Permutation-Invariant Representation of Neural Networks with Neuron Embeddings

Ryan Zhou[0000−0002−2192−2667], Christian Muise[0000−0002−2728−6585], and Ting Hu[0000−0001−6382−0602]

Queen's University, Kingston, Ontario, Canada K7L 2N8
{20rz11,christian.muise,ting.hu}@queensu.ca

Abstract. Neural networks are traditionally represented in terms of their weights. A key property of this representation is that there are multiple representations of a network which can be obtained by permuting the order of the neurons. These representations are generally not compatible between networks, making recombination a challenge for two arbitrary neural networks - an issue known as the "permutation problem" in neuroevolution. This paper proposes an indirect encoding in which a neural network is represented in terms of interactions between neurons rather than explicit weights, and which works for both fully connected and convolutional networks. In addition to reducing the number of free parameters, this encoding is agnostic to the ordering of neurons, bypassing a key problem for direct weight-based representation. This allows us to transplant individual neurons and layers into another network without accounting for the specific ordering of neurons. We show through experiments on the MNIST and CIFAR-10 datasets that this method is capable of representing networks which achieve comparable performance to direct weight representation, and that combining networks this way preserves a larger degree of performance than through direct weight transfer.

Keywords: Neuroevolution · Indirect Encoding · Neural Networks · Convolutional Neural Networks · Crossover · Permutation Invariance

1 Introduction

One of the main challenges in neuroevolution is developing an effective crossover operation. This is in large part due to what is known as the competing conventions or permutation problem [37]: given any particular neural network, an equivalent network can be obtained by permuting the order of the neurons along with the corresponding weights. In other words, functionally identical networks - that is, networks with the same computation graph - can have different representations simply because the units comprising them are defined in a different order. This implies two things: that the representation contains unnecessary information about the ordering of neurons, and that the internal representations for two networks are overwhelmingly likely to be incompatible. Crossover between incompatible representations will generally be destructive to learned relationships.

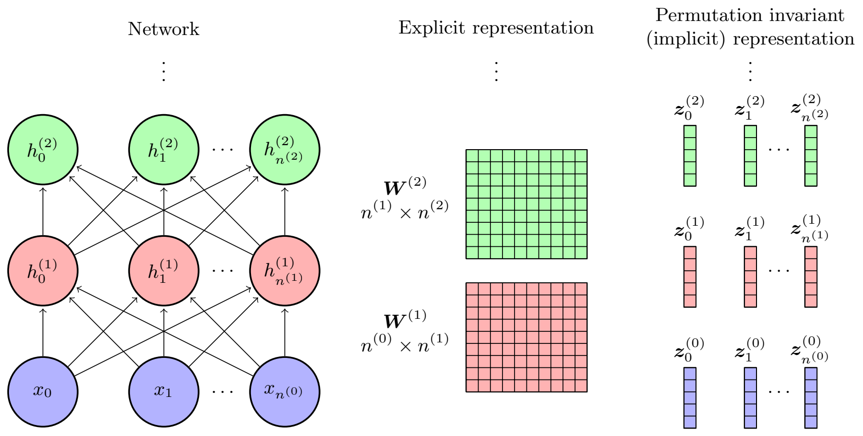

Network
Explicit representation
Permutation invariant (implicit) representation
$h_0^{(2)}$ $h_1^{(2)}$ $h_{n^{(2)}}^{(2)}$
$h_0^{(1)}$ $h_1^{(1)}$ $h_{n^{(1)}}^{(1)}$
x_0 x_1 $x_{n^{(0)}}$
$\boldsymbol{W}^{(2)}$ $n^{(1)} \times n^{(2)}$
$\boldsymbol{W}^{(1)}$ $n^{(0)} \times n^{(1)}$
$\boldsymbol{z}_0^{(2)}$ $\boldsymbol{z}_1^{(2)}$ $\boldsymbol{z}_{n^{(2)}}^{(2)}$
$\boldsymbol{z}_0^{(1)}$ $\boldsymbol{z}_1^{(1)}$ $\boldsymbol{z}_{n^{(1)}}^{(1)}$
$\boldsymbol{z}_0^{(0)}$ $\boldsymbol{z}_1^{(0)}$ $\boldsymbol{z}_{n^{(0)}}^{(0)}$

Fig. 1. A neural network (left) with $n^{(i)}$ hidden units per layer is traditionally represented by explicitly specifying the weights of the connections, usually as a matrix or tensor $W^{(i)}$ of dimension $n^{(i-1)} \times n^{(i)}$ (middle). We propose instead to view the network as sets of neurons (right), with a neuron j in layer i represented as a vector $\boldsymbol{z}_j^{(i)}$. Weights are generated implicitly by calculating alignment coefficients between neurons. This representation is parameter efficient and there is no explicit ordering within each layer, rendering it permutation invariant.

Representing the network in a way that is agnostic to the neuron order, i.e, is permutation invariant with respect to the neurons, can reduce this problem of incompatible representations. In this paper, we propose neuron embeddings, an indirect encoding method for representing a neural network which views the network as unordered sets of neurons rather than ordered lists of weights (Figure 1). This builds permutation invariance directly into the representation. The key to our approach is that the weights are not fixed, but are generated dynamically by individual neurons based on the other neurons present in the network. This allows neurons not only to be reordered but also moved between models in crossover operations. A neuron that is moved will adapt its weights to the new population it finds itself in. In addition, because direct weight representation implicitly contains information about neuron order, a permutation invariant indirect representation can be made smaller and more parameter efficient.

Our experiments with the proposed representation show that crossover using neuron embeddings significantly improves the performance of the resulting network compared to the same operation done using direct representation. We also propose and demonstrate a method by which this can be extended to convolutional neural networks, allowing the network to be permutation invariant with respect to the ordering of convolutional filters.

In the following section, Section 2, we will present some context and motivation and contrast our approach with existing ones. In Section 3, we introduce the proposed concept of neuron embeddings, self-contained representations of individual neurons, and the corresponding representation of the network as unordered sets of these embeddings. In Section 4 we present our experiments and results. We find that neuron embedding representation achieves comparable network performance to direct representation in fewer parameters, and that crossover with embeddings preserves a larger degree of functionality than with direct representation. We provide some concluding remarks in Section 5.

2 Related Work

Neuroevolution and Indirect Encoding Neuroevolution is the application of evolutionary methods to neural networks. A key component of many evolutionary algorithms is recombination, but applying it to neural networks has been challenging because of the permutation problem. Addressing it has been a central focus of neuroevolution work [35]. Previous methods have approached this by looking for analogous structures in the network to limit the impact of permutation [37], or by sorting the neurons based on their connections [5]. However, these methods do not scale to the sizes of networks in modern deep learning. We propose that a more efficient solution is to build permutation invariance into the representation, thereby avoiding the problem.

A second challenge for large-scale neuroevolution is the large number of weights in a neural network, leading to impractically large genomes if direct encoding is used - that is, if each weight is coded for individually in the genome. Indirect encoding is an alternative approach which represents the network using a small number of parameters and uses rules to generate the weights [31,38]. This concept has proved successful at allowing larger networks to be trained with evolution [14,21,36]. Modern neural network architectures can also be viewed in this light; notably, convolution [11,23] and attention [42] generate large numbers of effective weights from small numbers of explicit parameters. We use indirect encoding in our method, generating weights based on a small number of vector representations.

Permutation Invariance in Neural Networks Permutation invariance refers to the property that a function remains unchanged even when some aspect of it is permuted. Previous work has been done on introducing various forms of permutation invariance (PI) to neural networks, primarily focused on allowing neural networks to exhibit permutation invariance over the inputs. [9] and [45] introduce methods which use pooling operations to perform permutation-invariant operations for set inputs. [2] introduce permutation invariance into the features themselves by recombining pairs of features. Set Transformer [25] builds upon these by using self-attention to capture higher order interactions. Sensory neurons [39] use similar modular units to produce a PI policy. These methods address permutation invariance in the inputs rather than the network representation itself. We draw on these ideas in order to do the opposite - to represent an arbitrary neural network (which may or may not be permutation invariant with respect to the inputs) in a manner that is PI to shuffling of the neurons.

Neuron-based Representation Neuron-based representations have also previously been employed in the literature, often in the context of evolving individual neurons [12, 26, 32] or compact representations of networks [8, 10, 17, 30]. Our work makes use of neuron-based representation to achieve permutation invariance, but is aimed at bridging the gap between these two applications. Our aim is not to train individual neurons in a population-based manner but instead to represent entire pretrained networks and discover structures which can be transferred between networks. Compared to previous work on full network representations, our approach not only represents single networks but also aims to improve cross-model compatibility between multiple networks by reducing networks down to transferable units. As such, the approach we propose is designed to make the individual neuron representations as self-contained as possible, without any interaction with network-specific structures such as hypernetworks.

Attention Attention [42] is a highly successful mechanism which underpins many modern deep neural networks. The key strength of attention is its ability to generate a large number of attention scores using only a small number of parameters, and to do so dynamically, which can be seen as form of indirect encoding [40]. In addition, it does so in a permutation-invariant way, by only depending on the features of the two endpoints. Because of this key property, we base our model on the attention kernel with appropriate modifications. Attention as used in models such as Transformers [42] operates between the tokens given as inputs to the network; our method differs in that we use as endpoints the neurons themselves, generating a set of weights which are agnostic to the input.

Model Compression and Tensor Decomposition Neural network compression refers to the general goal of reducing the size of a model in order to reduce the amount of storage or computation required without significantly impacting performance. One method of achieving this is through tensor decomposition. Because weights in neural networks may be represented with tensors, it is possible to express the full tensor as a product or sum of lower-rank or smaller tensors. Several methods for providing exact or approximate decompositions exist [1, 20]; commonly used methods include CP [18], Tucker [41] and tensor train [27] decomposition. The method we describe in this paper can be viewed as a low-rank decomposition of the weight tensors, similar to the methods described in [16] and [44]. That is, for a weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ with rank r, we approximate $\boldsymbol{W}$ with the product $\boldsymbol{W} = \boldsymbol{X}\boldsymbol{Y}$ with $\boldsymbol{X} \in \mathbb{R}^{m \times r}$ and $\boldsymbol{Y} \in \mathbb{R}^{r \times n}$. This reduces the number of parameters from mn to $r(m+n)$ [6]. There are two major points of contrast between our method and other tensor decompositions: first, our primary goal is to generate self-contained representations of neurons and so the embedding for each neuron is used twice - once to determine the incoming weights, and once to determine the outgoing weights. For this reason, our method imposes a symmetry constraint such that the two embeddings are identical in order to produce a single representation of the "role" of a neuron. Second, our method is only a decomposition in the case of the linear dot-product kernel; other attention kernels allow it to represent a broader class of functions.

3 Method

We will first describe how our method works for a simple feedforward network. Then, we will describe how convolutional neural networks can be represented as well. In short, we replace all weights in the network with a set of vector representations of the neurons present in the network. Weights are then generated in an attention-like way, with some modifications.

It is important that each neuron's representation contains all the information necessary to perform its function so that it can be moved between networks - thus, there is no equivalent to the query, key and value networks of attention which would need to be external to the neuron. This ensures a neuron's representation is fully self-contained, allowing it to be transplanted into a second neural network and generate new weights without requiring information from the original neural network.

Neuron Embedding The core idea of our method is to introduce a learnable vector embedding $\mathbf{z}$ for each neuron (Figure 1). This is simply a d-dimensional vector which represents the role of the neuron and can be trained via gradient descent. This is used to generate weight scores between it and all neurons in the previous layer using a kernel $K(\cdot,\cdot)$. We calculate the alignment score α_{ij} in a manner similar to attention by using a dot product kernel, and assign this score as the weight. That is, we take the dot product between the embedding $z_i^{(l)}$ of neuron i in layer l and the embedding $z_j^{(l+1)}$ of neuron j in layer $l+1$ [42] with an optional nonlinearity σ:

$$\alpha_{ij} = K(\boldsymbol{z}_i^{(l)}, \boldsymbol{z}_j^{(l+1)}) = \sigma(\boldsymbol{z}_i^{(l)} \boldsymbol{z}_j^{(l+1)^\top}) \tag{1}$$

This is done efficiently as a matrix operation by packing the embeddings for both layers into the matrices $\boldsymbol{Z}^{(l)} \in \mathbb{R}^{n_l \times d}$ and $\boldsymbol{Z}^{(l+1)} \in \mathbb{R}^{n_{l+1} \times d}$, where n_i is the number of hidden units in the layer i. The activation vector $\boldsymbol{h}^{(l)}$ of layer l takes the place of the value function, giving us:

$$\mathit{Attention}(\boldsymbol{Z}^{(l)}, \boldsymbol{Z}^{(l+1)}, \boldsymbol{h}^{(l)}) = \sigma(\boldsymbol{Z}^{(l)} \boldsymbol{Z}^{(l+1)^\top}) \boldsymbol{h}^{(l)} \tag{2}$$

This can be implemented simply by assigning the matrix of attention scores to be the weight matrix $\boldsymbol{W}^{(l)}$. Note that unlike the Transformer formulation of attention, we use the unscaled dot product here. Scaling the dot product by $\frac{1}{\sqrt{d}}$ corrects the variance of the product to be 1 when the input embeddings have variance 1; however, we find in practice it is more effective to scale the initialization of the embeddings. Each component of the embedding is initialized to be normally distributed with standard deviation $\frac{1}{\sqrt{d}}$ or equivalently variance $\frac{1}{d}$, where d is the dimensionality of the embedding:

$$z_i \sim N(0, \frac{1}{d}) \tag{3}$$

This ensures the magnitude of the embedding vectors has a mean of 1, removing the need for scaling.

Bias In addition to the embedding, each neuron contains a learnable bias b in order to match the overall function of a feedforward network. This bias has the same role as the bias in a feedforward layer, and is added after the weights are applied. Since each bias is specific to a single neuron, it can be considered part of the self-contained representation and moved to a different network.

Input Encoding To generate the weights for the first layer, it is necessary to provide an embedding for each input to the network, which can be learned from the data [7]. A second possibility is to provide predefined embeddings; for example, through positional encodings [42]. We tested sinusoidal positional embeddings for one and two dimensions [42, 43] as well as localized wavelets, but found that in practice, these fixed embeddings performed poorly. We allow a model to learn the input embeddings from the dataset, which can then be shared with subsequent models trained on the same dataset. This is important for cross-model transfer, as it provides the two models a common basis from which to work.

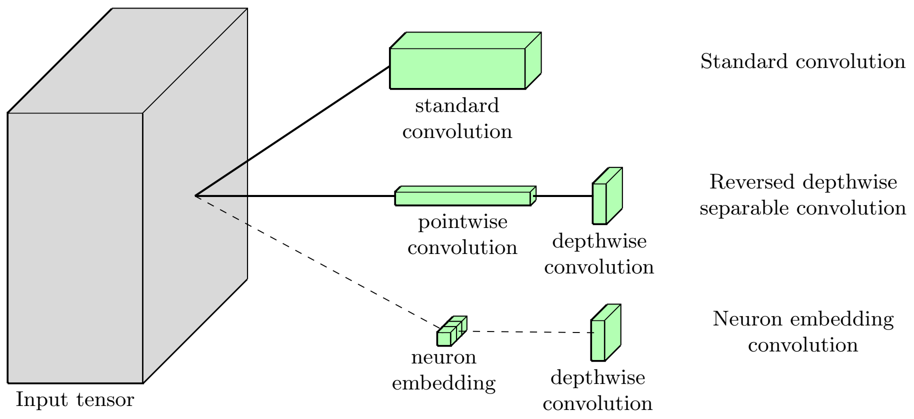


Fig. 2. Representation of a convolutional neuron. The standard representation explicitly specifies all the weights in the kernel. Depthwise separable convolutions provide an approximate replacement by splitting the kernel into a pointwise convolution, which mixes information across channels, and a depthwise convolution which applies one spatial kernel per channel. We replace the pointwise convolution with an implicit representation using neuron embeddings but keep the depthwise convolution, rendering the network permutation invariant to the ordering of filters but preserving spatial structure. Each neuron embedding and depthwise convolution pair represents a single output filter.

Encoding Convolutional Networks Convolutional neural networks present a unique challenge. For a $k \times k$ filter with $n^{(i)}$ input channels, we have $k^2 \cdot n^{(i)}$ incoming weights. However, we only have $n^{(i)}$ embeddings in the layer below. In addition, we would like to do this in a way that can be encapsulated as a single neuron, allowing it to operate in a self-contained manner.

Our solution (Figure 2) is to employ reversed order depthwise separable convolutions [3]. The standard order is to apply the $n^{(i)}$ depthwise convolutions first, followed by the pointwise convolution to expand the number of channels from $n^{(i)}$ to $n^{(i+1)}$. However, in order to produce self-contained representations, we would like to treat each pointwise-depthwise pair as a single neuron; for this, we need $n^{(i+1)}$ depthwise kernels. Thus, we reverse the order of operations, performing the pointwise convolution first to produce $n^{(i+1)}$ different channels in the output, and then assign each channel its own depthwise convolution. Since the pointwise convolution can be seen as a feedforward network along the channel dimension, we can represent this using neuron embeddings, with one embedding per output channel. Performing the steps in reverse order is also known as a blueprint separable convolution and exhibits improved training properties [13].

4 Experiments

We now present a series of experiments designed to test the ability of our method to represent equivalent networks to direct weight encoding, and to evaluate its ability to preserve performance under crossover. We use the MNIST [24] and CIFAR-10 [22] datasets to evaluate the models. All models were implemented in Python using the PyTorch library [29], and the code can be found on GitHub at `https://github.com/ryanz8/neuron-embedding`. Experiments were performed on a single computer with an NVIDIA RTX3090 GPU.

Hyperparameter Optimization Hyperparameters for the direct weight representation models were manually tuned following empirical guidelines [28, 33] with a small random search over learning rate and weight decay. As the focus of this paper is on the relative efficacy of the representation methods rather than overall performance, we did not perform heavy hyperparameter optimization. Rather, we attempt to showcase the models under similar starting conditions. As such, the hyperparameters of the neuron embedding representations were matched to those of the direct representations. This should favor the direct representation slightly; however, there is the possibility that the results will differ or the performance gap will be greater under different hyperparameters.

4.1 Training from Random Initialization

Our first experiment tests the ability of our method to achieve comparable performance to weight encoding when trained from random initialization. The intent is to test whether neuron embeddings can be trained the same way as direct weight representations without any special tuning. We compared two types of architectures: fully connected and convolutional, each using direct weight representation, against equivalents using neuron embedding representation. We chose training settings which yielded high performance after a short amount of training for the direct weight representations, and used the same settings without modification for the neuron embedding representations.

Dataset	Model	Parameters	Layers	Acc. (%)	CE Loss
MNIST	FC (direct)	318010	2 fc	98.05	0.0672
MNIST	**FC (emb.)**	**76416**	**2 fc**	**97.43**	**0.0999**
MNIST	FC (direct)	417640	5 fc	98.14	0.0710
MNIST	**FC (emb.)**	**97536**	**5 fc**	**97.44**	**0.1077**
MNIST	Conv. (direct)	160070	3 conv 2 fc	99.38	0.0294
MNIST	Conv. (sep.)	84750	3 conv 2 fc	99.27	0.03732
MNIST	**Conv. (emb.)**	**51598**	**3 conv 2 fc**	**99.00**	**0.0412**
CIFAR-10	ResNet9 (direct)	2438794	8 conv 1 fc	89.40	0.3962
CIFAR-10	ResNet9 (sep.)	287818	8 conv 1 fc	88.21	0.4312
CIFAR-10	**ResNet9 (emb.)**	**98298**	**8 conv 1 fc**	**86.90**	**0.4469**

Table 1. Performance when trained from random initialization for fully connected (FC) models and convolutional (conv) models. "Direct" models use direct (explicit) weight representation. "Sep." models use reverse order depthwise separable convolutions (blueprint separable convolutions). "Emb." models (ours, bolded) use neuron embedding representation.

All models unless otherwise specified were trained with cross-entropy loss [19], using the Adam optimizer on MNIST and SGD with momentum on CIFAR-10. Network widths are noted in brackets, with convolutional layers denoted with a superscript c. We test a 2-layer (400,10) and 5-layer (400,400,400,400,10) feedforward network and a 5-layer convolutional network (16^c,40^c,1000,100,10) on MNIST, and a 9-layer ResNet (64^c,128^c,128^c,128^c,256^c,256^c,256^c,256^c,10) [15] on CIFAR-10 designed based on the results of the DAWNBench benchmark [4, 28]. For models using neuron embedding, we set the nonlinearity σ to be the identity for faster training. All models use ReLU activation for all layers except the output. Comparison was done using the best model found after 2000 steps of training as determined by cross-validation on a holdout set of 10000 data points. With Adam, we use a one-cycle learning rate schedule [34] and cosine annealing, with a learning rate of 0.01 and batch size of 1000 which has been shown to work well in combination with this schedule [33]. For stochastic gradient descent, we use linear annealing with a maximum learning rate of 2×10^{-4} obtained by hyperparameter search and a batch size of 512. The dimensionality of the neuron embeddings is set to 64 for fully connected models and 48 for convolutional models.

The results in Table 1 show that representation using neuron embeddings is able to achieve comparable performance to direct weight representation, when using standard training settings without modification. The slight difference in performance we attribute to the use of training settings optimized for direct weight representation; as we will show next, it is not due to the smaller number of parameters leading to a gap in expressiveness for this problem. We note that training time is also not impacted, and in some cases is actually reduced which we attribute to the smaller number of parameters.

Model	Free Parameters	Accuracy (%)	MSE
Reference	318010	97.48	-
Neuron embedding (64 dims)	76416	97.48	0.00036
Neuron embedding (32 dims)	38208	97.15	0.00053
Neuron embedding (16 dims)	19104	75.08	0.00095
Neuron embedding (8 dims)	9552	65.61	0.00177
Neuron embedding (4 dims)	4776	20.72	0.00414

Table 2. Results for training to a 2-layer reference network. An embedding dimension of 64 is sufficient to match the performance of this network within margin of error, while decreasing the embedding dimension degrades the performance. MSE refers to the mean squared deviation of the weights in the neuron embedding representation from the weights in the reference network. The mean-squared amplitude of the weights in the reference network is 0.0152.

4.2 Compression Ability

Our next experiment tests the ability of neuron embeddings to exactly reproduce the weights of a reference fully connected network. This tests the expressiveness of the neuron embeddings. We expect that if the network is able to reproduce the weights, then performance should match that of the reference network. We tested different values for d, the embedding dimension to show the effect of embedding expressiveness on the final accuracy.

To force the embeddings to replicate the weights, we train the embeddings by minimizing the mean squared loss over all the generated weights when compared to the reference network. This was chosen as it corresponds to minimizing the quantity

$$\sum_{i=1}^{N} \frac{1}{m_i n_i} \|\mathbf{W}_i - \mathbf{Z}_{i-1}\mathbf{Z}_i^T\|_F^2. \tag{4}$$

That is, it approximates the full-rank decomposition of the weight matrices normalized by the number of elements. Here $\mathbf{W}_i$ is the weight matrix for layer i, m_i and n_i are the dimensions of $\mathbf{W}_i$, $\mathbf{Z}_{i-1}$ and $\mathbf{Z}_i$ are the neuron embeddings for the layers $i-1$ and i, and $\|\cdot\|_F$ is the Frobenius norm. Models were trained using the Adam optimizer with a learning rate of 0.002 for 2000 steps.

Results are shown in Table 2. As can be seen, with sufficient d models are able to almost exactly match the performance of a directly encoded network. Insufficient expressiveness as a result of a too small d harms the performance of the network, but even with only 8 dimensions a significant fraction of the knowledge was still represented (with an accuracy of 65% versus the 10% of random chance). In all cases, the number of parameters of the neuron embedding model was smaller than that of the fully connected reference network, despite being able to match the weights.

4.3 Cross-Model Compatibility

Our next experiment tests whether neuron-based representations enable better compatibility between different models. Our goal is to determine the degree to which the function of a neuron is preserved when moved to a different setting. This evaluates the potential of this representation for crossover operations and cross-model transfer learning.

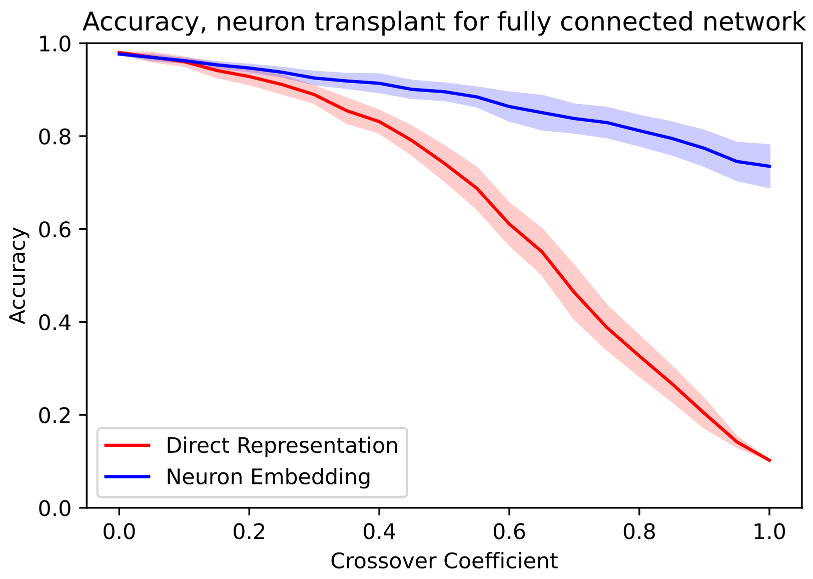
We trained two models from random initialization, producing two different networks to act as a source network and a target network. We then trained two neuron embedding models to replicate the weights of each direct encoding parent. We use the same learned input encodings for both neuron embedding models, done by copying the learned input encodings from the target network to the source network before training. This did not affect the weights themselves and it was possible to replicate both the weights of the source and target network to high accuracy using the same embeddings for the inputs but different neuron embeddings for all subsequent layers.

We performed this process for both fully connected and convolutional models. The fully connected models contained 8 hidden layers with 400 neurons each and a 10 neuron output layer. The convolutional models consisted of three 3x3 reverse-order depthwise separable convolutional layers with 20, 40 and 80 neurons, followed by a 100 neuron fully connected hidden layer and the 10 neuron output layer.

Neuron transplant We tested compatibility for both pairs of models by transferring a variable number of neurons in the first hidden layer from the source network to the target network, which we refer to as a crossover operation. If the internal representations are compatible, we expect models to retain a greater degree of performance under this operation. Here, a crossover coefficient of 0.8 indicates that 80% of the neurons in that layer of the target network have been replaced and 20% of the neurons remain. A coefficient of 1.0 indicates that the entire layer has been replaced with the layer from the source network. Neurons are chosen in random order for this, and we repeat each experiment 10 times and report the mean and 95% confidence interval.

The results in Figure 3 show that transplanting neurons in the hidden layer results in minor loss of performance for both models until roughly 1/3 of the neurons were replaced, after which performance deteriorates rapidly. When the entire layer was transferred, performance was close to chance for the direct encoding. This is as expected as the weights of the layer are adapted to their original setting and do not store information in a form usable by the new model. However, in the case of transfer through neuron embedding, we are able to preserve a larger fraction of the relationships even when the entire layer is transplanted to a new network.

We stress that the direct representation and the neuron embedding representation both encode the same networks with the same weights; thus, the greater information transfer is due entirely to the way in which the layers are encoded.



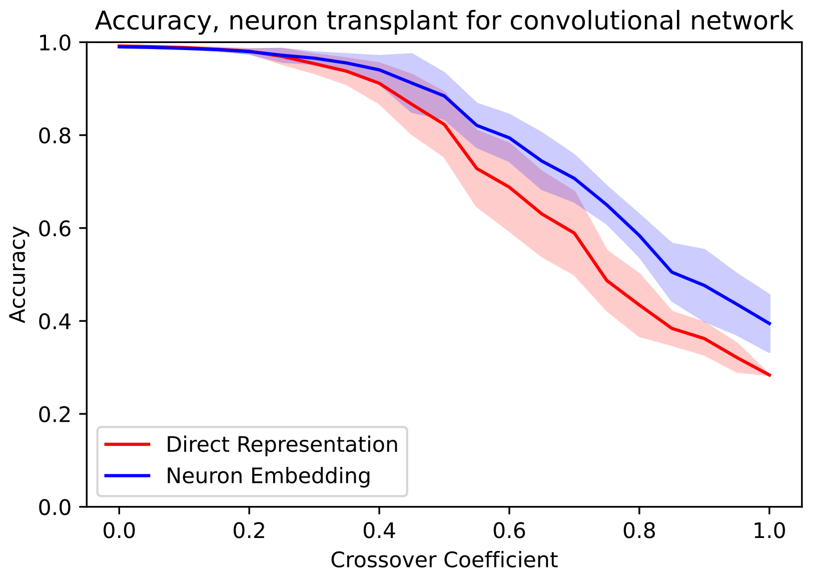


Fig. 3. Accuracy under neuron transplant for fully connected (top) and convolutional (bottom) models. Bold lines show the mean over 10 runs, and the shaded region indicates a 95% confidence interval for the mean. Crossover coefficient (horizontal axis) represents the fraction of neurons in the layer replaced by neurons from another model. We compare two identical networks encoded in two ways - direct encoding or neuron embedding. At 100% crossover, an entire layer from the source network is directly transplanted to the recipient network without any further training. We observe that the same network when encoded with neuron embedding maintains significantly more performance, and can function even when the entire layer is replaced.

Linear interpolation To investigate whether these results are an artifact of the neuron transplant method, we perform a second experiment, but rather than transferring single neurons we apply linear interpolation to every neuron in the layer simultaneously. For the direct representation, we linearly interpolate between the weights of the two models, and for the embedding representation we linearly interpolate between the corresponding embedding vectors of the neuron representation. Results of this operation are shown in Figure 4. We observe similar results to the previous experiment for the fully connected model, suggesting that the representation itself is responsible for the results. However, we note slightly worse performance by both representations on the convolutional model. It is worth noting that the embedding vectors themselves are interpolated, producing entirely new embeddings; this suggests that it is possible to perform crossover on the neuron level, as well as on the network level.

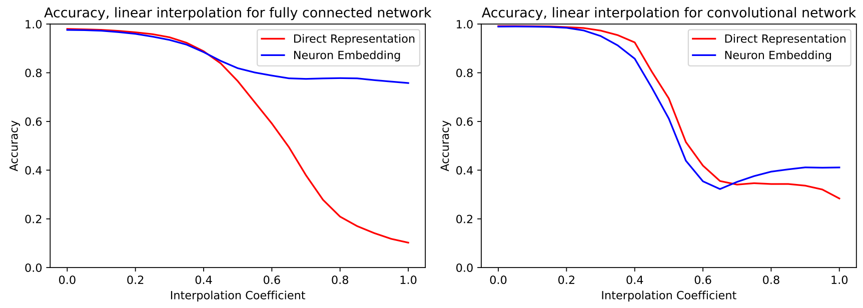


Fig. 4. Model accuracy under linear interpolation for fully connected model (left) and convolutional model (right). The weights and embedding vectors are directly interpolated by taking a weighted average, and all neurons in the layer are interpolated simultaneously. We observe similar results to the previous experiment on the fully connected network. Note that embedding vectors themselves are being changed; this suggests the possibility of neuron-level as well as network-level crossover.

5 Conclusion

In this paper we presented neuron embeddings, an indirect encoding method for representing a neural network in terms of unordered sets of individual neurons. This is a parameter-efficient representation which is also invariant to permutation of the neurons, allowing for better compatibility when performing crossover. Our method encapsulates the role of a neuron into a single self-contained representation which is used to generate the weights implicitly, allowing them to be transferred into a second neural network and still preserve some degree of function, even when the two networks are trained independently. This opens the door to new possibilities for neuroevolution, as this removes one important roadblock for crossover in neural networks, and can be used in conjunction with other methods such as those based on neuron alignment. In addition, the self-contained nature of the representations may prove useful for methods which evolve individual neurons, rather than complete networks. Of interest for future work is the extension of this method to larger hierarchical structures, which may also enable more efficient neural architecture search.

This work also has potential applications for cross-dataset knowledge transfer and transfer learning, which we intend to investigate in more depth moving forward. For example, it may be possible to transfer knowledge from multiple models or to improve upon existing methods of imitation learning. We also would like to further investigate whether neuron-based representation can aid in visualizing the patterns and knowledge contained in a neural network. If this is the case, this could lead to future applications for interpretability.

References

1. Bacciu, D., Mandic, D.P.: Tensor Decompositions in Deep Learning. Computational Intelligence p. 10 (2020)
2. Chen, X., Cheng, X., Mallat, S.: Unsupervised Deep Haar Scattering on Graphs. In: Advances in Neural Information Processing Systems. vol. 27. Curran Associates, Inc. (2014)
3. Chollet, F.: Xception: Deep Learning with Depthwise Separable Convolutions. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1800–1807. IEEE, Honolulu, HI (Jul 2017). https://doi.org/10.1109/CVPR.2017.195
4. Coleman, C., Narayanan, D., Kang, D., Zhao, T., Zhang, J., Nardi, L., Bailis, P., Olukotun, K., Ré, C., Zaharia, M.: DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS ML Systems Workshop p. 10 (2017)
5. Das, A., Hossain, M.S., Abdullah, S.M., Islam, R.U.: Permutation free encoding technique for evolving neural networks. In: International Symposium on Neural Networks. pp. 255–265. Springer (2008)
6. Deng, L., Li, G., Han, S., Shi, L., Xie, Y.: Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE **108**(4), 485–532 (2020). https://doi.org/10.1109/JPROC.2020.2976475
7. Devlin, J., Chang, M.W., Lee, K., 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. Association for Computational Linguistics, Minneapolis, Minnesota (Jun 2019). https://doi.org/10.18653/v1/N19-1423, `https://www.aclweb.org/anthology/N19-1423`
8. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with Analog Genetic Encoding. In: Runarsson, T.P., Beyer, H.G., Burke, E., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) Parallel Problem Solving from Nature - PPSN IX. pp. 671–680. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11844297_68

9. Edwards, H., Storkey, A.: Towards a Neural Statistician. 5th International Conference on Learning Representations (ICLR 2017) pp. 1–13 (2017)
10. Eliasmith, C., Anderson, C.H.: Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems. Computational Neuroscience Series, A Bradford Book, Cambridge, MA, USA (Oct 2002)
11. Fukushima, K., Miyake, S.: Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In: Competition and cooperation in neural nets, pp. 267–285. Springer (1982)
12. Gomez, F.J.: Robust Non-Linear Control through Neuroevolution. Ph.D. thesis, University of Texas at Austin (Aug 2003)
13. Haase, D., Amthor, M.: Rethinking Depthwise Separable Convolutions: How Intra-Kernel Correlations Lead to Improved MobileNets. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 14588–14597. IEEE, Seattle, WA, USA (Jun 2020). https://doi.org/10.1109/CVPR42600.2020.01461
14. Hausknecht, M., Khandelwal, P., Miikkulainen, R., Stone, P.: Hyperneat-ggp: A hyperneat-based atari general game player. In: Proceedings of the 14th annual conference on Genetic and evolutionary computation. pp. 217–224 (2012)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (Jun 2016). https://doi.org/10.1109/CVPR.2016.90
16. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up Convolutional Neural Networks with Low Rank Expansions. In: Proceedings of the British Machine Vision Conference 2014. pp. 88.1–88.13. British Machine Vision Association, Nottingham (2014). https://doi.org/10.5244/C.28.88
17. Karaletsos, T., Dayan, P., Ghahramani, Z.: Probabilistic Meta-Representations Of Neural Networks. arXiv:1810.00555 [cs, stat] (Oct 2018)
18. Kiers, H.: Towards a Standardized Notation and Terminology in Multiway Analysis. Journal of Chemometrics - J CHEMOMETR **14**, 105–122 (May 2000). https://doi.org/10.1002/1099-128X(200005/06)14:33.0.CO;2-I
19. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
20. Kolda, T.G., Bader, B.W.: Tensor Decompositions and Applications. SIAM Review **51**(3), 455–500 (Aug 2009). https://doi.org/10.1137/07070111X
21. Koutník, J., Cuccu, G., Schmidhuber, J., Gomez, F.: Evolving large-scale neural networks for vision-based reinforcement learning. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation. pp. 1061–1068 (2013)
22. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. Tech. Rep. TR-2009 (2009)
23. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation **1**(4), 541–551 (1989)
24. LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist **2** (2010)
25. Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., Teh, Y.W.: Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In: International Conference on Machine Learning. pp. 3744–3753. PMLR (May 2019)
26. Moriarty, D.E., Mikkulainen, R.: Efficient Reinforcement Learning through Symbiotic Evolution. Machine Learning **22**(1), 11–32 (Jan 1996). https://doi.org/10.1023/A:1018004120707

27. Oseledets, I.: Tensor-Train Decomposition. SIAM J. Scientific Computing **33**, 2295–2317 (Jan 2011). https://doi.org/10.1137/090752286
28. Page, D.: How to Train Your ResNet (Sep 2018)
29. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703 (2019)
30. Reisinger, J., Miikkulainen, R.: Acquiring evolvability through adaptive representations. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation - GECCO '07. p. 1045. ACM Press, London, England (2007). https://doi.org/10.1145/1276958.1277164
31. Schmidhuber, J.: Discovering neural nets with low kolmogorov complexity and high generalization capability. Neural Networks **10**(5), 857–873 (1997)
32. Schmidhuber, J., Wierstra, D., Gagliolo, M., Gomez, F.: Training Recurrent Networks by Evolino. Neural Computation **19**(3), 757–779 (Mar 2007). https://doi.org/10.1162/neco.2007.19.3.757
33. Smith, L.N.: A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. arXiv preprint arXiv:1803.09820 (2018)
34. Smith, L.N., Topin, N.: Super-convergence: Very fast training of neural networks using large learning rates. In: Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications. vol. 11006, p. 1100612. International Society for Optics and Photonics (2019)
35. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence **1**(1), 24–35 (2019)
36. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artificial life **15**(2), 185–212 (2009)
37. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)
38. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life **9**(2), 93–130 (2003)
39. Tang, Y., Ha, D.: The Sensory Neuron as a Transformer: Permutation-Invariant Neural Networks for Reinforcement Learning. arXiv:2109.02869 [cs] (Sep 2021)
40. Tang, Y., Nguyen, D., Ha, D.: Neuroevolution of self-interpretable agents. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference. pp. 414–424 (2020)
41. Tucker, L.R.: Some mathematical notes on three-mode factor analysis. Psychometrika **31**(3), 279–311 (Sep 1966). https://doi.org/10.1007/BF02289464
42. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 6000–6010. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (Dec 2017)
43. Wang, Z., Liu, J.C.: Translating math formula images to latex sequences using deep neural networks with sequence-level training (2019)
44. Yu, X., Liu, T., Wang, X., Tao, D.: On Compressing Deep Models by Low Rank and Sparse Decomposition. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 67–76. IEEE, Honolulu, HI (Jul 2017). https://doi.org/10.1109/CVPR.2017.15
45. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep Sets. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)