Point Cloud Network: An Order of Magnitude Improvement in Linear Layer Parameter Count

Charles Hetterich
chetterich@utexas.edu

September 8, 2023

Abstract

This paper introduces the Point Cloud Network (**PCN**) architecture, a novel implementation of linear layers in deep learning networks, and provides empirical evidence to advocate for its preference over the Multilayer Perceptron (**MLP**) in linear layers. We train several models, including the original **AlexNet**, using both MLP and PCN architectures for direct comparison of linear layers [8]. The key results collected are model parameter count and top-1 test accuracy over the **CIFAR-10** and **CIFAR-100** datasets [7]. AlexNet-PCN$_{16}$, our PCN equivalent to AlexNet, achieves comparable efficacy (*test accuracy*) to the original architecture with a **99.5%** reduction of parameters in its linear layers. All training is done on cloud *RTX 4090* GPUs, leveraging pytorch for model construction and training. Code is provided for anyone to reproduce the trials from this paper.

1 Introduction

The Multilayer Perceptron is the simplest type of Artificial Neural Network (**ANN**). Since its inception in the mid-20th century, it has held firmly as one of the most popular structures in deep learning. MLPs were the first networks used with backpropagation and are relied on heavily in the attention mechanisms of the popular transformer architectures [11,12].

Typically, networks that employ MLPs suffer from an extremely large parameter count. This is because the amount of trainable parameters present in an MLP scale by $O(n^2)$ relative to the number of input features. A parameter count so large that models have to be run across several GPUs because the parameters alone cannot fit into just one. GPT-3 and GPT-4 are two well-known models today, both of which rely on MLPs in their transformer architectures, with GPT-3 holding 175 billion trainable parameters [1,10].

AlexNet, widely regarded as the catalyst of the modern deep learning boom

over a decade ago, popularized the convolutional operation [8]— the key feature of the convolution being its reduction in parameter count in processing image data [9].

Despite the value demonstrated by the parameter reduction present in convolutional networks, MLPs are still prevalent simply because there is currently no accessible alternative implementation of linear layers. The PCNs presented in this paper cut the parameter count present in linear layers by **an order of magnitude**, $O(n^2) \rightarrow O(n)$, while still maintaining a comparable efficacy to their equivalent MLP counterpart.

1.1 Related Work

Low-rank compression of ANNs is an emerging area of research which is closely related to PCNs [3]. Most work in this area relies on *Singular Value Decomposition* and can be divided into one of two categories (1) finding a low-rank factorization of a pre-trained network [3,5], or (2) training a low-rank network directly [6,13]. The latter is more closely related to a PCN.

1.2 Contribution

This work offers a rephrasing of the same problem that low-rank factorization networks aim to solve. In low-rank factorization, we start with a weight matrix, W, and look to find an optimal *compression* that maintains efficacy [3]. The PCN starts with an already small set of parameters, and looks to find an optimal *expansion* of those parameters that will perform with comparable efficacy to W.

We outline a light-weight implementation of the PCN architecture that is practical and generalizable to most existing deep learning networks, with source code that makes it trivial to implement.

We also provide a set of key results that demonstrate that a PCN can substantially reduce the number of parameters in linear layers while still maintaining a comparable efficacy to an MLP.

2 Background– Multilayer Perceptron Architecture

Two terms commonly used in describing ANNs are **neurons** and **weights**. In MLPs, neurons are the space where outputs from one layer and inputs to the next layer may be found. The weights are the *things in between neurons*. They are what processes information from one layer to the next. In most current deep learning architectures this is where nearly all of the trainable parameters can be found.

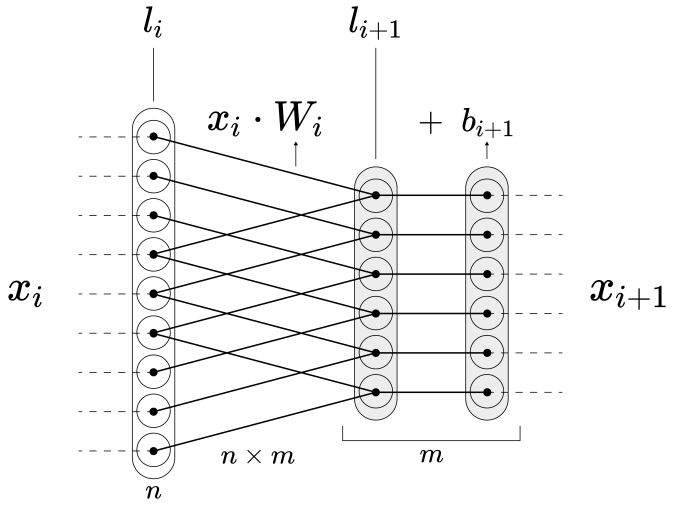


Figure 1: visual representation of MLP forward function

Let's say we have two layers of neurons in a deep neural network, l_i, and l_{i+1}, holding n and m neurons, respectively. l_i takes input array x_i and processes that through l_{i+1} into x_{i+1}. Between these two layers there will be trainable parameters W_i, a matrix of size $n \times m$. There is a also bias term, b_{i+1}, an array of size m. We define the MLP forward function as,

$$x_{i+1} = x_i \cdot W_i + b_{i+1}$$

noting that this operation contains $O(mn)$ trainable parameters.

3 Point Cloud Network Architecture

In contrast to an MLP, the trainable parameters of a PCN are all *neuron-centric*. What is learned are *features of the neurons themselves*, rather than *something in between*. In an MLP, we would say that the bias term, b is *neuron-centric*, but not W which contains a large majority of MLP parameters.

We will treat the features of neurons as positional information (i.e. each neuron is a point in space, hence the name). The rest of this section explains step-by-step how to use these neuron features to process input data in the same way, and with the same expressiveness, as an MLP.

3.1 Distance Matrix

Going back to the prior example network— this time we'll say l_i, and l_{i+1} are actually trainable parameters, where l_i is of shape $n \times d$ and l_{i+1} is of shape $m \times d$. d is a hyperparameter representing the number of features each of our neurons have, or we can say this is the *dimensionality* of the space our neurons exist in. d is an especially interesting hyperparameter because it allows us to scale up or down the number of parameters in our network without affecting the number of features in a given layer. We'll also use bias term b_{i+1} of size m again.

The W from an MLP is of shape $n \times m$. In this step, we can generate an equally shaped distance matrix $D(l_i, l_{i+1})$, where $D_{j,k}$ is the distance between neurons $l_{i,j}$ and $l_{i+1,k}$.

$$D_{j,k}(l_i, l_{i+1}) = \sqrt{\sum_{c=1}^{d} (l_{i,j,c} - l_{i+1,k,c})^2}$$

The intention is to replace W with D as follows,

$$x_{i+1} = x_i \cdot D(l_i, l_{i+1}) + b_{i+1}$$

However, D only contains *nonnegative* numbers, whereas $W \in \mathbb{R}^{n \times m}$. Using W, a network can choose to *flip* and *scale* the signal passed forward from one neuron to another, whereas using D, a network can only *scale* signals. This would make our network using D fundamentally less expressive than one using W. D is also prone to exploding/vanishing gradients.

In this work D is given as the euclidean distance between d-dimensional points, but $D_{j,k}$ has many possible implementations. The important feature of D is that it outputs an appropriately shaped matrix that facilitates interaction between every neuron in l_i, with every neuron in l_{i+1}. An example of an alternate implementation would be to omit the square root in the definition above. Another example would be the product $l_i l_{i+1}^\top$ giving D a similar property to the multiplication of *keys and queries* in transformers [12] or $UV^\top$ in low-rank factorization [5,6,13,3]. There likely exists a more optimal definition of D than the one defined here.

3.2 Distance-Weight-Function

The *distance-weight-function*, denoted here as F, is an element-wise function to pass D through. The goal of F is to project D into a space that makes it as expressive as W and to provide regularization properties. In this paper the **triangle wave** is selected for F. Let $F_{\lambda,\epsilon}$ be an element-wise triangle wave function centered around 0 with amplitude λ and period ϵ, with a regularization term included.

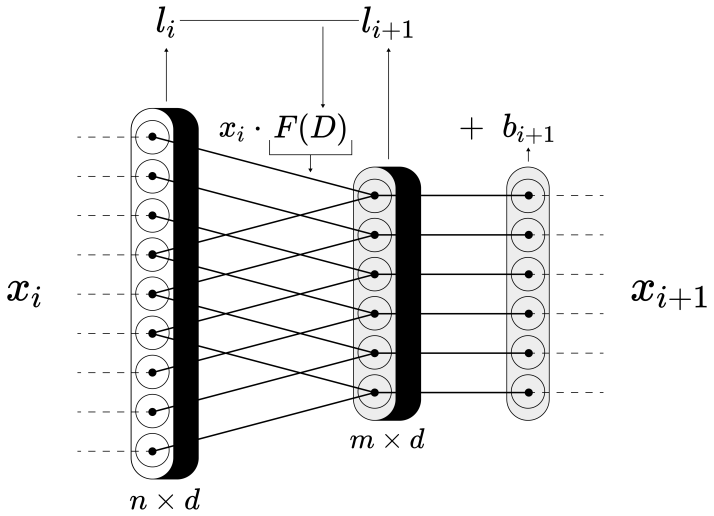


Figure 2: visual representation of PCN forward function

$$F_{\lambda,\epsilon}(z) = \frac{\mathbf{1}}{\sqrt{\boldsymbol{n}}} \cdot \frac{\lambda}{\epsilon} \cdot (\epsilon - |z \bmod 2\epsilon - \epsilon| - \frac{\epsilon}{2})$$

There is room for simplification, but the above equation is what is used in this work. $\frac{\mathbf{1}}{\sqrt{\boldsymbol{n}}}$ is selected as the regularization term in order to maintain a stable signal moving forward through the network, agnostic of layer size and network depth. This term was found through a trial-and-error approach observing the variance of signal passed through untrained networks, which can be found in the provided source code. A better regularization term likely exists.

Selection of The Triangle Wave. The triangle wave is selected for two desirable properties. Firstly, it takes any number $\in \mathbb{R}$ and clamps it to the range $[-\lambda, \lambda]$. This provides important control over the stability of our signal moving forward through the network, ensuring that no weights are excessively large in magnitude, regardless of how much neurons may explode away from, or implode into one another during the learning process. This in turn allows for a steady flow of gradients during backpropagation.

The second property that is specific to the triangle wave is its constant gradient and continuity— informed by the prevalence of the ReLU shape for nonlinearities[8]. Cos/sin have saddle points where gradients may get stuck. Square waves' gradients are flat and saw waves are discontinuous which may lead to the network *pushing* or *pulling* a weight *up* or *down* the saw wave's drop off.

3.3 Forward Function

The PCN forward function is given as follows,

$$x_{i+1} = x_i \cdot F_{\lambda,\epsilon}(D(l_i, l_{i+1})) + b_{i+1}$$

which has $O(n + m)$ trainable parameters, in contrast to the $O(nm)$ trainable parameters in an MLP.

4 Training

This section details the model architectures implemented as well as the full training process.

Techniques such as random image augmentation, batch normalization, or residual connections are refrained from being used in favor of the direct comparison of linear layer performance of MLPs and PCNs over achieving state of the art (**SOTA**) performance. Additionally, a limited compute budget informs several design choices seen in this section.

4.1 Model Definitions

A modest variety of model categories are trained to evaluate the PCNs performance in different circumstances. For each model category there is a single baseline model that uses MLPs and one or more equivalent PCN models. All PCN models use hyperparameters $\lambda = 1, \epsilon = 0.1$.

Details about the shape and depth of each network can be found in figure 3.

4.1.1 LinearNet

Baseline Network. LinearNet-MLP consists of linear layers followed by ReLUs. A linear layer with no ReLU is applied to produce the final output.

PCN Network. Four LinearNet-PCN$_d$ models are trained, each differing only in their dimensionality ($d \in [4, 8, 16, 32]$). Each PCN takes the baseline definition and replaces all MLP layers with equally shaped PCN layers with no further modification.

4.1.2 ConvNet

Baseline Network. ConvNet-MLP has a *feature extractor* network consisting of convolutional layers followed by ReLUs. The feature extractor network is then fed into the *classifier* network, consisting of linear layers followed by ReLUs with a linear layer at the end.

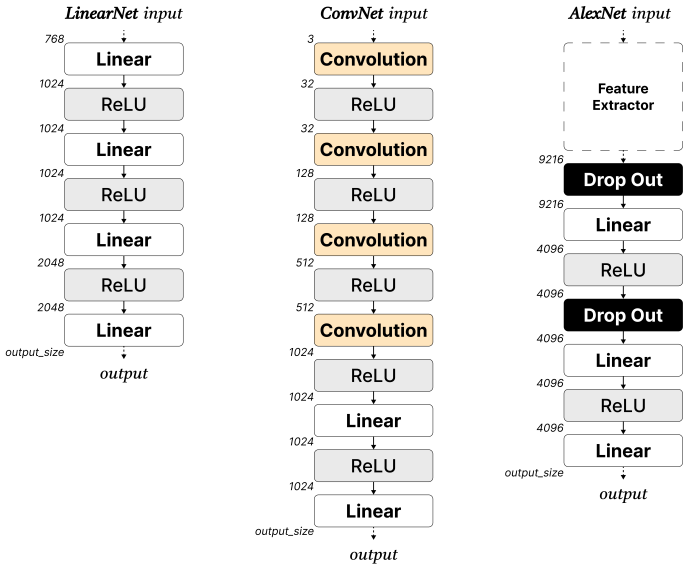


Figure 3: Illustration of LinearNet, ConvNet, and AlexNet architectures. On the left side of each network are the sizes of the signal passed forward through the network.

PCN Network. ConvNet-PCN$_{16}$ uses the same feature extractor as ConvNet-MLP. For the classifier a PCN is used instead of an MLP for linear layers with no further modifications.

4.1.3 AlexNet

Baseline Network. AlexNet-MLP is an untrained replica of the original model with a single modification made to the last linear layer in order to output the appropriate number of class predictions for each of CIFAR-10 and CIFAR-100. Like the previous ConvNet, AlexNet also consists of a convolutional *feature extractor* network followed by a linear *classifier* network. The classifier network employs $dropout_{\mathbf{=0.5}}$ before each linear layer, besides an isolated linear layer at the end [8].

PCN Network. AlexNet-PCN$_{16}$ uses the same feature extractor as AlexNet-MLP. In the classifier, MLPs are replaced with PCNs for linear layers. AlexNet-PCN$_{16}$ also features $dropout_{\mathbf{=0.5}}$ layers as used in the original.

4.2 Datasets

CIFAR-10 and CIFAR-100 are two popular image classification datasets. Both are labeled subsets of the tiny images dataset. CIFAR-10 consists of 60000

32x32 images divided into 10 classes, with 6000 images per class. The dataset is split into 50000 training images and 10000 withheld test images with exactly 1000 images of each class in the test set. CIFAR-100 is the same as CIFAR-10 but with 600 images per class, and follows the same principal for train/test split. The images and classes used in CIFAR-10 are mutually exclusive from those in CIFAR-100 [7].

The CIFAR datasets are chosen for benchmarks in order to strike balance between task difficulty and compute required. The MNIST dataset is too easy to solve— very small networks can achieve close to 100% test accuracy— so it is difficult to extract conclusive results about a PCN's efficacy in comparison to an MLP on this dataset. ImageNet, the dataset AlexNet was originally trained on, would require too much compute. The CIFAR datasets are sufficiently difficult tasks, while also being small enough to train the largest models in a reasonable amount of time given compute constraints.

Although MNIST is not used as a benchmark in this work, it was a valuable resource in performing rapid preliminary testing of the PCN architecture. The MNIST dataset was used in making all of the architecture and regularization choices seen throughout this paper [2].

4.3 Preprocessing

For training/validation of LinearNet models, images are scaled down from 32x32 to 16x16, reducing the first linear layer's input size from $3072 \rightarrow 768$. Conversely, All images are scaled up to 227x227 for AlexNet models to match the original paper [8].

4.4 Initialization

MLP and convolutional parameters use default initializations given by *torch.nn.Linear* and *torch.nn.Conv2d*, respectively.

PCN neuron positional values are initialized uniformly over the range $[-1, 1]$ and bias terms uniformly over the range $[-0.1, 0.1]$.

4.5 Loss, Gradient, and Optimizers

Loss for all models are calculated using *torch.nn.CrossEntropyLoss*, and parameter gradients are calculate using pytorch's autograd feature.

MLP and convolutional parameters are updated with *stochastic gradient descent* (**SGD**), via, *torch.optim.SGD*.

PCN parameters are updated with a slightly modified version of SGD that is informed by layer size. Given layer size n, PCN parameters l_i, b_i, gradients Δl_i, Δb_i, and learning-rate γ we perform a PCN's SGD update as follows:

$$l_i \coloneqq l_i - \gamma \Delta l_i \frac{\boldsymbol{n}}{\boldsymbol{\log_2 n}}$$
$$b_i \coloneqq l_i - \gamma \Delta b_i \mathbf{10^5}$$

Both of the terms $\frac{n}{\log_2 n}$ and 10^5 are used in order to make parameters throughout the network learn at close to the same rate, agnostic of layer size. These values were selected in early tests by observing the variance in gradients during the training process over a variety of network shapes. This optimization strategy does not account for irregularities in gradients resulting from network depth and artifacts of this fact may become pronounced in the loss/accuracy curves when attempting to train deep PCN networks, although residual connections may alleviate this problem [4]. For the purpose of the trials done in this paper, the above optimization strategy is sufficient.

4.6 Training Loop Details

All models are trained with a **batch size of 1024** and **learning-rate of 0.0001**, for **3.5k epochs**. For each training iteration, we aggregate the *loss*, and for each epoch we aggregate both *training accuracy* and *test accuracy*, seen in figure 5.

5 Results

This section presents the results of training all LinearNet, ConvNet, and AlexNet architectures over the CIFAR-10 and CIFAR-100 datasets. Key results are collected in table 1. Reported train/test accuracies are generated with the the final models after training. During the training of ANNs, it is normal for accuracies to fluctuate from epoch-to-epoch which introduces minor variance into these results. Loss, training accuracy, and test accuracy curves are collected in figure 5 of the appendix, which display more stable trends.

Discussion focuses on linear parameter count and test accuracy.

5.1 LinearNet

Four LinearNet-PCN$_d$ models and LinearNet-MLP are trained. For $d = 4, 8$ there is a degradation in performance relative to the MLP. At $d = 16, 32$, the PCN outperforms the MLP on both datasets. As d increases, the PCNs experience more overfitting. The MLP experiences substantially more overfitting than all PCNs. LinearNet-PCN$_{32}$, the largest PCN in this class of models, has 161k parameters, which is a **95.9%** reduction from 3.95 million parameters in the MLP. Additionally, figure 5 displays a consistent increase in PCN performance with an increase in d.

		CIFAR-10		CIFAR-100	
model	**# linear params (millions)**	top-1 acc. (train)	**top-1 acc. (test)**	top-1 acc. (train)	**top-1 acc. (test)**
LinearNet-PCN$_4$	**0.024**	48.4	**46.0**	23.5	**20.9**
LinearNet-PCN$_8$	**0.044**	53.4	**48.9**	26.2	**22.7**
LinearNet-PCN$_{16}$	**0.083**	61.4	**53.0**	31.9	**26.1**
LinearNet-PCN$_{32}$	**0.161**	66.9	**52.8**	39.1	**28.1**
LinearNet-MLP	**3.957**	96.8	**52.1**	78.7	**25.2**
ConvNet-PCN$_{16}$	**0.035**	88.4	**60.0**	72.9	**26.1**
ConvNet-MLP	**1.06**	98.9	**58.1**	99.8	**27.3**
AlexNet-PCN$_{16}$	**0.296**	85.7	**78.9**	51.6	**43.7**
AlexNet-MLP	**54.575**	84.1	**78.6**	52.5	**47.5**

Table 1: Train and test accuracies (%) over both CIFAR-10 and CIFAR-100 datasets for each model, along with the parameter counts of their linear layers.

5.2 ConvNet

Both ConvNet-MLP and ConvNet-PCN$_{16}$ have 5.35 million convolutional parameters. ConvNet-PCN$_{16}$ outperforms ConvNet-MLP by **1.9%** on CIFAR-10 and underperforms by **1.2%** on CIFAR-100. Similarly to LinearNet, ConvNet-MLP experiences more overfitting than ConvNet-PCN$_{16}$. ConvNet-PCN$_{16}$ has 35k linear parameters, which is a **96.7%** reduction from 1.06 million linear parameters in the MLP.

5.3 AlexNet

Both AlexNet-MLP and AlexNet-PCN$_{16}$ have 2.47 million convolutional parameters. AlexNet-PCN$_{16}$ outperforms AlexNet-MLP by **0.3%** in CIFAR-10, and underperforms by **3.8%** in CIFAR-100. Both models experience similar amounts of overfitting. AlexNet-PCN$_{16}$ has 296k linear parameters, which is a **99.5%** reduction from 54.6 million linear parameters in the MLP.

6 Limitations and Future Work

6.1 Memory Requirements

As has been demonstrated by this work, the PCN architecture can substantially reduce the number of parameters needed to train linear layers. However, the implementation seen here does not actually reduce the memory requirements. This is due to my reliance on pytorch's native autograd feature and *torch.cdist* to find D. During the forward pass, D in its entirety is calculated and stored in memory, which is the same size as W. A fused kernel function for calculating $x_{i+1,k}$ that never stores D but instead calculates $D_{j,k}$ as needed could be used.

$$\sigma_k(x_i) = b_{i+1,k} + \sum_{j=1}^{n} x_{i,j} \cdot F_{\lambda,\epsilon}(D_{j,k}(l_{i,j}, l_{i+1,k}))$$

Successfully implementing this along with its corresponding gradient functions on accelerated hardware would reduce memory consumption $O(n^2) \rightarrow O(n)$ during training and inference.

6.2 Compute Requirements

Two limiting factors of deep learning are **memory** and **compute**. The PCN architecture can alleviate memory consumption, but requires $O(d)$ times more compute than an MLP.

6.3 Network Stability

As has been stated previously, all regularization terms used in the PCNs presented in this work were found through trial-and-error rather than rigorous math. Because of this, these PCNs are not resilient to their hyperparameters and a more robust PCN definition should be investigated.

6.4 Applying PCNs Elsewhere

In this work the PCN architecture is applied to linear layers. The same concept can be applied to the convolutional layers along the *channel* axis, and to graph layers along the *node-feature* axis.

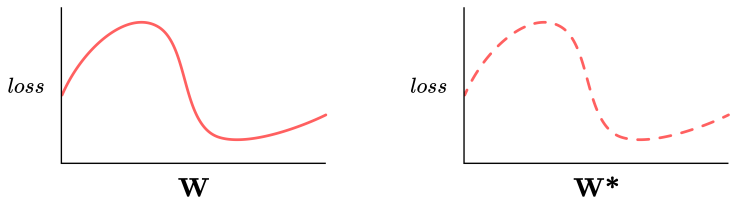


Figure 4: On the left: loss distributed across $\boldsymbol{W}$. On the right: loss distributed across $\boldsymbol{W^*}$, *projected into the space of* $\boldsymbol{W}$. Blank space represents where $\boldsymbol{W^*}$ does not occur.

6.5 Conjecture— Why PCNs Work

The concept of a PCN can be boiled down to an MLP where we generate a plausible W, similarly to low-rank factorization[5,6,13,3]. Let $\boldsymbol{W} = \mathbb{R}^{n \times m}$ be the set of all possible values for W, $\boldsymbol{W^*} \subseteq \boldsymbol{W}$ be the set of all possible values for $F(D)$, and $\overline{L}_{\boldsymbol{W}}$ be the mean loss w.r.t. $\boldsymbol{W}$. If $\overline{L}_{\boldsymbol{W^*}} = \overline{L}_{\boldsymbol{W}}$, then $F(D)$ should have a comparable efficacy to W. Consequently, if $\overline{L}_{\boldsymbol{W^*}} < \overline{L}_{\boldsymbol{W}}$ or

$\overline{L}_{\boldsymbol{W}^*} > \overline{L}_{\boldsymbol{W}}$, then $F(D)$ would be expected to perform better or worse than W, respectively.

It may be interesting to investigate $F(D)$ that *maximizes* $\overline{L}_{\boldsymbol{W}} - \overline{L}_{\boldsymbol{W}^*}$.

7 Ethical Concerns

With the exception of recent high profile publications, it seems a relatively uncommon practice to include an ethics section in a deep learning paper like this one. I use this section as a platform to attempt to mindfully outline some of my concerns. I include this section to advocate for a culture within academia that normalizes, legitimizes, and prioritizes this conversation— hoping that a more organized practice forms.

Downstream Consequences. Deep Learning is a unique technology in that it is largely task-agnostic. Because of this, the set of downstream applications is uncharacteristically large compared to other technology. Although the PCNs presented in this paper are applied to test datasets, the intention is to integrate this into existing deep learning architectures for which there are existing harmful applications. This makes it important to be cognizant of and acknowledge these harmful applications.

Mindful Conversations. Having productive conversations about A.I. safety is a bit paradoxical. It is surely helpful to be aware of potential negative applications of deep learning, yet it may actually be harmful to indulge in any unnecessary details that don't move the conversation forward. For example, I would consider media outlets echoing unproductive details about harmful applications to be an unethical practice.

8 Acknowledgements

I would like to thank Ryan Schaake for offering fruitful comments, review, and insight.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.,

2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[2] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29 (6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

[3] Moonjung Eo, Suhyun Kang, and Wonjong Rhee. An effective low-rank compression with a joint rank selection followed by a compression-friendly training. *Neural Networks*, 161:165–177, 2023. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2023.01.024. URL https://www.sciencedirect.com/science/article/pii/S0893608023000242.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[5] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán. Low-rank compression of neural nets: Learning the rank of each layer. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8046–8056, 2020. doi: 10.1109/CVPR42600.2020.00807.

[6] Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N. Gomez. Exploring low rank training of deep neural networks. 2022.

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[9] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[10] OpenAI. Gpt-4 technical report. 2023.

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[13] Kiran Vodrahalli, Rakesh Shivanna, Maheswaran Sathiamoorthy, Sagar Jain, and Ed H. Chi. Nonlinear initialization methods for low-rank neural networks. 2022.

A Supplemental Material

A.1 Source Code

Source code and other materials can be found at https://gitlab.com/cHetterich/pcn-paper-and-materials.

A.2 Figures

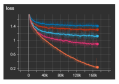
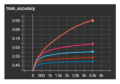
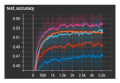
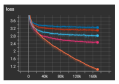
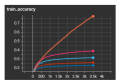
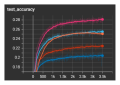
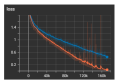
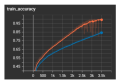
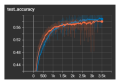
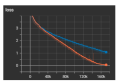
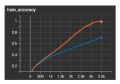
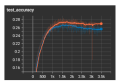
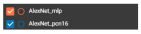
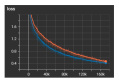
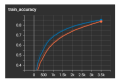
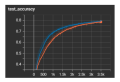
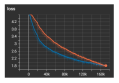
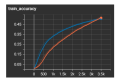
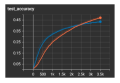
Model Class	Key	Dataset	Loss	Train Accuracy	Test Accuracy
LinearNet		CIFAR-10			
		CIFAR-100			
ConvNet		CIFAR-10			
		CIFAR-100			
AlexNet		CIFAR-10			
		CIFAR-100			

Figure 5: A collection of training accuracy, test accuracy, and loss curves for all models.