A Conceptual Framework For White Box Neural Networks

Maciej Satkiewicz
314 Foundation
Warsaw/Krakow
maciej.satkiewicz@314.foundation

Abstract

This paper introduces *semantic features* as a general conceptual framework for fully explainable neural network layers. A well-motivated proof of concept model for relevant subproblem of MNIST consists of 4 such layers with the total of ~5k learnable parameters. The model is easily interpretable, achieves human-level **adversarial** test accuracy with no form of adversarial training, requires little hyperparameter tuning and can be quickly trained on a single CPU. The general nature of the technique bears promise for a paradigm shift towards radically democratised and truly generalizable white box neural networks. The code is available at https://github.com/314-Foundation/white-box-nn.

1 Introduction

Current state of the art (SOTA) neural networks are large, unwieldy black boxes offering little insight into their decision-making process, require prohibitive amounts of resources and are prone to adversarial attacks not seen during training [10, 3]. The latter implies that models rely on unintended features shared between train and test, giving false sense of generalisability [4, 6]. These issues drastically limit the usability of current generation of neural networks, especially in critical domains such as healthcare, cybersecurity or autonomous systems. Yet animal brains apparently learn rich and robust patterns in mostly unsupervised sensory input - even brains as simple as drosophila brain with ~135k neurons. This discrepancy indicates a need for radically simplified AI.

This paper aims to lay foundation for a new paradigm for training models by introducing a conceptual framework for building white box neural networks. The emphasis is put on theoretical and qualitative aspects of proposed solutions and less on tweaking quantitative metrics. Following this spirit it's important to train a proof of concept (PoC) network on *Minimum Viable Dataset* (MVD) - problem easy enough to get rid of all the obfuscating details but still hard enough to be relevant. MNIST dataset is falsely considered to be "solved" or "too easy" while SOTA models are hardly adequate in terms of their complexity, lack interpretability and are vulnerable to adversarial perturbations not seen during training [9]. This is true not only for the entire 10-digit dataset but even for certain pairs of digits [11] and therefore the MVD for this paper is chosen to be the binary subset of MNIST consisting of images of "3" and "5" (one of the most often confused pairs), which actually might be even harder than the entire MNIST in terms of average adversarial accuracy. This is not to say that similar results cannot be obtained on more complex datasets - preliminary experiments suggest that it is exactly the opposite - but in fact one of the goals of such selection of MVD is to make a clear and concise argument for the effectiveness and potential of presented method.

The structure of the paper is as follows:

- Section 2. gives a rigorous definition of *semantic feature* and argues for the generality of the notion;
- Section 3. builds a carefully motivated PoC 4-layer white box neural network for selected MVD;
- Section 4. analyses the trained model both qualitatively and quantitatively in terms of explainability, reliability and adversarial robustness.

The paper ends with ideas for further research and acknowledgments.

2 Semantic features

In machine learning inputs are represented as tensors. Yet the standard Euclidean topology in tensor space usually fails to account for small domain-specific variations of features. For example shifting an image by 1 pixel is semantically negligible but usually results in a distant output in L_2 metric.

This observation inspires the following definition:

Definition: A *semantic feature* $f_{\mathbf{PL}}$ of dimension $s \in \mathbb{N}$ is a tuple $(f, \mathbf{P}, \mathbf{L})$ where:

- f - *base*, $f \in \mathbb{R}^s$,
- $\mathbf{P}$ - *parameter set*, $\mathbf{P} \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$
- $\mathbf{L}$ - *locality function* : $\mathbf{P} \to \text{Aut}(\mathbb{R}^s, \mathbb{R}^s)$ where Aut is the family of differentiable automorphisms of $\mathbb{R}^s$

Additionally the feature $(f, \mathbf{P}, \mathbf{L})$ is called *differentiable* iff $\mathbf{L}$ is differentiable (on some neighbourhood of $\mathbf{P}$).

Intuitively a semantic feature consists of a base vector f and a set of it's "small" variations $f_p = \mathbf{L}(p)(f)$ for all $p \in \mathbf{P}$; these variations may be called *poses* after [8]. Thus $(f, \mathbf{P}, \mathbf{L})$ is *locality-sensitive* in the adequate topology capturing semantics of the domain. The differentiability of $\mathbf{L}(p)$ and of $\mathbf{L}$ itself allow for gradient updates of f and $\mathbf{P}$ respectively. The intention is to learn both f and $\mathbf{P}$ while defining $\mathbf{L}$ explicitly as an inductive bias[1] for the given modality.

2.1 Examples of semantic features

To better understand the definition consider the following examples of semantic features:

1. **real-valued** $f_{\mathbf{PL}}$
   - $f \in \mathbb{R}$
   - $\mathbf{P}$ - a real interval $[p_{min}, p_{max}]$ for $p_{min} <= 0 <= p_{max}$
   - $\mathbf{L}$ - function that maps a real number p to the function $g \mapsto g + p$ for $g \in \mathbb{R}$
2. **convolutional** $f_{\mathbf{PL}}$
   - f - square 2D convolutional kernel of size (c_{out}, c_{in}, k, k)
   - $\mathbf{P}$ - set of 2×2 rotation matrices
   - $\mathbf{L}$ - function that maps a rotation matrix to the respective rotation of a 2D kernel[2]
3. **affine** $f_{\mathbf{PL}}$
   - f - vector representing a small 2D spatial feature (e.g. image patch containing a short line)
   - $\mathbf{P}$ - set of 2×3 invertible augmented matrices[3]
   - $\mathbf{L}$ - function that maps a 2×3 matrix to the respective 2D affine transformation of images[2]
4. **xor** $f_{\mathbf{PL}}$
   - f - single-entry vector of dimension $n \in \mathbb{N}$
   - $\mathbf{P} \subset \{0, 1, \ldots, n-1\}$
   - $\mathbf{L}$ - function that maps a natural number p to the function that rolls the non-zero element of g by p coordinates to the right, where g is a single-entry vector of dimension n
5. **logical** $f_{\mathbf{PL}}$
   - f - concatenation of k single-entry vectors and some dense vectors in-between
   - $\mathbf{P} \subset \{0, 1, \ldots, n_0 - 1\} \times \ldots \times \{0, 1, \ldots, n_{k-1} - 1\}$
   - $\mathbf{L}$ - function that maps $p \in \mathbf{P}$ to the respective rolls of single-entry vectors (leaving the dense vectors intact)

[1] by the "no free lunch" theorem a general theory of learning systems has to adequately account for the inductive bias.
[2] can be made differentiable as in [7, 5].
[3] preferably close to the identity matrix.

2.2 Matching semantic features

The role of semantic features is to be matched against the input to uncover the general structure of the data. Suppose that we have already defined a function *match*: $(\mathbb{R}^c, \mathbb{R}^s) \to \mathbb{R}$ that measures an extent to which datapoint $d \in \mathbf{R}^c$ contains a feature $g \in \mathbb{R}^s$. Now let's define

$$\mathrm{SFmatch}(d, (f, \mathbf{P}, \mathbf{L})) = \max_{p \in \mathbf{P}} \mathrm{match}(d, \mathbf{L}(p)(f)) \tag{1}$$

If $\mathbf{L}$ captures domain topology adequately then we may think of matching $f_{\mathbf{PL}}$ as a form of local inhibition along $\mathbf{L_P}(f) = \{L(p)(f), p \in P\}$. Therefore semantic feature defines a **XOR** gate over $\mathbf{L_P}(f)$. On the other hand every $f_p \in \mathbf{L_P}(f)$ can be viewed as a conjunction (**AND** gate) of its non-zero coordinates[4]. Therefore semantic features turn out to be a natural way of expressing logical claims about the classical world - in the language that can be "understood" by neural networks. This is especially apparent in the context of logical semantic features which allow to explicitly express logical sentences, e.g. sentence $(A \vee B) \wedge \neg C \wedge \neg D$ can be expressed by a logical $f_{\mathbf{PL}}$ where:

- $f = [1, 0, -1, -1]$
- $\mathbf{P} = \{0, 1\}$
- $\mathbf{L}(0)(f) = [1, 0, -1, -1]$ and $\mathbf{L}(1)(f) = [0, 1, -1, -1]$

Semantic features capture the core characteristic of any semantic entity - **having many possible states but being at exactly one state at a time**. As the remainder of the paper will show this turns out to be a sufficiently strong regularization - backpropagation through appropriately chosen $\mathbf{L}$ results in easily interpretable f and $\mathbf{P}$.

It remains to characterise the *match* function. Usually it can be defined as the scalar product $d \cdot emb(g)$ where $emb : \mathbb{R}^s \to \mathbb{R}^c$ is some natural embedding of the feature space to the input space. In particular for our previous examples lets set the following:

- for real-valued $f_{\mathbf{PL}}$: $\mathrm{match}(d, g) = \mathrm{int}(d == g)$ where $d, g \in \mathbb{R}$
- for convolutional $f_{\mathbf{PL}}$: $\mathrm{match}(d, g) = \max_{0<=i<c_{out}}(d * \frac{g^i}{||g^i||_2})$ where g is a 2D convolutional kernel of shape (c_{out}, c_{in}, k, k), g^i is the i-th convolutional filter of g, d is an image patch of shape (c_{in}, k, k) [5]
- for affine $f_{\mathbf{PL}}$: $\mathrm{match}(d, g) = d \cdot \frac{g}{||g||_2}$ where g is a 2D image of shape (c_{in}, k, k) and d is a 2D image patch of the same shape
- for logical $f_{\mathbf{PL}}$: $\mathrm{match}(d, g) = d \cdot g$ for $g, d \in \mathbb{R}^c$

The occasional L_2 normalization in the above examples is to avoid favouring norm-expanding $\mathbf{L}(p)$.

3 Network architecture

In this section we will use semantic features to build a PoC white box neural network model that classifies MNIST images of "3" and "5" in a transparent and adversarially robust way. We will build the network layer by layer arguing carefully for every design choice. Note that the network construction is not an abstract process and must reflect the core characteristics of the chosen dataset. There is no free lunch, if we want to capture the semantics of the dataset we need to encode it in the architecture - more or less explicitly. The framework of semantic features allows to do this in a pretty natural way. Note that a single neural network layer will consist of many parallel semantic features of the same kind.

The network consists of the following 4 layers stacked sequentially. Note that for simplicity we don't add bias to any of those layers. Te resulting model has around 4.8K parameters.

3.1 Two Step Layer

MNIST datapoint is a 28×28 grayscale image. Its basic building block is a pixel $x \in [0, 1]$. Despite being allowed to assume any value between 0 and 1 it is actually (semantically) a ternary object - it's either ON, OFF or MEH[6]. Therefore the semantic space should squash the $[0, 1]$ interval into those 3 values.

[4] i.e. f_p is a conjunction of lower-level features.

[5] alternatively we could treat every g^i as a separate semantic feature but will we stick to the *max* option for simplicity.

[6] the MEH state is an undecided state in-between.

A real-valued $f_{\mathbf{PL}}$ with *match* and $\mathbf{L}$ defined as in the previous section identifies numbers in certain interval around f. This means that a layer consisting of multiple real-valued $f_{\mathbf{PL}}$, provided that the relevant intervals don't overlap, can be represented as a single locally-constant real-valued function. This allows us to simplify the implementation of such layer and instead of learning the constrained parameters of multiple $f_{\mathbf{PL}}$ we can learn a single parameterised real-valued function that serves as an entire layer. In order to group pixel intensities into 3 abstract values we can draw inspiration from the $\mathrm{softsign}(x) = \frac{x}{1+x}$ function or rather from it's parameterised version $\mathrm{softsign}(x, t, s) = \frac{x-t}{s+x}$ and "glue" two of such step functions together to obtain (Parameterised) Two Step function shown in Figure 2. The exact implementation of the Two Step function is provided in Appendix A. We initialize the layer setting init_scales_div=10.

3.2 Convolutional Layer

A pixel-wise function is not enough to classify pixel as ON or OFF - the semantics of our MVD require a pixel to exist in a sufficiently bright region to be considered as ON. Therefore the next layer will consist of a **single** convolutional semantic feature $(f, \mathbf{P}, \mathbf{L})$:

- f - 2D kernel of shape $(2, 1, 5, 5)$ (concatenation of (g^0, g^1), both of shape $(1, 1, 5, 5)$)
- $\mathbf{P}$ - fixed set of k distinct rotations by a multiple of the $\frac{360^\circ}{k}$ angle, for $k = 32$
- $\mathbf{L}$ - as defined earlier

The layer performs $\mathrm{SFmatch}(d(x, y), f_{\mathbf{PL}})$ with every pixel (x, y) of image d. This means that it matches the two rotated filters g^0 and g^1 with (x, y) and takes the maximum across all the matches[7]. Intuitively this layer checks if the pixel has a sufficiently bright neighbourhood.

The layer is followed by ReLU activation. We initialize g^0 and g^1 as the identity kernel of shape $(1, 1, 5, 5)$ plus Gaussian noise $\sim \mathcal{N}(0, 0.1)$.

3.3 Affine Layer

The basic semantic building blocks of MNIST digits are fragments of various lines together with their rough locations in the image (same line at the top and the bottom often has different semantics). In short they are shapes at locations. The semantic identity of shape is not affected by small[8] affine transformations. The next shape-extracting layer will therefore consist of **8** affine semantic features $(f, \mathbf{P}, \mathbf{L})$ defined as follows:

- f - 2D image patch of shape $(1, 20, 20)$
- $\mathbf{P}$ - set of 32 affine 2×3 augmented matrices
- $\mathbf{L}$ - as defined earlier

For simplicity every $f_{\mathbf{PL}}$ in this layer is located in the center of 28×28 image, i.e. we zero-pad f on every side before applying the affine transformations. Intuitively this matches the image against a localized shape in a way that is robust to small affine perturbations of the shape.

f The layer is followed by ReLU activation. We initialize affine matrices as 2×2 identity matrix plus Gaussian noise $\sim \mathcal{N}(0, 0.01)$ concatenated with 1×2 matrix filled with Gaussian noise $\sim \mathcal{N}(0, 1)$.

3.4 Logical Layer

After Affine Layer has extracted predictive shapes it remains to encode logical claims such as "number 3 is A or B and not C and not D" etc. Since the previous layer is learnable we can enforce a preferable structure on the input neurons to the Logical Layer. This layer will consist of **2** logical $f_{\mathbf{PL}}$ (one for every label) defined as follows:

- f - vector of dimension 8
- $\mathbf{P} = \{0, 1, 2, 3\}$
- $\mathbf{L}$ - as defined earlier

We initialize $f^0 = [\frac{4}{10}, 0, 0, 0, -\frac{1}{10}, -\frac{1}{10}, -\frac{1}{10}, -\frac{1}{10}]$ and $f^1 = [-\frac{1}{10}, -\frac{1}{10}, -\frac{1}{10}, -\frac{1}{10}, \frac{4}{10}, 0, 0, 0,]$. The exact initialization values are less important then their sign and relative magnitude, however they should not be too large as we will train the network using CrossEntropyLoss.

[7] a single number, not a pair of numbers, as in the chosen definition of match for convolutional $f_{\mathbf{PL}}$.

[8] close to the identity augmented matrix

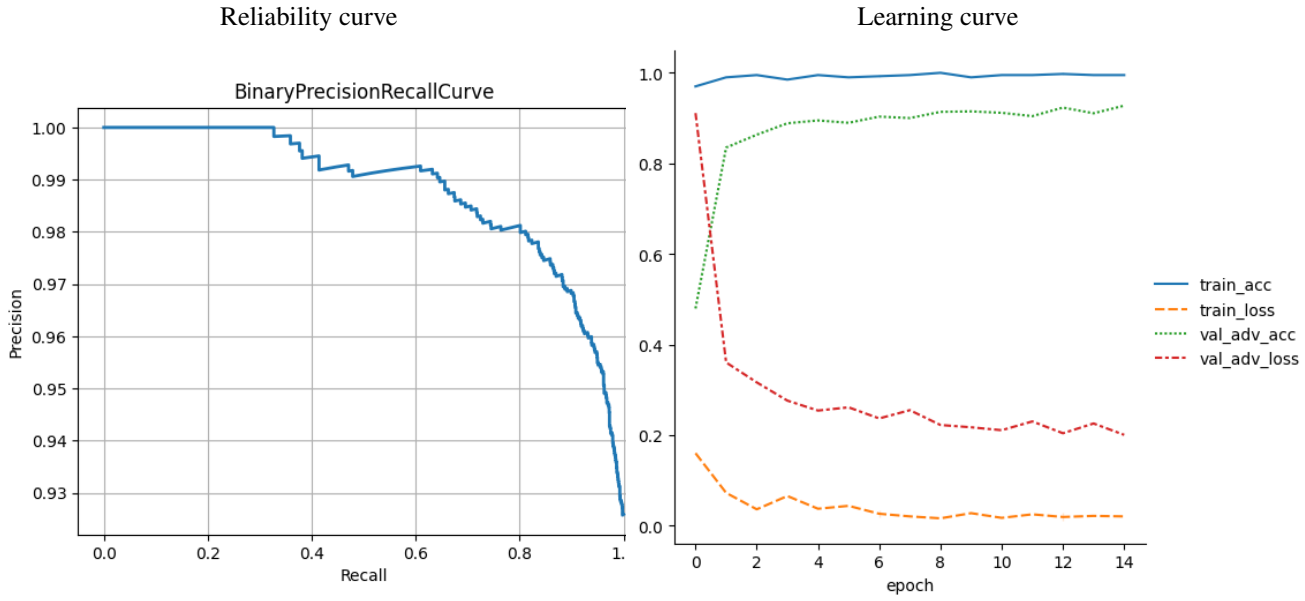

Figure 1: (left) Reliability test-time curve for the classic 40-step PGD Attack. (right) Learning curve.

4 Results

4.1 Training

The model was trained to optimize the CrossEntropyLoss for 15 epochs with Adam optimizer, learning rate $3e{-}3$, weight decay $3e{-}6$. The only augmentation was the Gaussian noise $\sim \mathcal{N}(0.05, 0.25)$ applied with 0.7 probability[9]. Those parameters were chosen "by hand" after few rounds of experimentation. The results seem pretty robust to the selection of hyperparameters as long as they are within reasonable bounds. Training a single epoch on a single CPU takes around 9s.

4.2 Quantitive results

We don't bother measuring the clean test metrics as it is a flawed approach that tells little about the true generalisation capability of the model. Every metric reported here is computed under strong adversarial regime.

The model achieves ~92% accuracy under AutoAttack[2] (with default parameters: norm='inf', eps=0.3, step_size=0.1). The metrics are almost identical for the much faster 40-step PGD Attack. Samples are usually misclassified with low confidence and therefore we compute a *reliability curve* - a precision-recall curve where the positive class is defined as the set of correctly classified samples (this means that the precision for 100% recall is exactly equal to the accuracy). It turns out that for 80% recall **model achieves human-level 98% adversarial accuracy** (Figure 1). This means that **in practice human intervention would be required only for the easily detected 20% of adversarial samples** to achieve human level reliability of the entire system (and most real-life samples are **not** adversarial).

4.3 Layer inspection

Lets inspect the trained model layers.

4.3.1 Two Step Layer

Figure A show the initial and learned shapes of the Two Step function. Te function smoothly thresholds the input at 0.365 and then at 0.556 which means that the interval $[0, 0.365]$ is treated as OFF, $[0.365, 0.556]$ as MEH and $[0.556, 1]$ as ON. Bear in mind that this is not the final decision regarding the pixel state - it's just the input to the next layer.

[9] mean is slightly positive because of the sparsity of data - subtracting pixel value is on average more harmful than adding it.

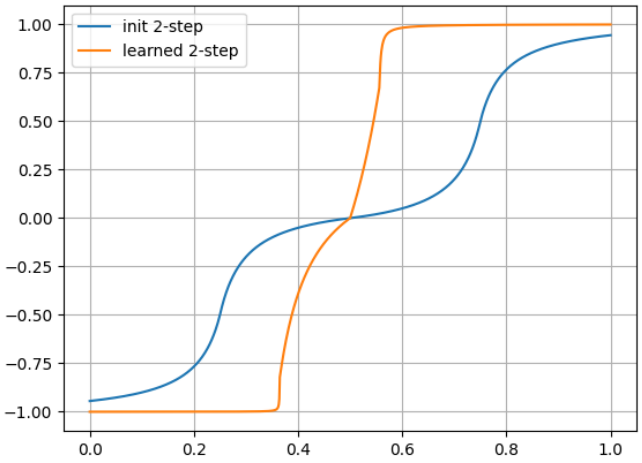

Figure 2: Two Step layer - initial and learned.

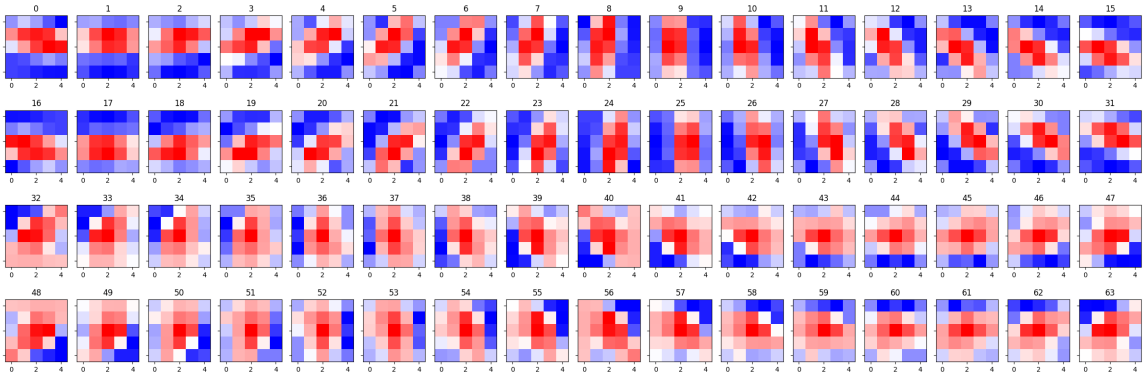

Figure 3: Learned convolutional kernels in 32 predefined poses.

4.3.2 Convolutional Layer

Quick inspection of the Figure 3 shows that the Convolutional Layer has indeed learned a meaningful filters that check if the central pixel lays inside an adequately structured region - intuitively the matching value corresponds to the degree the model considers the pixel as being ON.

4.3.3 Affine Layer

Figure 4 shows the learned affine base features. Figure B in the Appendix B shows learned features in 32 learned feature-specific poses. It's hard to imagine substantially more meaningful features.

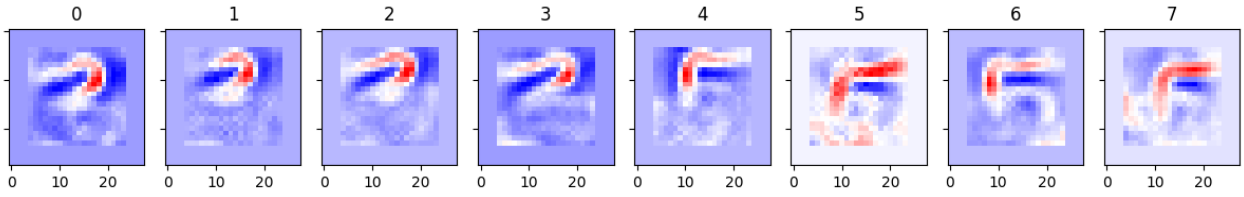

Figure 4: Learned affine base features.

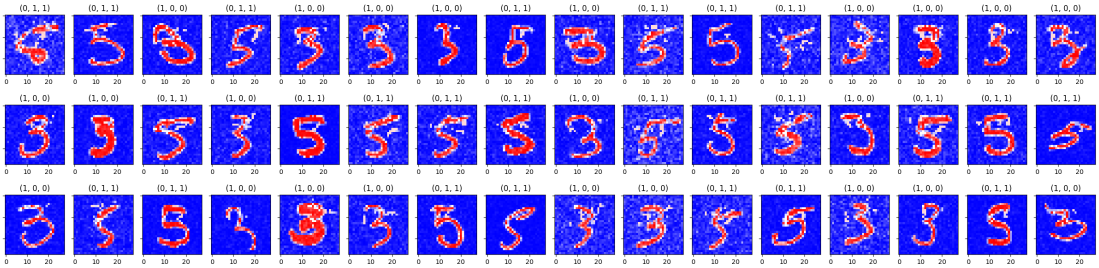

Figure 5: Boundary Attack: minimal perturbations that flip model prediction (model confidence at around 50%).

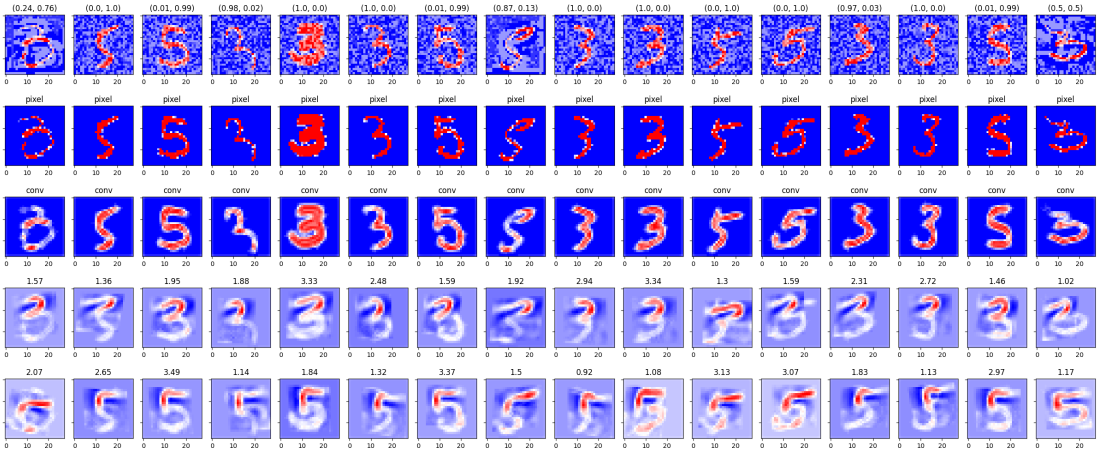

Figure 6: Visualization of test data as "seen" by the first 3 consecutive layers.

4.3.4 Logical Layer

The two learned logical base features are as follows:

- $f^0 = [1.0700, 0.0000, 0.0000, 0.0000, -0.7800, -0.7100, -0.8000, -0.8100]$
- $f^1 = [-0.8100, -0.7600, -0.8600, -0.8000, 1.0500, 0.0000, 0.0000, 0.0000]$

This is exactly as expected - $f^0_{\mathbf{PL}}$ defines number "3" as an entity having large SFmatch with **any** of the first 4 affine semantic features and small SFmatch with **all** of the remaining 4 affine semantic features. For $f^1_{\mathbf{PL}}$ its the other way round.

4.4 Decision boundary

Figure 5 shows the minimum perturbations required to change model's predictions under Boundary Attack[1] for sample test images. The added perturbations are easily spotted and understood by humans. Note that by definition model is extremely unconfident on those samples and in practice the perturbations required to fool the entire system would have to be much larger.

Figure 6 shows how the data is "seen" by the first 3 layers. The consecutive rows and lables denote respectively:

1. adversarial input and predicted class probabilities
2. output of Two Step Layer
3. output of Affine Layer
4. top pose of affine features predicting label "3" (with the match value)[10]
5. top pose of affine features predicting label "5" (with the match value)[10]

[10]These rows are actually overlaid with the middle row to make it easier to see what has been matched.

Note that in Figure 6 there are 3 misclassified examples and for every one of them **it's easy to understand why the model made the mistake**.

5 Significance and further research

The importance of explainability is hard to overestimate. An automatic system discovering meaningful predictive patterns in data could significantly offload the task of finding the underlying causal structure of observed phenomena. This has a great potential to impact science, diagnostics, big data and many other fields. Combined with the demonstrated level of efficiency the resulting democratisation could open up exciting areas for growth worldwide, favouring diverse open source solutions and providing considerable level of robustness against centralisation.

The general nature of semantic features makes it reasonable to expect that extending these results to more complex datasets and domains should be relatively straightforward - but may require some honest engineering. The obvious ideas for further work include:

- use semantic features in self-supervised setting to obtain interpretable dimensionality reduction;
- study more complex logical semantic features in more elaborate scenarios;
- figure out a way to make logical semantic features differentiable, i.e. make $\mathbf{P}$ learnable;
- design richer modes of spatial variation, e.g. hierarchical affine semantic features to capture variations of compound objects (objects consisting of affine parts are themselves affine if you "zoom out" appropriately);
- find adequate semantic topology for color space;
- integrate affine features with sliding window approach, i.e. implement weight sharing of similar features at distant locations;
- define semantic features for sound (inspiration for semantic invariants could be drawn from music theory);
- define semantic features for text, e.g. to capture semantic invariance of permutations of words;

6 Acknowledgments

This research has been entirely self-funded (a.k.a. I saved money, left my job and took the risk of following independently a depressingly underfunded and underestimated line of research).

However it needs to be noted that I had the opportunity to do preliminary research on adversarial robustness as a half-time activity during my employment at MIM Solutions. Special thanks go to Piotr Sankowski, who introduced me to the issue of adversarial vulnerability, and both to Piotr Sankowski and Piotr Wygocki who - as my former employers - decided to trust me with that research as a half-time part of my contract. It was precisely thanks to this opportunity that I developed a strong intuition that modern neural network architectures are inherently flawed and need to be fundamentally re-examined.

The story of this paper - should the results be recognized by the community - indicates a need for deep systemic and cultural changes that would empower individuals and smaller institutions to allow them to afford putting values and long-term informed risks over short-term profits. This is especially needed in larger public institutions like universities. Otherwise we might get stuck following "easy" directions which leads to a lethal combination of demoralisation and deterioration of quality.

References

[1] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models, 2018. arXiv:1712.04248.

[2] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020. arXiv:2003.01690.

[3] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, November 2020. URL: http://dx.doi.org/10.1038/s42256-020-00257-z, doi:10.1038/s42256-020-00257-z.

[4] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features, 2019. arXiv:1905.02175.

[5] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks, 2016. arXiv:1506.02025.

[6] Binghui Li, Jikai Jin, Han Zhong, John E. Hopcroft, and Liwei Wang. Why robust generalization in deep learning is difficult: Perspective of expressive power, 2022. arXiv:2205.13863.

[7] E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020. URL: https://arxiv.org/pdf/1910.02190.pdf.

[8] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules, 2017. arXiv:1710.09829.

[9] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on mnist, 2018. arXiv:1805.09190.

[10] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014. arXiv:1312.6199.

[11] Qi Tian, Kun Kuang, Kelu Jiang, Fei Wu, and Yisen Wang. Analysis and applications of class-wise robustness in adversarial training, 2021. arXiv:2105.14240.

A Implementation of Two Step Layer

```
class TwoStepFunction(Module):
    """Simplified real-valued SFLayer for inputs in [0, 1]"""
    def init_weights(self):
        self.a0 = nn.Parameter(torch.tensor(0.5))
        self.a1 = nn.Parameter(torch.tensor(0.5))
        self.t0 = nn.Parameter(torch.tensor(0.25))
        self.t1 = nn.Parameter(torch.tensor(0.75))

        self.scales = nn.Parameter(
            (torch.ones((4,)) / self.init_scales_div)
        )

    def softsign(self, x, threshold, scale, internal):
        a = self.a1 if threshold > 0 else self.a0

        x = x - threshold
        x = x / (scale + x.abs())

        if internal:
            shift = threshold
            shift = shift / (scale + shift.abs())
            x = x + shift
            x = a * x / shift.abs()

            return x

        div = x.abs().max() / (1 - a)
        x = x / div
        x = x + threshold.sgn() * a

        return x

    def transform(self, x):
        return 2 * x - 1

    def forward(self, x):
        x = self.transform(x)
        t0, t1 = self.transform(self.t0), self.transform(self.t1)

        params = [
            (t0, self.scales[0], False),
            (t0, self.scales[1], True),
            (t1, self.scales[2], True),
            (t1, self.scales[3], False),
        ]
        xs = [self.softsign(x, *p) for p in params]

        masks = [
            (x < t0).float(),
            (t0 <= x).float() * (x < 0.0).float(),
            (0.0 <= x).float() * (x < t1).float(),
            (t1 <= x).float(),
        ]
        x = sum(m * xx for (m, xx) in zip(masks, xs))
        return x
```

B Learned Affine Features With Learned Poses