Learning Compact Vector Representations from Weight Matrices

Chris Liu

Abstract

In natural language processing, well-trained word embeddings can not only transfer to downstream tasks but also distinguishes words from each other by semantics (e.g., synonyms), which can be reflected by vector distance. Ideally, distance calculation between neural networks is supposed to be easier, as we can directly calculate distance between matrices. However, we first identified a problem: weight matrices do not have such a convenient property naturally by illustrating it using 2-d visualizations. We propose to extract *a compact vector-like representation of deep neural networks*. More specifically, we study the problem of learning to construct compact representations of neural network weight matrices by linearly projecting them into a smaller space. Such representations are derived from the task of generalization performance prediction (i.e., predicting generalization performance based on model weights) and have properties that can be used to distinguish good and bad weights, similar to discrete word embeddings or dynamic sentence embeddings. We analyzed the compact vectors by projecting them into a 3-d space and observed evidence that supports our hypothesis: compact vectors obtained by linearly transforming the neural network weight matrices not only separate those with high and low accuracies and are also preserve the original performance ranking.[1]

1. Introduction

Two of the most relevant techniques to learning compressed representations are word (Mikolov et al., 2013a) and sentence (Reimers & Gurevych, 2019) embedding and autoencoders (Hinton & Salakhutdinov, 2006). In the generic form of word embedding, given a training corpus, each token is first initialized as a dense vector and is then trained using backpropagation with some objective. A trained word vector can then be used as an input to a fully-connected layer or a recurrent layer for further feature extraction. To obtain a sentence/paragraph vector, Le & Mikolov (2014) simply concatenate or average the known word vectors to obtain a representation of a text span (e.g., a sentence, a paragraph, or even a document). Recently, a common approach is to calculate the average of the BERT (Devlin et al., 2019) output given a sentence as input. Instead of having an explicit embedding matrix, an autoencoder uses an encoder module to encode a high-dimensional input into a latent vector, which is often low-dimension and thus efficient (Hinton & Salakhutdinov, 2006). The objective is to reconstruct the original input features given only the intermediate latent representation. However, an autoencoder is usually trained in an unsupervised fashion, as the "true label" is the original input.

In the problem of generalization performance predictions of deep neural networks, one tries to employ information from data, training procedure, and model to predict the generalization performance (Jiang et al., 2019; Yak et al., 2019; Eilertsen et al., 2020; Martin & Mahoney, 2020; Unterthiner et al., 2020; Martin & Mahoney, 2021; Martin et al., 2021). In this work, we only consider settings where such information comes from the weight matrices (Unterthiner et al., 2020), because we hope the learned vector can directly represent the original network. More formally, the predictor $f_\theta : \mathcal{X} \to \mathcal{Y}$ is a function that maps x, a set of input weights from a base network, to y, the generalization accuracy, a positive real number between 0 and 1. Unterthiner et al. (2020) shows that using the flattened weight matrices from all layers yields a good baseline with a fully-connected neural network as the predictor, and incorporating only statistics, such as mean, variance, and quantiles, from each layer as features is already sufficient for such task.

While the two approaches in the work above are promising in terms of obtaining the expected test accuracy based on weight matrices, we argue for two drawbacks given the goal of this work: 1) due to the black box nature of deep neural networks, it is unclear how the predictor infers the test accuracy by utilizing features in weight matrices, rendering the connection between weight matrices and the generalization performance unclear; 2) the inputs of flattened weight matrices cannot be scaled to large models, because the first layer of our predictor needs to have $M \cdot H$ parameters, where M is the dimension of the flattened weight matrices (or the number of parameters in our base network), and H is the hidden size of the first layer. We also identified a problem

[1] Supplementary materials are included here.

in neural network weights: networks with similar performance do not consistently stay close to each other. However, such a property could be beneficial in both retrieving similar networks without doing inference at all and analysis of the neural network weights themselves. We use 2-d plots and heat maps to illustrate such property in Figure 6.

To address the above drawbacks, we borrow methods from word and sentence embedding and autoencoders to encode weight matrices into a lower-dimensional space. We introduce a set of small and dynamically initialized encoders as the first layer of our generalization performance predictor to encode a weight matrix into a compact vector (ComVeX) representations of some fixed or varying latent embedding dimension. We choose the generalization prediction task because we hope the compact vector representation can reflect the generalization performance of the whole network. The vectors can then be either concatenated or averaged and fed into later layers for the regression task. Once trained, given a neural network of the same architecture, the encoders alone can encode it as a set of vector representations (one for each layer), and we hope that distance metrics can be used to distinguish networks that perform well from those that perform badly. From here, we will also explore the possibility of using compact vectors to construct a small network that partially represents the original network.

While we have not demonstrated how to use such embeddings in this work, we do emphasize several of its real-world application. First, such embeddings could be used for more efficient vector search and storage. For example, given a set of neural network weights, the original model may take gigabytes of space to store. However, with the proposed approach, we only need to store their embeddings and the weights of the encoders. We can then recover the weights or the original network on demand by reversing matrix multiplication or deconvolution. Also, such compact vectors are faster to search naturally. Moreoever, we also inject the performance information to the compact vectors, which allow us to infer its performance without doing inference at all.

Our contributions are listed as follows:

- We combine ideas from word and sentence embeddings and autoencoders to learning compact vector representations for weight matrices in neural networks.
- We address the scalability problem in Unterthiner et al. (2020) by using a set of small linear encoders as the first layer of the predicting to generate dynamic compact vector representations in the forward pass of training.
- We show that the compact vectors still preserve the "goodness" of a network (generalization performance) and provide visualizations in a 2-d space.

2. Related Work

2.1. The Generalization Performance Prediction Problem

The problem setting in this work largely follows Unterthiner et al. (2020), as introduced above. Experiment-wise, Unterthiner et al. (2020) considers under-parameterized convolutional neural networks trained on four image classification datasets. They also experiment with adding hyperparameters as features but do not observe significant performance gain. They show that the predictor learned in this fashion can also generalize across architectures. However, we aim to address the black-box nature of the original prediction task and make the predictor network scalable. Eilertsen et al. (2020) studies the opposite problem to Unterthiner et al. (2020): predicting the hyperparameter based on weight matrices. Similar problems are also studied in recent years (Jiang et al., 2019; Yak et al., 2019; Eilertsen et al., 2020; Martin & Mahoney, 2020; 2021; Martin et al., 2021).

Instead of predicting the expected test performance, Jiang et al. (2019) and Yak et al. (2019) focus on predicting the generalization gap (i.e., the difference between training and test performance). In this setting, the predictor needs to learn not only how well the base model does on unseen data but also potentially the performance on the training data, if in an implicit manner. Jiang et al. (2019) approximates the minimal distances to class boundary for each example in each hidden layer and use the margin distributions as the input to learning such a mapping to the generalization gap, instead of using the raw weight matrices. Yak et al. (2019) also incorporates the margin distribution information, except that they employ a recurrent neural network architecture to handle varying network depth. We handle this by using a dynamic set of small encoders to handle the weight matrix in each layer, and the size of this set can be easily computed during the initialization phase.

Martin & Mahoney (2020) and Martin et al. (2021) empirically confirms findings in Unterthiner et al. (2020). Previously, Martin & Mahoney (2021) showed that a better correlation between weights and the performance from models in vision and language domains can be obtained by using even more complex statistics other than mean, variance, and quantiles.

2.2. Encoding words and sentences

Before the pre-training era, Word2Vec was the most popular word embeddings (Mikolov et al., 2013a;b). It combines two unsupervised methods for learning word vectors: contunuous bag-of-words (CBOW) and skip-gram. The CBOW model tries to predict the current word from a window of context words, and the skip-gram model does the opposite: it predicts the context words given the current word. The

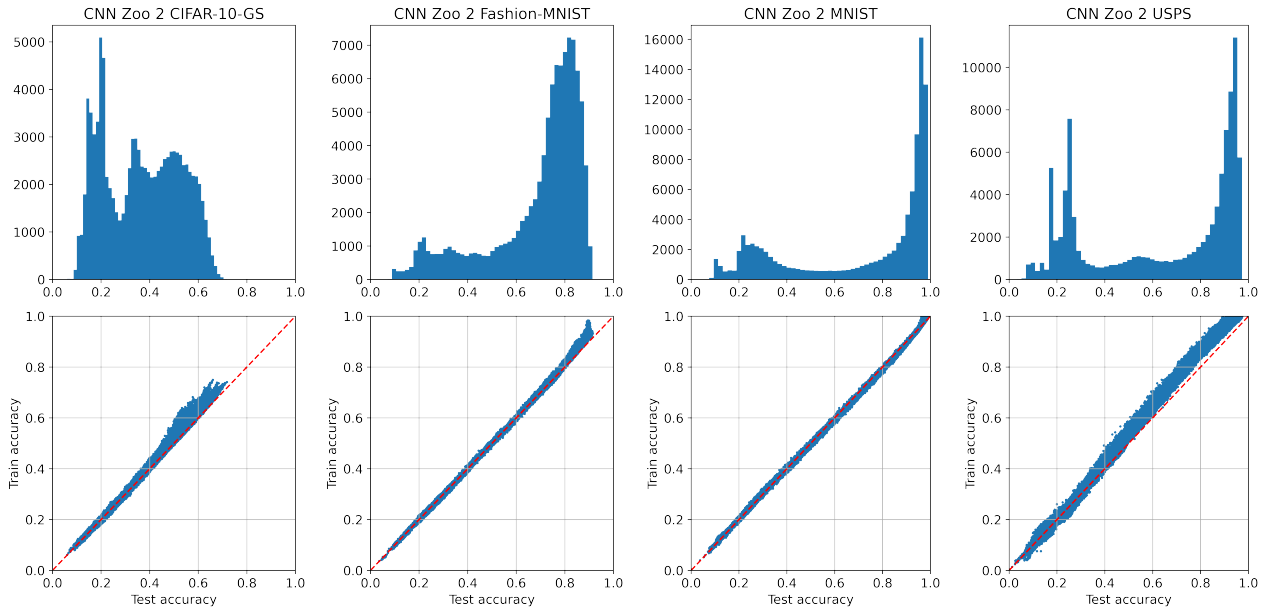

Figure 1. **Top**: The test accuracy distribution of the CNN Zoo 2 dataset. **Bottom**: Test accuracy versus training accuracy. This reflects whether base CNN models are overfitting.

other popular word embedding approach is GloVe (Pennington et al., 2014). It is also trained in an unsupervised fashion by focusing on words co-occurrences over the training corpus, resulting in an correlation between the distances between word vectors and their semantic similarity. Later, ELMo is the first model that can generate dynamic word embeddings based on a word and its context (Peters et al., 2018). It achieves context-dependence by employing a bidirectional LSTM, as it can predicts the next word from both directions. After the emergence of Transformer (Vaswani et al., 2017), BERT (Devlin et al., 2019) became the second model that can produce contextual representations of words due to the advantage of positional embedding along with the masked language modeling and next sentence prediction tasks.

To generate sentence embeddings, it is infeasible to learn a representation for every single pemutation of the words, so it is crucial to aggregate information from word vectors. Le & Mikolov (2014) considers concatenation and averaging over the individual word vectors. Transformer-like architectures, such as BERT, can also be used to directly generate sentence embeddings. A representative architecture is Sentence-BERT (Reimers & Gurevych, 2019), which modifies the original BERT architecture to a siamese network and uses triplet network structures. Sentence-BERT is able to produce sentence embeddings that reflect semantic similarities using distance metrics like cosine-similarities. Other relevant works include: Reimers & Gurevych (2020); Thakur et al. (2021a); Reimers & Gurevych (2021); Wang et al. (2021); Thakur et al. (2021b).

3. Method

We now formally describe our dataset collection method, the proposed compact vector representations from weight matrices, and the encoder-predictor architecture for learning such representations. Figure 2 shows the proposed encoder-predictor architecture.

3.1. CNN Zoo 2

Unterthiner et al. (2020)'s CNN Zoo dataset was collected by training around 30,000 models and taking a single epoch after a particular training session. This strategy created a diverse set of model weights and test accuracies, but it failed to resemble the real world hyperparameter tuning scenario. Therefore, we employed a more realistic model dataset collection method: for a particular image classification dataset, we train 1000 base CNN models and take all the epoch-wise checkpoints from the training session. This dataset collection strategy gave us a dataset of 1000×100 (epochs) models. We believe incorporating checkpoints from all epochs can allow a predictor to learn more fine-grained training dynamics than using checkpoints from distinct training trajectories.

We performed this strategy on four datasets: CIFAR-10 (Krizhevsky et al., 2009), MNIST (Cireşan et al., 2011), Fashion-MNIST (Xiao et al., 2017), and USPS (Carlucci

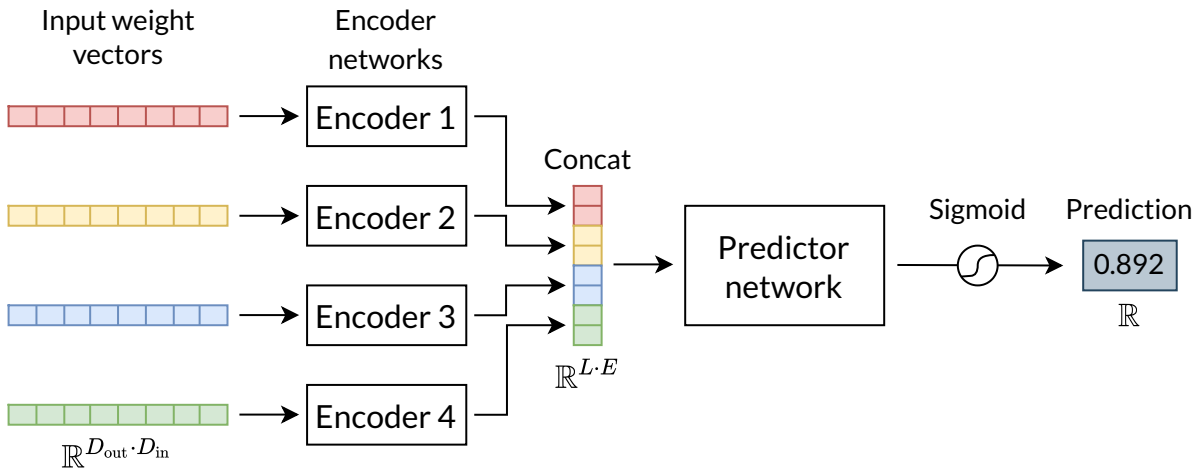

Figure 2. For each of the L layers in a base CNN, we use a individual encoder network to do projection. In our case, L is 4. The reduced vectors for all layers are concatenated together to feed into the predictor network. The sigmoid transformation is used to ensure the final accuracy is strictly between [0, 1].

et al., 2019). We chose USPS instead of SVNH (like in CNN Zoo version 1) because of the large number of samples in SVHN and time constraints. Like Unterthiner et al. (2020), all datasets are converted to grayscale images.

For the base CNNs, we increased the filter size from 16 to 64 but did not change the number of layers. The numbers of parameters in each layer are `[640, 36928, 36928, 650]`, resulting in a 75,146-parameter model. We split the dataset using 0.8/0.1/0.2 for training/validation/testing. The final CNN Zoo 2 dataset (with 1000 base CNNs trained on one image classification dataset) is approximately 26.1 GB. In Figure 1, we present the test accuracy distributions and the test accuracy versus training accuracy plots for base CNNs trained on all four image classification dataset.

3.2. Extracting compact vector representations

A naive approach to directly use a linear layer with identity activation as the first layer but with some small output dimension to form a bottleneck. This bottleneck can recover the introduced distributive architecture if we combine all linear encoders into one single layer. In our case, we use 256 as the output dimension because we set the embedding dimension to be 64, and we have 4 layers in the base CNN models. We call such an encoder $\text{FC}_{\text{Linear}}$.

We now describe a way to encode each weight matrix separately. Let f_θ denote a fully-connected neural network parameterized by weights θ and W_l the weight matrix from layer l. A weight matrix W_l has shape $D_{\text{out}} \times D_{\text{in}}$, where D_{in} is the dimension of the input vector (e.g., the dimension of a flattened image), and D_{out} is the output dimension, or, equivalently, the hidden size of layer l. Each encoder E_l by default is a shallow and narrow fully-connected network, which takes a flattened weight matrix W_l^{flat} as its input (an alternative is to use a 1-D convolutional layer to reduce the dimension of the original weight matrix). The hidden size H of each emebdding network corresponds to the latent embedding dimension D_{emb} of a particular weight matrix W_l. Here, note that if we select to concatenate the output compact vector representations, the latent embedding dimension for each weight matrix (or the hidden size of each encoder) does not have to be the same, and the resulting feature vector will have size $\sum_i^k = D_{\text{emb}}^i$. The benefit of using a varying latent embedding dimension is that we can choose to represent a layer with more parameters by a larger compact vector. However, this does not directly fit into the averaging approach, because it is infeasible to average across varying length vectors. Therefore, we consider concatenation with fixed latent dimension only in this paper and leave the averaging approach for future work. Once we obtain the aggregated feature vector, we can use any architecture in later layers as long as it accepts a 1-d feature vector as input and outputs a scalar.

The encoder-predictor architecture can be trained using standard regression task framework. Like Unterthiner et al. (2020), we also add a sigmoid transformation at the end of the network to constraint the output value in the range $[0, 1]$. We apply mean squared error loss to calculate the difference between the prediction and the true label.Once the encoder-predictor model is trained, we can decouple

Architecture	Complexity
FC network	$O(L \cdot \phi \cdot H)$
$\text{FC}_{\text{Linear}}$	$O(L \cdot \phi \cdot E)$
$\text{ComVeX}_{\text{Linear}}$	$O(L \cdot \phi \cdot E)$
$\text{ComVeX}_{\text{Conv}}$	$O(L \cdot \frac{3 \cdot \phi}{h_i} \cdot E)$

Table 1. Complexity of the number of parameters in the first-layer encoders. $\text{FC}_{\text{Linear}}$ and $\text{ComVeX}_{\text{Linear}}$ both have the same complexity because the first layer of $\text{FC}_{\text{Linear}}$ is simply a linear layer with identity activation, and the first layer in $\text{ComVeX}_{\text{Linear}}$ performs the encoding operations in a distributive fashion. For all models, E is usually about seven times smaller than H.

Architecture	Parameters	Test loss	Test R^2
FC network	35608249	0.011631	0.5214
$\text{FC}_{\text{Linear}}$	19974475	0.016832	0.2971
$\text{ComVeX}_{\text{Linear}}$	5954785	0.010816	0.5541
$\text{ComVeX}_{\text{Conv}}$	391325	0.001233	0.9496

Table 2. Predictors' performance on CNN Zoo 2 CIFAR-10.

the two modules and take the outputs from the encoders as the final compact vector representations. We can then use the set of vectors for similarity studies. We call this model $\text{ComVeX}_{\text{Linear}}$. Technically, $\text{ComVeX}_{\text{Linear}}$ is not so different from $\text{FC}_{\text{Linear}}$ because their parameter complexity in the first layer is identical. However, $\text{ComVeX}_{\text{Linear}}$ emphasizes the encoding of each layer separately with multiple smaller encoders, whose weights are also backpropagated separately.

3.3. Convolutional compact vector extractor

One problem with the default fully-connected encoders is still the scalability. Now we illustrate why it is the case. Let i denote the index of layer in the base CNN, L the number of layers in the base CNN, H the hidden size of the first layer of the predictor, E the latent embedding dimension of a single linear encoder in the proposed architecture. With a fully-connected network, the number of parameters in the first layer is

$$H + \sum_{i=1}^{L} |\phi_i| \cdot H, \tag{1}$$

where $|\phi_i|$ is the number of parameters in layer i of the base CNN, and H is the hidden size. The additional H term represents the number of biases. With the proposed linear encoder, we reduce the size of the first layer to

$$\sum_{i=1}^{L} \left[|\phi_i| \cdot E + E \right], \tag{2}$$

where $E < H$. However, because of the fully-connected nature of linear layer, the number of parameters in each encoder could still explode as the number of parameters in a certain layer of the base CNN is large. Here a weight matrix is flattened. To further reduce the weights in the first-layer encoders, we also proposed linear encoders using convolution. Previously, each hidden neuron of the linear encode connects to each of the input weight. With a convolutional encoder, instead of flattening a weight matrix, we reshape it into 2-d. Recall that a weight matrix for a convolution layer has shape $(D_{\text{out}}, D_{\text{in}}, F_{\text{height}}, F_{\text{width}})$, corresponding to (number of filters, input dimension, filter height, filter width). We combine the weights and bias and reshape such a weight matrix in every layer of the base CNN model into $(D_{\text{out}}, *)$, where * is the product of all other dimensions. For example, if a convolutional layer has input dimension = 1, 64 filters, kernel size of (3, 3), its weight matrix will have shape (64, 1, 3, 3), and the bias vector will be (64, 1). We can reshape (64, 1, 3, 3) to (64, 9) and concatenate both in the second dimension, resulting in a matrix of (64, 4). If each row (64 in total) in a weight matrix corresponds to a neuron and each column (10 in total) represent the number of connections to a neuron (or simply the flattened filter size + 1[2]), each filter of size $(3, F_{\text{height} \times \text{width}} + 1)$ slides through the dimension of neuron. Each convolution operation produces a scalar, and the entire convolution process through the weight matrix results in a vector. We then perform max-pooling over time to obtain a since scalar, and the final output will be a vector whose dimension equals the number of filters in one convolutional layer. Such convolution operation is similar to Kim (2014), except that we only use one convolution layer as a single encoder. Now, the number of parameters in the first layer will be

$$\sum_{i=1}^{L} \left[3 \cdot \frac{|\phi_i|}{h_i} \cdot E + E \right]. \tag{3}$$

Here, we use h_i to denote the number of filters in our base CNN, which is 16 for CNN Zoo version 1 and 64 for CNN Zoo 2. While E represents the latent embedding dimension, it also serves as the number of filters for a convolutional encoder. We call this architecture $\text{ComVeX}_{\text{Conv}}$.

To summarize, a linear encoder takes as input a flattened weight matrix of shape $(D_{\text{out}} \times D_{\text{in}},)$ from layer i, which is essentially a vector. A convolutional encoder takes as input a weight matrix, which is reshaped into $(D_{\text{out}}, D_{\text{in}})$ beforehand.

[2]The +1 is because we concatenated the weight matrix and the bias vector.

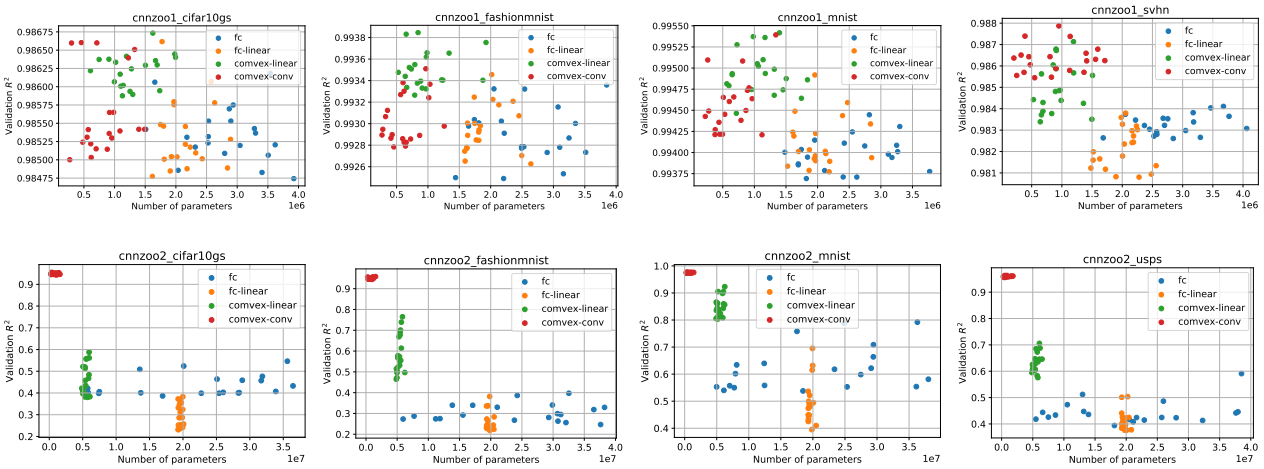

Figure 3. Number of parameters versus validation R^2 scores. For each model architecture, we tuned 100 sets of hyperparameters and took the top-20 candidates ranked by validation R^2 scores. In most scenarios, ComVeX-Linear and ComVeX-Conv achieved the best performances with the least number of parameters.

Architecture	**Parameters**	**Test loss**	**Test R^2**
FC network	32530569	0.023807	0.3966
$\text{FC}_{\text{Linear}}$	19827601	0.024885	0.3718
$\text{ComVeX}_{\text{Linear}}$	5851759	0.008128	0.7828
$\text{ComVeX}_{\text{Conv}}$	1530671	0.00173	0.9562

Table 3. Predictors' performance on CNN Zoo 2 Fashion-MNIST.

Architecture	**Parameters**	**Test loss**	**Test R^2**
FC network	36295603	0.02099	0.7596
$\text{FC}_{\text{Linear}}$	19930939	0.031575	0.6361
$\text{ComVeX}_{\text{Linear}}$	6249599	0.00694	0.9205
$\text{ComVeX}_{\text{Conv}}$	1400207	0.002254	0.9741

Table 4. Predictors' performance on CNN Zoo 2 MNIST.

Architecture	**Parameters**	**Test loss**	**Test R^2**
FC network	38509217	0.04628	0.4995
$\text{FC}_{\text{Linear}}$	20236345	0.057672	0.373
$\text{ComVeX}_{\text{Linear}}$	6065281	0.037939	0.5748
$\text{ComVeX}_{\text{Conv}}$	843887	0.002792	0.9693

Table 5. Predictors' performance on CNN Zoo 2 USPS.

4. Experiments and Results

4.1. Model selection

For all architectures and datasets, we randomly sampled 100 sets of hyperparameter configurations and train a target network using early stopping. The hyperparameter tuning procedure is largely similar to Unterthiner et al. (2020), but we listed the details in appendix. For CNN Zoo 2, we significantly increase the strength of dropout and weight decay to mitigate the overfitting issue in the vanilla fully-connected network.

4.2. Generalization performance prediction

Before studying the encoding power, we first trained the four model architectures on the original CNN Zoo version 1 to test the effectiveness of the proposed encoder-predictor architecture. We followed the same hyperparameter search procedure[3] as Unterthiner et al. (2020) and could obtain similar results with a fully-connected network. We reported the number of parameters, test loss, and test R^2 in Table 6, Table 7, Table 8, and Table 9 in Appendix. For test loss and test R^2, all four architecture have similar results, approaching to nearly perfect R^2 scores on the test set. However, only $\text{ComVeX}_{\text{Linear}}$ or $\text{ComVeX}_{\text{Conv}}$ achieve on par performance with the least amount of parameters. At best, we achieved similar performance with approximately 1/8 of the parameters of the vanilla FC network.

We then also evaluated these architectures on CNN Zoo

[3]One exception is that we only searched 100 hyperparameter configuration to find the best predictor because of the time limitation. Unterthiner et al. (2020) used 1,000 configurations.

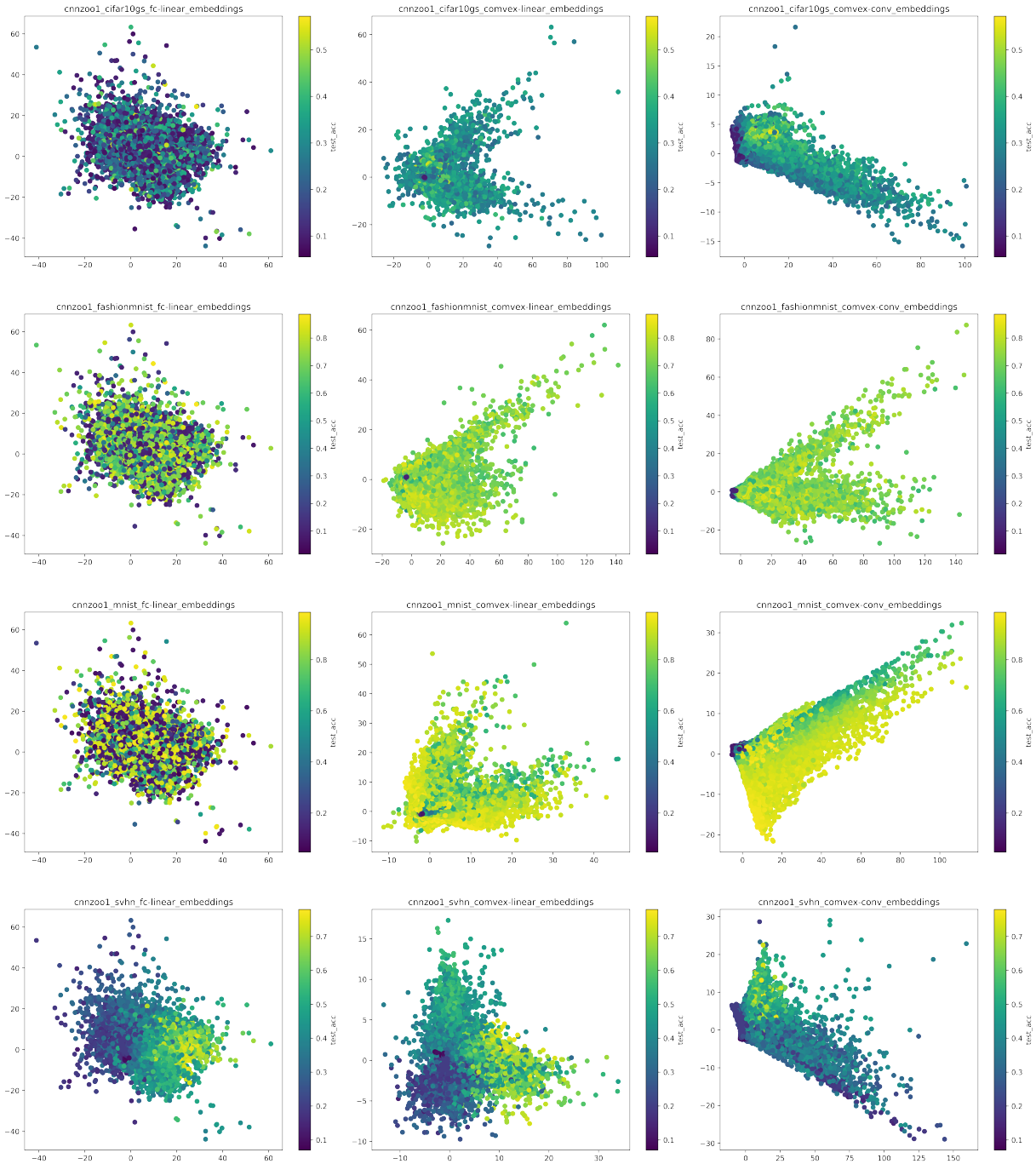

Figure 4. A 2-d visualization of the 256-dimensional embedding for different base CNN models in the test set of CNN Zoo version 1. We used PCA for dimensionality reduction. Each data point represent a single base CNN model's embedding, and its color reflects its true performance. We directly used the embeddings generated by FC-Linear and concatenated the four (one for each layer in a base CNN) 64-dimensional vectors produced by ComVeX-Linear and ComVeX-Conv so that all vectors' dimensions matched. The FC-Linear model struggles to distinguish high-quality models from low-quality ones, whereas the distributive embeddings generated by ComVeX-Linear and ComVeX-Conv show clear separation of dark and light color points. ComVeX has the best distinguishing behavior among the three.

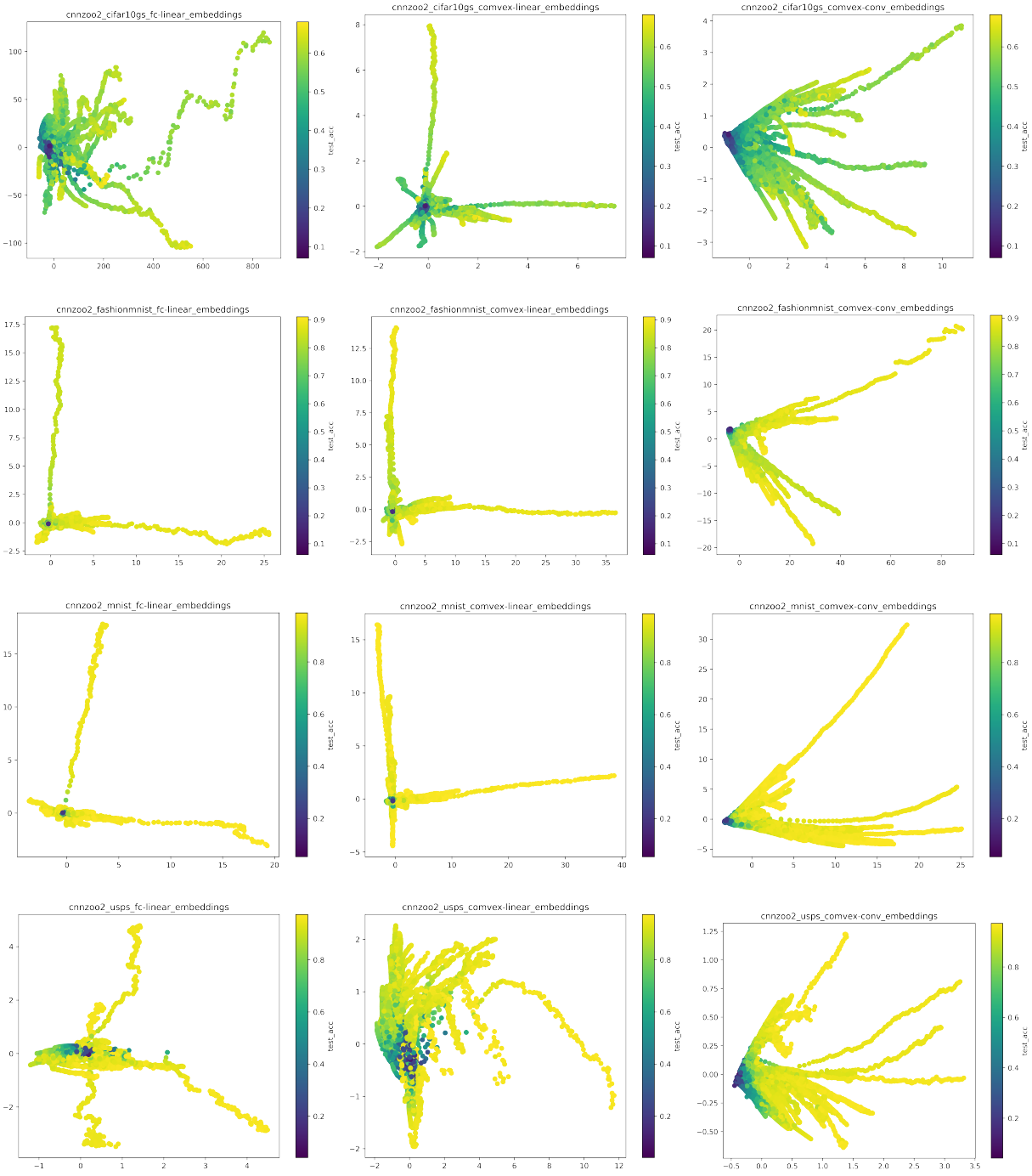

Figure 5. This figure is the same as Figure 5, except that these base models are from CNN Zoo 2.

2, which reflect a more realistic setting in hyperparameter tuning. The vanilla fully-connected networks in Unterthiner et al. (2020) could no longer do well because of the large model size, which led to overfitting. The test R^2 scores on both the fully-connected networks and $\text{FC}_{\text{Linear}}$ are substantially lower than their scores on CNN Zoo 1. This potentially means that they failed to learn the learning dynamics during training, as these networks are "consecutive" in epochs (or time). What's even worse is that the overfitting is extremely hard to mitigate because more than 90% of the weights reside in the first layer, and such overfitting can hardly be addressed by common regularization techniques even if heavy dropout and weight decay are used. However, with $\text{ComVeX}_{\text{Linear}}$, it could still obtain much higher test R^2 scores with reasonable model size, and $\text{ComVeX}_{\text{Conv}}$ achieves negligible performance loss with far less parameters. We also illustrate this in Figure 3.

4.3. Embedding analysis

We then proceeded to analyze both the raw features and the embeddings generated by our linear encoders by projecting them into 2-d plane using PCA. Specifically, the raw features refers to the flattened and concatenated weight matrices, which is essentially a long vector whose length is equal to the total number of parameters in a particular base CNN. In Figure 6, networks with similar performance are aligned on some line, indicating they are similar in some dimension but not others. This observation applies to both CNN Zoo 1 and CNN Zoo 2. This prevents us from using vector distance to find similar networks, because it requires all elements in those vectors to be close. In the case of 2-d visualization, we would like networks with similar performance to form clusters.

We then tested non-distributive embeddings. In the first columns of Figure 4 and Figure 5, non-distributive embeddings failed to separate networks (trained on CIFAR-10, MNIST, and Fashion-MNIST) with low accuracy and those with high accuracy. However, it did a better job on SVHN. Our hypothesis is that non-distributive embeddings might still have distinguishing power in higher dimensions. For distributive embeddings produced by $\text{ComVeX}_{\text{Linear}}$ and $\text{ComVeX}_{\text{Conv}}$, they have much better clustering properties, as the dark and light points are well separated. Looking closely, $\text{ComVeX}_{\text{Conv}}$ produce a denser grouping than $\text{ComVeX}_{\text{Linear}}$. It is also worth noting that, while these linear encoders are still not clustering networks with high accuracy well, they do center most networks with low accuracy, as shown in the dark area in Figure 4 and Figure 5.

5. Conclusions

In this work, we demonstrated that it is possible to use linear transformation to project all weights and biases in a neural network into a low-dimensional space without losing their performance information. We first observed that raw neural networks are not well aligned in 2-d. We then verified that a vanilla fully-connected network struggled to produce embeddings of weight matrices with distinguishing properties. However, the proposed distributive encoding using linear layer and convolutional layer with identity activation preserved the network quality in a better way: they tend align networks with similar performance more consistently. Besides, the proposed networks can also outperform the vanilla fully-connected networks with far less parameters. However, one limitation of the proposed method lies in the task itself. Unlike information retrieval tasks, where we usually have queries and document pairs. Such structure does not naturally exist in our task. The other potential limitation is the lack of separating property in embeddings. We hypothesize that using objectives with contrastive information could result in better embedding learning. We hope our work can inspire more research into the neural network weight representations.

References

Carlucci, F. M., Russo, P., Tommasi, T., and Caputo, B. Hallucinating agnostic images to generalize across domains. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3227–3234. IEEE, 2019.

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. High-performance neural networks for visual object classification. *ArXiv preprint*, abs/1102.0183, 2011. URL https://arxiv.org/abs/1102.0183.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Eilertsen, G., Jönsson, D., Ropinski, T., Unger, J., and Ynnerman, A. Classifying the classifier: dissecting the weight space of neural networks. *ArXiv preprint*, abs/2002.05688, 2020. URL https://arxiv.org/abs/2002.05688.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *science*, 313 (5786):504–507, 2006.

Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S. Predicting the generalization gap in deep networks with margin distributions. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HJlQfnCqKX.

Kim, Y. Convolutional neural networks for sentence classification proceedings of the 2014 conference on empirical methods in natural language processing, emnlp 2014, october 25-29, 2014, doha, qatar, a meeting of sigdat, a special interest group of the acl. *Association for Computational Linguistics, Doha, Qatar*, 2014.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Le, Q. V. and Mikolov, T. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pp. 1188–1196. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/le14.html.

Martin, C. H. and Mahoney, M. W. Heavy-tailed universality predicts trends in test accuracies for very large pre-trained deep neural networks. In Demeniconi, C. and Chawla, N. V. (eds.), *Proceedings of the 2020 SIAM International Conference on Data Mining, SDM 2020, Cincinnati, Ohio, USA, May 7-9, 2020*, pp. 505–513. SIAM, 2020. doi: 10.1137/1.9781611976236.57. URL https://doi.org/10.1137/1.9781611976236.57.

Martin, C. H. and Mahoney, M. W. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *Journal of Machine Learning Research*, 22(165):1–73, 2021.

Martin, C. H., Peng, T. S., and Mahoney, M. W. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*, 12(1):1–13, 2021.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *ArXiv preprint*, abs/1301.3781, 2013a. URL https://arxiv.org/abs/1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Ghahramani, Z., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 3111–3119, 2013b. URL https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html.

Pennington, J., Socher, R., and Manning, C. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2227–2237, New Orleans, Louisiana, 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202.

Reimers, N. and Gurevych, I. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410.

Reimers, N. and Gurevych, I. Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4512–4525, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.365. URL https://aclanthology.org/2020.emnlp-main.365.

Reimers, N. and Gurevych, I. The curse of dense low-dimensional information retrieval for large index sizes. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 605–611, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-short.77. URL https://aclanthology.org/2021.acl-short.77.

Thakur, N., Reimers, N., Daxenberger, J., and Gurevych, I. Augmented SBERT: Data augmentation method for improving bi-encoders for pairwise sentence scoring

tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 296–310, Online, 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.28. URL https://aclanthology.org/2021.naacl-main.28.

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., and Gurevych, I. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *ArXiv preprint*, abs/2104.08663, 2021b. URL https://arxiv.org/abs/2104.08663.

Unterthiner, T., Keysers, D., Gelly, S., Bousquet, O., and Tolstikhin, I. Predicting neural network accuracy from weights. *ArXiv preprint*, abs/2002.11448, 2020. URL https://arxiv.org/abs/2002.11448.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Wang, K., Reimers, N., and Gurevych, I. Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning. *ArXiv preprint*, abs/2104.06979, 2021. URL https://arxiv.org/abs/2104.06979.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yak, S., Gonzalvo, J., and Mazzawi, H. Towards task and architecture-independent generalization gap predictors. *ArXiv preprint*, abs/1906.01550, 2019. URL https://arxiv.org/abs/1906.01550.

A. Hyperparameter tuning

For CNN Zoo 1, we used the hyperparameter sampling listed below.

- Number of layers: uniformly from range [2, 8].[4]
- Hidden size: uniformly from range [256, 512).
- Dropout rate: uniformly from range [0.0, 0.2].
- Weight decay: log-uniformly from range [1e-8, 1e-3].
- Optimizers: uniformly from [adam, adamw, adamax, nadam, radam].
- Batch size: uniformly from [64, 128, 256, 512].
- Initializer: uniformly from [xavier, he, orthogonal, original] (all from the uniform distribution (default in PyTorch).

For CNN Zoo 2, we used the following hyperparameter sampling.

- Number of layers: uniformly from range [2, 8].
- Hidden size: uniformly from range [64, 512).
- Dropout rate: uniformly from range [0.0, 0.7].
- Weight decay: log-uniformly from range [1e-5, 1e-2].
- Optimizers: uniformly from [adam, adamw, adamax, nadam, radam].
- Batch size: uniformly from [64, 128, 256, 512].
- Initializer: uniformly from [xavier, he, orthogonal, original] (all from the uniform distribution (default in PyTorch).

Note that we significantly increase both dropout and weight decay to mitigated overfitting in the regular FC network.

B. CNN Zoo 1 Performance

Architecture	**Parameters**	**Test loss**	**Test R^2**
FC network	3538509	0.000171	0.9866
$\text{FC}_{\text{Linear}}$	1774519	0.000168	0.9867
$\text{ComVeX}_{\text{Linear}}$	1623967	0.000169	0.9869
$\text{ComVeX}_{\text{Conv}}$	458153	0.000167	0.9868

Table 6. Predictors' performance on CNN Zoo 1 CIFAR-10.

Architecture	**Parameters**	**Test loss**	**Test R^2**
FC network	3848853	0.000558	0.993
$\text{FC}_{\text{Linear}}$	2017377	0.000564	0.9929
$\text{ComVeX}_{\text{Linear}}$	842399	0.000505	0.9935
$\text{ComVeX}_{\text{Conv}}$	966321	0.000513	0.9936

Table 7. Predictors' performance on CNN Zoo 1 Fashion-MNIST.

Architecture	**Parameters**	**Test loss**	**Test R^2**
FC network	2817777	0.000658	0.9944
$\text{FC}_{\text{Linear}}$	1956661	0.000586	0.995
$\text{ComVeX}_{\text{Linear}}$	1397399	0.000521	0.9955
$\text{ComVeX}_{\text{Conv}}$	1339361	0.000533	0.9955

Table 8. Predictors' performance on CNN Zoo 1 MNIST.

Architecture	**Parameters**	**Test loss**	**Test R^2**
FC network	3675685	0.000265	0.9834
$\text{FC}_{\text{Linear}}$	2054869	0.000269	0.9832
$\text{ComVeX}_{\text{Linear}}$	1192327	0.000206	0.9871
$\text{ComVeX}_{\text{Conv}}$	950801	0.000203	0.9875

Table 9. Predictors' performance on CNN Zoo 1 SVHN.

C. Raw weights visualization

See Figure 6 on next page.

D. Simialrity matrices

[4]This does not include the first layer for the proposed architectures, and we add one to the number of layers for the regular fully-connected networks during initialization for fair comparison.

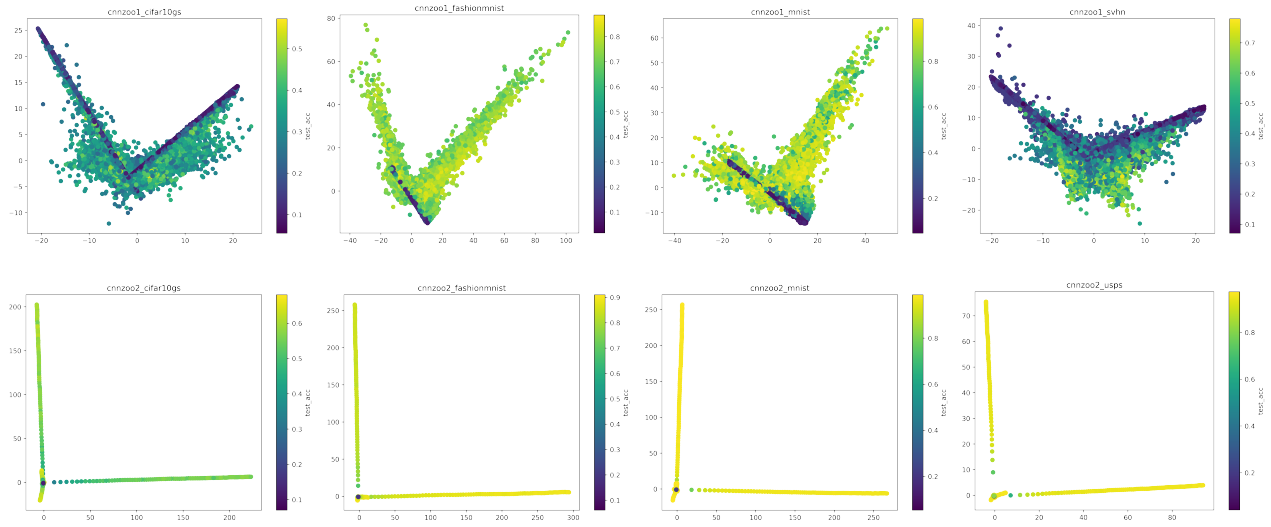

Figure 6. A 2-d visualization of the raw neural network weights in CNN Zoo 1 (top) and CNN Zoo 2 (bottom). Networks with similar performance align inconsistently on lines instead of together as a cluster.

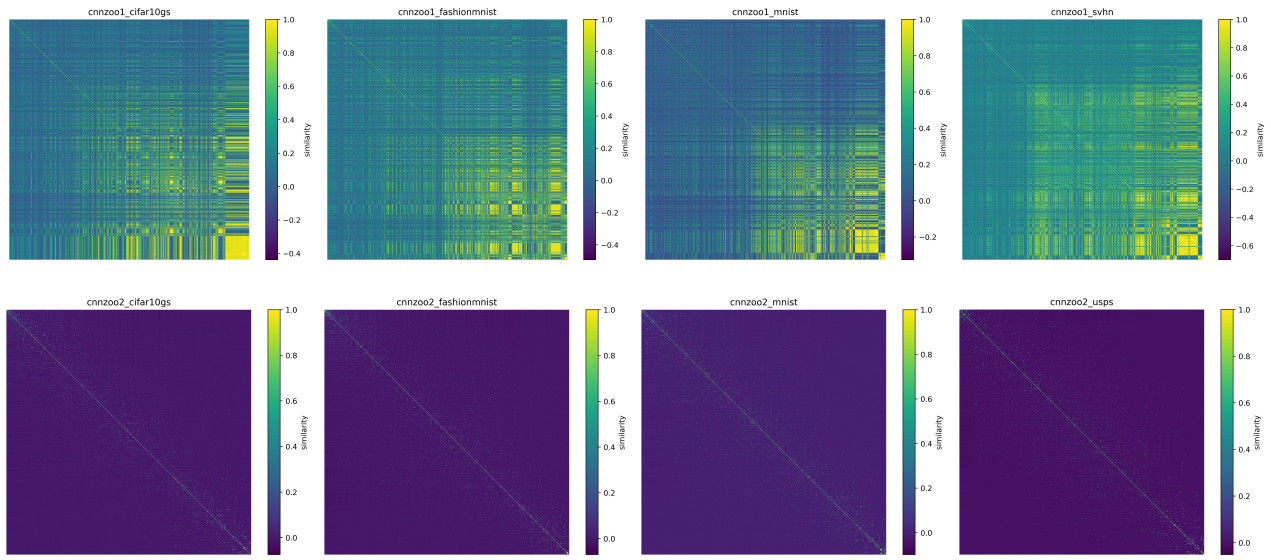

Figure 7. Similarity matrices of raw weights. On both axes, models are sort by their accuracy in descending order (from left to right for x, orfrom top to bottom for y). Lighter means higher similarity.

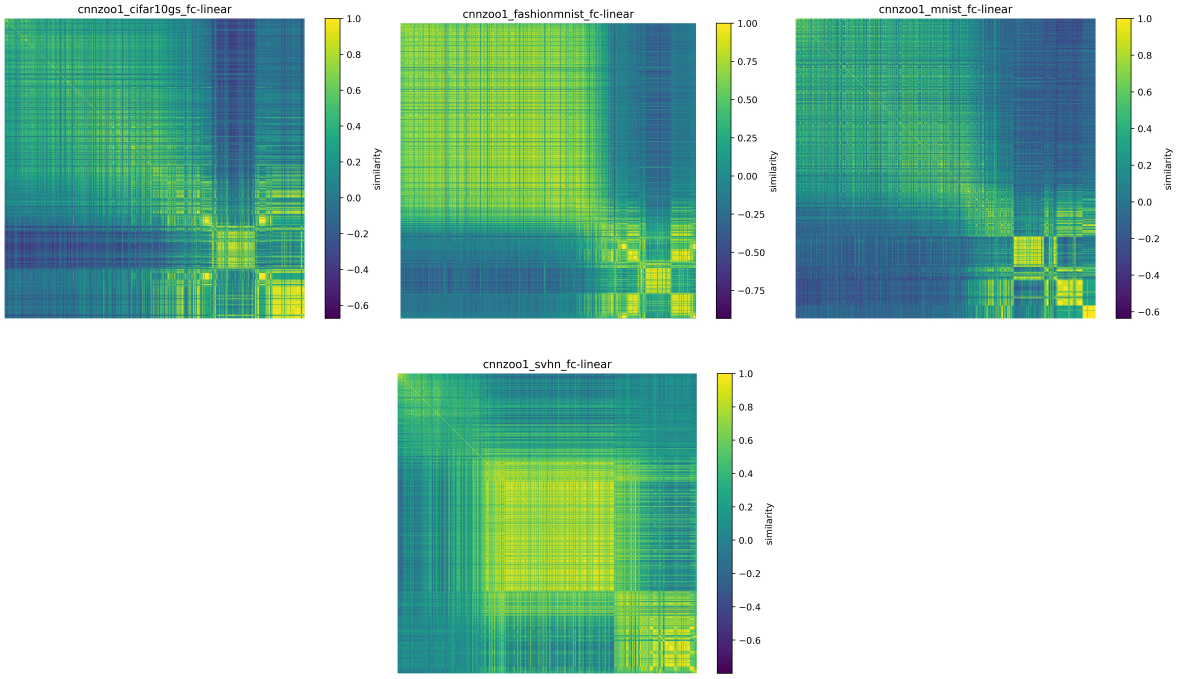

Figure 8. CNN Zoo 1 similarity matrices

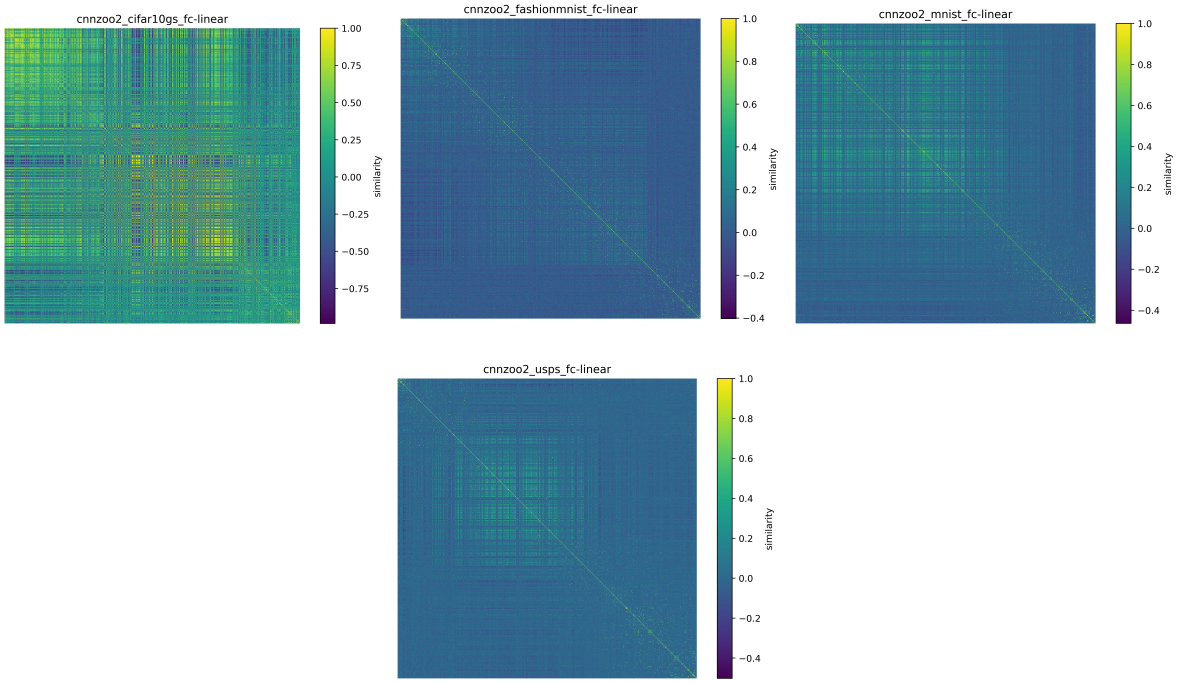

Figure 9. CNN Zoo 2 similarity matrices

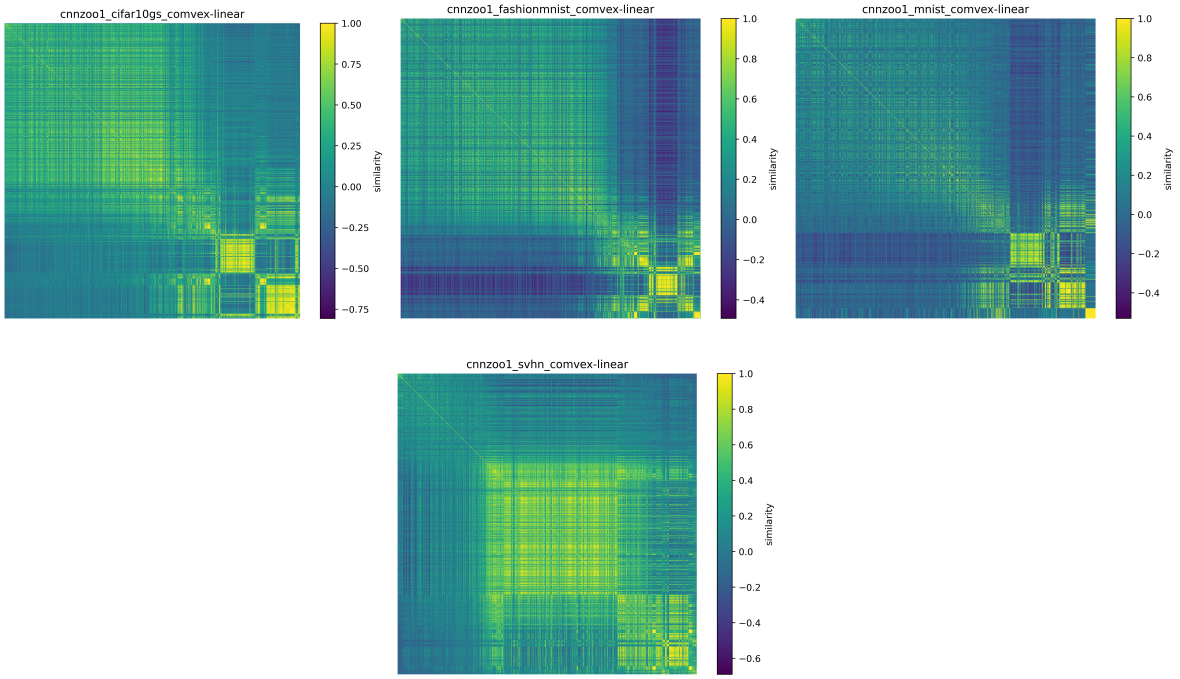

Figure 10. CNN Zoo 1 similarity matrices

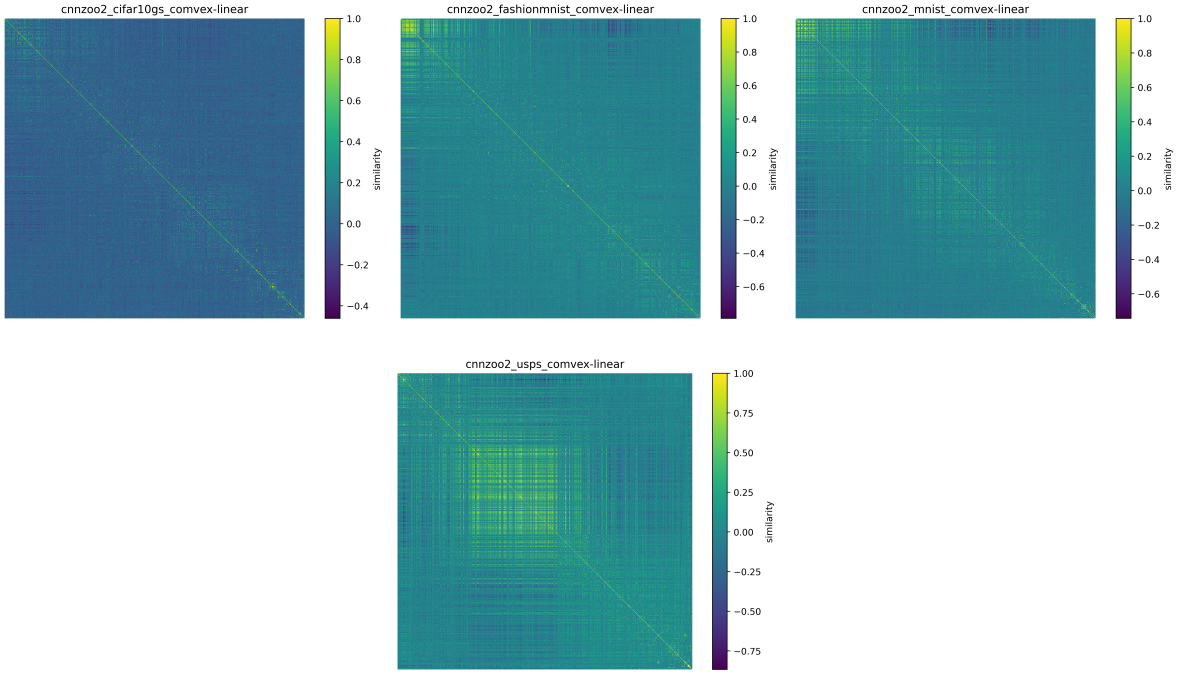

Figure 11. CNN Zoo 2 similarity matrices

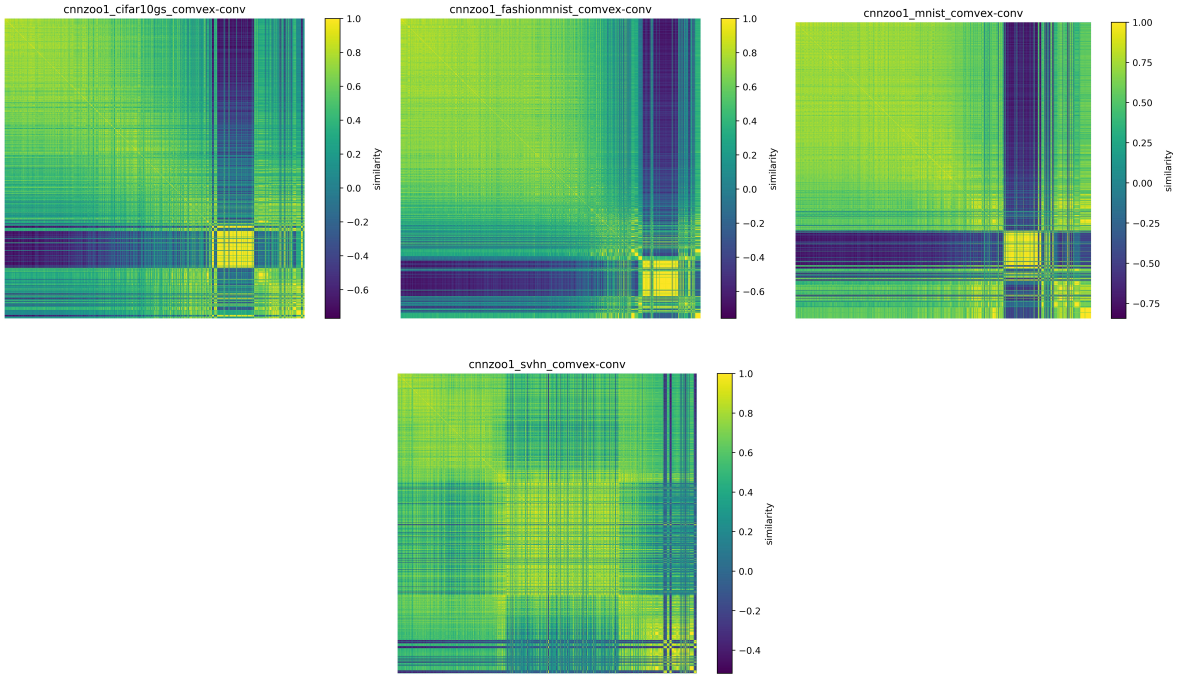

Figure 12. CNN Zoo 1 similarity matrices

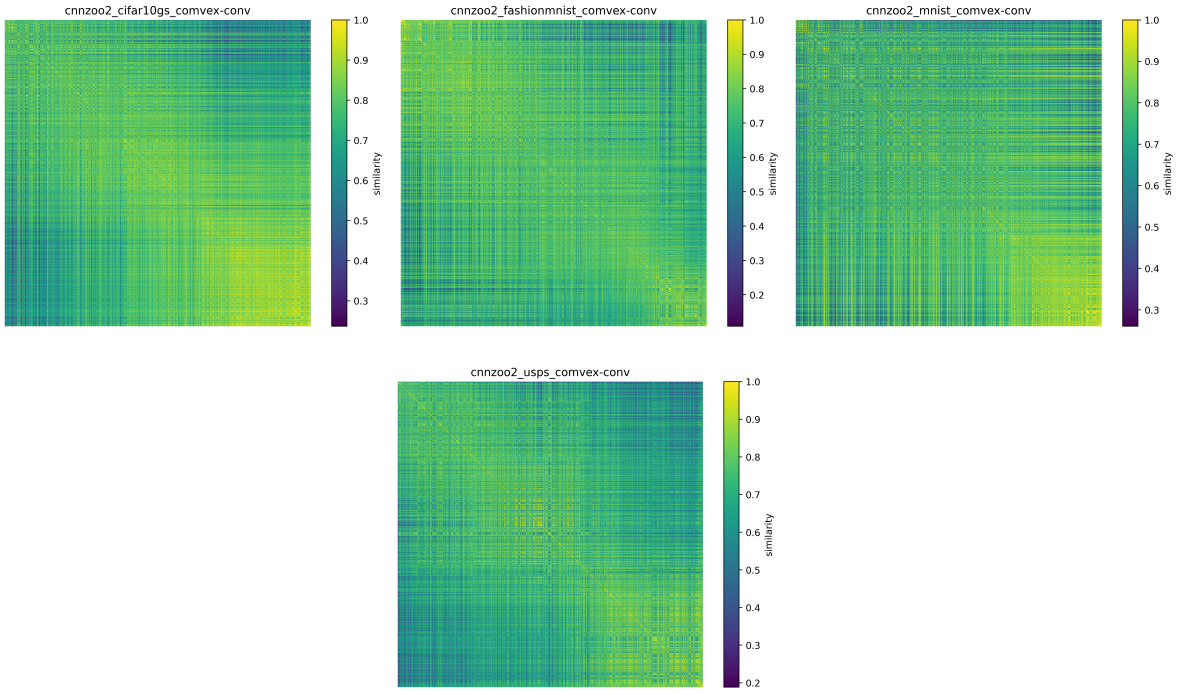

Figure 13. CNN Zoo 2 similarity matrices