A PERMUTATION-INVARIANT REPRESENTATION OF NEURAL NETWORKS WITH NEURON EMBEDDINGS

Anonymous authors
Paper under double-blind review

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 and attempting to transfer part of a network without the preceding layers is usually destructive to any learned relationships. This paper proposes a method by which a neural network is represented in terms of interactions between neurons rather than explicit weights. In addition to reducing the number of free parameters, this encoding is agnostic to the ordering of neurons, bypassing a key problem for weight-based representations. This allows us to transplant individual neurons and layers into another network and still maintain their functionality. This is particularly important for tasks like transfer learning and neuroevolution. We show through experiments on the MNIST and CIFAR10 datasets that this method is capable of representing networks which achieve identical performance to direct weight representation, and that transfer done this way preserves much of the performance between two networks that are distant in parameter space.

1 INTRODUCTION

Neural networks are traditionally represented by their weights. This entails directly storing the weights in a structure such as a tensor. However, this type of representation has the property that for any given 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.

We propose that 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 these two problems. Current methods of reducing model size, such as pruning, quantization, tensor decomposition and knowledge distillation manipulate the weight representations and do not directly address the issue of permutation. In this paper, we take a different approach by thinking of networks in terms of neurons rather than weights. We present a method of representing the network which considers neural networks as unordered sets of neurons, building permutation invariance directly into the representation.

Our experiments with this representation show that considering networks as collections of neurons rather than weights provides benefits in addition to reducing the size of the model. Because the number of possible permuted representations of even a single layer is the factorial of the number of neurons, it is extremely likely that two networks which are trained on the same task will have incompatible representations, even if functionally similar. This means that cross-model transfer of information is generally impossible for arbitrary models (Neyshabur et al., 2020). We show that viewing networks as neurons rather than weights increases the robustness of the network to such operations. This has important implications for transfer learning as well as opening the door to population-based learning methods such as neuroevolution (Stanley et al., 2019).

In the following sections, we first present some context and motivation for considering permutation-invariant representation, and contrast our approach with existing ones. We then introduce neuron

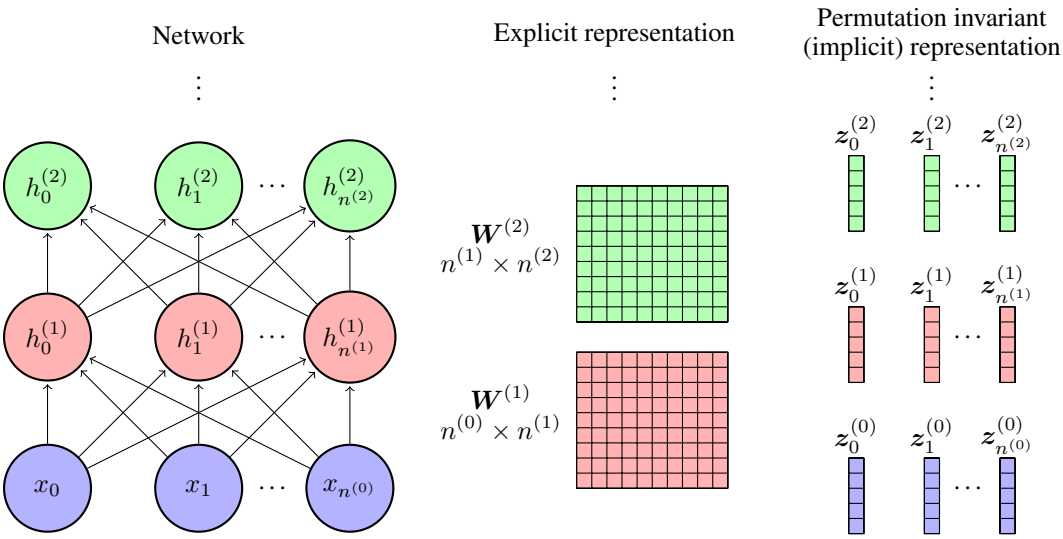

Figure 1: A neural network (left) with $n^{(i)}$ hidden units per layer is traditionally represented by explicitly specifying the weights of the connections, usually as a matrix or tensor $W^{(i)}$ of dimension $n^{(i-1)} \times n^{(i)}$ (middle). We propose instead to view the network as sets of neurons (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.

embeddings, self-contained representations of individual neurons, and the corresponding representation of the network as a series of unordered sets of neurons. The key to this approach is the self-contained nature of the neuron embeddings; weights are not fixed, but are generated dynamically in an attention-inspired way allowing them to be reordered and even moved between layers and models. We then perform experiments showing that this representation is able to achieve comparable performance to weight-based representations in fewer numbers of parameters. Finally, we test models on tasks designed to determine whether neuron-based representation transfers knowledge across models better than weight-based representation, and show that our method leads to a larger amount of information transferred.

2 Related Work

2.1 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. Edwards & Storkey (2017) and Zaheer et al. (2017) introduce methods which use pooling operations to perform permutation-invariant operations for set inputs. Chen et al. (2014) introduce permutation invariance into the features themselves by recombining pairs of features. Set Transformer (Lee et al., 2019) builds upon these by using self-attention to capture higher order interactions. Sensory neurons (Tang & Ha, 2021) use similar modular units to produce a PI policy. All these methods address permutation invariance in the inputs rather than the network representation. Our aim is 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.

Transfer Learning Transferring knowledge learned by a model from a source domain to a related target domain (Bozinovski, 2020; Pratt, 1992) is an effective method of improving the performance of models when target domain data is scarce. This may be done in a number of ways: by augmenting

the dataset in the target domain with relevant or modified data points from the source domain, by finding common features between the two domains, and directly transferring the network parameters by reusing part of the network (Pan & Yang, 2010; Tan et al., 2018). This is generally done by copying the first n layers of the network. It has been shown that one of the characteristics of a good transfer is that different instances of a model trained on the same task should lie close together in parameter space (Neyshabur et al., 2020); that is, they have compatible features and can be combined in a non-destructive manner without severe degradation in performance. Permutation dependence over the feature order is one source of incompatibility; even if the preceding layers generate the exact same features, it is likely they will still be in the "wrong" order for two different representations. Our goal in this paper is to eliminate this source of incompatibility, which may open up the possibility for transfer learning from multiple sources or transfer of partial layers. While still reliant on the exact features generated by preceding layers, permutation invariance ensures that the network is no longer sensitive to the ordering of the features; thus, the network is no longer reliant on co-adaptation of the later layers to the feature order, but only to the content of the features themselves.

Neuroevolution and Indirect Encoding Neuroevolution is the application of evolutionary methods to neural networks, involving the training of a population of networks. A key component of many evolutionary algorithms is recombination, which merges two individuals to create an offspring individual in order to preserve and propagate useful innovations (Eiben & Smith, 2015; Gangwani & Peng, 2018). Applying this to neural networks has been challenging precisely because of the "permutation problem" - the incompatibility of representations between networks due to permutation - and addressing it has been a central focus of neuroevolution work (Stanley et al., 2019). Previous methods in the neuroevolution literature addressed this by looking for analogous structures in the network to limit the impact of permutation (Stanley & Miikkulainen, 2002), or by sorting the neurons based on their connections (Das et al., 2008). 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 bypassing the problem.

A second challenge for scaling these methods is that weights are encoded directly, meaning each weight is specified explicitly. Indirect encoding is an alternative approach which represents the network using a small number of parameters and uses rules to generate the weights, with each parameter being responsible for multiple weights (Schmidhuber, 1997; Stanley & Miikkulainen, 2003). This is inspired by biological neural networks - it is clear that brains are not initialized randomly, but rather a number of innate capabilities are encoded for in the genome in an efficient way (Koulakov et al., 2021). However, the estimated 100 trillion connections in the human brain (Ackerman et al., 1992) far exceeds the capacity of our DNA to represent, and thus it is necessary to encode the information in a more efficient manner while still producing the required structure. This concept has proved successful at allowing larger networks to be trained with evolution (Stanley et al., 2009; Hausknecht et al., 2012; Koutník et al., 2013). Modern neural network architectures can also be viewed in this light; notably, convolution (Fukushima & Miyake, 1982; LeCun et al., 1989) and attention (Vaswani et al., 2017) generate large numbers of effective weights. In order to achieve better scaling, we also use indirect encoding, generating weights based on a small number of vector representations.

Neuron-based Representation Neuron-based representations have also previously been employed in the literature, often in the context of evolving individual neurons (Moriarty & Mikkulainen, 1996; Gomez, 2003; Schmidhuber et al., 2007) or compact representations of networks (Eliasmith & Anderson, 2002; Dürr et al., 2006; Reisinger & Miikkulainen, 2007; Karaletsos et al., 2018; Karaletsos & Bui, 2020). Our work makes use of neuron-based representation to achieve permutation invariance, but is aimed at bridging the gap between these two applications. In contrast to evolutionary approaches which view neurons as individually interchangable units to be optimized, 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 (Vaswani et al., 2017) is a highly successful mechanism which underpins most modern models for natural language processing. 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 (Tang et al., 2020). 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 (Vaswani et al., 2017) operates between the tokens given as inputs to the network; our method differs in that we use as endpoints the neurons themselves. This means the weights are input agnostic and tied to the neurons present in the network.

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. A number of approaches have been explored to achieve this which can be broadly classified into pruning, quantization, knowledge distillation and tensor decomposition approaches (Gale et al., 2019; Blalock et al.; Choudhary et al., 2020; Deng et al., 2020). The first three categories are orthogonal to our approach; although they all reduce the size of the model, they do so in a different manner and can be used in conjunction. The approach we describe here is closely related to the fourth category, 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 using tensor decomposition. Several methods for providing exact or approximate decompositions exist (Kolda & Bader, 2009; Bacciu & Mandic, 2020); commonly used methods include CP (Kiers, 2000), Tucker (Tucker, 1966) and tensor train (Oseledets, 2011) 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 Jaderberg et al. (2014) and Yu et al. (2017). 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 $r(m + n)$ (Deng et al., 2020). 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 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 allows for different attention kernels other than the linear dot-product; thus, it is only the simplest case that can be represented as an approximate factorization.

3 METHOD

We will first describe how our method works for a simple feedforward network. Then, we will describe how CNNs can be represented as well. In summary, our method removes all stored weights from the network and replaces them with a set of vector representations of the neurons present in the network. Weights are then generated in an attention-like way, with some important modifications.

It is important that each neuron's representation contains all the information necessary to perform its function - thus, there is no equivalent to the query, key and value networks of attention which the representations pass through before the weights are generated. 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 $K(\cdot, \cdot)$. Inspired by attention, we use as the weight the alignment score α_{ij}, calculated using a dot product kernel on the embedding $z_i^{(l)}$ of neuron i in layer l and the embedding $z_j^{(l+1)}$ of neuron j in layer $l + 1$ (Vaswani et al., 2017) with an optional nonlinearity σ:

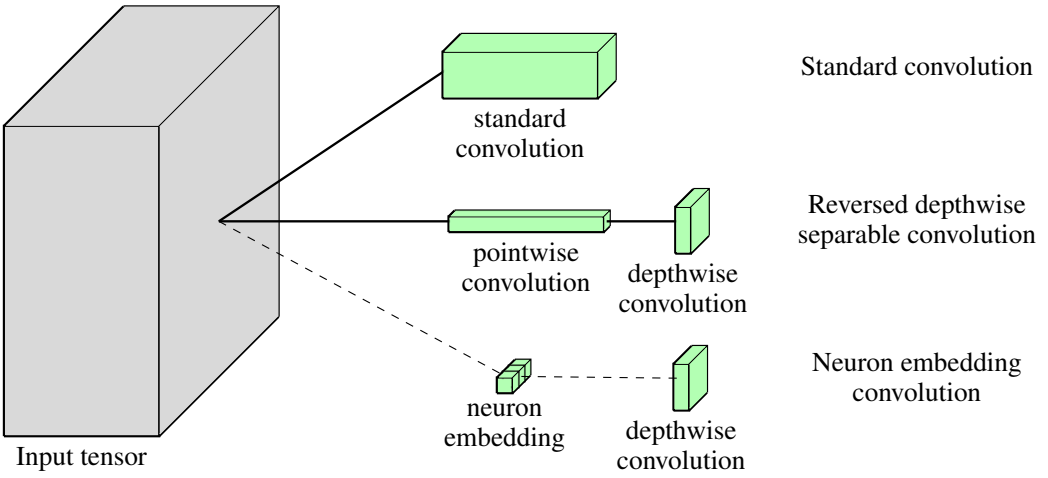

Figure 2: Representation of a convolutional neuron. The standard representation explicitly specifies all the weights in the kernel. Depthwise separable convolutions provide an approximate replacement by splitting the kernel into a pointwise kernel, which mixes information across channels, and a depthwise 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.

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

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:

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

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 (Devlin et al., 2019). A second possibility is to provide predefined embeddings; for example, through positional encodings (Vaswani et al., 2017). We tested sinusoidal positional embeddings for one and two dimensions (Vaswani et al., 2017; Wang & Liu, 2019) 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.

Encoding CNNs CNNs 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 (Chollet, 2017). 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 (Haase & Amthor, 2020).

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 (LeCun et al., 2010) and CIFAR-10 (Krizhevsky, 2009) datasets to evaluate the models. All models were implemented in Python using the PyTorch library (Paszke et al., 2019). For reproducibility, the code will be made available on GitHub upon publication. 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 (Smith, 2018; Page, 2018) 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-based representations 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-based 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 representations.

All models unless otherwise specified were trained with cross-entropy loss (Kingma & Ba, 2015), 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) (He et al., 2016) on CIFAR-10 based on the results of the DAWNBench benchmark (Coleman et al., 2017; Page, 2018). 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 (Smith & Topin, 2019) 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 (Smith, 2018). 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 dimen-

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**
CIFAR10	ResNet9 (direct)	2438794	8 conv 1 fc	89.40	0.3962
CIFAR10	ResNet9 (sep.)	287818	8 conv 1 fc	88.21	0.4312
CIFAR10	**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) use neuron embedding representation.

sionality of the neuron embeddings is set to 64 for fully connected models and 48 for convolutional models.

Our results 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.

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 pre-

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 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 calculated by neuron embedding from the weights in the reference network. The mean-squared amplitude of the weights in the reference network is 0.0152.

served when it is moved to a different setting. This evaluates the potential of this representation for crossover operations in neuroevolution and cross-model transfer learning.

We trained two 2-layer fully connected 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 fully connected parent, using the same learned input encodings for both. This was 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 is possible to replicate both the weights of the source and target network to high accuracy using the same input embedding but different neuron embeddings (Table 3).

Linear interpolation We first interpolated between the two sets of models to determine the shape of the loss landscape in between the models. We linearly interpolate between the weights of the direct representation, and between the corresponding embedding vectors of the neuron representation. We make no effort to pick analogous neurons for this, and corresponding embeddings and weights are simply chosen in the order they are stored. Results of this operation are shown in Figure 3. We observe that in both cases accuracy is impacted, but performance is similar for both representations suggesting similar loss landscapes along the direct line connecting the models.

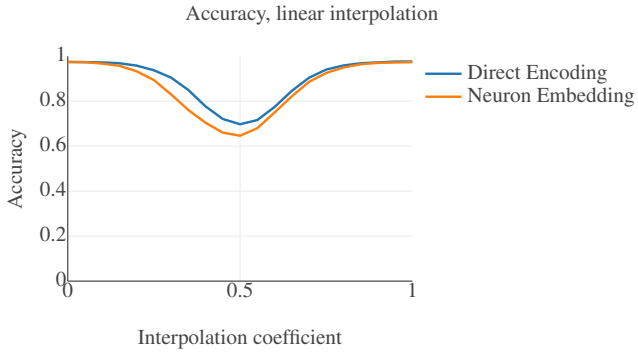

Figure 3: Model accuracy under linear interpolation. Weights/embeddings are directly interpolated.

Neuron transplant Next, we tested compatibility for both pairs of models by transferring a variable number of neurons in the 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 the that layer of 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 arbitrarily for this, as there is no particular ordering.

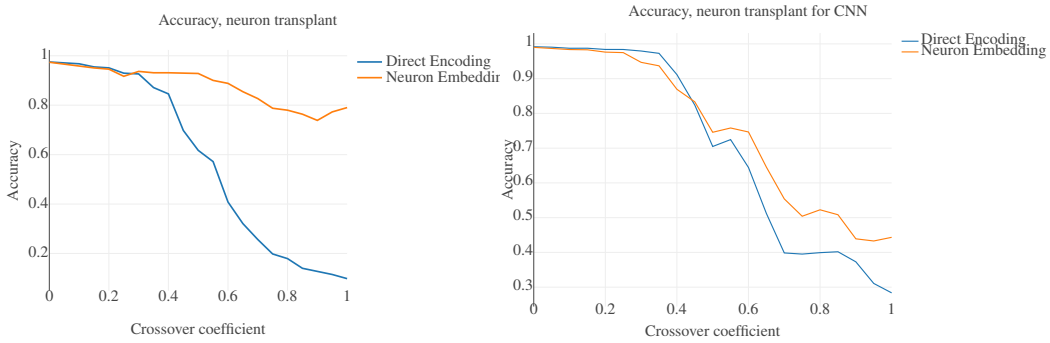

Figure 4: Accuracy under neuron transplanting. Crossover coefficient indicates the proportion of neurons in the layer replaced by neurons from another model. At 100% crossover, an entire layer from the source network is directly transplanted to the recipient network without any further training.

Encoding Method	Target Model Acc. (%)	Source Model Acc. (%)	50% Crossover Acc. (%)	100% Crossover Acc. (%)
Direct Encoding	97.48	97.66	61.73	9.83
Neuron Embedding	97.38	97.41	92.83	79.03

Table 3: Results for transferring a hidden layer neuron by neuron. 50% crossover refers to transfer of 50% of the neurons in the hidden layer from a source network to a target network, without modifying the other layers. 100% crossover means that the entire layer has been transferred.

The results in Figure 4.3 and Table 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 encoding and the neuron embedding network represent the same networks with the same weights; thus, the greater information transfer is due entirely to the way in which the layers are encoded.

5 Conclusion

In this paper we presented neuron embeddings, a method of 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, which allows for better cross-model compatibility. Because our method encapsulates the role of a neuron in a single compact representation, which we use to generate the weights implicitly, we are able to transfer all or part of layers from one network neuron-by-neuron while preserving some degree of function, even for two networks trained independently. This opens the door to the possibility of neuroevolution at scale, as it addresses a critical roadblock to crossover in neural networks, and could provide improvements to current population-based and ensemble methods. In addition, the encapsulated nature of the representations may allow for evolutionary methods to be applied to a single network, by treating the neurons within the network as a population. 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

Sandra Ackerman et al. *Discovering the brain*. National Academies Press, Washington, D.C., 1992.

Davide Bacciu and Danilo P Mandic. Tensor Decompositions in Deep Learning. *Computational Intelligence*, pp. 10, 2020.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the State of Neural Network Pruning? *Proceedings of Machine Learning and Systems 2 (MLSys 2020)*, pp. 18.

Stevo Bozinovski. Reminder of the First Paper on Transfer Learning in Neural Networks, 1976. *Informatica*, 44, September 2020. doi: 10.31449/inf.v44i3.2828.

Xu Chen, Xiuyuan Cheng, and Stephane Mallat. Unsupervised Deep Haar Scattering on Graphs. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

Francois Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, Honolulu, HI, July 2017. IEEE. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.195.

Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, pp. 1–43, 2020.

Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. *NIPS ML Systems Workshop*, pp. 10, 2017.

Anupam Das, Md Shohrab Hossain, Saeed Muhammad Abdullah, and Rashed Ul Islam. Permutation free encoding technique for evolving neural networks. In *International Symposium on Neural Networks*, pp. 255–265. Springer, 2008.

Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4): 485–532, 2020. doi: 10.1109/JPROC.2020.2976475.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

Peter Dürr, Claudio Mattiussi, and Dario Floreano. Neuroevolution with Analog Genetic Encoding. In Thomas Philip Runarsson, Hans-Georg Beyer, Edmund Burke, Juan J. Merelo-Guervós, L. Darrell Whitley, and Xin Yao (eds.), *Parallel Problem Solving from Nature - PPSN IX*, Lecture Notes in Computer Science, pp. 671–680, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-38991-0. doi: 10.1007/11844297_68.

Harrison Edwards and Amos Storkey. Towards a Neural Statistician. *5th International Conference on Learning Representations (ICLR 2017)*, pp. 1–13, 2017.

A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer-Verlag, Berlin Heidelberg, 2 edition, 2015. ISBN 978-3-662-44873-1. doi: 10.1007/978-3-662-44874-8. URL https://www.springer.com/gp/book/9783662448731.

Chris Eliasmith and Charles H. Anderson. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Computational Neuroscience Series. A Bradford Book, Cambridge, MA, USA, October 2002. ISBN 978-0-262-05071-5.

Kunihiko Fukushima and Sei Miyake. 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.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv e-prints*, arXiv:1902.09574, 2019. URL https://arxiv.org/abs/1902.09574.

Tanmay Gangwani and Jian Peng. Policy optimization by genetic distillation. In *6th International Conference on Learning Representations*, 2018.

Faustino John Gomez. *Robust Non-Linear Control through Neuroevolution*. PhD thesis, University of Texas at Austin, August 2003.

Daniel Haase and Manuel Amthor. 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, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. doi: 10.1109/CVPR42600.2020.01461.

Matthew Hausknecht, Piyush Khandelwal, Risto Miikkulainen, and Peter Stone. 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.

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)*, pp. 770–778, June 2016. doi: 10.1109/CVPR.2016.90.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. In *Proceedings of the British Machine Vision Conference 2014*, pp. 88.1–88.13, Nottingham, 2014. British Machine Vision Association. ISBN 978-1-901725-52-0. doi: 10.5244/C.28.88.

Theofanis Karaletsos and Thang D Bui. Hierarchical gaussian process priors for bayesian neural network weights. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17141–17152. Curran Associates, Inc., 2020.

Theofanis Karaletsos, Peter Dayan, and Zoubin Ghahramani. Probabilistic Meta-Representations Of Neural Networks. *arXiv:1810.00555 [cs, stat]*, October 2018.

Henk Kiers. Towards a Standardized Notation and Terminology in Multiway Analysis. *Journal of Chemometrics - J CHEMOMETR*, 14:105–122, May 2000. doi: 10.1002/1099-128X(200005/06)14:33.0.CO;2-I.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 51 (3):455–500, August 2009. ISSN 0036-1445, 1095-7200. doi: 10.1137/07070111X.

Alexei Koulakov, Sergey Shuvaev, and Anthony Zador. Encoding innate ability through a genomic bottleneck. *bioRxiv*, 2021.

Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. 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.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical Report TR-2009, 2009.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *International Conference on Machine Learning*, pp. 3744–3753. PMLR, May 2019.

David E. Moriarty and Risto Mikkulainen. Efficient Reinforcement Learning through Symbiotic Evolution. *Machine Learning*, 22(1):11–32, January 1996. ISSN 1573-0565. doi: 10.1023/A:1018004120707.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 512–523. Curran Associates, Inc., 2020.

Ivan Oseledets. Tensor-Train Decomposition. *SIAM J. Scientific Computing*, 33:2295–2317, January 2011. doi: 10.1137/090752286.

David Page. How to Train Your ResNet, September 2018.

Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010. ISSN 1558-2191. doi: 10.1109/TKDE.2009.191.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Lorien Y. Pratt. Discriminability-Based Transfer between Neural Networks. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pp. 204–211, San Francisco, CA, USA, November 1992. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-274-8.

Joseph Reisinger and Risto Miikkulainen. Acquiring evolvability through adaptive representations. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation - GECCO '07*, pp. 1045, London, England, 2007. ACM Press. ISBN 978-1-59593-697-4. doi: 10.1145/1276958.1277164.

Jürgen Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

Jürgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez. Training Recurrent Networks by Evolino. *Neural Computation*, 19(3):757–779, March 2007. ISSN 0899-7667. doi: 10.1162/neco.2007.19.3.757.

Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, pp. 1100612. International Society for Optics and Photonics, 2019.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Kenneth O Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pp. 270–279. Springer, 2018.

Yujin Tang and David Ha. The Sensory Neuron as a Transformer: Permutation-Invariant Neural Networks for Reinforcement Learning. *arXiv:2109.02869 [cs]*, September 2021.

Yujin Tang, Duong Nguyen, and David Ha. Neuroevolution of self-interpretable agents. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 414–424, 2020.

Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3): 279–311, September 1966. ISSN 1860-0980. doi: 10.1007/BF02289464.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, December 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.

Zelun Wang and Jyh-Charn Liu. Translating math formula images to latex sequences using deep neural networks with sequence-level training, 2019.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On Compressing Deep Models by Low Rank and Sparse Decomposition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 67–76, Honolulu, HI, July 2017. IEEE. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.15.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.