Logical Activation Functions: Logit-space equivalents of Probabilistic Boolean Operators

Anonymous Author(s)
Affiliation
Address
email

Abstract

The choice of activation functions and their motivation is a long-standing issue within the neural network community. Neuronal representations within artificial neural networks are commonly understood as logits, representing the log-odds score of presence of features within the stimulus. We derive logit-space operators equivalent to probabilistic Boolean logic-gates AND, OR, and XNOR for independent probabilities. Such theories are important to formalize more complex dendritic operations in real neurons, and these operations can be used as activation functions within a neural network, introducing probabilistic Boolean-logic as the core operation of the neural network. Since these functions involve taking multiple exponents and logarithms, they are computationally expensive and not well suited to be directly used within neural networks. Consequently, we construct efficient approximations named AND_{AIL} (the AND operator Approximate for Independent Logits), OR_{AIL}, and XNOR_{AIL}, which utilize only comparison and addition operations, have well-behaved gradients, and can be deployed as activation functions in neural networks. Like MaxOut, AND_{AIL} and OR_{AIL} are generalizations of ReLU to two-dimensions. While our primary aim is to formalize dendritic computations within a logit-space probabilistic-Boolean framework, we deploy these new activation functions, both in isolation and in conjunction to demonstrate their effectiveness on a variety of tasks including image classification, transfer learning, abstract reasoning, and compositional zero-shot learning.

1 Introduction

Non-linear activation functions are essential in artificial neural networks to form higher-order representations, since otherwise the network would be degeneratively equivalent to a single linear layer. Most activation functions represent a simple non-linearity despite the fact that there is a lot of evidence of much more complex non-linear integration and computations in dendrites (Hentschel et al., 2004; London & Häusser, 2005; Payeur et al., 2019). For example, Gidon et al. (2020) recently demonstrated that a single biological neuron can compute the XOR of its inputs, a property long known to be lacking from artificial neurons (Minsky & Papert, 1969). While simple activation functions work in artificial neural networks in the sense that more complex operations can be formed from the combination of several layers of neurons, understanding the function and impact of advanced dendritic operations in networks is important.

Here we develop a theoretical underpinning of recent proposals for higher-order activation functions like MaxOut (Goodfellow et al., 2013) in a probabilistic framework. 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. We derive logit-space operators equivalent to probabilistic Boolean logic-gates AND, OR, and XNOR for independent probabilities. We then

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

construct computationally feasible approximations to these functions with well-behaved gradients. These new activation functions, which are generalizations of ReLU to two-dimensions, are then applied on benchmark datasets to demonstrate their effectiveness on a diverse range of tasks including image classification, transfer learning, abstract reasoning, and compositional zero-shot learning. The new principled approach we present introduces new ways to redistribute computation from the network into the neuronal mechanisms, and build more parameter-efficient models. We demonstrate effective activation functions based on these ideas and expect to future work to build on this.

2 Background

The earliest artificial neural networks 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).

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 activation function (Fukushima, 1980; Jarrett et al., 2009; Nair & Hinton, 2010), $\mathrm{ReLU}(x) = \max(0, x)$, does not have this problem, since in its non-zero 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 popular choice of activation function. The chief drawback to ReLU is that it has no sensitivity to changes across half of its input domain, which prevents updates on stimuli which trigger its "off" state and can even lead to neuronal death[1]. Whilst sigmoid is equivalent to converting our feature logits into probabilities, the ReLU activation function is equivalent to truncating our logits and denying any evidence for the absence of a feature. We hypothesize that omitting negative evidence in this way is undesirable and reduces the generalization ability of ReLU-based networks.

Variants of ReLU have emerged, aiming to smooth out its transition between domains and provide a gradient in its inactive regime. These include ELU (Clevert et al., 2016), CELU (Barron, 2017), SELU (Klambauer et al., 2017), GELU (Hendrycks & Gimpel, 2020), SiLU (Elfwing et al., 2017; Ramachandran et al., 2017), and Mish (Misra, 2019). However, all these activation functions still bear the general shape of ReLU and truncate negative logits.

Fuzzy logic operators are generalizations of Boolean logic operations to continuous variables, using rules similar to applying logical operators in probability space. Previous work has explored networks of fuzzy logic operations, including neural networks which use an activation functions that constitutes a learnable interpolation between fuzzy logic operators (Godfrey & Gashler, 2017). In our work, the activation functions which we introduce are motivated similarly to fuzzy logic operators, but are designed to operate in logit space instead of in probability space, which better reflects the behaviour and space of pre-activation neural units.

In this work we contribute to the theoretical underpinning of neural activation functions by developing activation functions based on the principle that neurons encode logits — scores that represent the presence of features in the log-odds space. In Section 3 we derive and define these functions in detail

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

for different logical operators, and then consider their performance on numerous task types including parity (Section 4.1), image classification (Sections 4.3 and 4.4), transfer learning (Section 4.5), abstract reasoning (Appendix A.17), soft-rule guided classification as exemplified by the Bach chorale dataset (Section 4.2), and compositional zero-shot learning (Appendix A.18). These tasks were selected to (1) survey the performance of the new activations on existing benchmark tasks, and (2) evaluate their performance on tasks which we suspect in particular may require logical reasoning and hence benefit from activation functions which apply these logical operations to logits.

3 Derivation

Manipulation of probabilities in logit-space is known to be more efficient for many calculations. For instance, the log-odds form of Bayes' Rule (Equation 9) states that the posterior logit equals the prior logit plus 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 (see Appendix A.3). 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. 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 $y = w^T x + b$. 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 generalize 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 the solution for OR 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.

3.1 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 $\mathrm{P}(X, Y) = \mathrm{P}(X \wedge Y) = \mathrm{P}(X) \; \mathrm{P}(Y) = \sigma(x) \, \sigma(y)$.

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

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

which we coin as $\mathrm{AND}_{\mathrm{IL}}$, the AND operator for independent logits (IL). 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 $\mathrm{AND}_{\mathrm{IL}}$ 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 $\mathrm{AND}_{\mathrm{IL}}$ is the sum of their logits.

We thus propose $\mathrm{AND}_{\mathrm{AIL}}$, a linear-approximate AND function for independent logits (AIL, i.e. approximate IL).

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

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

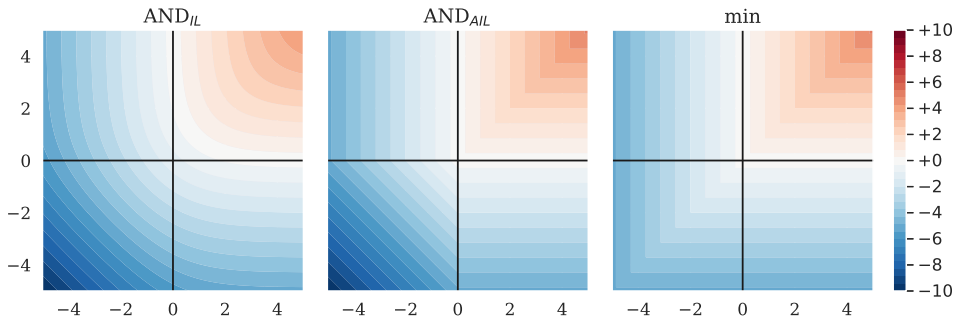


Figure 1: Heatmap comparing the outputs for the exact logit-space probabilistic-and function for independent logits, $\mathrm{AND}_{\mathrm{IL}}(x, y)$; our constructed approximation, $\mathrm{AND}_{\mathrm{AIL}}(x, y)$; and $\max(x, y)$.

3.2 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

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

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

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

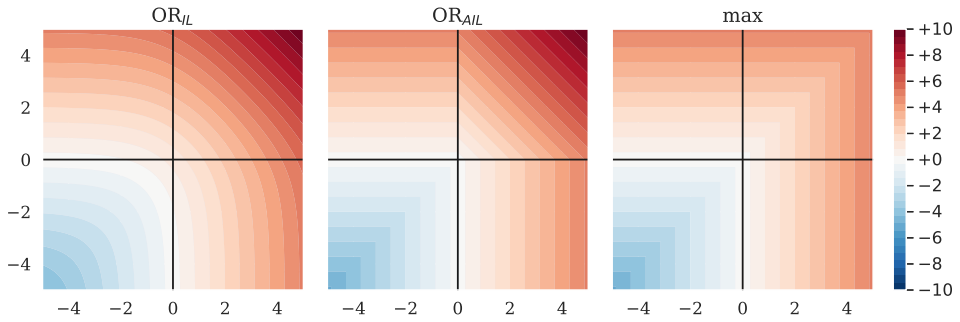


Figure 2: Comparison of the exact logit-space probabilistic-or function for independent logits, $\mathrm{OR}_{\mathrm{IL}}(x, y)$; our constructed approximation, $\mathrm{OR}_{\mathrm{AIL}}(x, y)$; and $\max(x, y)$.

3.3 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 this matches $\mathrm{XNOR}_{\mathrm{IL}}$ less closely, and has a divergent gradient along both $x = 0$ and $y = 0$.

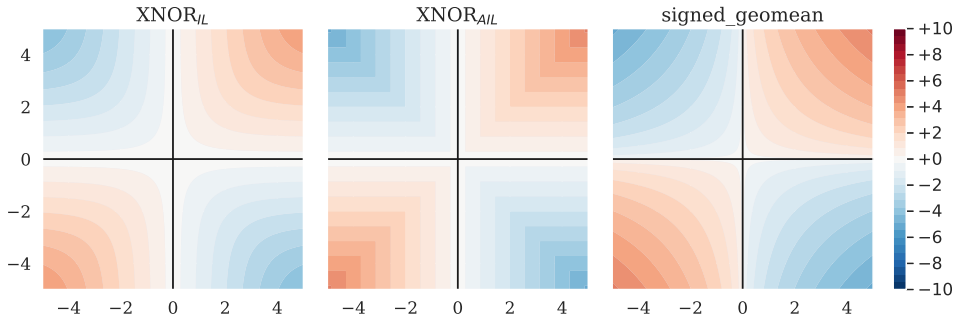


Figure 3: Comparison of the exact logit-space probabilistic-xnor function for independent logits, $\mathrm{XNOR}_{\mathrm{IL}}(x, y)$; our constructed approximation, $\mathrm{XNOR}_{\mathrm{AIL}}(x, y)$; and $\mathrm{signed_geomean}(x, y)$.

3.4 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 generalization 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 generalized 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 this work to consider only $k = 2$, but note these activation functions also generalize to higher dimensions.

3.5 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 (illustrated in Appendix A.6).

In the partition (p) strategy, we split the n_c dimensional pre-activation 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 be 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.

A negative weight in a network is equivalent to the logit-NOT operator. Hence with sufficient width and depth, a multi-layer network using only the $\mathrm{OR}_{\mathrm{IL}}$ activation function should be able to replicate any probabilistic logic circuit.

Utilizing $\mathrm{AND}_{\mathrm{AIL}}$, $\mathrm{OR}_{\mathrm{AIL}}$ 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.

We hypothesised that training a network with all three of our activation functions in an ensemble in this manner could yield better results since the network would not need to expend layers having to combine $\mathrm{OR}_{\mathrm{IL}}$ operations together to yield other Boolean operations.

3.6 Normalization

Our AIL activation functions are close approximations to exact AND, OR, and XNOR operations in logit-space. However, when deploying non-linearities within a neural network, it is important that the activation functions have a gain of 1 in order to improve stability during training (Klambauer et al., 2017), a property the AIL activations do not possess. We constructed normalized variants of the exact logit operators and our approximations, dubbed NIL and NAIL, respectively. This was achieved by subtracting the expected mean, and dividing by the expected standard deviation, assuming that the operands are sampled independently from the standard normal distribution, $\mathcal{N}(0, 1)$. For more details, see Appendix A.7.

4 Experiments

We evaluated the performance of our AIL activation functions, both individually and together in an ensemble, on a range of benchmark 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 only show results for $\mathrm{OR}_{\mathrm{AIL}}$ not both individually.

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

4.1 Parity

In a simple initial experiment, we constructed a synthetic dataset whose labels could be derived directly by stacking the logical operation XNOR. Each sample had four input logits, and target value equal to the parity of the number of positive inputs. A very small model using XNOR with two hidden layers (of 4, then 2 neurons) should be capable of perfect classification accuracy on this dataset with a sparse weight matrix by learning to nest pairwise binary relationships. We trained such an MLP with either ReLU or $\mathrm{XNOR}_{\mathrm{AIL}}$ activations, for 100 epochs using Adam, one-cycle learning rate schedule, max LR 0.01, weight decay 1×10^{-4}.

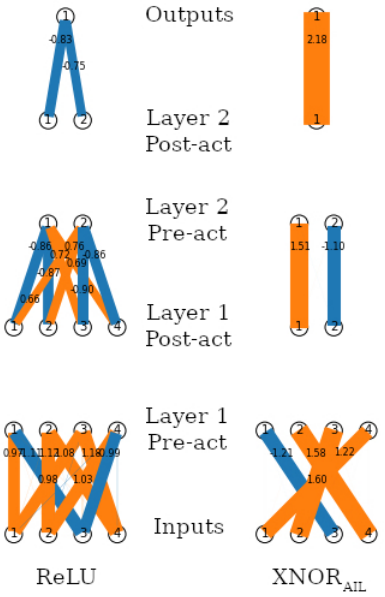


Figure 4: Visualization of weight matrices learnt by two-layer MLPs on a binary classification task, where the target output is the parity of the inputs. Widths of lines indicate weight magnitudes (orange: +ve values, blue: -ve). Network with ReLU: 60% accuracy; $\mathrm{XNOR}_{\mathrm{AIL}}$: 100% accuracy.

The MLP with $\mathrm{XNOR}_{\mathrm{AIL}}$ activation learned a sparse weight matrix able to perfectly classify any input combination, shown in Figure 4. In comparison, with ReLU the network was only able to produce 60% classification accuracy. The accuracy with ReLU was improved by increasing the MLP width/depth, but this did not result in a sparse weight matrix. This experiment demonstrates that $\mathrm{XNOR}_{\mathrm{AIL}}$ can be utilized by a network to find the simplest relationship between inputs that satisfies the objective. For additional results, see Appendix A.10.

4.2 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 ∼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 and of the other voices, e.g. "two voices a fifth apart ought not to move in parallel with one another". The task we used was to determine whether a given short four-part musical excerpt is taken from a Bach chorale or not. Negative examples were created by stochastically corrupting chorale excerpts (see Appendix A.12). We trained 2-layer MLPs with a sigmoid output (summarized in Figure 18), and found that $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)$\}$ performed best, but that overall the results were comparable ($p < 0.1$ between best and worst, Student's t-test, 10 random inits).

Additionally, we investigated the independence between logits in the trained pre-activation embeddings. We 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 took the cosine similarity between rows of the weight matrix. Since the inputs to all features in a given layer are the same, this is equivalent to measuring the similarity between corresponding pair of pre-activation features. We measured correlations between adjacent pre-activations (paired operands for the logical activation functions), and also between non-adjacent pairs of pre-activations. Our results indicate that the network learns features which are independent when they are not passed to the same 2D activation, and *anti-correlated* features when they are. For more details, see Appendix A.13.

4.3 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 comprised of the last 10k images of the training partition.

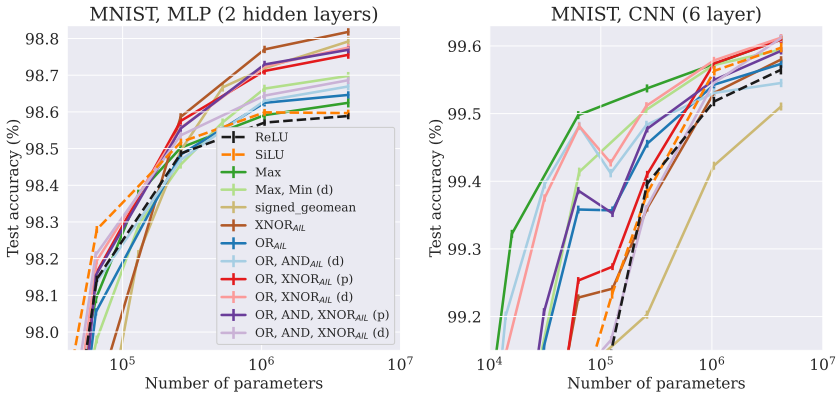


Figure 5: We trained CNN on MNIST, MLP on flattened-MNIST, using Adam (1-cycle, 10 ep), hyperparams determined by random search. Mean (bars: std dev) of $n = 40$ weight inits.

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 pooling (stride 2) after every other conv layer, followed by flattening and three MLP layers. The layer widths were scaled up to explore a range of model sizes (see Appendix A.14 for more details).

For the MLP, $\mathrm{XNOR}_{\mathrm{AIL}}$ performed best along with signed_geomean ($p < 0.1$), ahead of all other activations ($p < 0.01$; Figure 5 left panel). With the CNN, five activation configurations ($\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)$\}$, $\{\mathrm{OR}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d/p)$\}$, Max, and SiLU) performed best ($p < 0.05$; Figure 5, right panel). Additionally, we note that CNNs which used $\mathrm{OR}_{\mathrm{AIL}}$ or Max (alone or in an ensemble) maintained high performance with an order of magnitude fewer parameters (3×10^4) than networks which did not (3×10^5 params).

4.4 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 the 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, w. The network was trained on CIFAR-10/-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)$\}$.

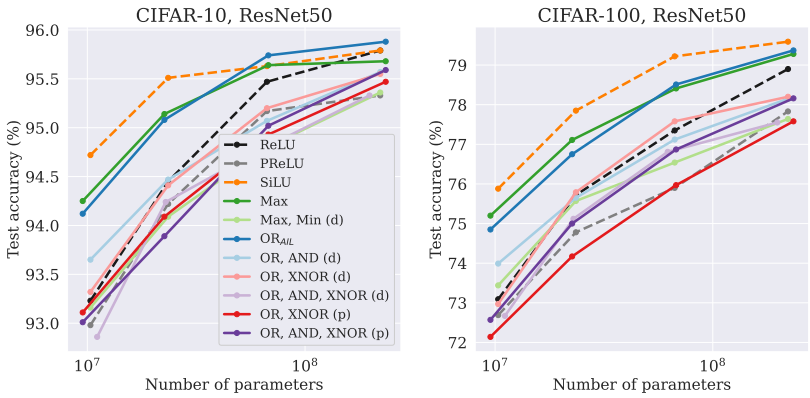


Figure 6: ResNet50 on CIFAR-10/100, varying the activation function used through the network. The width was varied to explore a range of network sizes (see text). Trained for 100 ep. with Adam, using hyperparams as determined by random search on CIFAR-100 with width factor $w = 2$.

For both CIFAR-10 and 100, we find that SiLU, $\mathrm{OR}_{\mathrm{AIL}}$, and Max outperform ReLU across a wide range of width values, as shown in Figure 6. These three activation functions hold up their performance best as the number of parameters is reduced. Since SiLU was previously discovered as an optimal $1 \rightarrow 1$ activation function for deep residual image models (Ramachandran et al., 2017), we expected it to perform highly in this setting. 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.

4.5 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 from these embeddings, using various activation functions. For a comprehensive set of baselines, we compared against every activation function built into PyTorch 1.10 (see Appendix A.16). To make the number of parameters similar, we used a width of 512 for activation functions with $1 \rightarrow 1$ mapping (e.g. ReLU), a width of 650 for activation functions with a $2 \rightarrow 1$ mapping (e.g. Max, $\mathrm{OR}_{\mathrm{AIL}}$), and a width of 438 for $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)$\}$. See Appendix A.16 for more details.

Our results are shown in Table 1. We found that all our AIL activation functions outperformed ReLU on every transfer task. Normalization had little impact on the performance of $\mathrm{OR}_{*\mathrm{IL}}$, but a large impact on $\mathrm{XNOR}_{*\mathrm{IL}}$. The best overall performance was attained by the exact $\mathrm{XNOR}_{\mathrm{NIL}}$, closely followed by our approximate $\mathrm{XNOR}_{\mathrm{NAIL}}$ and ensembles containing either of these. Surprisingly, our approximation $\mathrm{OR}_{*\mathrm{AIL}}$ outperformed the exact form $\mathrm{OR}_{*\mathrm{IL}}$. The proposed activation functions were beaten only by PReLU, on Stanford Cars. On Caltech101, all MLPs were beaten by a linear layer, suggesting our MLPs were overfitting for that dataset. For further discussion, see Appendix A.16.

Table 1: Transfer learning from a frozen ResNet-18 architecture pretrained on ImageNet-1k to other computer vision datasets. Mean (std. error) of $n = 5$ random initializations of the MLP (same pretrained encoder). Bold: best. Underlined: top two. Italic: no sig. diff. from best (two-sided Student's t-test, $p > 0.05$). Background: linear color scale from ReLU baseline (white) to best (black).

	Test Accuracy (%)						
Activation function	Cal101	CIFAR10	CIFAR100	Flowers	StfCars	STL-10	SVHN
Linear layer only	***88.35***±0.15	78.56±0.09	57.39±0.09	92.32±0.20	33.51±0.06	94.68±0.02	46.60±0.14
ReLU	86.58±0.17	81.63±0.05	58.04±0.11	90.71±0.26	30.97±0.26	94.62±0.06	53.26±0.08
PReLU	*87.83*±0.21	81.03±0.13	58.90±0.18	*93.17*±0.19	***39.84***±0.18	94.54±0.05	53.47±0.08
SELU	87.74±0.09	79.93±0.13	58.24±0.06	92.27±0.13	37.51±0.17	94.53±0.07	50.94±0.12
GELU	87.10±0.15	81.39±0.09	58.51±0.13	91.51±0.15	33.43±0.15	94.62±0.06	53.43±0.23
Mish	87.11±0.12	81.09±0.11	58.37±0.10	91.61±0.15	33.75±0.14	94.61±0.05	53.05±0.12
Tanh	87.48±0.06	80.56±0.07	57.35±0.08	90.32±0.20	29.51±0.12	*94.63*±0.07	51.86±0.05
Max	86.96±0.20	81.76±0.14	58.60±0.12	90.98±0.18	33.37±0.15	*94.70*±0.06	53.53±0.16
Max, Min (d)	87.23±0.13	82.31±0.10	59.05±0.10	91.68±0.18	34.91±0.12	94.64±0.04	53.91±0.13
XNOR_{IL}	85.01±0.17	79.62±0.09	57.14±0.07	84.76±0.43	1.34±0.11	94.51±0.03	51.99±0.16
OR_{IL}	87.11±0.08	79.75±0.05	58.07±0.11	91.12±0.36	33.12±0.12	94.60±0.03	51.21±0.17
XNOR_{NIL}	87.25±0.22	***82.88***±0.08	***60.78***±0.08	***93.26***±0.26	*39.47*±0.20	***94.83***±0.06	***55.34***±0.19
OR_{NIL}	87.19±0.16	79.61±0.05	58.44±0.10	91.65±0.29	35.82±0.04	94.58±0.03	50.95±0.15
OR, AND_{NIL} (d)	86.82±0.18	80.09±0.09	58.60±0.07	91.44±0.20	37.03±0.11	94.65±0.05	52.49±0.09
OR, XNOR_{NIL} (d)	*87.82*±0.19	*82.67*±0.05	*60.60*±0.11	*92.93*±0.12	39.22±0.17	94.63±0.06	*54.87*±0.09
OR, AND, XNOR_{NIL} (d)	87.41±0.27	*82.84*±0.06	60.38±0.10	*92.98*±0.17	*39.42*±0.18	*94.71*±0.03	*55.11*±0.12
XNOR_{AIL}	86.97±0.18	81.83±0.06	58.46±0.10	90.93±0.15	32.56±0.10	*94.71*±0.06	53.75±0.14
OR_{AIL}	87.45±0.14	81.88±0.07	59.10±0.09	92.00±0.15	36.01±0.12	*94.69*±0.04	53.68±0.14
$\text{XNOR}_{\text{NAIL}}$	87.61±0.23	82.38±0.07	59.77±0.13	*93.07*±0.20	*39.77*±0.04	*94.81*±0.03	53.91±0.05
OR_{NAIL}	87.19±0.16	81.79±0.09	59.40±0.09	92.12±0.12	37.32±0.17	94.65±0.04	53.82±0.21
OR, AND_{NAIL} (d)	87.62±0.11	82.28±0.10	59.71±0.05	92.10±0.20	37.70±0.12	*94.61*±0.08	53.86±0.10
OR, $\text{XNOR}_{\text{NAIL}}$ (d)	*87.85*±0.22	82.52±0.11	60.02±0.10	*93.12*±0.13	*39.64*±0.09	*94.75*±0.03	54.13±0.05
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (d)	87.78±0.14	*82.67*±0.06	60.01±0.21	*93.12*±0.21	*39.65*±0.14	*94.78*±0.03	54.58±0.12

4.6 Additional results

For results on Covertype, abstract reasoning and compositional zero-shot learning, please see Appendix A.11, Appendix A.17 and Appendix A.18, respectively.

5 Conclusion

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 probabilistic Boolean operations, and demonstrated their effectiveness on a wide range of tasks.

Although our activation functions assume independence (which is generally approximately true for the pre-activation features learnt with 1D activation functions), we found the network learnt to induce anti-correlations between features which were paired together by our activation functions (Appendix A.13). This suggests that exact independence of the features is not essential to the performance of our proposed activation functions.

We found the $\text{XNOR}_{*\text{AIL}}$ 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 $\text{XNOR}_{*\text{IL}}$ or an ensemble of AIL activation functions together works best, and that the duplication ensembling strategy outperforms partitioning. In this work we have restricted ourselves to only considering using a single activation function (or ensemble) throughout the network, however our results together indicate that stronger results may be found by using $\text{OR}_{*\text{AIL}}$ for feature extraction and either $\text{XNOR}_{*\text{AIL}}$ or an ensemble of $\{\text{OR}, \text{AND}, \text{XNOR}_{*\text{AIL}}$ (d)$\}$ for later higher-order reasoning layers within the network.

Our work shows that there is more to learn about the importance of more complex activation functions, both for future more complex applications of artificial neural networks and to understand the importance of biological findings of non-linear dendritic integration in real neuronal networks.

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, 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, Colorado State University, 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, 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), 2016.

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.

François Fleuret, Ting Li, Charles Dubout, Emma K. Wampler, Steven Yantis, and Donald Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 108(43):17621–17625, 2011. ISSN 0027-8424. doi: 10.1073/pnas.1109168108. URL `https://www.pnas.org/content/108/43/17621`.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980. ISSN 1432-0770. doi: 10.1007/BF00344251. URL `https://doi.org/10.1007/BF00344251`.

Albert Gidon, Timothy Adam Zolnik, Pawel Fidzinski, Felix Bolduan, Athanasia Papoutsi, Panayiota Poirazi, Martin Holtkamp, Imre Vida, and Matthew Evan Larkum. Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, 367(6473):83–87, 2020. doi: 10.1126/science.aax6239. URL `https://www.science.org/doi/abs/10.1126/science.aax6239`.

Luke B. Godfrey and Michael S. Gashler. A parameterized activation function for learning fuzzy logic operations in deep neural networks. *CoRR*, abs/1708.08557, 2017. URL `http://arxiv.org/abs/1708.08557`.

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.

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), 2020.

H. G. E. Hentschel, Alan Fine, and C. S. Pencea. Biological computing with diffusion and excitable calcium stores. *Mathematical Biosciences Engineering*, 1(1):147–159, 2004.

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, 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.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pp. 2146–2153, 2009. doi: 10.1109/ICCV.2009.5459469.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

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, 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, University of Toronto, 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.

Michael London and Michael Häusser. Dendritic computation. *Annu. Rev. Neurosci.*, 28(1):503–532, 2005.

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, 2019.

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

Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines. volume 27, pp. 807–814, 06 2010.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and 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.

Alexandre Payeur, Jean-Claude Béïque, and Richard Naud. Classes of dendritic information processing. *Curr. Opin. Neurobiol.*, 58:78–85, October 2019.

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

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment, 2017.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

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

Jason Rennie, Lawrence Shih, Jaime Teevan, and David Karger. Tackling the poor assumptions of naive bayes text classifiers. *Proceedings of the Twentieth International Conference on Machine Learning*, 41, 07 2003.

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.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] The methodology we discuss is general-purpose and any societal impacts we could discuss would be highly general and completely speculative at this stage.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1–4.5 and Appendix A.10–A.18.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All results figures and tables have error over several seeds, and the number of seeds used, clearly indicated.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A.9. The time to run each experiment is also indicated in the README files which accompany the code.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Appendix A.8, Table 3.
   (b) Did you mention the license of the assets? [No] It is surprisingly difficult to discover the license under which popular ML datasets, such as CIFAR-10, are released! Most of the popular datasets within the community do not appear to be released under open licenses.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code to replicate our results is included in the supplemental material zip.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We describe the nature of the datasets we are using. There is no reason to expect that the data we are using contains personally identifiable information.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Approximating logistic AND and OR using ReLU

In Figure 7 (upper panels), we show a representation of a 2D linear layer followed by the ReLU activation function. Changing projection used in the linear layer allows us to rotate and stretch the output only; each unit is, by construction, empty in half the plane.

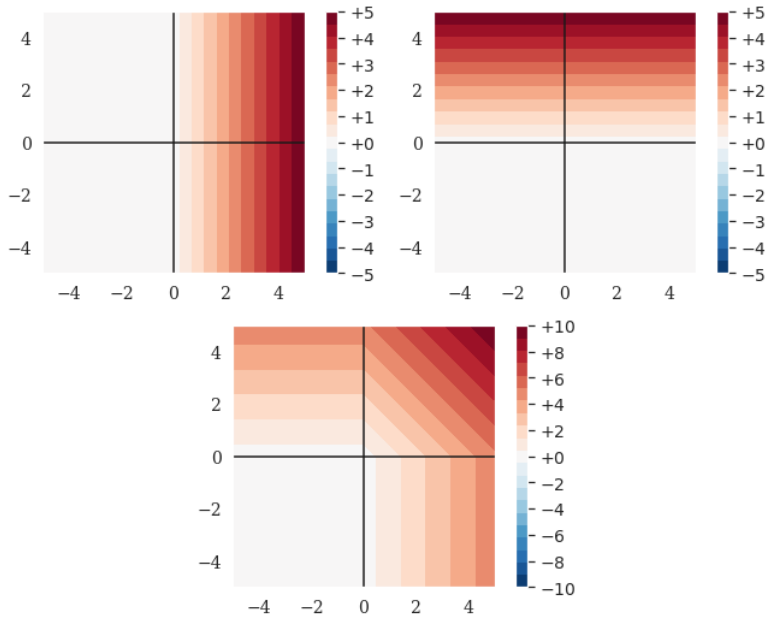
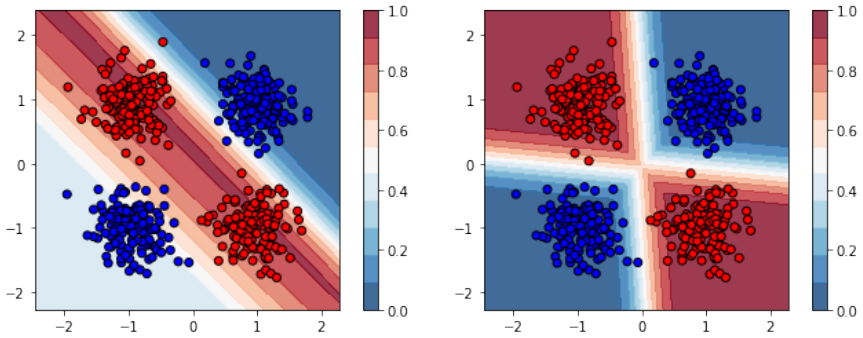


Figure 7: ReLU unit, and ReLU units followed by a linear layer to try to approximate $\mathrm{OR}_{\mathrm{IL}}$, leaving a dead space where the negative logits should be.

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 with $z = \mathrm{ReLU}(x) + \mathrm{ReLU}(y)$. However, as shown in Figure 7, lower panel, the solution using ReLU leaves a quadrant of the output space set to zero due to its behaviour at truncating away information. This illustrates why a ReLU-based neural network can not accurately approximate the logistic AND and OR functions between a pair of logits with only two hidden units. Comparing this with Figures 1 and 2 demonstrates the advantage of $\mathrm{AND}_{\mathrm{AIL}}$ and $\mathrm{OR}_{\mathrm{AIL}}$ in situations where the network needs to use probabilistic Boolean logic as its basis.

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 generalize to rotations and offset versions of the task.



(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.3 Linear layers and Bayes' Rule in log-odds form

Bayes' Theorem or Bayes' Rule is given by

$$\mathrm{P}(H|X) = \frac{\mathrm{P}(X|H)\ \mathrm{P}(H)}{\mathrm{P}(X)}. \tag{8}$$

In this case, we update the probability of our hypothesis, H, based on the event of the observation of a new piece of evidence, X. Our prior belief for the hypothesis is $\mathrm{P}(H)$, and posterior is $\mathrm{P}(H|X)$. To update our belief from the prior and yield the posterior, we multiply by the Bayes factor for the evidence which is given by ${}^{\mathrm{P}(X|H)}/_{\mathrm{P}(X)}$.

Converting the probabilities into log-odds ratios (logits) yields the following representation of Bayes' Rule.

$$\log\left(\frac{\mathrm{P}(H|X)}{\mathrm{P}(H^C|X)}\right) = \log\left(\frac{\mathrm{P}(H)}{\mathrm{P}(H^C)}\right) + \log\left(\frac{\mathrm{P}(X|H)}{\mathrm{P}(X|H^C)}\right) \tag{9}$$

Here, H^C is the complement to H (the event that the hypothesis is false), and $\mathrm{P}(H^C) = 1 - \mathrm{P}(H)$. Our prior log-odds ratio is $\log\left({}^{\mathrm{P}(H)}/_{\mathrm{P}(H^C)}\right)$, and our posterior after updating based on the observation of new evidence X is $\log\left({}^{\mathrm{P}(H|X)}/_{\mathrm{P}(H^C|X)}\right)$. To update our belief from the prior and yield the posterior, we add the log-odds Bayes factor for the evidence which is given by $\log\left({}^{\mathrm{P}(X|H)}/_{\mathrm{P}(X|H^C)}\right)$.

In log-odds space, a series of updates with multiple pieces of independent evidence can be performed at once with a summation operation.

$$\log\left(\frac{\mathrm{P}(H|\boldsymbol{x})}{\mathrm{P}(H^C|\boldsymbol{x})}\right) = \log\left(\frac{\mathrm{P}(H)}{\mathrm{P}(H^C)}\right) + \sum_i \log\left(\frac{\mathrm{P}(X_i|H)}{\mathrm{P}(X_i|H^C)}\right). \tag{10}$$

This operation can be represented by the linear layer in an artificial neural network, $z_k = b_k + \boldsymbol{w}_k^T \boldsymbol{x}$. Here, the bias term $b_k = \log\left({}^{\mathrm{P}(H)}/_{\mathrm{P}(H^C)}\right)$ is the prior for hypothesis (the presence of the feature represented by the k-th neuron), and the series of weighted inputs from the previous layer, $w_{ki}\, x_i$ provide evidential updates. This is also equivalent to the operation of a multinomial naïve Bayes classifier, expressed in log-space, if we choose $w_{ki} = \log p_{ki}$ (Rennie et al., 2003).

A.4 Difference between AIL and IL functions

Here, we measure and show the difference between the true logit-space operations and our AIL approximations, shown in Figure 9, Figure 10, and Figure 11.

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.

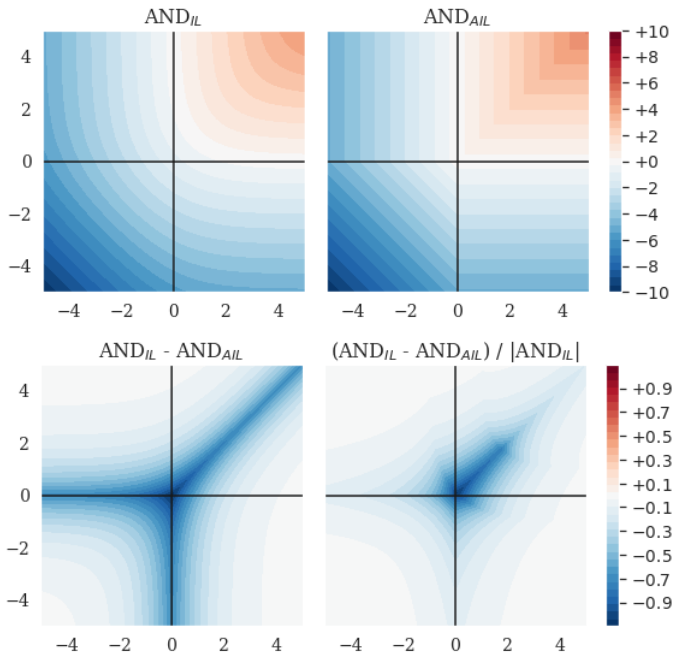


Figure 9: Heatmaps showing $\mathrm{AND}_{\mathrm{IL}}$, $\mathrm{AND}_{\mathrm{AIL}}$, their difference, and their relative difference.

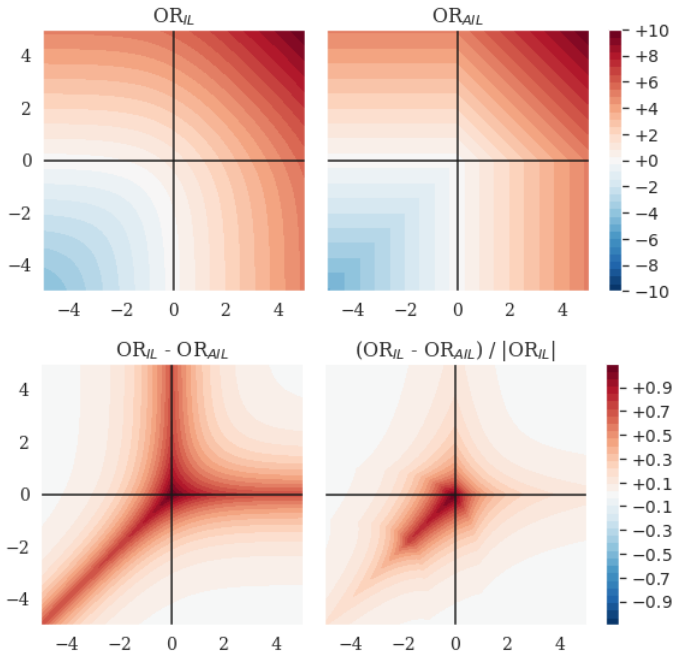


Figure 10: Heatmaps showing $\mathrm{OR}_{\mathrm{IL}}$, $\mathrm{OR}_{\mathrm{AIL}}$, their difference, and their relative difference.

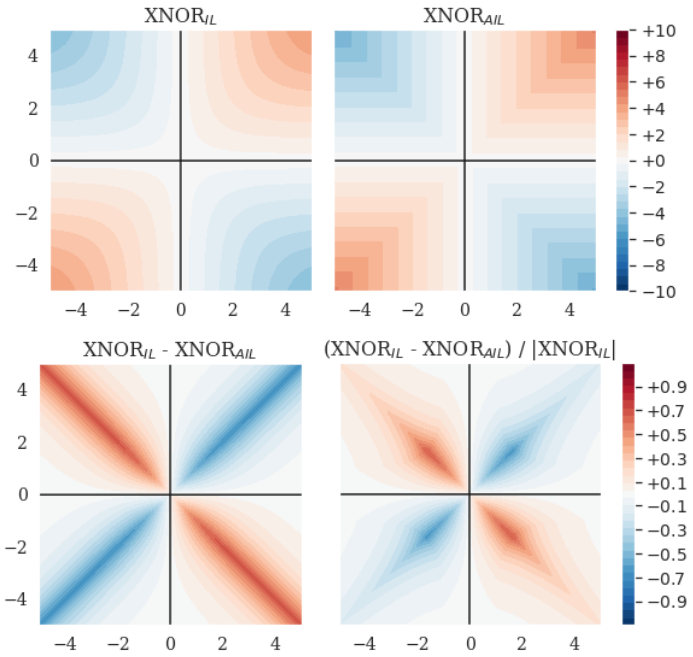


Figure 11: Heatmaps showing $\mathrm{XNOR}_{\mathrm{IL}}$, $\mathrm{XNOR}_{\mathrm{AIL}}$, their difference, and their relative difference.

A.5 Gradient of AIL and IL functions

We show the gradient of each of the logit-space Boolean operators and their AIL approximates in Figure 12, Figure 13, and Figure 14. 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).

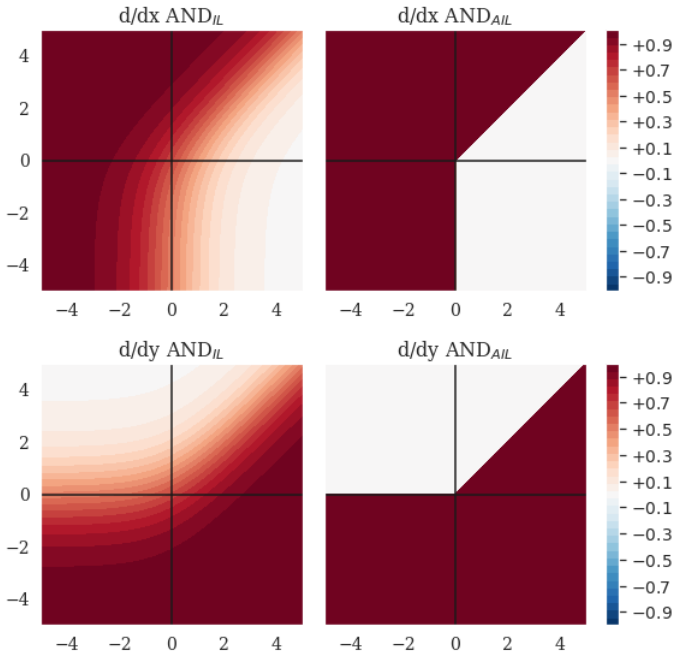


Figure 12: Heatmaps showing the gradient with respect to x and y of $\mathrm{AND}_{\mathrm{IL}}$ and $\mathrm{AND}_{\mathrm{AIL}}$.

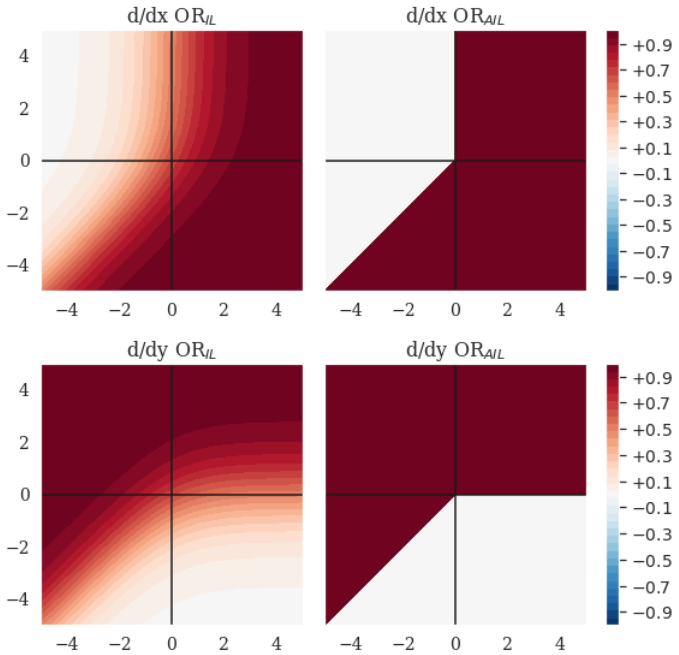


Figure 13: Heatmaps showing the gradient with respect to x and y of $\mathrm{OR}_{\mathrm{IL}}$ or $\mathrm{OR}_{\mathrm{AIL}}$.

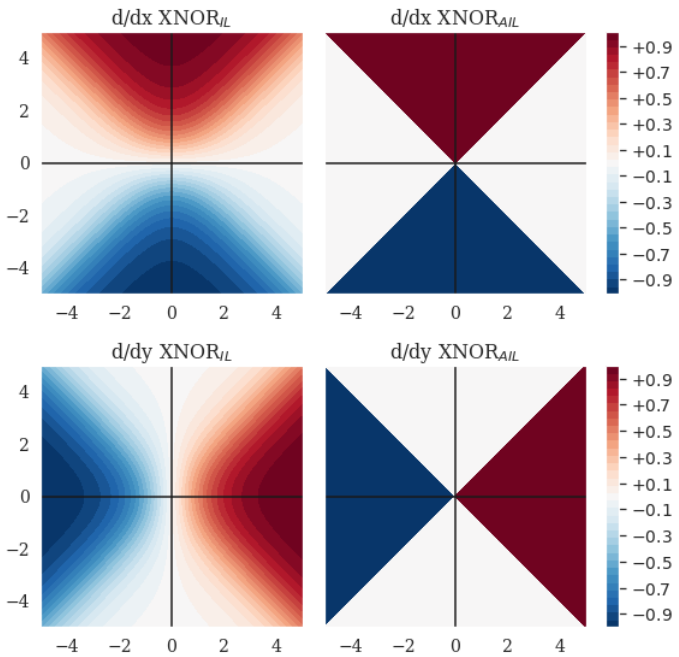
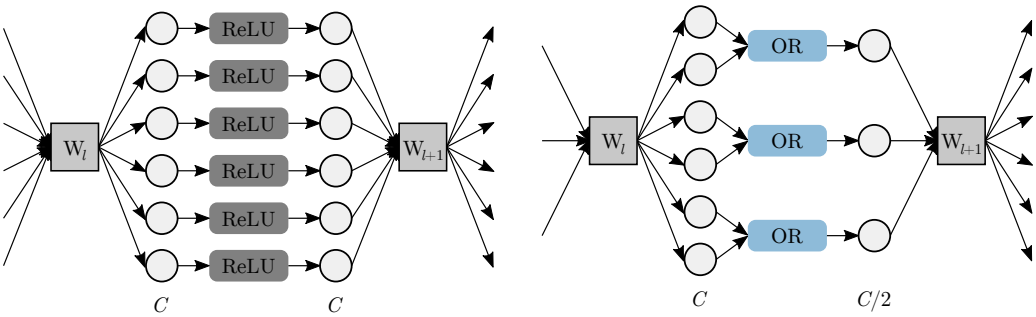


Figure 14: Heatmaps showing the gradient with respect to x and y of $\mathrm{XNOR}_{\mathrm{IL}}$ xnor $\mathrm{XNOR}_{\mathrm{AIL}}$.

A.6 Activation pathing diagrams

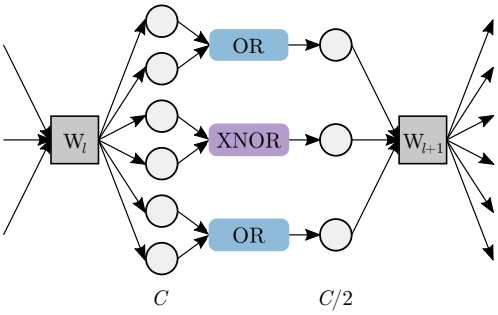
Here we illustrate the implications of using a $2 \rightarrow 1$ activation function on the network achitecture, and the layout of the partition and duplication ensembling methods described in Section 3.5.

A standard 1D activation function with a $1 \rightarrow 1$ mapping such as ReLU processes each pre-activation logit independently, and the number of post-activation channels equals the pre-activation channels (Figure 15a). The total number of parameters per layer is $C^2 + C \sim C^2$.

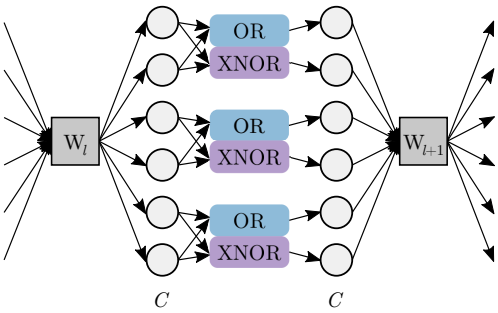


(a) ReLU layer ($1 \rightarrow 1$ activation function).

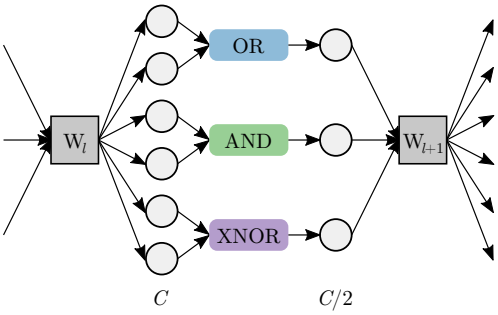
(b) $\mathrm{OR}_{\mathrm{AIL}}$ layer ($2 \rightarrow 1$ activation function).



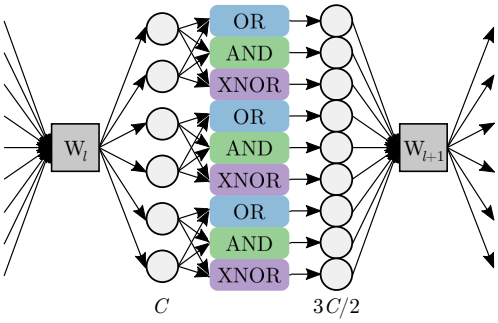
(c) Partition ensembling strategy, using OR and XNOR (overall $2 \rightarrow 1$).



(d) Duplication ensembling strategy, OR and XNOR (overall $2 \rightarrow 2$).



(e) Partition ensembling strategy, using OR, AND,and XNOR (overall $2 \rightarrow 1$).



(f) Duplication ensembling strategy, OR, AND and XNOR (overall $2 \rightarrow 3$).

Figure 15: Network architectures for one layer within an MLP network using various activation functions/ensembles (centered around the activation function). The preceding weight matrix for this layer, W_l, and the weight matrix for the next layer, W_{l+1}, each produce C features (channels). The activation functions may change the number of features, depending on their mapping.

For a $2 \rightarrow 1$ activation function such as $\mathrm{OR}_{\mathrm{AIL}}$, each activation function requires two arguments and produces a single output (Figure 15b). The number of output channels is half that of the input, and there are $\frac{C^2}{2} + C \sim \frac{C^2}{2}$ parameters per layer.

If we use a combination of $2 \rightarrow 1$ activation functions with the partition ensembling strategy, each pre-activation neuron is used once, and only seen by one activation function. The number of output channels is again half that of the input, as seen in Figure 15c and e.

Using the duplication ensembling strategy, each $2 \to 1$ activation function is applied in parallel to the same set of operands (Figure 15d and f). The number of output channels scales up with the number of activation functions used in the ensemble. With two activation functions, the ensemble performs a $2 \to 2$ mapping; the number of channels is unchanged by the layer and the number of parameters per layer is equal to that of a $1 \to 1$ activation function, $C^2 + C \sim C^2$.

A.7 Normalization methodology

To normalize the IL and AIL functions, we assumed the pair of arguments to the activation function were drawn independently from the standard normal distribution, N(0, 1), with probability density function

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}. \tag{11}$$

We found the expected value of the operator, and the standard deviation, and then subtracted by the mean and divided by the standard deviation,

$$\mathrm{OP}_{N*IL} = \frac{\mathrm{OP}_{*IL} - \mu}{\sigma}. \tag{12}$$

For the normalized version of the exact operators, NIL, the mean and standard deviation were estimated empirically using 600M samples. For our AIL functions, we determined the mean and standard deviation analytically by integration. The values are shown in Table 2.

Table 2: Mean and standard deviation of IL and AIL activation functions, for inputs drawn independently from the standard normal distribution.

Activation function	Mean	Standard deviation
$\mathrm{OR}_{\mathrm{IL}}$	1.29895	0.94834
$\mathrm{AND}_{\mathrm{IL}}$	-1.29895	0.94834
$\mathrm{XNOR}_{\mathrm{IL}}$	0	0.36641
$\mathrm{OR}_{\mathrm{AIL}}$	0.68104	0.97229
$\mathrm{AND}_{\mathrm{AIL}}$	-0.68104	0.97229
$\mathrm{XNOR}_{\mathrm{AIL}}$	0	0.60281

A.7.1 Derivations for $\mathrm{OR}_{\mathrm{AIL}}$

Derivations to follow in supplementary material zip.

Final result:

$$\mathbb{E}[\mathrm{OR}_{\mathrm{AIL}}(x_0, x_1)] = \frac{1}{\sqrt{2\pi}} + \frac{1}{2\sqrt{\pi}}$$
$$\mathrm{Var}(\mathrm{OR}_{\mathrm{AIL}}(x_0, x_1)) = \frac{5}{4} - \frac{1}{\sqrt{2\pi}} - \frac{1}{4\pi}$$

A.7.2 Derivations for $\mathrm{XNOR}_{\mathrm{AIL}}$

Derivations to follow in supplementary material zip.

Final result:

$$\mathbb{E}[\mathrm{XNOR}_{\mathrm{AIL}}(x_0, x_1)] = 0$$
$$\mathrm{Var}(\mathrm{XNOR}_{\mathrm{AIL}}(x_0, x_1)) = 1 - \frac{2}{\pi}$$

A.8 Dataset summary

The datasets used in this work are summarized in Table 3.

Table 3: Dataset summaries.

Dataset	№ Samples Train	№ Samples Test	Classes	Reference
Bach Chorales	229	77	2	Boulanger-Lewandowski et al. (2012)
Caltech101	6162	1695	101	Fei-Fei et al. (2006)
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)
I-RAVEN	6000	2000	—	Hu et al. (2020)
MIT-States	30 338	12 995	245 obj, 115 attr	Isola et al. (2015)
MNIST	60 000	10 000	10	LeCun et al. (1998)
Oxford Flowers	6552	818	102	Nilsback & Zisserman (2008)
Stanford Cars	8144	8041	196	Krause et al. (2013)
STL-10	5000	8000	10	Coates et al. (2011)
SVHN	73 257	26 032	10	Netzer et al. (2011)

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

The Covertype dataset was chosen as a dataset that contains only simple features (and not pixels of an image) on which we could study a simple MLP architecture, and was selected based on its popularity on the UCI ML repository.

The Bach Chorales dataset was chosen because — in addition to being in continued use by ML researchers for decades — it presents an interesting opportunity to consider a task where logical activation functions are intuitively applicable, as it is a relatively small dataset that has also been approached with rule-based frameworks, e.g. the expert system by Ebcioglu (1988).

MNIST, CIFAR-10, CIFAR-100 are standard image datasets, commonly used. We used small MLP and CNN architectures for the experiments on MNIST so we could investigate the performance of the network for many configurations (varying the size of the network). We used ResNet-50, a very common deep convolutional architecture within the computer vision field, on CIFAR-10/100 to evaluate the performance in the context of a deep network.

The datasets used for the transfer learning task are all common and popular natural image datasets, with some containing coarse-grained classification (CIFAR-10), others fine-grained (Stanford Cars), and with a varying dataset size (5000—75 000 training samples). We chose to do an experiment involving transfer learning because it is a common practical situation where one must train only a small network that handles high-level features, and is the sort of situation which involves manipulating high-level features, relying on the pretrained network to do the feature extraction.

We considered other domains where logical reasoning is involved as a component of the task, and isolated abstract reasoning and compositional zero-shot learning as suitable tasks.

For abstract reasoning, we wanted to use an IQ style test, and determined that I-RAVEN was a state-of-the-art dataset within this domain (having fixed some problems with the pre-existing RAVEN dataset). We determined that the SRAN architecture from the paper which introduced I-RAVEN was still state-of-the-art on this task, and so used this.

Another problem domain in which we thought it would be interesting to study our activation functions was compositional zero-shot learning (CZSL). This is because the task inherently involves combining an attribute with an object (i.e. the AND operation). For CZSL, we looked at SOTA methods on `https://paperswithcode.com`. The best performance was from SymNet, but this was only implemented in TensorFlow and our code was set up in PyTorch so we built our experiments on the back of the second-best instead, which is the TMN architecture. In the TMN paper, they used two datasets: MIT-States and UT-Zappos-1. In our preliminary experiments, we found that the network started overfitting on MIT-States after around 6 epochs, but on UT-Zappos-1 it was overfitting after the first or second epoch (one can not tell beyond the fact the val performance is best for the first epoch). In the context of zero-shot learning, an epoch uses every image once, but there are also only a finite number of tasks in the dataset. Because there are multiple samples for each noun/adjective pair, and each noun only appears with a handful of adjectives and vice versa, there is in a way fewer tasks in one epoch than there are images. Hence it is possible for a zero-shot learning model to overfit to

the training tasks in less than one epoch (recall that the network includes a pretrained ResNet model for extracting features from the images). For simplicity, we dropped UT-Zappos-1 and focused on MIT-States.

A.9 Experiment configuration and hardware

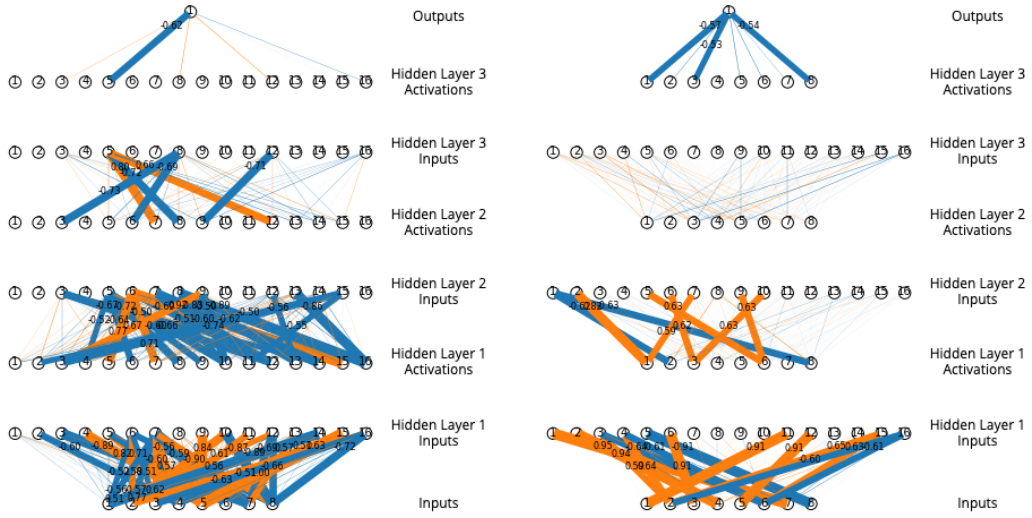
Experiments were run using PyTorch (typically v1.10) on an internal cluster of NVIDIA RTX-6000 and Tesla T4 GPUs. Our experiments each used either 1 or 4 GPUs.

A.10 Parity Experiments

Following on from the parity experiment described in the main text (Section 4.1), 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:

$$\text{XNOR}_{\text{IL}}(\,\text{XNOR}_{\text{IL}}(\text{XNOR}_{\text{IL}}(x_2, x_5), \text{XNOR}_{\text{IL}}(x_3, x_4)),$$
$$\text{XNOR}_{\text{IL}}(\text{XNOR}_{\text{IL}}(x_6, x_7), \text{XNOR}_{\text{IL}}(x_0, x_1)).$$

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 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.



(a) ReLU learned weight matrix

(b) XNOR_{AIL} learned weight matrix

Figure 16: Training results, regression experiment on second synthetic dataset.

While this time the model was not able to learn a sparse weight matrix that exactly reflected our labelling function (see Figure 16), the model was again able to leverage the XNOR_{AIL} activation function to significantly outperform an identical model utilizing the ReLU activation function.

[2]We hypothesize that, because our XNOR_{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 bottleneck in the later layers of the network which restricts the amount of information that made its way through to the final layer

We found that a simple model with three hidden layers, each with eight neurons, utilizing $\mathrm{XNOR}_{\mathrm{AIL}}$ 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 $\mathrm{XNOR}_{\mathrm{AIL}}$ model, we had to increase the model size by 32 times to 256 neurons at each hidden layer.

A.11 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) from the UCI Machine Learning Repository. The Covertype dataset is a classification task consisting of 581 012 samples of forest coverage across 7 classes. Each sample has 54 attributes.

Networks were trained using one-cycle (Smith & Topin, 2017; Smith, 2018) for 50 epochs, with a batch size of 1024, without weight decay. We used a fixed random 80:20 train:test split throughout our experiments. The learning rate was selected using an automated learning rate finder approach, verified against a validation partition formed of 15% of the training data. For our final experiments, the validation partition was included in the training data. No data augmentation was performed.

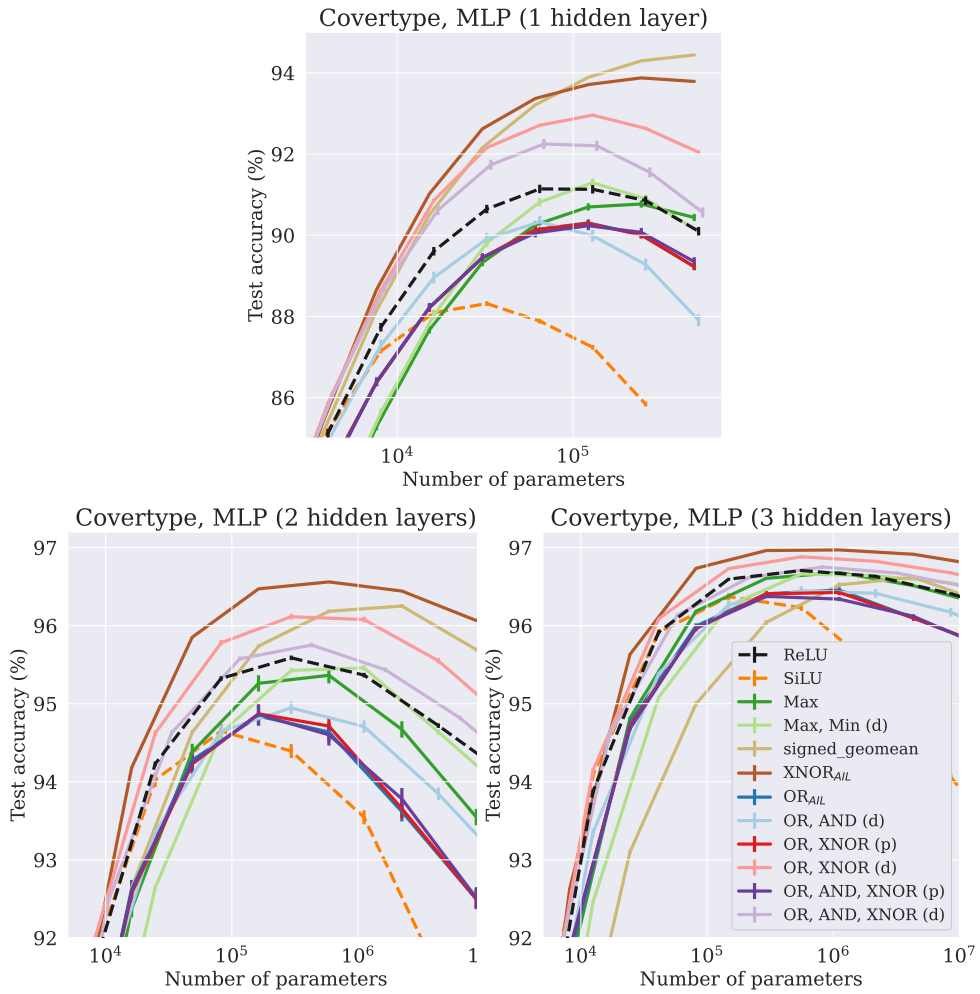


Figure 17: We trained MLPs on the Covertype dataset, with a fixed 80:20 random split. Trained with Adam, 50 ep., 1-cycle, using LRs determined automatically with LR-finder. Mean (bars: std dev) of $n = 10$ weight inits.

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. As shown in Figure 17, Appendix A.11, $\mathrm{XNOR}_{\mathrm{AIL}}$ 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 $\mathrm{signed_geomean}$. The duplication strategy outperforms partition.

A.12 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 $(L, 4, 37)$, where we set the sequence length to be $L = 4$.

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 $4\,L$ 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

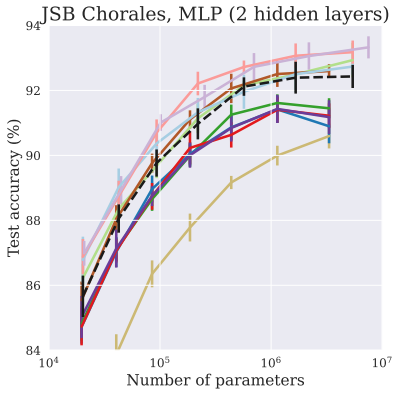


Figure 18: We trained 2-layer MLPs discriminators on JSB Chorales using Adam (constant LR 1×10^{-3}, 150 ep.), Mean (bars: std dev) of $n = 10$ weight inits.

A.13 Correlations between pre-activations

Results on correlations between weights in the JSB Chorale models are shown in Figure 19. We found that when taking all pre-activations 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.

We found that randomly selected pairs of preactivation features within the same layer have correlations that are given by a Gaussian-like distribution centered around zero. This was the case for all of the activation functions we tested. The behaviour of randomly selected pairs of features is thus reasonably consistent with the assumption of independence which we have made. We also investigated the correlation between the pairs of preactivation features which were passing into our two-dimensional activation functions. Here, we found the correlation structure is different, and the correlation depends

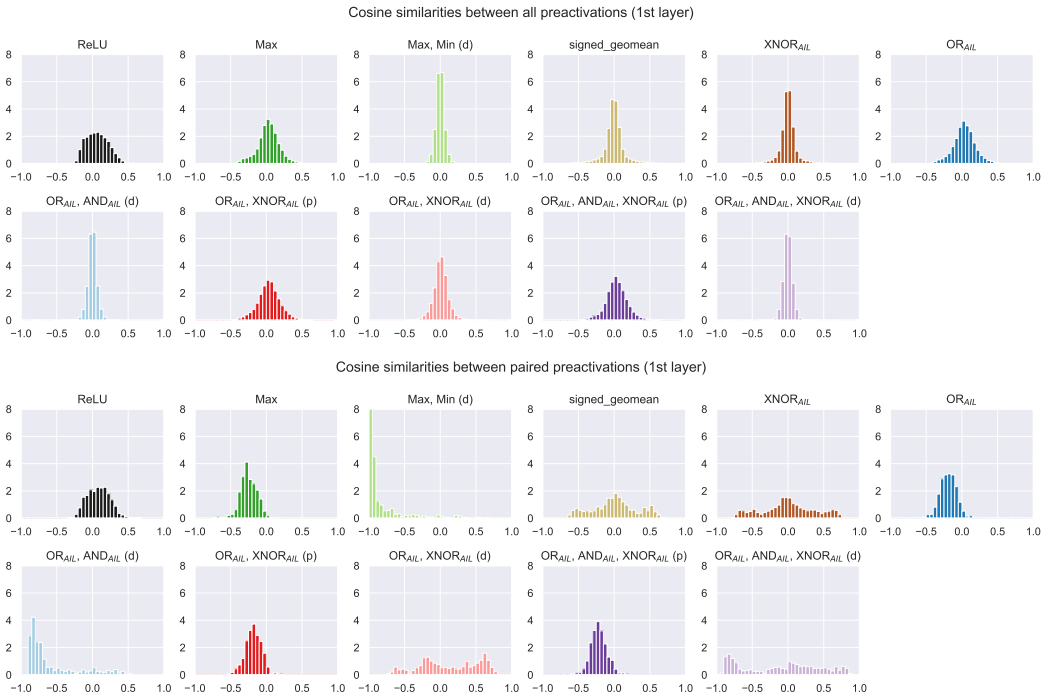


Figure 19: Cosine similarities between pre-activation weights of two activation functions in the first layer of an MLP trained on JSB Chorales.

on the activation function being used. With Max and $\mathrm{OR}_{\mathrm{AIL}}$ activations, the network learns to make the columns of the weight matrix (and hence the preactivation scores for the pair of features) be inversely correlated. With $\mathrm{XNOR}_{\mathrm{AIL}}$, the network learns features which are either positively or negatively correlated (a wider distribution of correlations than seen with random pairs of features). We observe that (in all cases) the network learns to make the features passed to the AIL activation functions be correlated instead of independent, despite our assumption of independence. So it appears clear that the assumption of independence is violated, but also that it doesn't *really* matter because the network is choosing to break the assumption and induce these correlations between the features to get better performance.

A.14 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 selected through a random search of the hyper-parameter space. We chose to do random search instead of grid search because it typically yields better results for the same number of test cases.

For the hyper-parameter search, we trained on the first $50\,000$ samples of the training partition and used the final $10\,000$ samples as a validation set. We ran the search for four iterations, with each iteration sampling 120 different hyper-parameter settings. The initial bounds for our hyper-parameter samples were set as described in Table 4. These bounds were chosen to be suitably wide such that the optimal configuration should be contained within them for all activation functions considered.

The bounds on our uniform random variable x were tightened at each iteration by selecting the top-5 performing hyper-parameter settings, taking 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

Table 4: Hyper-parameter random search parameters.

Hyper-parameter	Variable	Sampling (initial bounds)
Adam beta1	$1 - 10^x$	$x \sim \text{Uniform}(-3, -0.5)$
Adam beta2	$1 - 10^x$	$x \sim \text{Uniform}(-5, -1)$
Adam epsilon	10^x	$x \sim \text{Uniform}(-10, -6)$
Weight decay	10^x	$x \sim \text{Uniform}(-7, -3)$
One-cycle max LR	10^x	$x \sim \text{Uniform}(-4, 0)$
One-cycle peak	x	$x \sim \text{Uniform}(0.1, 0.5)$

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.

We found that the XNOR activation function required quite different hyper-parameters than the other activation functions, but OR and Max activation functions used similar hyper-parameters to ReLU. However, we have no reason to believe that the proposed AIL activation functions are more susceptible than others to the choice of hyper-parameters or the way in which hyper-parameters are selected.

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 $[c, 2c, 4c, 4c, 8c, 8c]$, and the number of pre-activation neurons produced by the subsequent linear layers were $[32c, 16c]$. Similar to the MLP model, c 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 1×10^6 in the CNN model, we changed the structure of the convolution layers and linear layers to $[c, c, c, 1.5c, 1.5c]$ and $[2c, 1.5c]$, 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.15 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 derived for CIFAR-10 by AutoAugment (Cubuk et al., 2018), with batch size of 128. The hyper-parameters for the optimizer and scheduler were determined through a random search of the hyper parameter space on the CIFAR-100 dataset[3]. The hyper-parameter search was tuned against a partition of 10% of the CIFAR-100 training samples. Our hyper-parameter search follows a similar approach to the one used for our MLP and CNN models on MNIST (Appendix A.14), but due to compute constraints with our larger models here we only trained 100 seeds each iteration, and only re-examined the top 10 seeds in the final round.

For PReLU, we did not perform a hyper-parameter search and merely re-used the same hyper-parameters as discovered in the search with ReLU.

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

Once our optimal hyper-parameters were selected we trained a ResNet50 model with automatic 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 $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (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. For the $2 \rightarrow 1$ activation functions (Max, $\mathrm{OR}_{\mathrm{AIL}}$, $\mathrm{XNOR}_{\mathrm{AIL}}$, $\{\mathrm{OR}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)$\}$, and $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)$\}$) we use width values of 0.75, 1.5, 3, and 6. For the $2 \rightarrow 3$ activation function $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)$\}$, we use width values of 0.4, 0.75, 1.5, and 3.

A.16 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 for these experiments, 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. For the SVHN dataset, horizontal flip was not performed. Pixel intensity normalization was done against the ImageNet mean and standard deviation.

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 normalization 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 650 for activation functions which map $2 \rightarrow 1$, and 438 for $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)$\}$ which maps features $3 \rightarrow 2$. These values control the total number of parameters in the head to be the same for datasets with 100 classes (the median number of classes in the datasets we considered). Our results with widths 438/512/650 and $\sim$600k parameters are shown in the main results, Table 1. To investigate a comprehensive set of baselines, we compared against every activation function implemented in PyTorch v1.10. This was too many comparisons to include in the main results, so for brevity we narrowed the activation functions down by omitting some activation functions where there were several similar functions and kept the better performing one (SiLU vs Mish, LeakyReLU vs PReLU, etc). The full set of results is shown in Table 5. Additionally, the precise widths and number of trainable parameters used for these experiments are shown Table 6.

Since the transfer learning experiments were performed without retraining the base network, we found (Table 1) the performance is limited by the features which the base network produces, whose relevance depends on the overlap between the domain of the pretraining task (ImageNet) and the new task. This information bottleneck, and variation in its utility, means the variation between performance on different datasets is much larger than the variation for different activation functions within the same dataset.

For transfer tasks which involve coarse-grained discrimination on images which are similar to ImageNet, the embedding generated by the pretrained model is already sufficient to separate the classes in the new dataset. Examples of this are Caltech101, where linear layer beats out using additional layers with non-linearities; and STL-10, where our best model is on-par with the linear model. The results on these two transfer learning tasks are too simple to be of interest to study. The performance is limited by the information retained by the pretrained embedding, but it appears that what task-relevant information is there is readily available with a linear layer without needing additional logic to interpret it.

Other transfer learning tasks we attempted are less trivial and show a larger difference in performance across the activation functions, and a gap from the linear layer readout model. The Stanford Cars dataset involves fine-grained discrimination between different car models, for which features generated by a model pretrained on ImageNet are not effective. The SVHN dataset, which contains

Table 5: Transfer learning from a frozen ResNet-18 architecture pretrained on ImageNet-1k to other computer vision datasets. Mean (std. error) of $n = 5$ random initializations of the MLP (same pretrained encoder). Bold: best. Underlined: top two. Italic: no sig. diff. from best (two-sided Student's t-test, $p > 0.05$). Background: linear color scale from ReLU baseline (white) to best (black).

	Test Accuracy (%)						
Activation function	Cal101	CIFAR10	CIFAR100	Flowers	StfCars	STL-10	SVHN
Linear layer only	***88.35***±0.15	78.56±0.09	57.39±0.09	92.32±0.20	33.51±0.06	94.68±0.02	46.60±0.14
ReLU	86.58±0.17	81.63±0.05	58.04±0.11	90.71±0.26	30.97±0.26	94.62±0.06	53.26±0.08
LeakyReLU	86.60±0.13	81.67±0.11	58.01±0.09	90.73±0.32	31.09±0.24	94.61±0.05	53.24±0.10
PReLU	*87.83*±0.21	81.03±0.13	58.90±0.18	*93.17*±0.19	***39.84***±0.18	94.54±0.05	53.47±0.08
Softplus	86.16±0.18	79.13±0.08	56.58±0.07	89.39±0.29	21.23±0.13	94.63±0.03	48.79±0.08
ELU	87.18±0.09	80.44±0.08	58.08±0.10	91.71±0.14	34.70±0.06	94.55±0.05	51.89±0.09
CELU	87.18±0.09	80.44±0.08	58.08±0.10	91.71±0.14	34.70±0.06	94.55±0.05	51.89±0.09
SELU	87.74±0.09	79.93±0.13	58.24±0.06	92.27±0.13	37.51±0.17	94.53±0.07	50.94±0.12
GELU	87.10±0.15	81.39±0.09	58.51±0.13	91.51±0.15	33.43±0.15	94.62±0.06	53.43±0.23
SiLU	86.91±0.11	80.53±0.11	58.14±0.12	91.37±0.18	32.15±0.17	94.59±0.05	52.35±0.16
Hardswish	87.12±0.12	80.10±0.10	58.25±0.10	91.56±0.25	33.17±0.23	94.62±0.05	51.82±0.13
Mish	87.11±0.12	81.09±0.11	58.37±0.10	91.61±0.15	33.75±0.14	94.61±0.05	53.05±0.12
Softsign	81.47±0.18	80.03±0.09	54.84±0.09	82.34±0.22	17.33±0.10	*94.70*±0.03	51.25±0.08
Tanh	87.48±0.06	80.56±0.07	57.35±0.08	90.32±0.20	29.51±0.12	*94.63*±0.07	51.86±0.05
GLU	86.71±0.31	79.19±0.07	57.64±0.10	90.34±0.19	27.04±0.12	94.57±0.03	50.12±0.19
Max	86.96±0.20	81.76±0.14	58.60±0.12	90.98±0.18	33.37±0.15	*94.70*±0.06	53.53±0.16
Max, Min (d)	87.23±0.13	82.31±0.10	59.05±0.10	91.68±0.18	34.91±0.12	94.64±0.04	53.91±0.13
XNOR_{IL}	85.01±0.17	79.62±0.09	57.14±0.07	84.76±0.43	1.34±0.11	94.51±0.03	51.99±0.16
OR_{IL}	87.11±0.08	79.75±0.05	58.07±0.11	91.12±0.36	33.12±0.12	94.60±0.03	51.21±0.17
OR, AND_{IL} (d)	87.22±0.14	79.56±0.06	57.91±0.12	91.44±0.18	36.04±0.07	94.44±0.05	50.42±0.25
OR, XNOR_{IL} (p)	87.11±0.08	79.75±0.05	58.07±0.11	91.12±0.36	33.12±0.12	94.60±0.03	51.12±0.16
OR, XNOR_{IL} (d)	86.76±0.12	80.83±0.08	58.55±0.10	90.76±0.07	33.56±0.14	94.47±0.07	52.54±0.09
OR, AND, XNOR_{IL} (p)	86.82±0.14	79.65±0.07	58.04±0.09	91.10±0.18	33.05±0.11	94.58±0.03	50.75±0.14
OR, AND, XNOR_{IL} (d)	86.78±0.10	80.18±0.02	58.70±0.17	91.68±0.07	34.82±0.14	94.49±0.04	52.09±0.18
XNOR_{NIL}	87.25±0.22	***82.88***±0.08	***60.78***±0.08	***93.26***±0.26	*39.47*±0.20	***94.83***±0.06	***55.34***±0.19
OR_{NIL}	87.19±0.16	79.61±0.05	58.44±0.10	91.65±0.29	35.82±0.04	94.58±0.03	50.95±0.15
OR, AND_{NIL} (d)	86.82±0.18	80.09±0.09	58.60±0.07	91.44±0.20	37.03±0.11	94.65±0.05	52.49±0.09
OR, XNOR_{NIL} (p)	87.25±0.15	79.53±0.07	58.62±0.11	91.50±0.28	35.78±0.05	*94.65*±0.06	50.38±0.11
OR, XNOR_{NIL} (d)	*87.82*±0.19	*82.67*±0.05	*60.60*±0.11	*92.93*±0.12	39.22±0.17	94.63±0.06	*54.87*±0.09
OR, AND, XNOR_{NIL} (p)	87.03±0.17	79.53±0.09	58.66±0.07	91.57±0.21	35.79±0.11	*94.65*±0.07	50.27±0.07
OR, AND, XNOR_{NIL} (d)	87.41±0.27	*82.84*±0.06	60.38±0.10	*92.98*±0.17	*39.42*±0.18	*94.71*±0.03	*55.11*±0.12
XNOR_{AIL}	86.97±0.18	81.83±0.06	58.46±0.10	90.93±0.15	32.56±0.10	*94.71*±0.06	53.75±0.14
OR_{AIL}	87.45±0.14	81.88±0.07	59.10±0.09	92.00±0.15	36.01±0.12	*94.69*±0.04	53.68±0.14
OR, AND_{AIL} (d)	87.43±0.11	82.38±0.06	59.90±0.08	92.07±0.18	37.16±0.15	94.55±0.05	54.05±0.07
OR, XNOR_{AIL} (p)	87.42±0.12	81.92±0.07	59.09±0.10	91.93±0.12	35.99±0.17	94.68±0.03	53.65±0.14
OR, XNOR_{AIL} (d)	87.09±0.21	82.20±0.04	59.44±0.07	91.90±0.10	36.88±0.10	*94.69*±0.06	54.03±0.11
OR, AND, XNOR_{AIL} (p)	87.43±0.14	81.78±0.06	59.27±0.13	91.98±0.29	35.90±0.09	94.66±0.03	53.71±0.15
OR, AND, XNOR_{AIL} (d)	87.49±0.11	82.50±0.08	59.83±0.12	92.37±0.08	37.60±0.20	*94.72*±0.02	54.55±0.09
$\text{XNOR}_{\text{NAIL}}$	87.61±0.23	82.38±0.07	59.77±0.13	*93.07*±0.20	*39.77*±0.04	*94.81*±0.03	53.91±0.05
OR_{NAIL}	87.19±0.16	81.79±0.09	59.40±0.09	92.12±0.12	37.32±0.17	94.65±0.04	53.82±0.21
OR, AND_{NAIL} (d)	87.62±0.11	82.28±0.10	59.71±0.05	92.10±0.20	37.70±0.12	*94.61*±0.08	53.86±0.10
OR, $\text{XNOR}_{\text{NAIL}}$ (p)	87.23±0.19	81.85±0.08	59.46±0.08	92.20±0.09	37.22±0.15	94.64±0.05	53.44±0.07
OR, $\text{XNOR}_{\text{NAIL}}$ (d)	*87.85*±0.22	82.52±0.11	60.02±0.10	*93.12*±0.13	*39.64*±0.09	*94.75*±0.03	54.13±0.05
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (p)	87.39±0.13	81.73±0.10	59.45±0.08	92.27±0.08	37.13±0.12	94.63±0.06	53.46±0.03
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (d)	87.78±0.14	*82.67*±0.06	60.01±0.21	*93.12*±0.21	*39.65*±0.14	*94.78*±0.03	54.58±0.12

images of house numbers, is coarse-grained but uses images which are outside the domain of ImageNet, which makes the transfer-learning task more difficult.

As shown in Table 5, we found the duplication ensemble strategy consistently outperformed the partition strategy on the transfer learning experiments. Compared to using OR_{NAIL}, there was no benefit to using an ensemble with the partition strategy. However, it was beneficial to ensemble OR_{NAIL} with AND_{NAIL}, and better still with $\text{XNOR}_{\text{NAIL}}$, under the duplication strategy. The performance of duplication-ensembles using $\text{XNOR}_{\text{NAIL}}$, i.e. {$\text{OR}_{\text{NAIL}}$, AND_{NAIL}, $\text{XNOR}_{\text{NAIL}}$ (d)} and {OR_{NAIL}, AND_{NAIL}, $\text{XNOR}_{\text{NAIL}}$ (d)}, was comparable to using $\text{XNOR}_{\text{NAIL}}$ alone, so in this case an ensemble was neither beneficial nor detrimental.

Additionally, we ran the experiment again with a pre-activation width of $w = 512$ for all activation functions. The results with constant pre-activation width $w = 512$ are shown in Table 7. By using

Table 6: Hidden pre-activation widths and number of trainable parameters used in transfer learning experiments (corresponding to results shown in Table 1).

			№ Trainable Parameters						
Activation function	Map	Width	Cal101	CIFAR10	CIFAR100	Flowers	StfCars	STL-10	SVHN
Linear layer only			52k	5k	51k	52k	101k	5k	5k
ReLU	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
LeakyReLU	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
PReLU	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
Softplus	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
ELU	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
CELU	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
SELU	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
GELU	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
SiLU	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
Hardswish	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
Mish	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
Softsign	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
Tanh	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
GLU	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
Max	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
Max, Min (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{XNOR}_{\mathrm{IL}}$	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}_{\mathrm{IL}}$	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{AND}_{\mathrm{IL}}$ (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{IL}}$ (p)	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{IL}}$ (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{IL}}$ (p)	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{IL}}$ (d)	$2 \to 3$	438	579k	519k	579k	580k	642k	519k	519k
$\mathrm{XNOR}_{\mathrm{NIL}}$	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}_{\mathrm{NIL}}$	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{AND}_{\mathrm{NIL}}$ (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{NIL}}$ (p)	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{NIL}}$ (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{NIL}}$ (p)	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{NIL}}$ (d)	$2 \to 3$	438	579k	519k	579k	580k	642k	519k	519k
$\mathrm{XNOR}_{\mathrm{AIL}}$	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}_{\mathrm{AIL}}$	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{AND}_{\mathrm{AIL}}$ (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)	$2 \to 3$	438	579k	519k	579k	580k	642k	519k	519k
$\mathrm{XNOR}_{\mathrm{NAIL}}$	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}_{\mathrm{NAIL}}$	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{AND}_{\mathrm{NAIL}}$ (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{NAIL}}$ (p)	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{NAIL}}$ (d)	$1 \to 1$	512	577k	530k	577k	578k	626k	530k	530k
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{NAIL}}$ (p)	$2 \to 1$	650	578k	549k	578k	579k	609k	549k	549k
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{NAIL}}$ (d)	$2 \to 3$	438	579k	519k	579k	580k	642k	519k	519k

a constant pre-activation width, the number of parameters used is not consistent across activation functions. The number of trainable parameters in each experiment is shown in Table 8. Note: some experiments on SVHN are missing due to an issue with the job scheduler which was not noticed in time to run the missing experiments.

We found that reducing the width from 650 to 512 (and hence reducing total number of trainable parameters) reduced the performance of $2 \to 1$ activation functions $\mathrm{XNOR}_{\mathrm{AIL}}$, $\mathrm{OR}_{\mathrm{AIL}}$, $\{\mathrm{OR}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)$\}$, and $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)$\}$. Increasing the width from 438 to 512 increased the performance of $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)$\}$.

Table 7: Transfer learning from a frozen ResNet-18 architecture pretrained on ImageNet-1k to other computer vision datasets. As with Table 1, but in this case we show 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. The number of parameters is shown in Table 8. Mean (standard error) of $n = 5$ inits of the MLP (same pretrained network).

	Test Accuracy (%)						
Activation function	Cal101	CIFAR10	CIFAR100	Flowers	StfCars	STL-10	SVHN
Linear layer only	***88.35***±0.15	78.56±0.09	57.39±0.09	92.32±0.20	33.51±0.06	94.68±0.02	46.60±0.14
ReLU	86.58±0.17	81.63±0.05	58.04±0.11	90.71±0.26	30.97±0.26	94.62±0.06	53.26±0.08
LeakyReLU	86.60±0.13	81.67±0.11	58.01±0.09	90.73±0.32	31.09±0.24	94.61±0.05	53.24±0.10
PReLU	*87.83*±0.21	81.03±0.13	58.90±0.18	***93.17***±0.19	***39.84***±0.18	94.54±0.05	53.47±0.08
Softplus	86.16±0.18	79.13±0.08	56.58±0.07	89.39±0.29	21.23±0.13	94.63±0.03	48.79±0.08
ELU	87.18±0.09	80.44±0.08	58.08±0.10	91.71±0.14	34.70±0.06	94.55±0.05	51.89±0.09
CELU	87.18±0.09	80.44±0.08	58.08±0.10	91.71±0.14	34.70±0.06	94.55±0.05	51.89±0.09
SELU	87.74±0.09	79.93±0.13	58.24±0.06	92.27±0.13	37.51±0.17	94.53±0.07	50.94±0.12
GELU	87.10±0.15	81.39±0.09	58.51±0.13	91.51±0.15	33.43±0.15	94.62±0.06	53.43±0.23
SiLU	86.91±0.11	80.53±0.11	58.14±0.12	91.37±0.18	32.15±0.17	94.59±0.05	52.35±0.16
Hardswish	87.12±0.12	80.10±0.10	58.25±0.10	91.56±0.25	33.17±0.23	94.62±0.05	51.82±0.13
Mish	87.11±0.12	81.09±0.11	58.37±0.10	91.61±0.15	33.75±0.14	94.61±0.05	53.05±0.12
Softsign	81.47±0.18	80.03±0.09	54.84±0.09	82.34±0.22	17.33±0.10	94.70±0.03	51.25±0.08
Tanh	87.48±0.06	80.56±0.07	57.35±0.08	90.32±0.20	29.51±0.12	94.63±0.07	51.86±0.05
GLU	86.34±0.16	79.35±0.05	57.22±0.12	90.10±0.20	26.72±0.13	94.63±0.05	50.64±0.10
Max	86.86±0.11	81.56±0.06	58.12±0.10	90.59±0.25	32.80±0.04	94.65±0.05	53.55±0.09
Max, Min (d)	87.23±0.13	82.31±0.10	59.05±0.10	91.68±0.18	34.91±0.12	94.64±0.04	53.91±0.13
XNOR_{IL}	85.11±0.10	79.77±0.08	56.92±0.09	84.29±0.29	1.78±0.21	94.56±0.05	
OR_{IL}	87.01±0.21	79.88±0.04	57.68±0.08	91.12±0.10	32.55±0.04	94.57±0.06	
OR, AND_{IL} (d)	87.22±0.14	79.56±0.06	57.91±0.12	91.44±0.18	36.04±0.07	94.44±0.05	50.42±0.25
OR, XNOR_{IL} (p)	86.99±0.21	79.88±0.04	57.68±0.08	91.12±0.10	32.55±0.04	94.58±0.06	
OR, XNOR_{IL} (d)	86.76±0.12	80.83±0.08	58.55±0.10	90.76±0.07	33.56±0.14	94.47±0.07	52.54±0.09
OR, AND, XNOR_{IL} (p)	86.84±0.20	79.75±0.05	57.71±0.08	91.46±0.10	32.58±0.08	94.58±0.04	
OR, AND, XNOR_{IL} (d)	86.98±0.24	80.22±0.10	58.89±0.10	91.56±0.26	35.27±0.08	94.49±0.05	
XNOR_{NIL}	87.94±0.08	*82.89*±0.09	60.22±0.12	*93.11*±0.22	38.93±0.12	94.78±0.03	55.23±0.11
OR_{NIL}	87.23±0.14	79.76±0.05	58.62±0.02	91.47±0.07	35.57±0.11	94.66±0.07	51.18±0.05
OR, AND_{NIL} (d)	86.82±0.18	80.09±0.09	58.60±0.07	91.44±0.20	37.03±0.11	94.65±0.05	52.49±0.09
OR, XNOR_{NIL} (p)	87.14±0.07	79.76±0.05	58.62±0.02	91.47±0.07	35.57±0.11	94.66±0.07	51.17±0.08
OR, XNOR_{NIL} (d)	*87.82*±0.19	82.67±0.05	*60.60*±0.11	*92.93*±0.12	39.22±0.17	94.63±0.06	54.87±0.09
OR, AND, XNOR_{NIL} (p)	86.97±0.10	79.68±0.09	58.54±0.05	91.56±0.20	35.64±0.17	94.65±0.04	51.10±0.06
OR, AND, XNOR_{NIL} (d)	87.33±0.26	***83.11***±0.10	***60.90***±0.16	*93.15*±0.28	*39.27*±0.18	*94.71*±0.09	***55.77***±0.13
XNOR_{AIL}	86.42±0.13	81.74±0.05	57.88±0.09	90.50±0.17	31.55±0.11	*94.78*±0.04	
OR_{AIL}	87.28±0.09	81.81±0.02	58.68±0.05	91.78±0.23	35.28±0.09	94.67±0.07	
OR, AND_{AIL} (d)	87.43±0.11	82.38±0.06	59.90±0.08	92.07±0.18	37.16±0.15	94.55±0.05	54.05±0.07
OR, XNOR_{AIL} (p)	87.26±0.06	81.85±0.06	58.72±0.09	91.80±0.24	35.27±0.09	94.64±0.07	
OR, XNOR_{AIL} (d)	87.09±0.21	82.20±0.04	59.44±0.07	91.90±0.10	36.88±0.10	94.69±0.06	54.03±0.11
OR, AND, XNOR_{AIL} (p)	87.03±0.19	81.75±0.07	58.72±0.08	91.85±0.14	35.46±0.14	94.69±0.04	
OR, AND, XNOR_{AIL} (d)	87.26±0.15	82.45±0.08	60.20±0.11	92.41±0.17	37.89±0.14	94.65±0.04	
$\text{XNOR}_{\text{NAIL}}$	87.47±0.12	82.38±0.05	59.34±0.08	*92.84*±0.18	39.03±0.05	***94.88***±0.02	54.23±0.15
OR_{NAIL}	87.33±0.14	81.78±0.03	59.27±0.06	91.83±0.14	36.79±0.20	*94.78*±0.07	53.79±0.15
OR, AND_{NAIL} (d)	87.62±0.11	82.28±0.10	59.71±0.05	92.10±0.20	37.70±0.12	94.61±0.08	53.86±0.10
OR, $\text{XNOR}_{\text{NAIL}}$ (p)	87.33±0.14	81.82±0.04	59.28±0.06	91.83±0.14	36.80±0.19	*94.78*±0.07	53.77±0.14
OR, $\text{XNOR}_{\text{NAIL}}$ (d)	*87.85*±0.22	82.52±0.11	60.02±0.10	*93.12*±0.13	*39.64*±0.09	94.75±0.03	54.13±0.05
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (p)	87.30±0.17	81.79±0.08	59.32±0.07	92.10±0.30	36.73±0.03	94.77±0.03	53.86±0.03
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (d)	87.42±0.10	82.83±0.05	*60.58*±0.15	*92.54*±0.20	*39.49*±0.15	*94.81*±0.11	55.10±0.04

Table 8: Number of trainable parameters used in transfer learning experiments (corresponding to results shown in Table 7).

			№ Trainable Parameters						
Activation function	Map	Width	Cal101	CIFAR10	CIFAR100	Flowers	StfCars	STL-10	SVHN
Linear layer only			52k	5k	51k	52k	101k	5k	5k
ReLU	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
LeakyReLU	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
PReLU	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
Softplus	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
ELU	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
CELU	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
SELU	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
GELU	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
SiLU	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
Hardswish	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
Mish	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
Softsign	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
Tanh	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
GLU	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
Max	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
Max, Min (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
XNOR_{IL}	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	
OR_{IL}	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	
OR, AND_{IL} (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
OR, XNOR_{IL} (p)	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	
OR, XNOR_{IL} (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
OR, AND, XNOR_{IL} (p)	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	
OR, AND, XNOR_{IL} (d)	$2 \rightarrow 3$	512	734k	664k	733k	735k	807k	664k	
XNOR_{NIL}	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
OR_{NIL}	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
OR, AND_{NIL} (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
OR, XNOR_{NIL} (p)	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
OR, XNOR_{NIL} (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
OR, AND, XNOR_{NIL} (p)	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
OR, AND, XNOR_{NIL} (d)	$2 \rightarrow 3$	512	734k	664k	733k	735k	807k	664k	664k
XNOR_{AIL}	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	
OR_{AIL}	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	
OR, AND_{AIL} (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
OR, XNOR_{AIL} (p)	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	
OR, XNOR_{AIL} (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
OR, AND, XNOR_{AIL} (p)	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	
OR, AND, XNOR_{AIL} (d)	$2 \rightarrow 3$	512	734k	664k	733k	735k	807k	664k	
$\text{XNOR}_{\text{NAIL}}$	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
OR_{NAIL}	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
OR, AND_{NAIL} (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
OR, $\text{XNOR}_{\text{NAIL}}$ (p)	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
OR, $\text{XNOR}_{\text{NAIL}}$ (d)	$1 \rightarrow 1$	512	577k	530k	577k	578k	626k	530k	530k
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (p)	$2 \rightarrow 1$	512	420k	397k	420k	420k	445k	397k	397k
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (d)	$2 \rightarrow 3$	512	734k	664k	733k	735k	807k	664k	664k

A.17 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.

In recent years, several challenges have been proposed to evaluate the ability of neural networks to perform abstract reasoning. The 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. Other abstract reasoning tasks have also been proposed (Fleuret et al., 2011; Johnson et al., 2017; Barrett et al., 2018).

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).

Table 9: Performance of SRAN-based models on the I-RAVEN dataset (Hu et al., 2020). Bold: best. Underlined: second best. Background: color scale from worst in to best, linear with accuracy value.

		I-RAVEN Test Acc (%)							
Activation function	Params	Acc	Center	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
PReLU	45.6M	54.5	69.2	45.4	40.5	64.4	47.6	57.5	57.3
CELU	45.6M	56.6	73.5	46.5	41.0	66.6	46.1	62.8	59.7
SELU	45.6M	53.5	67.2	43.8	40.1	62.8	44.5	58.0	58.1
GELU	45.6M	61.4	80.8	49.2	42.5	69.2	48.3	70.3	69.2
SiLU	45.6M	59.4	78.0	47.2	41.4	66.3	47.8	69.1	66.2
Max	44.6M	57.8	76.3	45.2	39.7	65.3	48.7	64.6	64.7
Max, Min (d)	45.6M	60.2	80.2	49.5	41.9	66.5	47.0	68.5	67.7
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**
OR, AND_{AIL} (d)	45.6M	57.5	74.7	45.6	41.3	68.3	44.9	64.5	63.0
OR, XNOR_{AIL} (p)	44.6M	59.8	80.5	45.8	41.4	67.2	48.2	67.5	68.1
OR, XNOR_{AIL} (d)	45.6M	62.8	83.7	49.1	43.3	68.1	**49.5**	73.8	72.2
OR, AND, XNOR_{AIL} (p)	44.6M	55.0	68.2	47.2	41.1	65.0	45.0	61.0	57.5

We found the network using OR_{AIL} activation function performed best overall, and across most of the subtasks. The second best performing activation functions were $\{\text{OR}_{\text{AIL}}, \text{XNOR}_{\text{AIL}}$ (d)$\}$ and ReLU.

A.18 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 both the task and its solution based on their previous experiences (a meta-learning task). 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 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 10). 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.

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). We used 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 10, we find that the trio of activation functions applied in parallel, $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)$\}$, performs best.

Table 10: 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)	$1 \to 1$		2.9	7.1	11.5
TMN repro. (ReLU)	$1 \to 1$	438 k	2.47 ± 0.07	6.42 ± 0.09	10.27 ± 0.11
Max	$2 \to 1$	650 k	2.42 ± 0.07	6.37 ± 0.08	10.28 ± 0.06
Max, Min (d)	$1 \to 1$	438 k	2.53 ± 0.06	6.69 ± 0.11	10.61 ± 0.14
$\mathrm{XNOR}_{\mathrm{AIL}}$	$2 \to 1$	650 k	1.22 ± 0.05	3.47 ± 0.12	5.82 ± 0.19
$\mathrm{OR}_{\mathrm{AIL}}$	$2 \to 1$	650 k	$\underline{2.65} \pm 0.05$	$\underline{6.80} \pm 0.08$	$\underline{10.78} \pm 0.13$
$\mathrm{OR}, \mathrm{AND}_{\mathrm{AIL}}$ (d)	$1 \to 1$	438 k	2.61 ± 0.05	6.73 ± 0.05	10.77 ± 0.12
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)	$2 \to 1$	650 k	$\underline{2.65} \pm 0.05$	$\underline{6.80} \pm 0.08$	$\underline{10.78} \pm 0.13$
$\mathrm{OR}, \mathrm{XNOR}_{\mathrm{AIL}}$ (d)	$1 \to 1$	438 k	1.89 ± 0.13	5.12 ± 0.21	8.35 ± 0.24
$\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)	$2 \to 1$	650 k	$\underline{\mathbf{2.67}} \pm 0.10$	$\underline{\mathbf{6.95}} \pm 0.14$	$\underline{\mathbf{10.96}} \pm 0.16$

Since we could not flexibly scale the total number of parameters in the network with the original architecture, we modified the TMN architecture by adding an additional linear layer to the end of each module which projects from the activation function to the embedding space. The modules would otherwise terminate with an activation function, which makes it difficult to handle activation functions which map $2 \to 1$. We dub the modified version of the network "TMNx."

Comparing the TMN results in Table 10 to TMNx in Table 11, 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 $1 \to 1$, and increased this to 70 for activation function which map $2 \to 1$, to maintain the total number of trainable parameters. To investigate a comprehensive set of baselines, we compared against every activation function implemented in PyTorch v1.10. The results are shown in Table 11.

We found that the ELU family of activations (ELU, CELU, SELU) performed best across all three top-k values. Our best activation functions were $\{\mathrm{OR}, \mathrm{AND}_{\mathrm{AIL}}$ (d)$\}$, followed by $\{\mathrm{OR}, \mathrm{AND}, \mathrm{XNOR}_{\mathrm{AIL}}$ (p)$\}$, which outperformed ReLU. The other AIL activation functions we

Table 11: Performance of TMNx networks at compositional zero-shot learning (CZSL) on the MIT-States dataset. Pre-activation width of 64 or 70 (depending on activation function, to keep the number of parameters approximately constant). Mean (standard error) of $n=5$ random initializations. Bold: best. Underlined: top two. Italic: no significant difference from best (two-sided Student's t-test, $p>0.05$). Background: color scale from second-worst in column to best, linear with accuracy value.

			MIT-States Test Accuracy (%)		
Activation function	Mapping	Params	Top-1	Top-2	Top-3
ReLU	$1\to1$	1.15M	2.76 ± 0.08	7.11 ± 0.08	11.26 ± 0.09
LeakyReLU	$1\to1$	1.15M	2.72 ± 0.09	6.99 ± 0.17	10.95 ± 0.20
PReLU	$1\to1$	1.15M	2.83 ± 0.06	7.25 ± 0.14	11.44 ± 0.17
Softplus	$1\to1$	1.15M	*2.92* ±0.11	*7.46* ±0.19	11.75 ± 0.18
ELU	$1\to1$	1.15M	*3.13* ±0.05	*7.78* ±0.14	*12.04* ±0.20
CELU	$1\to1$	1.15M	***3.17*** ±0.06	***7.81*** ±0.15	*12.00* ±0.19
SELU	$1\to1$	1.15M	*3.05* ±0.04	*7.77* ±0.10	***12.37*** ±0.16
GELU	$1\to1$	1.15M	2.81 ± 0.06	7.19 ± 0.11	11.34 ± 0.15
SiLU	$1\to1$	1.15M	2.87 ± 0.08	*7.34* ±0.14	11.60 ± 0.18
Hardswish	$1\to1$	1.15M	2.84 ± 0.08	7.23 ± 0.11	11.48 ± 0.11
Mish	$1\to1$	1.15M	2.90 ± 0.08	*7.37* ±0.17	11.59 ± 0.13
Softsign	$1\to1$	1.15M	2.90 ± 0.04	7.28 ± 0.09	11.53 ± 0.18
Tanh	$1\to1$	1.15M	2.93 ± 0.09	*7.47* ±0.16	11.64 ± 0.20
GLU	$2\to1$	1.16M	2.59 ± 0.06	6.86 ± 0.15	10.91 ± 0.24
Max	$2\to1$	1.16M	2.45 ± 0.08	6.52 ± 0.18	10.53 ± 0.29
Max, Min (d)	$1\to1$	1.15M	2.53 ± 0.10	6.61 ± 0.15	10.65 ± 0.15
XNOR_{IL}	$2\to1$	1.16M	2.03 ± 0.04	5.43 ± 0.02	8.82 ± 0.05
OR_{IL}	$2\to1$	1.16M	2.90 ± 0.05	*7.40* ±0.11	11.68 ± 0.14
OR, AND_{IL} (d)	$1\to1$	1.15M	2.68 ± 0.06	7.05 ± 0.08	11.17 ± 0.12
OR, XNOR_{IL} (p)	$2\to1$	1.16M	2.90 ± 0.05	*7.40* ±0.11	11.68 ± 0.14
OR, XNOR_{IL} (d)	$1\to1$	1.15M	2.60 ± 0.04	6.68 ± 0.12	10.71 ± 0.17
OR, AND, XNOR_{IL} (p)	$2\to1$	1.16M	2.76 ± 0.10	7.12 ± 0.15	11.36 ± 0.19
OR, AND, XNOR_{IL} (d)	$2\to3$	1.22M	2.44 ± 0.06	6.54 ± 0.13	10.54 ± 0.20
XNOR_{AIL}	$2\to1$	1.16M	1.88 ± 0.06	4.94 ± 0.15	7.99 ± 0.19
OR_{AIL}	$2\to1$	1.16M	2.61 ± 0.08	6.83 ± 0.12	10.97 ± 0.20
OR, AND_{AIL} (d)	$1\to1$	1.15M	2.81 ± 0.06	7.12 ± 0.10	11.27 ± 0.14
OR, XNOR_{AIL} (p)	$2\to1$	1.16M	2.61 ± 0.08	6.83 ± 0.12	10.97 ± 0.20
OR, XNOR_{AIL} (d)	$1\to1$	1.15M	2.46 ± 0.04	6.51 ± 0.11	10.45 ± 0.17
OR, AND, XNOR_{AIL} (p)	$2\to1$	1.16M	2.72 ± 0.11	7.15 ± 0.19	11.32 ± 0.22
OR, AND, XNOR_{AIL} (d)	$2\to3$	1.22M	2.73 ± 0.10	6.95 ± 0.16	11.03 ± 0.09
$\text{XNOR}_{\text{NAIL}}$	$2\to1$	1.16M	1.78 ± 0.04	4.84 ± 0.11	7.95 ± 0.19
OR_{NAIL}	$2\to1$	1.16M	2.51 ± 0.03	6.58 ± 0.12	10.56 ± 0.24
OR, AND_{NAIL} (d)	$1\to1$	1.15M	2.67 ± 0.04	6.98 ± 0.06	11.02 ± 0.16
OR, $\text{XNOR}_{\text{NAIL}}$ (p)	$2\to1$	1.16M	2.55 ± 0.05	6.67 ± 0.15	10.67 ± 0.26
OR, $\text{XNOR}_{\text{NAIL}}$ (d)	$1\to1$	1.15M	2.43 ± 0.06	6.32 ± 0.09	10.28 ± 0.18
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (p)	$2\to1$	1.16M	2.60 ± 0.06	6.83 ± 0.20	10.85 ± 0.27
OR, AND, $\text{XNOR}_{\text{NAIL}}$ (d)	$2\to3$	1.22M	2.63 ± 0.06	6.77 ± 0.08	10.81 ± 0.14

considered peformed less well, and XNOR performed particularly poorly on this task. This suggests this is a domain where logical activation functions are less well suited.