Origami Inspired Neural Networks

Anonymous Authors[1]

Abstract

One of the leading methods to improve the performance of deep neural networks is simply increasing the number of parameters, thereby leveraging neural scaling laws. However, improving parameter efficiency can also yield significant benefits by reducing training time, inference speed, and hardware expenses. We propose a new simple network layer, named the Fold Layer, based on a softened generalization of the Fold and Cut Theorem and the elegant principles of origami. This new non-linearity transforms the data with a geometric fold in an n-dimensional space, and can be parameterized to cut, fold, stretch or squeeze data. We experimentally explore the utility of this new nonlinear across a variety of tasks and algorithms, and explore its performance for parameter count, accuracy, and inference time.

1. Introduction

Neural networks have become the cornerstone of modern machine learning, with performance improvements primarily driven by scaling laws—larger models with more parameters tend to achieve better results. However, this approach has its limitations, including high computational costs, increased training time, and inefficiency in deployment in resource-constrained environments. Addressing these challenges requires innovative architectural designs that prioritize parameter efficiency without sacrificing performance.

We introduce the Fold Layer, a novel neural network component inspired by the Fold and Cut Theorem and the principles of origami. The fold layer operates by defining a hyperplane in an n-dimensional space and reflecting points across this hyperplane so that their distance from the plane remains constant. This transformation effectively simulates the combination of a fully connected linear layer and a ReLU activation function. This simplification requires significantly fewer parameters and reduces the spatial complexity from $O(n \times m)$ to $O(n)$. Inference complexity decreases similarly, enabling faster execution for time-critical applications.

The geometric nature of the fold layer provides a more interpretable function parameterized by a single vector. This efficient representation makes it a promising alternative to traditional linear layers, which can be seamlessly replaced by a single fold layer in any existing architecture, including transformers, and convolutional networks. Its intuitive design aligns naturally with data containing natural symmetries such as [EXAMPLE], potentially unlocking new avenues for applications in classification and reinforcement learning, among others. Moreover, its parameter efficiency makes it particularly advantageous for scenarios where hardware constraints limit computational resources, such as on-device large-language models.

This paper makes several key contributions: it introduces the fold layer as a novel type of nonlinearity for neural networks, demonstrates its theoretical foundations based on the Fold and Cut theorem, and benchmarks its performance across [TASKS]. Models incorporating fold layers achieve [COMPARISON] on [TASKS], showing promise as a substitute for the classical linear layer and ReLU activation pair.

To facilitate further exploration, we provide a GitHub repository with implementation details and an interactive notebook for hands-on experimentation. This paper is organized as follows: [OUTLINE].

2. Background

Neural networks have become indispensable across computational fields, including image recognition, natural language processing, and scientific modeling. However, as models grow in width and depth, they increasingly require substantial computational resources and longer training times, making deployment in resource-constrained environments challenging. This has spurred research into novel architectures and activation functions that can enhance accuracy and parameter efficiency.

One area of focus has been introducing different types of nonlinearities into deep learning models to boost their ac-

[1] Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

curacy and generalization. Notable nonlinear activation functions that have produced positive results are GELU (Gaussian Error Linear Units) (Hendrycks & Gimpel, 2016), and Swish (Ramachandran et al., 2017). GELU has been effective in improving the smoothness of the gradient as it propagates through the network and is commonly used in GPT models such as BERT. Meanwhile, Swish has demonstrated various improvements upon the common ReLU but it is worth noting that the accuracy improvements come with additional computational complexity.

These non-linearities change how the data is handled during the networks training process. Feedforward Neural Networks with a piecewise activation function, such as ReLU, can create individual polyhedron regions (Huchette et al., 2023). These can take complex geometric shapes, and the number and structure of these regions will depend on the networks depth, width and activation patterns. If one can better understand how these regions form, and how the data is being handled within these regions, then the generalization and learning of the model can be better understood. A similar theory was proposed to better understand how networks created local polytopes due to piecewise non-linear activation functions (Black et al., 2022). Rather than attempting to understand how each neuron impacts learning, we can address how these learned polytopes and groups of polytopes encode important features of the data.

The geometry of the a deep learning model can have an impact on how well it learns and then generalizes to new data. The type of data itself can also impact how we should build our model and utilize known properties of types of data when trying to construct a useful architecture. An example of this is when working with sparse data to create efficient solutions for ordinary and partial differential equations using PINNS (Physics-Informed Neural Networks) or more recently PIKANS (Physics-Informed Kolmogorov–Arnold Networks) (Toscano et al., 2024). When working with image data, the best networks have traditionally been CNNs (convolutional neural networks)(O'Shea & Nash, 2015) due to our understanding that pixels located near each other are more related than pixels located on the opposite side of the image. Thus a convolution operation is much better suited to learning images than a vanilla MLP.

A research team at the Samsung Israel Research Center has combined the analysis of nonlinear activation functions with data treatment strategies within neural networks. They explored how layers could be "folded" by removing redundant activation functions, resulting in improved model latency ((Dror et al., 2021)). If an activation function isn't impacting the data to a certain threshold, then it can be removed, allowing you to fold the 2 layers that were originally separated into one larger linear layer.

We approach the problem similarly by "folding the data". This intuition of how neural networks naturally "fold" data was explored by Christian Keup and Moritz Helias of the Institute of Computational and Systems Neuroscience, Aachen University (Keup & Helias, 2022). They argued that neural networks naturally fold data in order to make non-linearly separable data, linearly separable. This is done through the learnable affine transformation of the bias and weight matrix combined with the nonlinearity of the activation function. It also requires the width of the network to be greater than the dimensionality of the data manifold. This allows the network to fold into unoccupied dimensions. These higher dimensions creates linear separability of data that, if otherwise contained within the original sized data manifold, would have been non-linear separable, making it much more difficult for the model to learn simliarities and generalize to new data.

In order to exploit this natural folding capability of a neural network, we wanted to teach the network how to fold. Rather than folding onto a hyperplane, we desgined a nonlinear layer to learn how to fold through the hyper plane, which better represents the folding mechanism one would recognize in origami and paper folding. By utilizing a new nonlinearity, that is explored in more depth in the model architecture section of our paper, and combining this with the traditional nonlinearity of an activation function, such as ReLU or Sigmoid, we can determine how tightly a model should fold data, and where the data should be folded, allowing us to create tighter decision boundaries than typical multi-layer perceptrons that are attemtping to learn how to fold into higher dimensions.

This novel utilizes the natural symmetries of the data along with a final linear layer and softmax that acts like a cutting mechanism. This follows the algorithm provided by Erik Demaine (Demaine et al., 2000). This approach significantly reduces the number of parameters compared to a similar linear layer in a deep network while enhancing inference speed. This strategy achieves comparable or improved accuracy to MLPs while decreasing the parameter count necessary for large classification tasks.

3. Model Architecture

Fold layers leverage the mathematics of paper folding to define flexible, powerful decision boundaries in neural networks. Drawing on the Fold-and-Cut theorem, origami networks with a series of fold layers perform reflections across learnable hyperplanes—"folds"—then apply a final "cut" to separate classes. This inspiration yields a novel approach that can match or exceed traditional deep learning in accuracy while offering practical efficiency gains. We summarize the motivation behind the fold layer before introducing it in Section 3.3.

3.1. Irregular Polygons Capture Complex Decision Boundaries

Similar to the framework of support vector machines, classification tasks often involve drawing generalizable decision boundaries between data points in some space $\mathbb{R}^d$. In $\mathbb{R}^2$, the learnable, irregular polygons can represent decision boundaries in a flat plane. (show a 2D example in a figure) [we might be able to remove this section]

3.2. The Fold-and-Cut Theorem Can Construct Decision Boundaries

We can leverage the fold-and-cut theorem to define the flexible decision boundaries that create these irregular polygons:

Theorem: Given any planar shape composed of straight-line segments, it is always possible to fold a single sheet of paper flat such that a single straight cut will produce the shape exactly.

Applying this theorem to machine learning, if there exists an optimal polygon decision boundary for a particular problem in $\mathbb{R}^2$, then by the fold-and-cut theorem we can find a series of folds and a single cut that represent the decision boundary. Thus, if we define parameterized operations to perform folds and cuts, then there exists a guaranteed set of parameters to represent any decision boundary.

3.3. Reflections Across Learnable Hyperplanes act as High-Dimensional "Folds"

We generalize the definition of a “fold” as a reflection across a hyperplane "crease." This generalized definition equates to the traditional fold in $\mathbb{R}^2$ while providing a natural extension for the higher-dimensional spaces typical of machine learning tasks.

Concretely, let $\mathbf{x} \in \mathbb{R}^d$ be an input vector, and let $\mathbf{n}_p \in \mathbb{R}^d$ be a learnable vector (the normal or “support” vector) defining the hyperplane crease. Geometrically, $\mathbf{n}_p$ encodes both the orientation of the hyperplane and through its magnitude, how “far” the fold occurs from the origin.

To fold $\mathbf{x}$ about the hyperplane orthogonal to $\mathbf{n}_p$, we reflect any point $\mathbf{x}$ on the other side of the hyperplane using the folding formula:

$$L_p(\mathbf{x}, \mathbf{n}_p) = \mathbf{x} - 2\left(\mathbb{1}_{\{\mathbf{n}_p \cdot \mathbf{x} > \mathbf{n}_p \cdot \mathbf{n}_p\}}\right)\left(1 - \frac{\mathbf{x} \cdot \mathbf{n}_p}{\|\mathbf{n}_p\|}\right)\mathbf{n}_p,$$

Breaking down the equation, we only select points on the far side of the hyperplane to fold with the indicator $\mathbb{1}_{\{\mathbf{n}_p \cdot \mathbf{x} > \mathbf{n}_p \cdot \mathbf{n}_p\}}$. Among these selected points, $\left(1 - \frac{\mathbf{x} \cdot \mathbf{n}_p}{\|\mathbf{n}_p\|}\right)\mathbf{n}_p$ gives an orthogonal distance vector separating a point from the hyperplane. Subtracting this vector once from the point $\mathbf{x}$ would project the point onto the hyperplane, while subtracting the vector twice achieves the desired reflection.

Because each fold layer requires only one vector $\mathbf{n}_p$ per fold, its parameter count and forward-pass computation scale as $\mathcal{O}(d)$, in contrast with the $\mathcal{O}(d^2)$ cost of a typical fully connected linear layer. While the backpropagation step still resembles matrix operations with $\mathcal{O}(d^2)$ cost, the $\mathcal{O}(d)$ forward pass makes fold layers especially attractive for real-time applications.

3.4. Final Linear Layer with Softmax acts as a "Cut"

After our space is folded in the desired orientation, the fold-and-cut analogy suggests implementing a linear cut to outline the desired polygon region. Since a linear layer followed by a binary softmax activation function equates to a logistic regression [citation needed], then [choose the previous sentence or the following]. Additionally, logistic regressions are also known to define a separating hyperplane, i.e. a cut through the space defining a decision boundary. Extending to multi-class classification, we can likewise define "cut regions" with a final linear layer and corresponding softmax activation. Mathematically our cut is defined as follows:

$$\text{cut}(\mathbf{x}) = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

3.5. Layering Folds Yield Trainable Origami Networks

With fold and cut layers clearly defined, we construct a proof-of-concept, "origami net" by composing M consecutive folds followed by a final cut layer. Mathematically,

$$F(\mathbf{x}, \theta) = \text{softmax}\left(\mathbf{W}L_M(\cdots L_1(\mathbf{x}, \mathbf{n}_1)\cdots, \mathbf{n}_M) + \mathbf{b}\right)$$

describes the origami net with $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{n}_1, \cdots, \mathbf{n}_M)$ as the learnable parameters.

Incorporating our model $F(\mathbf{x}, \theta)$ into a cross entropy loss function, we now have a trainable origami net. SUPER parameter efficient because each layer is way smaller in parameters than a traditional linear layer. yup. edit this.

3.6. Origami Networks Work on Toy Datasets

(send one example to the appendix) We now demonstrate the design and functionality of the fold layer using several toy examples. While fold operations are easiest to visualize in two dimensions as in these cases, their mathematical formulation extends to arbitrary dimensions. Each example employs a simple model with two to three fold layers followed by a linear layer to define the final decision boundary that we do not visualize. The model’s objective is to take data with two input features $[-2, 2]$ and two output classes and transform the data in 2D space using 2D fold operations. We cherry picked model results for this demonstration to best clarify the fold layer’s capability, but the model learned each of the fold layer parameters.

We start with a simple XOR problem in Figure 1. Consider the data as a colored napkin. The model determines the optimal fold to rearrange the data for linear separation, indicated by the red line. This example requires two folds, though a single 3D fold could achieve separability for less complex data [CITATION TO APPENDIX]. In this case, the model only required six parameters: two for each of the fold layers and two for the linear separation after the model applies the fold transformations.

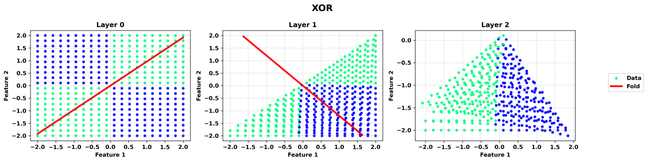

Figure 1. The model finds the lines of symmetry along each diagonal and folds the data to achieve linear separability on the XOR problem.

Our second example uses samples placed randomly according to the 2D normal distribution rather than a uniform distribution in Figure 2. In this case, one class completely encloses the other class. The model achieves linear separability by learning six parameters defining two folds, albeit less precisely than the XOR example.

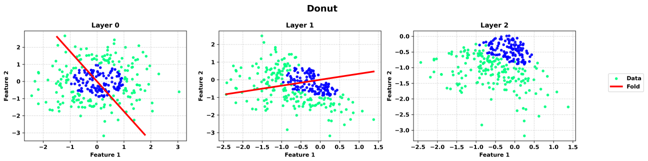

Figure 2. The model folds a circular, non-uniform distribution efficiently even when one class encloses the other

An extreme example highlights the fold layer's capacity to leverage data symmetries in Figure 3. Here we consider one class in the shape of a five-pointed star surrounded closely by another class. Each layer finds a line of symmetry in the data and folds to prevent different classes from overlapping. When symmetry lines are absent, the folds adapt by generating them, Layer 1 in the second chart demonstrates. Fold layers exhibit an inherent capacity for model selection. Redundant or unhelpful folds minimize their impact by reducing folding magnitude or positioning all training data on the unmoved side of the fold function. This behavior effectively removes one layer from the network.

Beyond geometric classification tasks, we can also effectively approximate toy regression functions as well. See appendix.

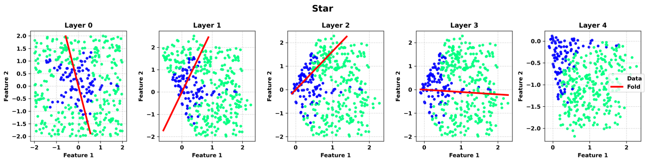

Figure 3. A more extreme star-shaped dataset with exploitable symmetries

3.7. Generalized Fold Layers allow more flexibility

Although these origami nets perform extremely well on geometrically symmetrical toy datasets, we provide the following adjustments for more practical learning tasks.

$$L_p(\mathbf{x}, \theta_p) = \mathbf{x} + s_p\, \sigma_p(\mathbf{x}) \left(1 - \frac{\mathbf{n}_p \cdot \mathbf{x}}{\|\mathbf{n}_p\|}\right) \mathbf{n_p}$$

3.7.1. Flexible Boundary Functions – $\sigma_p(\mathbf{x})$

We first relax the indicator function of the original fold to a general boundary function $\sigma_p(\mathbf{x})$. We define three different boundary functions as follows.

Hard Fold: The boundary function for the hard fold is the standard indicator function previously defined as:

$$\sigma_p(\mathbf{x}) = \mathbb{1}_{\{\mathbf{n}_p \cdot \mathbf{x} > \mathbf{n}_p \cdot \mathbf{n}_p\}}$$

Leaky Fold: Analogous to a leaky ReLU, the leak parameter $k \in (0, 1)$ allows gradients from all of the data to propagate through the fold layer.

$$\sigma_p(\mathbf{x}) = \left\{ \begin{array}{ll} 1, & \mathbf{n}_p \cdot \mathbf{x} > \mathbf{n}_p \cdot \mathbf{n}_p \\ k_p, & \text{otherwise} \end{array} \right\}$$

Soft Fold: A sigmoid boundary function with an optionally learnable "crease" parameter α_p creates a smoother transition across the fold. This function results in a fold resembling a hard fold when α_p is large and a 2D paper folded with minimal creasing when α_p is small.

$$\sigma_p(\mathbf{x}) = \frac{1}{1 + e^{-\alpha_p(\mathbf{n}_p \cdot \mathbf{x} - \|\mathbf{n}_p\|)}}$$

We can visually compare how these different boundary functions impact the folding mechanism below. As we travel in the direction $\mathbf{n}_p$ we can see:

[Insert figure of 3 functions]

3.7.2. Stretch

After performing a fold operation, we can add a fixed or learnable stretch parameter s_p that determines another translation to the data. In the 2D case, it looks like stretching or squeezing fabric instead of paper where $s_p = 2$ looks much like a normal fold and $s_p = 1$ results in a dimension reduction along the hyperplane.

3.7.3. Fold Direction

A standard fold layer always shifts data from the hyperplane's positive side towards the origin. We instead select the fold direction by multiplying s_p randomly by 1 or -1 during initialization so that some fold layers translate data from the negative side towards the positive side.

3.7.4. Higher Dimensions

The fold operation naturally extends to higher dimensions by folding data beyond its original space—analogous to partially folding a piece of paper, In practice we append 0s to the end of $\mathbf{x}$ and apply the function in this higher dimension.

3.8. Fold Layers are Compatible with any Existing Deep Learning Architecture

While origami nets composed strictly of fold layers with a final cut work well for our toy problems, fold layers are more practical when integrated into broader architectures. For instance, replacing multiple linear layers in a traditional MLP with fold layers enhances nonlinearity while reducing computational overhead. Similarly, integrating fold layers into advanced architectures like transformers holds promise for scaling efficiency without sacrificing expressive power. We refer to networks composed of multiple fold layers as origami networks.

3.9. Fold Layers are not the Fold-and-Cut Theorem

Although the fold-and-cut analogy serves as a helpful guide, practical use of our designed fold layer diverges from the theorem in multiple ways. Most blatant deviation is the extension to higher dimensions. The fold-and-cut theorem is proven only in 2D with no current proof of a higher dimensional corollary. Secondly, the original theorem allows for "back-folding" or a partial undoing of the fold. Our fold layers is more accurately described by a 'fold and glue' operation. Although some effect of this can be negated with higher dimensions [see appendix]. The fold and cut theorem also deals with finite paper, while we are folding an infinitely spanning vector space. [FIX: We also believe that fold layers are sensitive to initialization as slight deviations in structure can lead to drastic changes later on]. Expanding more on this in our results section, origami nets composed strictly of fold layers significantly under perform architectures that integrate additional fold layers, or replace deep linear layers. Finally, the generalizations also deviate from a traditional fold.

4. Methodology

4.1. Training Data

We used five classification datasets and three reinforcement learning tasks to evaluate the performance of our proposed nonlinearity. The HIGGS dataset from UCI contains 11,000,000 samples representing particle physics data, with 28 input features and 2 output classes. The Covertype dataset, also from UCI, includes 581,012 rows of tabular data describing forest cover types, with 10 possible classes. For image-based tasks, we utilized MNIST Digits and Fashion MNIST, both curated by Yann LeCun. MNIST Digits comprises 70,000 28x28 grayscale images of handwritten digits across 10 categories, while Fashion MNIST includes 70,000 28x28 grayscale images of clothing items spanning 10 categories. Finally, we incorporated the Cifar10 dataset, which provides 60,000 32x32 color images of 10 classes of common objects. [REINFORCEMENT LEARNING TASKS].

Preprocessing steps remained minimal to ensure reproducibility. We performed a train-test split and converted all data into tensor format compatible with modern deep learning frameworks. This straightforward preprocessing approach allows direct comparison with other methods and minimizes the introduction of external variables.

4.2. Hardware and Schedule

[MISSING] Specifications: CPU: Model, number of cores, clock speed. GPU: Model, VRAM size, and number of GPUs used. Memory: Total RAM. Storage: Type (e.g., SSD, HDD) and capacity if relevant. Frameworks and Libraries: Mention the version of the software frameworks used (e.g., PyTorch 2.0, TensorFlow 2.9). Cluster or Cloud Services (if applicable): Specify whether the training was conducted on a local machine, a computing cluster, or a cloud service (e.g., AWS, Google Cloud, Azure). Provide instance type or node details for cloud/cluster resources. Operating System: State the OS used (e.g., Ubuntu 20.04, Windows 11).

We allocated a maximum of 24 hours for training each model, with early stopping and checkpointing mechanisms in place to prevent unnecessary computation. Training proceeded with a batch size of 32 and a cap of 300 epochs. Each model ran in parallel across five trials to ensure robust performance evaluation. The full set of experiments required approximately 1,000 GPU hours, reflecting significant computational demands. Limited GPU availability constrained the number of repetitions and the total count of models tested, influencing the scope of the experimental design.

4.3. Optimizer and Regularization

We trained our models using the Adam optimizer implemented in torch.optim.Adam with a learning rate of 0.001. Training employed a custom gnome learning rate scheduler that began at one-third of the learning rate, rapidly increased to three times the learning rate, and then quickly decreased back to the baseline rate. Early stopping terminated training if validation accuracy showed no improvement over an extended period, preventing unnecessary computation. A hard stop ensured training did not exceed 300 epochs. Models trained with a batch size of 32, and no specialized weight initialization methods were applied. These configurations provided a stable and efficient training process for all experiments.

4.4. Model Selection

We conducted a hyperparameter sweep to evaluate the performance of our model. We varied between all possible combinations of the learning rates, fold layer type, stretch, depth, layer combinations, and layer widths as detailed in 1. To clarify the layer combinations, we tested alternating fold then linear layer until the desired depth is reached. For example, if the desired depth is 5 then we would have: fold, linear, fold, linear, fold. We also tested three consecutive fold layers then three linear layers (e.g. fold, fold, fold, linear for a depth of four. The layer width was varied across three schemes: [EXPLAIN].

Each configuration was repeated three times to ensure statistical robustness. We applied this hyperparameter tuning procedure uniformly to our models to ensure comparability. [MOVE table TO APPENDIX (maybe)]

Table 1. Hyperparameters for Model Selection

Hyperparameter	**Values Tested**
Learning Rate	0.01, 0.001, 0.0001
Fold Type	Hard, Soft
Has Stretch	True, False
Layer Count	1, 4, 7, 10
Layer Combinations	Fold then Linear (alternating), 3 Folds, 3 Linear (alternating)
Width Variation	Linearly increasing, Constant, Alternating
Repetitions per Model	3

4.5. Evaluation Method

We compared the models against each other based on their validation accuracy during the final epochs for classification tasks. While no conclusive differences emerged across most configurations, we observed that larger models with a learning rate of 0.001. In reinforcement learning tasks, models using SoftFolds and stretch functionality outperformed those with hard folds. In the following results section, we compare models based on their validation accuracy, inference speed, and training curve to highlight the advantages of the Fold layer in application.

5. Results

We present the results of a general classification problems and RL problems (and GNN if good)

5.1. Classification

color-coded architecture figure introduction and explanation of main plot showing the tradeoff curves additional ablation study

5.2. Reinforcement Learning

showing the architecture show the episode score plots created by Nathaniel

5.3. Resnet

yup

5.4. Transformers

yup

5.5. Graph Networks

Nathaniel also presents the results (if good)

6. Discussion

- Diagram of when to use fold layers ()

- Which architectures and variations worked well

Fold layers performed the best when coupled with linear layers and activation functions, placed after each ReLU in the traditional MLP. Since folding puts points on top of each other, repeated folding may make it difficult to separate points of one class from another. On the other hand, giving the model opportunities to stretch each time before folding may allow points to be moved and separated in a new way.

- Insights into why the fold layer performs well (or doesn't) in specific scenarios.

- Insights into why linear layer/fold layer architecture does well

- Discussion on efficiency, generalization, and scalability.

Due to its better big O and order-of-magnitude smaller parameter-per-layer count, fold layers have the potential to scale quite efficiently.

- Limitations and potential challenges.

In general, MLPs with folds took about twice as long to train. Why? (maybe more complicated computation graph?) It is hard to compete with the decades that have gone into optimizing matrix multiplication, but as fold layers become more widely adopted, there is potential for the training and inference of these layers to become even faster.

- Applications where the fold layer could excel.

Much research has gone into taking advantage of symmetries inherient in the data or system (Villar et al., 2022; Fuchs et al., 2020). Since the folds we have presented reflect one type of symmetry (symmetry about a hyperplane?), perhaps the idea of learnable folding could be extended to other symmetries.

- Discuss flexibility in integrating the fold layer into existing architectures or designing new ones.

The Fold and SoftFold are independent torch modules that can be added as layers in any model, allowing for many flexible and unique opportunities of usage.

7. Conclusion

- If someone read only the intro and the conclusion, they should know what we did/asked, what we found, and why it matters

In response to the question "How can the fold-and-cut theorem be applied to machine learning?" we have developed new Fold and SoftFold layers. These layers learn the hyperplane over which the space should be folded as part of classification or regression tasks. These layers are interpretable and require significantly less parameters than a linear layer.

- Restate the main contributions and findings. How good is it?

We've included a decision tree on when to include fold layers and when not to. Including fold layers in an MLP can significantly improve performance when important hyperparameters are not well tuned, for example in RL algorithms (just a guess, could also/instead say "MLPs with fold layers are significantly less sensitive to hyperparameters").

- Emphasize the significance of the fold layer in advancing neural network design.

Expanding the library of deep learning tools is important as the array of tasks remains varied. Fold layers present a unique approach to nonlinear transformations for machine learning tasks.

- Suggest directions for future work (e.g., adaptive number fold layers, domain-specific applications, part of transformers or reinforcement learning algorithms).

Fold layers have the potential to improve performance wherever linear layers are used, including graph neural networks, language models, vision transformers, reinforcement learning, recurrent neural networks, neural radiance fields, contrastive learning, neural ODEs, and in diffusion or flow-based models.

8. Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

9. References

References

Black, S., Sharkey, L., Grinsztajn, L., Winsor, E., Braun, D., Merizian, J., Parker, K., Guevara, C. R., Millidge, B., Alfour, G., and Leahy, C. Interpreting neural networks through the polytope lens, 2022. URL https://arxiv.org/abs/2211.12312.

Demaine, E. D., Demaine, M. L., and Lubiw, A. Folding and cutting paper. In Akiyama, J., Kano, M., and Urabe, M. (eds.), *Discrete and Computational Geometry*, pp. 104–118, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-46515-7.

Dror, A. B., Zehngut, N., Raviv, A., Artyomov, E., Vitek, R., and Jevnisek, R. Layer folding: Neural network depth reduction using activation linearization, 2021. URL https://arxiv.org/abs/2106.09309.

Fuchs, F. B., Worrall, D. E., Fischer, V., and Welling, M. Se(3)-transformers: 3d roto-translation equivariant attention networks, 2020. URL https://arxiv.org/abs/2006.10503.

Hendrycks, D. and Gimpel, K. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. URL http://arxiv.org/abs/1606.08415.

Huchette, J., Muñoz, G., Serra, T., and Tsay, C. When deep learning meets polyhedral theory: A survey, 2023. URL https://arxiv.org/abs/2305.00241.

Keup, C. and Helias, M. Origami in n dimensions: How feed-forward networks manufacture linear separability, 2022. URL https://arxiv.org/abs/2203.11355.

O'Shea, K. and Nash, R. An introduction to convolutional neural networks, 2015. URL https://arxiv.org/abs/1511.08458.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions, 2017. URL https://arxiv.org/abs/1710.05941.

Toscano, J. D., Oommen, V., Varghese, A. J., Zou, Z., Daryakenari, N. A., Wu, C., and Karniadakis, G. E. From pinns to pikans: Recent advances in physics-informed machine learning, 2024. URL https://arxiv.org/abs/2410.13228.

Villar, S., Yao, W., Hogg, D. W., Blum-Smith, B., and Dumitrascu, B. Dimensionless machine learning: Imposing exact units equivariance, 2022. URL https://arxiv.org/abs/2204.00887.

A. You *can* have an appendix here.

You can have as much text here as you want. The main body must be at most 8 pages long. For the final version, one more page can be added. If you want, you can use an appendix like this one.

The `\onecolumn` command above can be kept in place if you prefer a one-column appendix, or can be removed if you prefer a two-column appendix. Apart from this possible change, the style (font size, spacing, margins, page numbering, etc.) should be kept the same as the main body.

- high dimensions fixes the fold and glue problem

- one example (either egg or star) goes in the appendix

- regression problem in the appendix

- [MAYBE?] Parameters of sweep from methodologies

- hyper parameter sweep in more detail for figuring out what works best

- images of hyper parameter sweep results from training

- XOR problem folded from 2D into 3D

- GNN results even if they weren't good

- RL results that don't fit