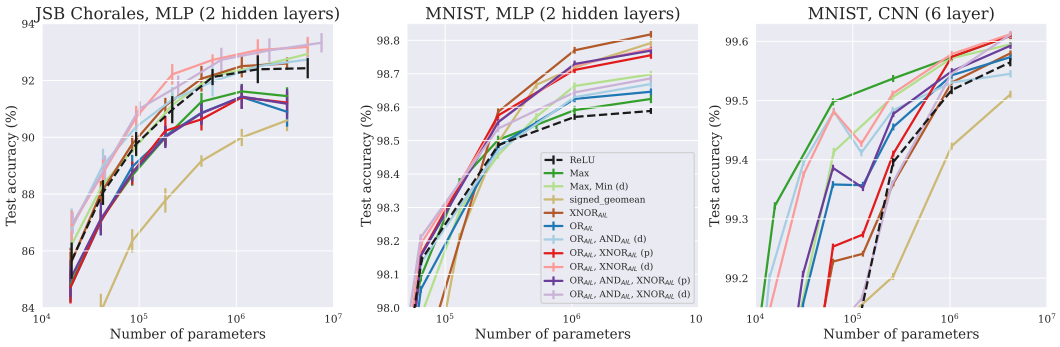


Figure 5: We trained 2-layer MLPs discriminators on JSB Chorales using Adam (LR 1×10^{-3}, 150 ep.), Mean (bars: std dev) of $n = 10$ weight inits. Additionally, we trained CNN on MNIST, MLP on flattened-MNIST, using Adam (1-cycle, 10 ep), hyperparams determined by random search. Mean (bars: std dev) of $n = 40$ weight inits.

pooling (stride 2) after every other conv layer. The output was flattened and followed by three MLP layers. The widths of the layers were scaled up approximately linearly to explore a range of model sizes (see subsection A.10 for more details).

For the MLP, $\mathrm{XNOR}_{\mathrm{AIL}}$ performed best along with $\mathrm{signed_geomean}$ ($p < 0.1$), ahead of all other activations ($p < 0.01$). With the CNN, four activation function configurations performed equally well: $\{\mathrm{OR}_{\mathrm{AIL}}, \mathrm{AND}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)$\}$, $\{\mathrm{OR}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d/p)$\}$ and Max. Additionally, note that CNNs that used $\mathrm{OR}_{\mathrm{AIL}}$ or Max (alone or in an ensemble) maintained high performance for an order of magnitude fewer parameters (3×10^4 params) than networks which did not (3×10^5 params).

3.5 RESNET50 ON CIFAR-10/100

We explored the impact of our AIL activation functions on the performance of deep networks by deploying them within a pre-activation ResNet50 model (He et al., 2015; 2016). We exchanged all ReLU activation functions in the network to a candidate activation function while maintaining the size of pass-through embedding. We experimented with changing the width of the network, scaling up the embedding space and all hidden layers by a common factor. The network was trained on CIFAR-10 (or -100) for 100 epochs using Adam (Kingma & Ba, 2017), 1-cycle Smith (2018); Smith & Topin (2017). Hyperparameters were optimized through a random search against a validation partition of CIFAR-100 for a fixed width factor $w = 2$ only. The same hyperparameters were reused for experiments of changing width, and for CIFAR-10. We used width factors of 0.5, 1, 2, and 4 for $1 \rightarrow 1$ activation functions (ReLU, etc), widths of 0.75, 1.5, 3, and 6 for $2 \rightarrow 1$ activation functions (Max, etc), and widths of 0.4, 0.75, 1.5, and 3 for $\{\mathrm{OR}_{\mathrm{AIL}}, \mathrm{AND}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)$\}$.

For both CIFAR-10 and 100, we find that $\mathrm{OR}_{\mathrm{AIL}}$ and Max outperform ReLU across a wide range of width values. These two activation functions hold up their performance best as the number of parameters is reduced. Meanwhile, other AIL activation functions perform similarly to ReLU when the width is thin, and slightly worse than ReLU when the width is wide. When used on its own and not part of an ensemble, the $\mathrm{XNOR}_{\mathrm{AIL}}$ activation function performed very poorly (off the bottom of the chart), indicating it is not well suited for this task.

3.6 TRANSFER LEARNING

We considered the task of transfer learning on several image classification datasets. We used a ResNet18 model (He et al., 2015) pretrained on ImageNet-1k. The weights were frozen (not fine-tuned) and used to generate embeddings of samples from other image datasets. We trained a two-layer MLP to classify the images, using various activation functions. To make the number of parameters similar, we used a width of 512 for activation functions with $1 \rightarrow 1$ mapping (e.g. ReLU) and a width of 720 for activation functions with a $2 \rightarrow 1$ mapping (e.g. Max, $\mathrm{OR}_{\mathrm{AIL}}$). See subsection A.12 for more details on the training process.

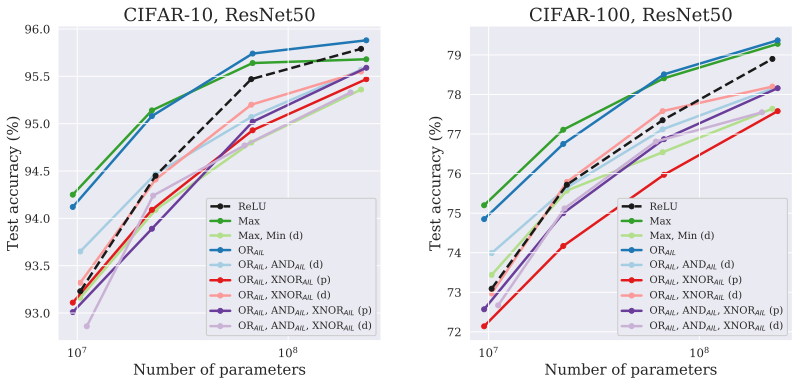


Figure 6: We trained ResNet50 on CIFAR-10/100 for 100 ep. using Adam. The same activation function (or ensemble) was used through the network. The width was varied to explore a range of network sizes (see text).

Table 1: Transfer learning with from a frozen ResNet-18 architecture pre-trained on ImageNet-1k to other computer vision datasets. An MLP head with two layers of pre-activation width either 512 or 720 (depending on activation function, to keep the number of params around 700k) was trained, without re-training the pretrained base model Trained with SGD, cosine annealed lr 0.01, for 25 epochs. Mean (standard error) of $n=5$ inits of the MLP (same pretrained network).

		Test Accuracy (%)						
Activation function	Params	Caltech101	CIFAR10	CIFAR100	Flowers	Cars	STL-10	SVHN
Linear layer only	5k–101k	**88.33**±0.19	78.56±0.09	57.39±0.09	92.50±0.11	33.51±0.06	94.69±0.03	45.42±0.06
ReLU	530k–626k	86.72±0.13	81.63±0.05	58.04±0.11	90.76±0.33	30.97±0.26	94.61±0.08	51.39±0.06
Max	633k–700k	86.87±0.10	81.93±0.11	58.76±0.13	91.76±0.26	33.69±0.10	94.66±0.04	51.62±0.10
Max, Min (d)	530k–626k	87.31±0.13	82.31±0.10	59.05±0.10	91.68±0.18	34.91±0.12	94.64±0.04	51.72±0.04
$\mathrm{OR}_{\mathrm{AIL}}$	633k–700k	87.17±0.16	82.07±0.17	59.28±0.09	92.49±0.12	36.80±0.17	94.69±0.03	51.66±0.09
$\mathrm{XNOR}_{\mathrm{AIL}}$	633k–700k	86.87±0.11	82.03±0.09	58.51±0.03	91.13±0.16	32.96±0.12	**94.70**±0.03	51.96±0.06
$\mathrm{OR}_{\mathrm{AIL}}, \mathrm{AND}_{\mathrm{AIL}}$ (d)	530k–626k	**87.41**±0.14	82.38±0.06	59.90±0.08	92.10±0.20	37.16±0.15	94.52±0.05	52.11±0.09
$\mathrm{OR}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)	633k–700k	87.17±0.16	82.06±0.14	59.25±0.06	92.51±0.12	36.76±0.16	94.67±0.02	51.73±0.07
$\mathrm{OR}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)	530k–626k	87.04±0.27	82.20±0.04	59.44±0.07	91.90±0.10	36.88±0.10	94.69±0.06	52.02±0.16
$\mathrm{OR}_{\mathrm{AIL}}, \mathrm{AND}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)	633k–700k	87.12±0.17	82.08±0.12	59.10±0.12	**92.56**±0.18	36.65±0.14	94.65±0.03	51.85±0.09
$\mathrm{OR}_{\mathrm{AIL}}, \mathrm{AND}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)	664k–807k	87.26±0.15	**82.45**±0.08	**60.20**±0.11	92.41±0.17	**37.89**±0.14	94.65±0.04	**52.19**±0.09

3.7 Abstract Reasoning

Abstract reasoning is challenging for neural networks to learn because their structure and objective function are more effective for tasks which come instinctively to humans ("System 1" of Kahneman, 2011), such as object recognition, as opposed to demanding ("System 2") logic tasks.

Raven's Progressive Matrices (Raven & Court, 1938) are a long-standing IQ test, which have been emulated by Barrett et al. (2018) with Procedurally Generated Matrices (PGM), and then the RAVEN task by (Zhang et al., 2019). Recently Hu et al. (2020) and Benny et al. (2021) have improved on that task with I-RAVEN and RAVEN-FAIR, respectively, both aiming to make the task more balanced.

We considered the application of AIL activation functions in the context of the I-RAVEN task, by adapting the Stratified Rule-Aware Network (SRAN) of Hu et al. (2020) to include our AIL activation functions. We first added LayerNorm to the network, and then swapped out the seven ReLU activations in the gating module. The architecture for the three ResNet-18 (He et al., 2015) base models were unchanged. Where necessary, the number of units per layer was modified to facilitate the change in dimensionality caused by our activation function. The networks were trained using the same procedure as described by Hu et al. (2020).

We found the network using $\mathrm{OR}_{\mathrm{AIL}}$ activation function performed best across most of the subtasks. The second best was $\{\mathrm{OR}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)$\}$.

Table 2: Performance of SRAN-based models on the I-RAVEN dataset (Hu et al., 2020).

		I-RAVEN Test Acc (%)							
Activation function	Param	Acc	Cntr	2×2G	3×3G	O-IC	O-IG	L-R	U-D
ReLU, Base SRAN (Hu et al., 2020)	44.0M	60.8	78.2	**50.1**	42.4	68.2	46.3	70.1	70.3
ReLU, SRAN+LayerNorm	45.6M	63.0	84.0	**50.0**	42.5	69.9	48.3	73.5	72.3
Max	44.6M	57.8	76.3	45.2	39.7	65.3	48.7	64.6	64.7
XNOR_{AIL}	44.6M	57.7	74.7	46.0	40.0	66.8	47.7	65.6	63.2
OR_{AIL}	44.6M	**64.3**	**84.4**	49.5	**44.0**	**71.5**	47.1	**76.5**	**77.0**
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}$ (d)	45.6M	57.5	74.7	45.6	41.3	68.3	44.9	64.5	63.0
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	44.6M	59.8	80.5	45.8	41.4	67.2	48.2	67.5	68.1
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)	45.6M	62.9	83.7	49.1	43.3	68.1	**49.5**	73.8	72.2
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	44.6M	55.0	68.2	47.2	41.1	65.0	45.0	61.0	57.5

3.8 Compositional Zero-Shot Learning

Zero-shot learning encompasses all problems which involve completing novel tasks which the subject has never seen before. The subject must infer the solution to the task from their previous experiences. Compositional zero-shot learning is a subset of zero-shot learning which involves combining knowledge about multiple stimulus properties together in novel pairings. For instance, if the network has been trained on "sliced bread," "sliced pear," and "caramelized pear," is it able to classify images of "caramelized bread" despite having never seen an example of this before?

We based our experiments on the Task-driven Modular Networks (TMN) proposed by Purushwalkam et al. (2019). We modified the TMN architecture by adding an additional linear layer to the end of each module (which would otherwise terminate with an activation function; see A.13 for details), and dub the modified version of the network "TMNx." Experiments were performed on the MIT-States dataset (Isola et al., 2015).We trained and tested on the corresponding partitions of the dataset as introduced by Purushwalkam et al. (2019).

Table 3: Performance of TMNx networks at compositional zero-shot learning (CZSL) on the MIT-States dataset. Mean (standard error) of $n=5$ random initializations.

			MIT-States Test AUC (%)		
Activation function	Mapping	Params	Top-1	Top-2	Top-3
ReLU	$1 \rightarrow 1$	1.66M	2.72 ± 0.07	6.84 ± 0.13	10.68 ± 0.22
Max	$2 \rightarrow 1$	1.55M	2.63 ± 0.09	6.80 ± 0.16	10.77 ± 0.18
Max, Min (d)	$1 \rightarrow 1$	1.66M	2.60 ± 0.09	6.79 ± 0.20	10.72 ± 0.29
XNOR_{AIL}	$2 \rightarrow 1$	1.55M	1.80 ± 0.09	4.86 ± 0.16	7.88 ± 0.24
OR_{AIL}	$2 \rightarrow 1$	1.55M	2.61 ± 0.07	6.89 ± 0.15	11.08 ± 0.15
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}$ (d)	$1 \rightarrow 1$	1.66M	2.81 ± 0.09	7.08 ± 0.20	11.14 ± 0.31
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	$2 \rightarrow 1$	1.55M	2.61 ± 0.07	6.89 ± 0.15	11.08 ± 0.15
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)	$1 \rightarrow 1$	1.66M	2.67 ± 0.06	6.75 ± 0.09	10.80 ± 0.12
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	$2 \rightarrow 1$	1.55M	$\mathbf{2.88} \pm 0.04$	$\mathbf{7.26} \pm 0.11$	$\mathbf{11.47} \pm 0.19$
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)	$2 \rightarrow 3$	1.78M	2.73 ± 0.05	7.01 ± 0.14	11.19 ± 0.26

4 Discussion

In this work we motivated and introduced novel activation functions analogous to boolean operators in logit-space. We designed the AIL functions, fast approximates to the true logit-space functions equivalent to manipulating the corresponding probabilities, and demonstrated their effectiveness on a wide range of tasks.

We found that the XNOR_{AIL} activation function was highly effective in the setting of shallow networks. Meanwhile, the OR_{AIL} activation function was highly effective for representation learning in the setting of a deep ResNet architecture trained on images. In scenarios which involve manipulating high-level features extracted by an embedding network, we find that using an ensemble of AIL activation functions together works best.

REFERENCES

Moray Allan and Christopher Williams. Harmonising chorales by probabilistic inference. In L. Saul, Y. Weiss, and L. Bottou (eds.), *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2005.

Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation, 2019.

David G. T. Barrett, Felix Hill, Adam Santoro, Ari S. Morcos, and Timothy Lillicrap. Measuring abstract reasoning in neural networks, 2018.

Jonathan T Barron. Continuously differentiable exponential linear units. *arXiv preprint arXiv:1704.07483*, 2017.

Yaniv Benny, Niv Pekar, and Lior Wolf. Scale-localized abstract reasoning, 2021.

Jock A. Blackard. *Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types*. PhD thesis, USA, 1998. AAI9921979.

Jock A Blackard and Denis J Dean. Comparative accuracies of neural networks and discriminant analysis in predicting forest cover types from cartographic variables. In *Second Southern Forestry GIS Conference*, pp. 189–199, 1998.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

Artem Chernodub and Dimitri Nowicki. Norm-preserving orthogonal permutation linear unit activation functions (oplu), 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018. URL `http://arxiv.org/abs/1805.09501`.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, 2017.

Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In Sanjoy Dasgupta and David McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1319–1327, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `https://proceedings.mlr.press/v28/goodfellow13.html`.

CG Gross. Genealogy of the "grandmother cell. *Neuroscientist*, 8 (5):512—518, 2002. doi: 10.1177/107385802237175.

Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hermann Hild, Johannes Feulner, and Wolfram Menzel. Harmonet: A neural net for harmonizing chorales in the style of JS Bach. In *Advances in neural information processing systems*, pp. 267–274, 1992.

Sheng Hu, Yuqing Ma, Xianglong Liu, Yanlu Wei, and Shihao Bai. Stratified rule-aware network for abstract visual reasoning, 2020.

Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. *arXiv preprint arXiv:1903.07227*, 2019.

Phillip Isola, Joseph J Lim, and Edward H Adelson. Discovering states and transformations in image collections. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1383–1391, 2015.

H M Dipu Kabir, Moloud Abdar, Seyed Mohammad Jafar Jalali, Abbas Khosravi, Amir F Atiya, Saeid Nahavandi, and Dipti Srinivasan. Spinalnet: Deep neural network with gradual input, 2020.

Daniel Kahneman. *Thinking, fast and slow.* Thinking, fast and slow. Farrar, Straus and Giroux, New York, NY, US, 2011. ISBN 0-374-27563-7 (Hardcover); 1-4299-6935-0 (PDF); 978-0-374-27563-1 (Hardcover); 978-1-4299-6935-2 (PDF).

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Proceedings of the 31st international conference on neural information processing systems*, pp. 972–981, 2017.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Feynman Liang. Bachbot: Automatic composition in the style of Bach chorales. *University of Cambridge*, 8:19–48, 2016.

Marvin Minsky and Seymour Papert. *Perceptrons.* Perceptrons. M.I.T. Press, Oxford, England, 1969.

Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 4:2, 2019.

Michael C Mozer. *Connectionist music composition based on melodic, stylistic, and psychophysical constraints*. University of Colorado, Boulder, Department of Computer Science, 1990.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729. IEEE, 2008.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001.

Senthil Purushwalkam, Maximilian Nickel, Abhinav Gupta, and Marc'Aurelio Ranzato. Task-driven modular networks for zero-shot compositional learning, 2019.

R Quian Quiroga, L Reddy, G Kreiman, C Koch, and I Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435:1102–1107, 2005. doi: 10.1038/nature03687.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

John C Raven and JH Court. *Raven's progressive matrices*. Western Psychological Services Los Angeles, CA, 1938.

Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018. URL http://arxiv.org/abs/1803.09820.

Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017. URL http://arxiv.org/abs/1708.07120.

UCI Machine Learning Repository. Covertype data set, Aug 1998. URL https://archive.ics.uci.edu/ml/datasets/Covertype.

Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. Raven: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

A Appendix

A.1 Linear ReLU examples

Here, we show a representation of what a 2-d linear layer followed by the ReLU activation function looks like. The output is the same up to rotation.

If we apply a second linear layer on top of the outputs of the first two units, we can try to approximate the logit AND or OR function. However, the solution using ReLU leaves a quadrant of the output space hollowed out as zero due to its behaviour at truncating away information.

A.2 Solving XOR

A long-standing criticism of artificial neural networks is their inability to solve XOR with a single layer Minsky & Papert (1969). Of course, adding a single hidden layer allows a network using ReLU to solve XOR. However, the way that it solves the problem is to join two of the disconnected regions together in a stripe (see Figure 8). Meanwhile, our $\mathrm{XNOR}_{\mathrm{AIL}}$ is trivial able to solve the XOR problem without any hidden layers. For comparison here, we include one hidden layer with 2 units for each network. Including a layer before the activation function makes the task harder for $\mathrm{XNOR}_{\mathrm{AIL}}$, which must learn how to project the input space in order to compute the desired separation. Also, including the linear layer allows the network to generalise to rotations and offset versions of the task.

A.3 Difference between AIL and IL functions

Here, we measure and show the difference between the true logit-space operations and our AIL approximations.

In each case, we observe that the magnitude of the difference is never more than 1, which occurs along the boundary lines in AIL. Since the magnitude of the three functions increase as we move away from the origin, the relative difference decreases in magnitude as the size of x and y increase.

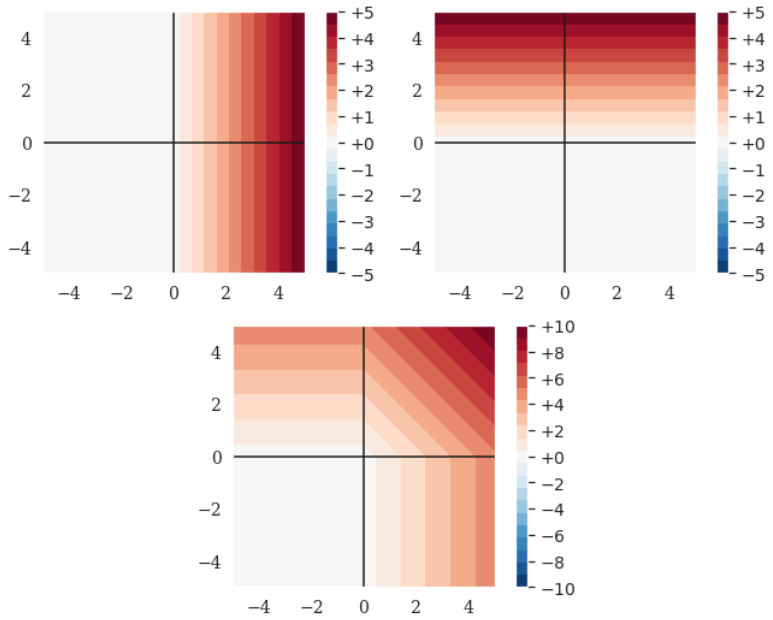
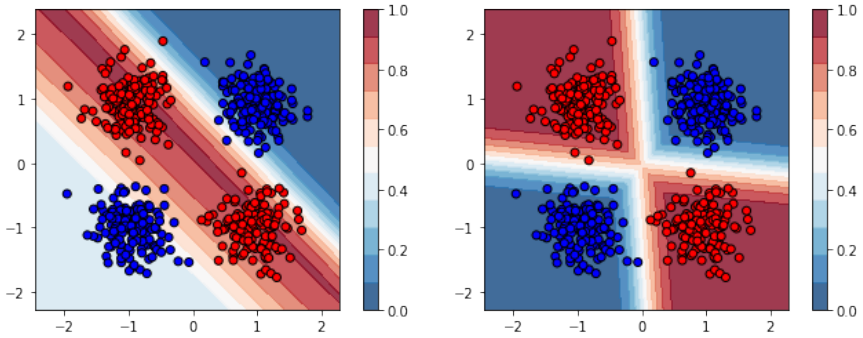


Figure 7: ReLU unit, and ReLU units followed by a linear layer to try to approximate $\mathrm{OR}_{\mathrm{IL}}$, leaving a dead space where the negative logits should be.



(a) ReLU (b) $\mathrm{XNOR}_{\mathrm{AIL}}$

Figure 8: Solving XOR with a single hidden layer of 2 units, using either ReLU or $\mathrm{XNOR}_{\mathrm{AIL}}$ activation. Circles indicate negative (blue) and positive (red) training samples. The heatmaps indicate the output probabilities of the two networks.

A.4 Gradient of AIL and IL functions

Here, we show the gradient of each of the logit-space boolean operators and their AIL approximates. By the symmetry of each of the functions, the derivative with respect to y is a reflected copy of the gradient with respect to x

We find that the gradient of each AIL function closely matches that of the exact form. Whilst there are "dead" regions where the gradient is zero, this only occurs for one of the derivatives at a time (there is always a gradient with respect to at least one of x and y).

A.5 Dataset summary

We used the same splits for Caltech101 as used in Kabir et al. (2020) (see Table 4).

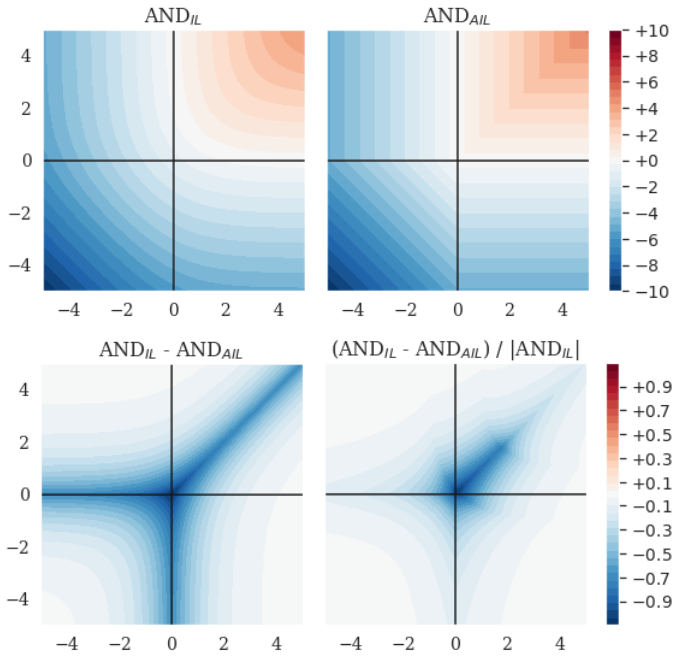


Figure 9: Heatmaps showing XNOR_{IL}, XNOR_{AIL}, their difference, and their relative difference.

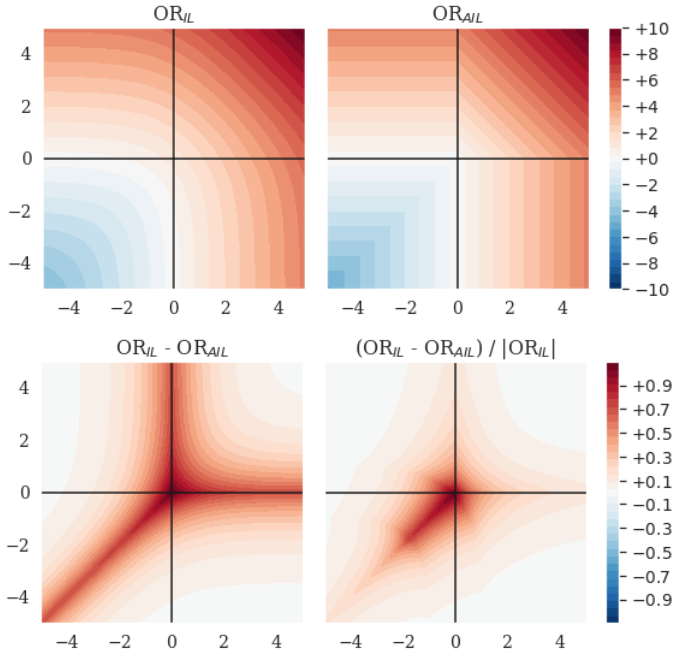


Figure 10: Heatmaps showing XNOR_{IL}, XNOR_{AIL}, their difference, or their relative difference.

A.6 Parity Experiments

We also introduced a second synthetic dataset with a labelling function that, while slightly more complex than the first, was still solvable by applying the logical XNOR operation to the network inputs. In this dataset increased our number of inputs to 8, and derived our labels by applying a set of nested XNOR_{IL} operations:

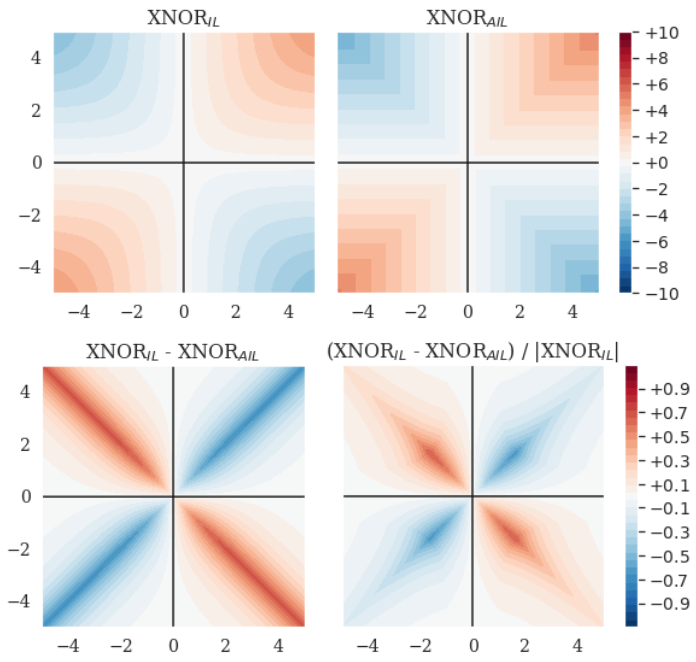


Figure 11: Heatmaps showing $\mathrm{XNOR}_{\mathrm{IL}}$, $\mathrm{XNOR}_{\mathrm{AIL}}$, their difference, xnor their relative difference.

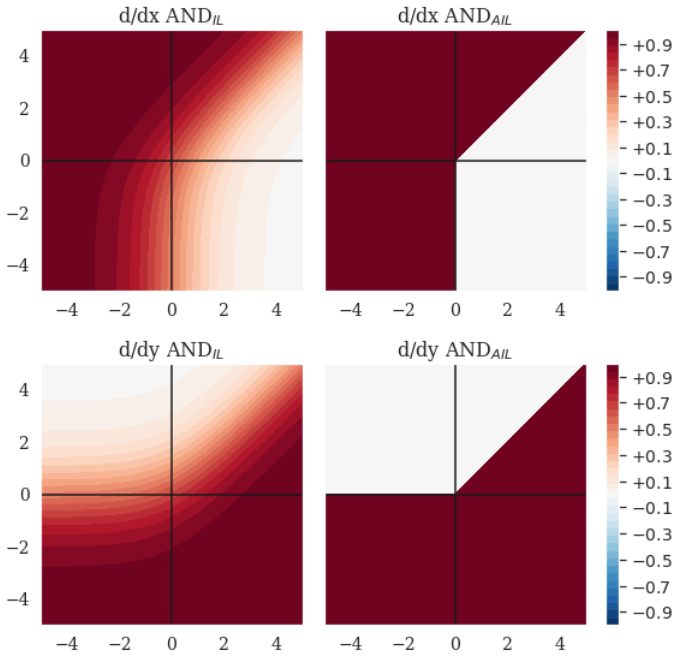


Figure 12: Heatmaps showing the gradient with respect to x and y of $\mathrm{AND}_{\mathrm{IL}}$ and $\mathrm{AND}_{\mathrm{AIL}}$.

$$\begin{aligned}\mathrm{XNOR}_{\mathrm{IL}}(&\mathrm{XNOR}_{\mathrm{IL}}(\mathrm{XNOR}_{\mathrm{IL}}(x_2, x_5), \mathrm{XNOR}_{\mathrm{IL}}(x_3, x_4)),\\ &\mathrm{XNOR}_{\mathrm{IL}}(\mathrm{XNOR}_{\mathrm{IL}}(x_6, x_7), \mathrm{XNOR}_{\mathrm{IL}}(x_0, x_1)).\end{aligned}$$

For this more difficult task we also reformulated our initial experiment into a regression problem, as the continuous targets produced by this labelling function are more informative than the rounded

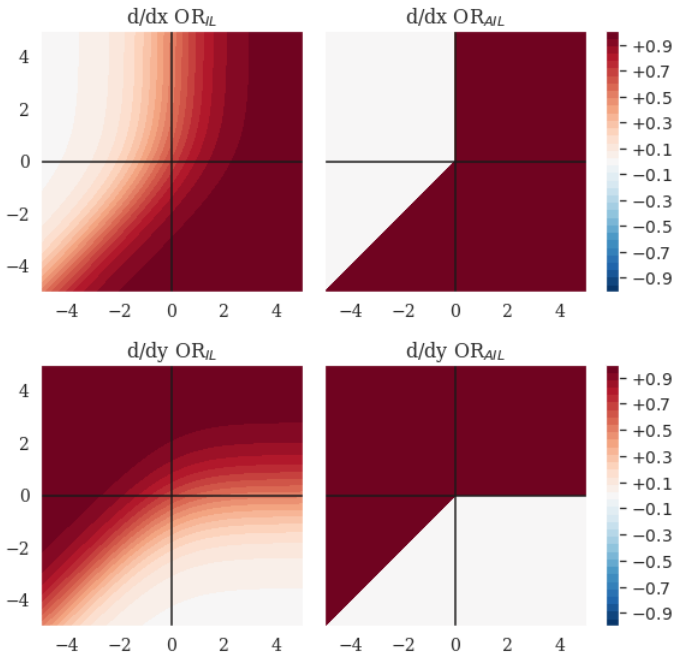


Figure 13: Heatmaps showing the gradient with respect to x and y of $\mathrm{OR}_{\mathrm{IL}}$ or $\mathrm{OR}_{\mathrm{AIL}}$.

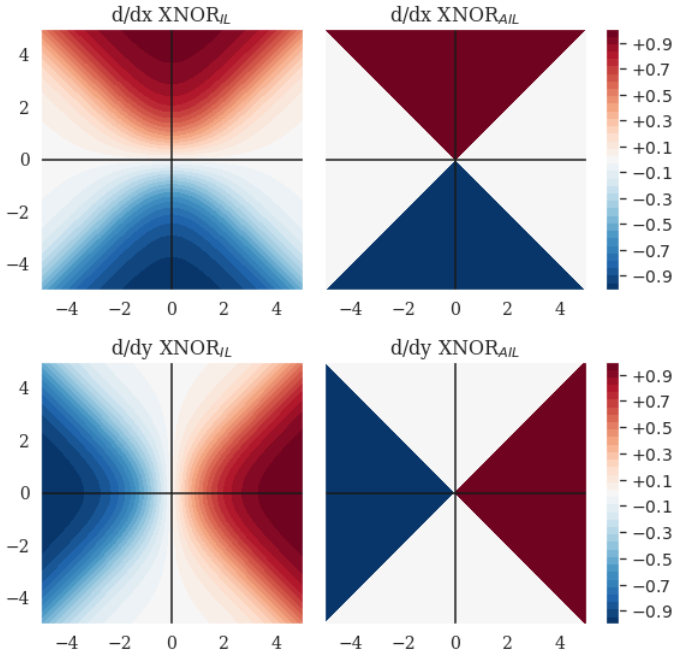


Figure 14: Heatmaps showing the gradient with respect to x and y of $\mathrm{XNOR}_{\mathrm{IL}}$ xnor $\mathrm{XNOR}_{\mathrm{AIL}}$.

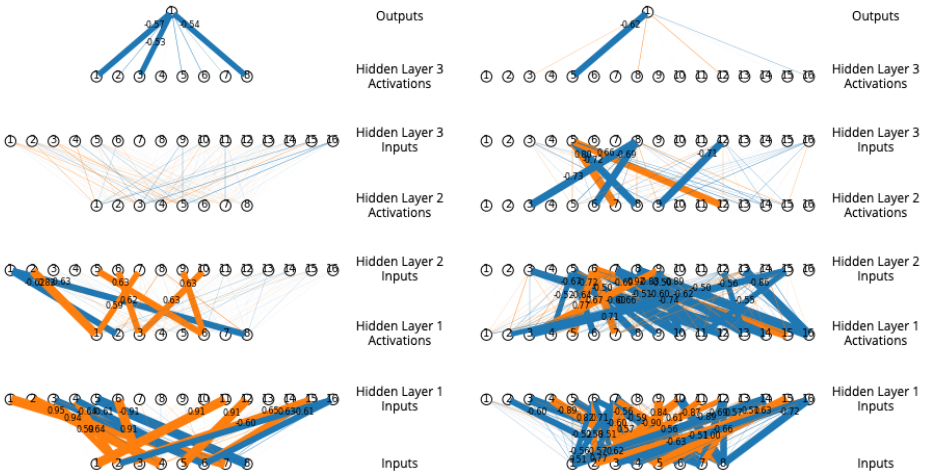
binary targets used in the first experiment. We also adjusted our network setup to have an equal number of neurons at each hidden layer as we found that this significantly improved model performance[2]. We again trained using the same model hyper-parameters for 100 epochs.

[2]We hypothesize that, because our $\mathrm{XNOR}_{\mathrm{AIL}}$ activation function reduces the number of hidden layer neurons by a factor of k, having a reduced number of neurons at each layer creates a bottle neck in the later layers of the network which restricts the amount of information that made its way through to the final layer

Table 4: Classification datasets.

Dataset	Train	Test	Classes	Reference
Bach Chorales	229	77	2	Boulanger-Lewandowski et al. (2012)
Caltech101	6162	1695	101	Fei-Fei et al. (2006)
Stanford Cars	8144	8041	196	Krause et al. (2013)
CIFAR-10	50 000	10 000	10	Krizhevsky (2009)
CIFAR-100	50 000	10 000	100	Krizhevsky (2009)
Covertype	464 810	116 202	7	Blackard (1998); Blackard & Dean (1998)
Oxford Flowers	6552	818	102	Nilsback & Zisserman (2008)
MNIST	60 000	10 000	10	LeCun et al. (1998)
STL-10	5000	8000	10	Coates et al. (2011)
SVHN	73 257	26 032	10	Netzer et al. (2011)

While this time the model was not able to learn a sparse weight matrix that exactly reflected our labelling function (see Figure 15), the model was again able to leverage the XNOR_{AIL} activation function to significantly outperform an identical model utilizing the ReLU activation function.



(a) XNOR_{AIL} learned weight matrix

(b) ReLU learned weight matrix

Figure 15: Training results, regression experiment on second synthetic dataset

We found that a simple model with three hidden layers, each with eight neurons, utilizing XNOR_{AIL} was able to go from a validation RMSE of 0.287 at the beginning of training to a validation RMSE of 0.016 after 100 epochs. Comparatively, an identical model utilizing the ReLU activation function was only able to achieve a validation RMSE of 0.271 after 100 epochs. In order for our ReLU network to match the validation RMSE of our 8-neuron-per-layer XNOR_{AIL} model, we had to increase the model size by 32 times to 256 neurons at each hidden layer.

A.7 Covertype

Our one, two, and three-layer MLPs were trained using 1-cycle (Smith & Topin, 2017; Smith, 2018) for 50 epochs, with a batch size of 1024, without weight decay. We used a subset of 15% of the training data for validation. The learning rate was selected using an automated learning rate finder approach, verified against the validation partition. For our final experiments, the validation partition was included in the training data. No data augmentation was performed.

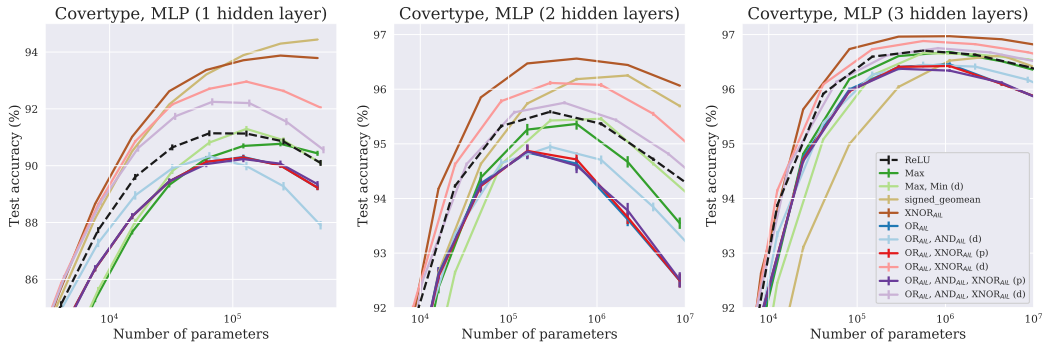


Figure 16: We trained MLPs on the Covertype dataset, with a fixed 80:20 random split. Trained with Adam, 50 ep., 1-cycle, using LRs determined automatically with LR-finder. Mean (bars: std dev) of $n = 10$ weight inits.

A.8 Bach Chorale training details

For each of the 4 voices, we restricted the available pitches to 3 octaves, resulting in 37 one-hot tokens (including silence). Since we only planned on feeding small time-windows of the chorale into our model, for pre-processing, we converted the data into a shape of $(\texttt{seq_len}, 4, 37)$, where we set $\texttt{seq_len}$ to 4.

To generate training examples for the discriminator, we transposed by $\{-5, -4, \ldots, 5, 6\}$ semitones, chosen uniformly at random, and there was a 0.5 probability that the sample was corrupted by the following method:

- Choose 2-3 notes in the $(\texttt{seq_len} \times 4)$ window to be corrupted
- For each note, corrupt by the following mixture distribution:
  - $(p = 0.6)$ Sample a pitch from a Gaussian centered on the existing note, with $\sigma = 3$ semitones, forcing the new pitch to be distinct
  - $(p = 0.2)$ Copy a pitch from the current voice, forcing the new pitch to be distinct
  - $(p = 0.2)$ Extend the previous note in time
  - $(p = 0.1)$ Sample uniformly from all 37 possible tokens

A.9 Correlations between Preactivations

Results on correlations between weights in the JSB Chorale models are shown in Figure 17. We found that when taking all preactivations into account, every activation function generally showed independence between features. Interestingly, the cosine similarities between inputs that were paired together for the bivariate activation functions showed anticorrelation in almost all cases where Max or $\mathrm{OR}_{\mathrm{AIL}}$ were used, and other cases generally showed more correlation than ReLU.

A.10 CNN and MLP on MNIST

In this experiment we trained MLP and CNN models for 10 epochs on the MNIST dataset using Adam optimizer, one cycle learning rate schedule, and cross entropy loss, with batch size of 256. We augmented our training samples using a random affine transformation from: rotation of ± 10 degrees, scale of factor 0.8 to 1.2, translation with max absolute fraction for horizontal and vertical directions of 0.08, and shear parallel to the x-axes of ± 0.3.

The hyper-parameters for the optimizer and scheduler were determined by undergoing a random search of the hyper parameter space, done on a validation set derived from by taking a train / test split on the MNIST training set (the first 50,000 images being used for training, the remaining 10,000 for validation). We ran the search for four iterations, with each iteration sampling 120 different hyper-parameter settings. The bounds for our hyper-parameter samples were initially very wide, and were tightened at each iteration by selecting the top-5 performing hyper-parameter settings, taking

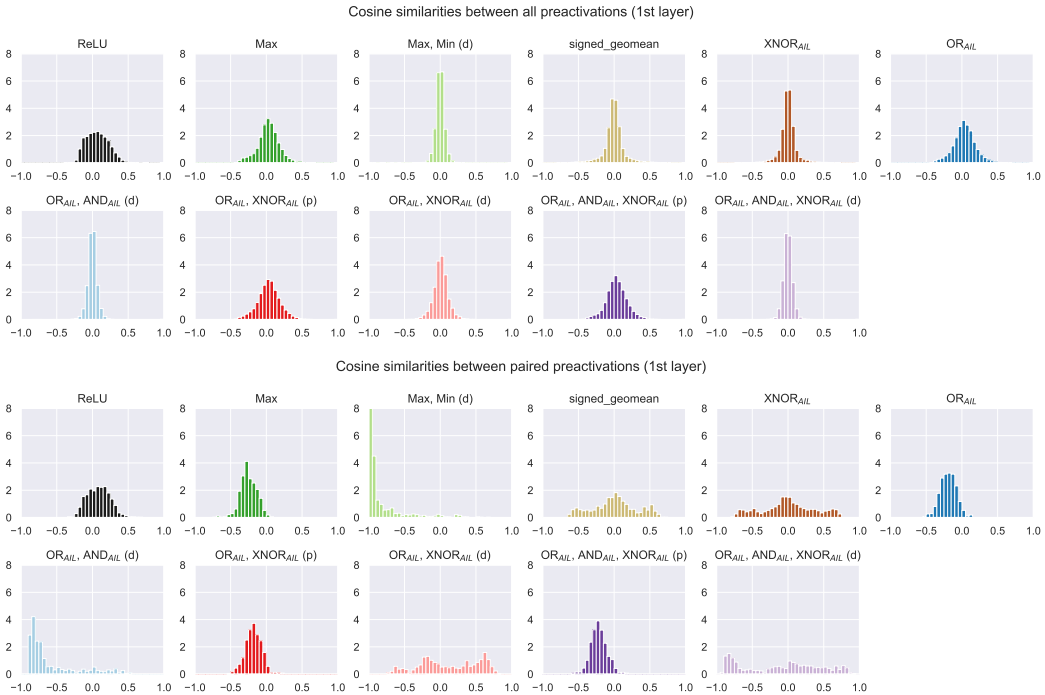


Figure 17: Cosine similarities between preactivation weights of two activation functions in the first layer of an MLP trained on JSB Chorales.

the mean $\bar{x}$ and weighted standard deviation σ across those settings, and re-setting the bounds for the next iteration to be equal to $\bar{x} \pm 1.5 * \sigma$. After the fourth iteration, we ran a final iteration where we selected the top-10 performing hyper-parameter settings across the four previous iterations, and then re-ran these for another 120 seeds (i.e. random weight initializations). We then selected the hyper-parameter settings which had the highest performance across all 120 seeds.

Our MLP model consisted of two hidden layers of equal size with batch norm applied to each. The number of neurons in each layer was set, taking into consideration the current activation function being tested, to ensure the number of trainable parameters in the network remained static across experiments

Our CNN model consisted of six layers, where each layer was comprised of a set of 2D convolution filters (kernel size 3, stride 1, padding 1), batch normalization, and a non-linear activation. The network also applied a pooling layer (kernel size 2, stride 2) at the end of every second layer. After the 6 CNN layers the network flattens the output and applies three linear layers. The number of output channels (ie. pre-activations) for the six 2D convolutions were $[x, 2x, 4x, 4x, 8x, 8x]$, and the number of pre-activation neurons produced by the subsequent linear layers were $[32x, 16x]$. Similar to the MLP model, x was chosen to ensure the number of trainable parameters in the network remained fixed across experiments. When varying the number of network parameters in our experiments, if the number of parameters dropped below $1e6$ in the CNN model, we changed the structure of the convolution layers and linear layers to $[x, x, x, x, 1.5x, 1.5x]$ and $[2x, 1.5x]$, respectively. This ensured that each layer had at least two channels for our activation functions to aggregate.

Once our optimal hyper-parameters were selected for both our MLP and CNN models, we then trained each model several times with varying number of trainable parameters. The reason we vary the number of parameters in the network instead of the network size / structure is because a network using our logical activation functions can have a significantly different number of parameters than an identical network using ReLU activation, because our logical activations aggregate across neurons at each layer.

A.11 RESNET50 ON CIFAR-10/100

For this experiment we trained a ResNet50 model for 100 epochs on CIFAR-10 and CIFAR-100 using Adam optimizer, one cycle learning rate schedule, cross entropy loss, and augmentations from the default CIFAR-10 auto-augmentCubuk et al. (2018) stack, with batch size of 128. The hyper-parameters for the optimizer and scheduler were determined by undergoing a random search of the hyper parameter space on the CIFAR-100 dataset[3], done on a validation set derived from by taking a train / test split on the training set (10% of the samples becoming the validation set). Our hyper-parameter search follows a similar approach to the one used by our MLP and CNN models on MNIST, but this time with due to compute constraints on our larger model we only trained for 100 seeds each iteration, and only looked at the top 10 seeds in the final round.

Once our optimal hyper-parameters were selected we trained a ResNet50 model with AMP mixed precision using four different width parameters: 0.5, 1, 2, and 4. Because activation functions such as $\mathrm{XNOR}_{\mathrm{AIL}}$ reduce the number of post-activation neurons by a factor of 2, and the $\mathrm{OR}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d) activation function increases the number of post-activation neurons by a factor of $3/2$, we adjust the width parameter for these activation function to result in all networks having approximately the same number of parameters as in our MNIST experiment. For the Max, $\mathrm{OR}_{\mathrm{AIL}}$, $\mathrm{XNOR}_{\mathrm{AIL}}$, $\mathrm{OR}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p) , and $\mathrm{OR}_{\mathrm{AIL}}, \mathrm{AND}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p) activation functions we use width parameters of 0.75, 1.5, 3, and 6. For the $\mathrm{OR}_{\mathrm{AIL}}, \mathrm{XNOR}_{\mathrm{AIL}}$ activation function we use width parameters of 0.4, 0.75, 1.5, and 3.

A.12 TRANSFER LEARNING

We used a pretrained ResNet50 model taken from the pytorch hub. The 2-layer MLP head was trained for 25 epochs on each dataset. Since we are interested in transfer learning in a data-limited regime, we used only a small amount of data augmentation. We applied horizontal flip ($p = 0.5$), scaling (0.7 to 1), aspect ratio stretching ($^3/_4$ to $^4/_3$), and colour jitter (intensity 0.4) only. Pixel intensity normalization was done against the ImageNet mean and std.

The MLP head was optimized using SGD, momentum 0.9. We used a batch size of 128, maximum learning rate 0.01, and weight decay 1×10^{-4}. Before training began, we passed one epoch worth of inputs through the network without updating the weights in order to refresh the batch normalisation statistics to the new dataset. We also performed one epoch of training with a warmup learning rate of 1×10^{-5} before commencing training at the maximum learning rate of 0.01. The learning rate was decayed with a cosine annealing schedule over 24 epochs. We report the performance of the final model at the end of the 25 epochs.

We used a pre-activation width of 512 neurons for ReLU and other and activation functions which map $1 \rightarrow 1$. To approximately match up the number of parameters, we used a width of 720 for activation functions which map $2 \rightarrow 1$. Our results with widths 512/720 and $\sim$ 700k parameters are shown in Table 1.

Additionally, we ran the experiment again with a width of $w = 512$ for the $2 \rightarrow 1$ activation functions. These results with constant width $w = 512$ are shown in Table 5.

For completeness, we also ran the experiment with a reduced width of $w = 384$ for the $1 \rightarrow 1$ activation functions. The results with widths 384/512 and ~ 400k parameters are shown in Table 6.

[3]We used the same set of hyper-parameters from this search for both the CIFAR-10 and CIFAR-100 experiments.

Table 5: Transfer learning with from a frozen ResNet-18 architecture pre-trained on ImageNet-1k to other computer vision datasets. As with Table 1, but in this case we show only results where the MLP has a pre-activation width of $w = 512$. Note that although the pre-activation width is constant, the number of parameters in the network is not consistent between experiments. Mean (standard error) of $n=5$ inits of the MLP (same pretrained network).

		Test Accuracy (%)						
Activation function	Params	Caltech101	CIFAR10	CIFAR100	Flowers	Cars	STL-10	SVHN
Linear layer only	5k–101k	**88.33**$\pm$0.19	78.56$\pm$0.09	57.39$\pm$0.09	**92.50**$\pm$0.11	33.51$\pm$0.06	94.69$\pm$0.03	45.42$\pm$0.06
ReLU	530k–626k	86.72$\pm$0.13	81.63$\pm$0.05	58.04$\pm$0.11	90.76$\pm$0.33	30.97$\pm$0.26	94.61$\pm$0.08	51.39$\pm$0.06
Max	397k–445k	86.93$\pm$0.12	81.56$\pm$0.06	58.12$\pm$0.10	90.67$\pm$0.31	32.80$\pm$0.04	94.64$\pm$0.06	51.15$\pm$0.08
Max, Min (d)	530k–626k	87.31$\pm$0.13	82.31$\pm$0.10	59.05$\pm$0.10	91.68$\pm$0.18	34.91$\pm$0.12	94.64$\pm$0.04	51.72$\pm$0.04
OR_{AIL}	397k–445k	87.29$\pm$0.11	81.81$\pm$0.02	58.68$\pm$0.05	91.78$\pm$0.23	35.28$\pm$0.09	94.64$\pm$0.07	51.27$\pm$0.12
XNOR_{AIL}	397k–445k	86.49$\pm$0.09	81.74$\pm$0.04	57.88$\pm$0.07	90.50$\pm$0.11	31.55$\pm$0.09	**94.75**$\pm$0.02	51.23$\pm$0.08
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}$ (d)	530k–626k	**87.41**$\pm$0.14	82.38$\pm$0.06	59.90$\pm$0.08	92.10$\pm$0.20	37.16$\pm$0.15	94.52$\pm$0.05	52.11$\pm$0.09
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	397k–445k	87.28$\pm$0.08	81.85$\pm$0.06	58.72$\pm$0.09	91.80$\pm$0.24	35.27$\pm$0.09	94.59$\pm$0.07	51.25$\pm$0.14
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)	530k–626k	87.04$\pm$0.27	82.20$\pm$0.04	59.44$\pm$0.07	91.90$\pm$0.10	36.88$\pm$0.10	94.69$\pm$0.06	52.02$\pm$0.16
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	397k–445k	87.16$\pm$0.18	81.75$\pm$0.07	58.72$\pm$0.08	91.85$\pm$0.14	35.46$\pm$0.14	94.66$\pm$0.03	51.32$\pm$0.09
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)	664k–807k	87.26$\pm$0.15	**82.45**$\pm$0.08	**60.20**$\pm$0.11	**92.41**$\pm$0.17	**37.89**$\pm$0.14	94.65$\pm$0.04	**52.19**$\pm$0.09

Table 6: Transfer learning with from a frozen ResNet-18 architecture pre-trained on ImageNet-1k to other computer vision datasets. As with Table 1, but in this case we show only results where the MLP has a pre-activation width of 384 or 512 depending on activation function. Mean (standard error) of $n=5$ inits of the MLP (same pretrained network).

		Test Accuracy (%)						
Activation function	Params	Caltech101	CIFAR10	CIFAR100	Flowers	Cars	STL-10	SVHN
Linear layer only	5k–101k	**88.33**$\pm$0.19	78.56$\pm$0.09	57.39$\pm$0.09	**92.50**$\pm$0.11	33.51$\pm$0.06	94.69$\pm$0.03	45.42$\pm$0.06
ReLU	328k–397k	86.50$\pm$0.18	81.40$\pm$0.07	57.59$\pm$0.08	90.22$\pm$0.18	29.37$\pm$0.13	94.64$\pm$0.03	51.04$\pm$0.06
Max	397k–445k	86.93$\pm$0.12	81.56$\pm$0.06	58.12$\pm$0.10	90.67$\pm$0.31	32.80$\pm$0.04	94.64$\pm$0.06	51.15$\pm$0.08
Max, Min (d)	328.k–397k	87.26$\pm$0.16	81.81$\pm$0.05	58.72$\pm$0.11	91.34$\pm$0.18	33.74$\pm$0.08	94.61$\pm$0.04	51.48$\pm$0.11
OR_{AIL}	397k–445k	87.29$\pm$0.11	81.81$\pm$0.02	58.68$\pm$0.05	91.78$\pm$0.23	35.28$\pm$0.09	94.64$\pm$0.07	51.27$\pm$0.12
XNOR_{AIL}	397k–445k	86.49$\pm$0.09	81.74$\pm$0.04	57.88$\pm$0.07	90.50$\pm$0.11	31.55$\pm$0.09	**94.75**$\pm$0.02	51.23$\pm$0.08
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}$ (d)	328k–397k	**87.32**$\pm$0.15	82.09$\pm$0.07	59.24$\pm$0.14	91.78$\pm$0.13	36.08$\pm$0.07	94.63$\pm$0.04	51.67$\pm$0.07
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	397k–445k	87.28$\pm$0.08	81.85$\pm$0.06	58.72$\pm$0.09	91.80$\pm$0.24	35.27$\pm$0.09	94.59$\pm$0.07	51.25$\pm$0.14
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)	328k–397k	87.28$\pm$0.21	81.98$\pm$0.02	59.00$\pm$0.11	91.68$\pm$0.35	35.74$\pm$0.19	94.67$\pm$0.05	51.65$\pm$0.06
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	397k–445k	87.16$\pm$0.18	81.75$\pm$0.07	58.72$\pm$0.08	91.85$\pm$0.14	35.46$\pm$0.14	94.66$\pm$0.03	51.32$\pm$0.09
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)	398k–501k	87.17$\pm$0.17	**82.19**$\pm$0.07	**59.55**$\pm$0.08	**92.41**$\pm$0.33	**36.84**$\pm$0.10	94.66$\pm$0.02	**51.91**$\pm$0.16

A.13 Compositional Zero-Shot Learning

Our experiments were based on the Task-driven Modular Networks (TMN) proposed by Purushwalkam et al. (2019). We used the code shared by the authors, but were unable to replicate the results they reported in the paper when using a different random seed (see Table 7). We adapted this network by changing out all the ReLU activation functions in the gate and module networks with a different activation function. Because the modules each terminate with an activation function, we needed to double the size of the hidden layer for some of our networks in order to maintain the dimensionality of the output. Consequently, some experiments had around 50% more parameters in total than others.

We trained the network using the same paradigm as Purushwalkam et al. (2019): Adam (Kingma & Ba, 2017) with a learning rate of 0.001 for the module network and 0.01 for the gating network, momentum 0.9, batch size 256, weight decay 5×10^{-5}. We evaluated the network using the AUC between seen and unseen samples Purushwalkam et al. (2019). The network was trained until the validation AUC had plateaued, determined by not increasing for 5 epochs. We selected the model from the epoch with highest validation AUC to apply to the test set. The best performance was typically attained after around 5 epochs.

As shown in Table 7, we find that the trio of activation functions applied in parallel, $\{\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)$\}$, performs best.

Table 7: Performance of TMN-based networks at compositional zero-shot learning (CZSL) on the MIT-States dataset. Mean (standard error) of $n=5$ random initializations.

			MIT-States Test AUC (%)		
Activation function	Mapping	Params	Top-1	Top-2	Top-3
TMN (Purushwalkam et al., 2019)	$1 \rightarrow 1$		2.9	7.1	11.5
TMN repro. (ReLU)	$1 \rightarrow 1$	438 k	2.47 ± 0.07	6.42 ± 0.09	10.27 ± 0.11
Max	$2 \rightarrow 1$	650 k	2.42 ± 0.07	6.37 ± 0.08	10.28 ± 0.06
Max, Min (d)	$1 \rightarrow 1$	438 k	2.53 ± 0.06	6.69 ± 0.11	10.61 ± 0.14
XNOR_{AIL}	$2 \rightarrow 1$	650 k	1.22 ± 0.05	3.47 ± 0.12	5.82 ± 0.19
OR_{AIL}	$2 \rightarrow 1$	650 k	2.65 ± 0.05	6.80 ± 0.08	10.78 ± 0.13
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}$ (d)	$1 \rightarrow 1$	438 k	2.61 ± 0.05	6.73 ± 0.05	10.77 ± 0.12
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	$2 \rightarrow 1$	650 k	2.65 ± 0.05	6.80 ± 0.08	10.78 ± 0.13
$\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)	$1 \rightarrow 1$	438 k	1.89 ± 0.13	5.12 ± 0.21	8.35 ± 0.24
$\text{OR}_{\text{AIL}}, \text{AND}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (p)	$2 \rightarrow 1$	650 k	$\mathbf{2.67} \pm 0.10$	$\mathbf{6.95} \pm 0.14$	$\mathbf{10.96} \pm 0.16$

Since we could not flexibly scale the total number of parameters in the network with the original architecture, we added an additional linear layer at the end of each module which projects from the activation function to the embedding space. Comparing the TMN results in Table 7 to TMNx in Table 3, we can see that adding the extra linear layer improved performance of the network in of itself. Intuitively, this makes sense since the output of the TMN modules are weighted with the output of the gating network and then summed, and this weighting and summing of evidence is best performed with on logits instead of the truncated output of ReLU units. But also, performance may have improved just because the model became larger.

We performed a hyperparameter search across the training parameters for the new network against the validation set using the ReLU activation function only. We adopted the hyperparameters discovered for ReLU for all other activation functions. The batch size was reduced to 128 due to the increase in size of the model. The discovered hyperparameters were a learning rate of 3×10^{-3} for both the module and gating network, and a weight decay of 1×10^{-5}. Other training hyperparameters, such as the ratio of negative samples to present, were left unchanged.

In order to match the number of parameters in the network, we used the original TMN hidden width of 64 for the module and gater networks with activation functions which map $1 \rightarrow 1$, and increased this to 96 for activation function which map $2 \rightarrow 1$. The main results are described in subsection 3.8