Grid-Functioned Neural Networks

Javier Dehesa [1 2] **Andrew Vidler** [2] **Julian Padget** [1] **Christof Lutteroth** [1]

Abstract

We introduce a new neural network architecture that we call “grid-functioned” neural networks. It utilises a grid structure of network parameterisations that can be specialised for different subdomains of the problem, while maintaining smooth, continuous behaviour. The grid gives the user flexibility to prevent gross features from overshadowing important minor ones. We present a full characterisation of its computational and spatial complexity, and demonstrate its potential, compared to a traditional architecture, over a set of synthetic regression problems. We further illustrate the benefits through a real-world 3D skeletal animation case study, where it offers the same visual quality as a state-of-the-art model, but with lower computational complexity and better control accuracy.

1. Introduction

In the last decades, a huge variety of neural network models and architectures have been put forward with the purpose of exploiting particular structures or the properties of different problem domains. Such is the case with convolutional neural networks for computer vision or recurrent neural networks for sequential data. However, beyond the structure of the data, network architectures rarely look at the data itself to condition their behaviour. The reason is simple: the analysis of the data values is left to the individual network units, according to the parameters that are learnt through the training process. That is, indeed, the whole intent of a neural network, whereby data can be largely regarded as an opaque container of information that is only made intelligible through the model. Nevertheless, this means the neural network must attempt to learn a single common function for the entire domain of the problem at hand. This is entirely possible, but in some cases it can be challenging for one model to reliably learn all the patterns hidden in data points scattered throughout the whole problem space. For example, predicting a geographical feature given a latitude and a longitude can be difficult for a neural network if it does not have an architecture that reflects the coordinate system articulating the data.

We introduce grid-functioned neural networks (GFNN), a novel kind of network architecture that overcomes this limitation through the definition of a multidimensional grid of neural network experts specialised in different subspaces of the input domain. The grid experts’ parameters are combined according to the given data, using an interpolation scheme, to produce a network configuration tailored to each particular input. As we will show, the result is a continuous and differentiable model that is able to capture complex local patterns better than conventional configurations, and in many cases with smaller inference complexity. Furthermore, GFNN supports a variety of inference methods that can be used to trade off between accuracy, memory use, and inference speed. Specifically, our contributions are:

1. The mathematical definition of GFNN.
2. An analysis of the computational complexity of inference for GFNN, as well as different approximate inference methods.
3. A comparison between the performance of GFNN and other architectures, namely a standard multilayer perceptron, using a small set of synthetic problems, and a state-of-the-art 3D animation synthesis model, as part of a case study.

2. Related Work

Ensembles of experts, in the general sense, have been used in machine learning for a long time. Classic algorithms like bagging (Breiman, 1996) or AdaBoost (Freund & Schapire, 1997), which aggregate the results of a collection of learners, take advantage of the diversity in the ensemble to mitigate the shortcomings of individual models. This kind of learner aggregation is very well suited for classification, and it can also be extended for regression (Drucker, 1997). However, traditional ensembles are unstructured sets of “weak” learners for the whole problem, as opposed to a structure of “strong” models for specific subspaces of it. This prevents them from exploiting the continuity and simplicity of local patterns in the data. In the field of neural networks,

[1] Department of Computer Science, University of Bath, Bath, UK [2] Ninja Theory, Cambridge, UK. Correspondence to: Javier Dehesa <javier.dehesa@bath.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

collections of experts have also appeared in some forms. The filters of convolutional networks used in computer vision have been shown to correspond to specific shapes and patterns of increasing complexity (Zeiler & Fergus, 2014), forming what could be regarded as an emergent hierarchy of experts. Variational autoencoders (Kingma & Welling, 2014) feature a probabilistic latent vector that also disentangles the space of the problem into a reduced number of expert parameters that "describe" the data. The success of these models comes from the structure they impose in the way they learn, which proves to be more effective than letting the optimisation process alone find these structures by itself. Nevertheless, these experts are all still emergent from the training data. As such, it is not always straightforward to tell what is the purpose of each expert, and whether the model will contain adequate experts for a given domain.

Holden et al.'s (2017) phase-functioned neural network model (PFNN) takes a different approach, where the expert structure consists of four neural network parameterisations that become more or less active depending on the given input data. Each expert specialises in a different, predefined aspect of the problem at hand (in the original work, 3D bipedal locomotion animation), which improves the accuracy of the model and significantly reduces the averaging effect commonly seen in neural networks, and together they cover the whole domain of the problem. However, parameterisations in PFNN can only specialise on one feature of the input data (the locomotion phase), so PFNN is only applicable to a limited range of scenarios. Subsequent models, like mode-adaptive neural networks (Zhang et al., 2018) or neural state machines (Starke et al., 2019), attempt to solve this limitation by using instead an unstructured pool of expert parameterisations that become active depending on the output of a second neural network. This again obscures the purpose of each expert and makes the overall model significantly more sensitive to initialisation. Our proposed GFNN model starts from PFNN, extending its applicability to an arbitrary number of specialisation dimensions and experts.

An alternative approach to exploiting locality was introduced by Liu et al. (2018) in their CoordConv model. Their work showed how conventional convolutional models fail on an apparently simple coordinate classification task. This is solved by extending the convoluted data with additional channels encoding the input coordinate, which allows the network to learn local patterns. Unlike our proposal, though, CoordConv does not impose any structure on the model itself, so the patterns are still discovered and encoded in the parameters of the whole network, instead of in dedicated subspaces.

Finally, our work can be examined from the point of view of implicit neural representations. This is a growing area in which recent advances are showing great promise (Peng et al., 2020; Sitzmann et al., 2020), and where our grid model could be directly applied. The recent work by Takikawa et al. (2021) for 3D surfaces is comparable to our approach, as it too associates vectors of parameters with particular points in space to learn highly-detailed local features. Similar to GFNN, these vectors are also interpolated across the space, with the difference that the interpolated vector is used as input to a neural network, and not as its parameters. While we do not analyse GFNN from the perspective of implicit representations here, we believe it has potential for application in this way.

3. Model Construction

Though the underlying concept of GFNN can be applied to any kind of neural network architecture (e.g. a convolutional network), we will consider here the case where it is applied to a traditional multilayer perceptron (MLP) model. Let us consider the problem of estimating a function $f\colon \mathbb{R}^{D_{In}} \to \mathbb{R}^{D_{Out}}$ from a set of examples $X \subset f$. An MLP for this problem would be comprised of $H \in \mathbb{N}$ hidden layers with corresponding sizes $\{D_1, \ldots, D_H\} \subset \mathbb{N}$. Assuming the same activation function a is applied to all hidden and output layers, the perceptron function $MLP\colon \mathbb{R}^{D_{In}} \to \mathbb{R}^{D_{Out}}$ is then expressed as:

$$\begin{aligned} MLP(\boldsymbol{x}; \boldsymbol{\theta}_{MLP}) &= \boldsymbol{x}_{H+1} \\ \boldsymbol{x}^0 &= \boldsymbol{x} \\ \boldsymbol{x}^i &= a(W_i \boldsymbol{x}^{i-1} + \boldsymbol{b}_i) \quad 0 < i \leq H+1 \end{aligned} \tag{1}$$

Where $\Theta_{MLP} = \{(W_1, b_1), \ldots, (W_{H+1}, b_{H+1})\} \in \mathbb{R}^T$ is the collection of T parameters to MLP, each pair $W_i \in \mathbb{R}^{D_i \times D_{i-1}}$ and $b_i \in \mathbb{R}^{D_i}$ being the weight matrix and bias vector of the i-th layer, with $D_0 = D_{In}$ and $D_{H+1} = D_{Out}$. This defines a basic neural network model to estimate f.

We now consider the case where f can be "characterised" by some features of the input. By this we mean there are $K > 0$ scalar functions computable from the input of f that are strongly correlated to the behaviour of the function in some sense. This set of functions, $\{p_1, \ldots, p_K\}$, only have the requirement that their value must be within a finite interval, which by convention we fix to $[0, 1]$, and so each of them is notated as $p_i\colon \mathbb{R}^{D_{In}} \to \mathbb{R}$. Each function can be assigned the property of being either "periodic" or "non-periodic", which in this context is meant to represent the quality of being a cyclic feature. For example, a feature representing an angle (with zero being $0°$ and one being $360°$) can be considered periodic, as the behaviour of f is expected to be similar when the feature takes values close to zero and close to one. The goal is to build a model with multiple parameterisations that specialise in the different possible combina-

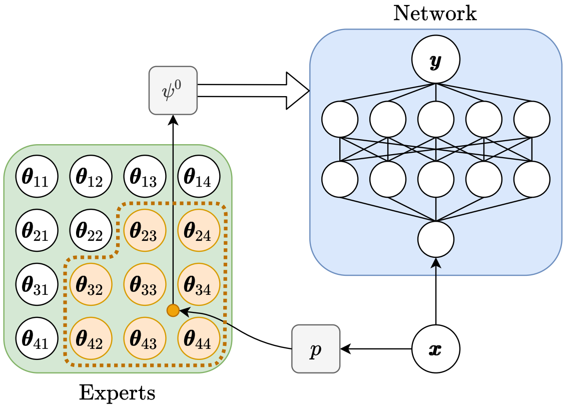

Network
$\boldsymbol{y}$
ψ^0
$\boldsymbol{\theta}_{11}$ $\boldsymbol{\theta}_{12}$ $\boldsymbol{\theta}_{13}$ $\boldsymbol{\theta}_{14}$
$\boldsymbol{\theta}_{21}$ $\boldsymbol{\theta}_{22}$ $\boldsymbol{\theta}_{23}$ $\boldsymbol{\theta}_{24}$
$\boldsymbol{\theta}_{31}$ $\boldsymbol{\theta}_{32}$ $\boldsymbol{\theta}_{33}$ $\boldsymbol{\theta}_{34}$
$\boldsymbol{\theta}_{41}$ $\boldsymbol{\theta}_{42}$ $\boldsymbol{\theta}_{43}$ $\boldsymbol{\theta}_{44}$
p
$\boldsymbol{x}$
Experts

Figure 1. Diagram of a GFNN model with a 4×4 grid of experts. The set of p functions determines a point in the expert grid space, which is used to blend the surrounding expert parameterisations (in orange) using the multidimensional cubic interpolation scheme defined by ψ^0. The result is used as the parameterisation to evaluate the input $\boldsymbol{x}$ on the base network architecture.

tions of values these feature functions may take. Figure 1 describes the overall approach for a two-dimensional case. We consider the feature vector space $P = [0,1]^K$ and slice it into an integer number of divisions across each dimension, $\{N_1, \ldots, N_K\} \subset \mathbb{N}$. These divisions are then arranged in a grid $G = \{1, \ldots, N_1\} \times \ldots \times \{1, \ldots, N_K\}$, and each $\boldsymbol{g} \in G$ has then an associated feature space location given by $c\colon G \to P$:

$$c(\boldsymbol{g})_i = \begin{cases} (g_i - 1)/N_i & \text{if } p_i \text{ is periodic} \\ (g_i - 1)/(N_i - 1) & \text{otherwise} \end{cases} \tag{2}$$

This means that, for non-periodic features, the first and last divisions are located exactly at zero and one in the corresponding dimension, while for periodic features the first division corresponds to both zero and one, while the last division is located before one.

The GFNN model is composed of a set of expert parameterisations $\Theta = \{\theta^K_{\boldsymbol{g}} \mid \boldsymbol{g} \in G\} \subset \mathbb{R}^T$, each one specialised in its corresponding $c(\boldsymbol{g})$. The value of the feature functions, however, will rarely fall exactly into any of these points. In the general case, a specialised parameterisation for the given input is computed as a blend of those in Θ. This blend could be defined in a variety of ways. Frequently, models with a smooth behaviour are preferable, and so C^1-continuity may be desirable. Multidimensional cubic spline interpolation satisfies this property (Catmull & Rom, 1974), so it is a good choice for us. This is built on the basis of the one-dimensional cubic spline interpolation function $S\colon [0,1] \times \mathbb{R}^{T \times 4} \to \mathbb{R}^T$, defined as:

$$\begin{aligned} S(\alpha, \boldsymbol{\tau}_1, \boldsymbol{\tau}_2, \boldsymbol{\tau}_3, \boldsymbol{\tau}_4) = {} & \boldsymbol{\tau}_1 \left(-\frac{1}{2}\alpha + \alpha^2 - \frac{1}{2}\alpha^3 \right) \\ & + \boldsymbol{\tau}_2 \left(1 - \frac{5}{2}\alpha^2 + \frac{3}{2}\alpha^3 \right) \\ & + \boldsymbol{\tau}_3 \left(\frac{1}{2}\alpha + 2\alpha^2 - \frac{3}{2}\alpha^3 \right) \\ & + \boldsymbol{\tau}_4 \left(-\frac{1}{2}\alpha^2 + \frac{1}{2}\alpha^3 \right) \end{aligned} \tag{3}$$

This defines a T-dimensional curve from $\boldsymbol{\tau}_2$ to $\boldsymbol{\tau}_3$ such that it can be concatenated to another curve from from $\boldsymbol{\tau}_1$ to $\boldsymbol{\tau}_2$ on one side and from $\boldsymbol{\tau}_3$ to $\boldsymbol{\tau}_4$ on the other side in a C^1-continuous fashion. Specifically, given a set of parameterisations $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N\}$ across a grid dimension corresponding to a feature function p (evaluated on an input vector $\boldsymbol{x} \in \mathbb{R}^{D_{In}}$), we can define the interpolating curve across all of them as:

$$\begin{aligned} & \widehat{S}(p(\boldsymbol{x}), \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N\}) = S(\alpha, \boldsymbol{\theta}_{t_1}, \boldsymbol{\theta}_{t_2}, \boldsymbol{\theta}_{t_3}, \boldsymbol{\theta}_{t_4}) \\ & \qquad \alpha = \beta - \lfloor \beta \rfloor \\ & \qquad \beta = \begin{cases} Np(\boldsymbol{x}) & \text{if } p \text{ is periodic} \\ (N-1)p(\boldsymbol{x}) & \text{otherwise} \end{cases} \\ & t_i = \begin{cases} 1 + (\lfloor \beta \rfloor + i - 2 \bmod N) & \text{if } p \text{ is periodic} \\ \min(\max(\lfloor \beta \rfloor + i - 1, 1), N) & \text{otherwise} \end{cases} \end{aligned} \tag{4}$$

To break down the expression, p is a value that falls in the interval between the divisions t_2 and t_3. $\boldsymbol{\theta}_{t_2}$ and $\boldsymbol{\theta}_{t_3}$ are therefore the closest expert parameterisations to the left and right of p respectively, while $\boldsymbol{\theta}_{t_1}$ and $\boldsymbol{\theta}_{t_4}$ are the second closest experts to the left and right respectively. The value $\alpha \in [0,1)$ is the relative position of p within the interval. The definition of the spline interpolation function ensures that the complete curve (open for non-periodic features, closed for periodic ones) traverses all the given parameterisations, all while preserving smooth continuity.

The function $\widehat{S}$ interpolates parameterisations across one dimension of the grid. Applying it multiple times we can then interpolate across all of its dimensions. We express this in terms of a set of functions $\psi^i_*\colon \mathbb{R}^{D_{In}} \to \mathbb{R}^T$ recursively defined as:

$$\begin{aligned} \psi^K_{j_1 \ldots j_K}(\boldsymbol{x}) &= \theta^K_{j_1 \ldots j_K} \\ \psi^i_{j_1 \ldots j_i}(\boldsymbol{x}) &= \widehat{S}(p_{i+1}(\boldsymbol{x}), \{\psi^{i+1}_{j_1 \ldots j_i 1}, \ldots, \psi^{i+1}_{j_1 \ldots j_i N_{i+1}}\}) \\ & \qquad 0 \leq i < K \end{aligned} \tag{5}$$

Each $\Psi_i(\boldsymbol{x}) = \{\psi^i_{\boldsymbol{h}}(\boldsymbol{x}) \mid \boldsymbol{h} \in \mathbb{N}^{N_1 \times \ldots \times N_i}\}$ is an i-dimensional grid of parameterisations obtained after $K - i$ dimensions have been interpolated. The process ends at $\psi^0(\boldsymbol{x}) = \widehat{S}(p_1, \Psi_1(\boldsymbol{x}))$. This is the interpolated expert parameterisation for the given value of $\boldsymbol{x}$. The final expression of the grid-functioned neural network model $GFNN : \mathbb{R}^{D_{In}} \to \mathbb{R}^{D_{Out}}$, parameterised by Θ, is:

$$GFNN(\boldsymbol{x}; \Theta) = MLP(\boldsymbol{x}; \psi^0(\boldsymbol{x})) \tag{6}$$

That is, the GFNN model can be summarily described as an MLP in which the parameters are computed as a smooth interpolation of a grid of experts.

4. Inference Complexity

Given a base MLP with T parameters, a grid model constructed on top of it (where each expert has T parameters) will have a computational cost of inference significantly higher than of the base model. However, since each expert in the grid deals with a subset of the problem, the size of the base model should be only a fraction of that of a regular MLP attempting to solve the entire problem. In terms of space, the grid model will have a total of $T_K = T \prod_{j=1}^{K} N_j$ parameters. This establishes a hyperparameter for the model by which, given a budget of model parameters, we can choose how to distribute them across grids of different sizes. In terms of time, we must consider two factors: the cost of the cubic spline interpolation and the cost of the MLP inference. The cost of the MLP inference depends on the specific architecture (number and size of layers), and it is well understood. The cost of the spline interpolation, on the other hand, is worth analysing. This cost can be estimated as the number of floating point operations carried out in the process, and it can be derived from eq. (5). Each intermediate interpolation $\boldsymbol{\psi}^i_{j_1 \ldots j_i}$ $(0 \leq i < K)$ aggregates N_{i+1} parameterisations. According to eq. (4) and eq. (3), this involves four interpolation weights (the coefficients that multiply each $\boldsymbol{\tau}$ term) for the four parameterisations closest to $p_{i+1}(\boldsymbol{x})$. Note these four coefficients are the same for every interpolation computed as part of Ψ^i, and they take a fixed number of floating point operations F_W to compute. The aggregation itself of the four vectors of T parameters (after computing the coefficients) takes four multiplications and three additions per parameter, totalling $7T$ floating point operations. The computation of each $\Psi_i(\boldsymbol{\gamma})$ requires interpolating $\prod_{j=1}^{i} \min(N_j, 4)$ grid parameterisations (note that even for grid dimensions with more than four divisions only four parameterisations participate in the interpolation). Putting everything together, the number of floating point operations required by the cubic spline interpolation F_{Interp} results in:

$$F_{Interp} = \sum_{i=0}^{K-1} \left(F_W + 7T \prod_{j=1}^{i} \min(N_j, 4) \right) \tag{7}$$

Or more simply:

$$F_{Interp} \in \mathcal{O}\left(KT \prod_{i=1}^{K} \min(N_i, 4) \right) \tag{8}$$

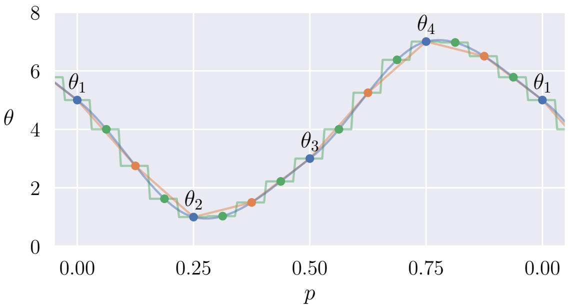


Figure 2. One-dimensional parameter interpolation using spline interpolation (blue) and linear (orange) and stepwise (green) piecewise approximations. Original grid has four points, linear approximation uses eight precalculated points and stepwise approximation uses sixteen precalculated points.

Following the same method proposed by Holden et al. (2017) for approximate inference of PFNN, it is possible to reduce the cost of the interpolation at the expense of more space and reduced accuracy. Specifically, once the grid model has been trained, parameter interpolations can be precalculated for a denser grid of locations, which can then be used to approximate the spline interpolation. This approximation can be linear, taking only the pair of parameterisations closer to each p, or stepwise, simply taking the value of the closest one. Figure 2 exemplifies this for one dimension. Using a larger number of precalculated parameterisations in the grid, a linear and a stepwise approximation can provide a reasonable estimation of the actual spline interpolation of the parameter.

In the case of linear interpolation, the computation involves taking the closest 2^K parameterisations across all dimensions in the grid and computing the weight for each of them, which takes KF_L operations, where F_L is a constant. The interpolation consists of multiplying the T parameters of each of those parameterisations with the corresponding weight and then adding them all. The total number of floating point operation F_{Linear} is then:

$$F_{Linear} = 2^K \left(T + KF_L\right) \in \mathcal{O}\left(2^K \left(T + K\right)\right) \tag{9}$$

Grid size	Hidden layers	Parameters	FLOPS
1×1 (MLP)	100, 100	10 501	20 801
3×3	31, 31	10 053	24 587
5×5	18, 18	10 375	15 352
7×7	12, 12	10 045	7 594

Table 1. Evaluated models. All models have two input units, one output unit and two hidden layers with the specified number of neurons.

For stepwise approximation, all the computation is reduced to finding the closest parameterisation in the grid. This takes a fixed number of floating point operations F_C per dimension, and thus the total $F_{Stepwise}$ is:

$$F_{Stepwise} = KF_C \in \mathcal{O}(K) \tag{10}$$

This range of evaluation methods offer a degree of flexibility over the usage of the model, particularly in contexts where computational resource for inference are limited.

5. Evaluation

In order to evaluate the potential of GFNN as compared to a conventional MLP model, we take a set of synthetic regression problems and compare the performance of different models with a comparable number of parameters. The problems are built from two 2D real functions, the Rosenbrock function (Rosenbrock, 1960) and the Ackley function (Ackley, 1987), shown in fig. 3. For each problem, 200 points are randomly sampled, 80% of which are used for training and 20% for evaluation.

Being naturally two-dimensional problems, these cannot be properly modelled with a one-dimensional structure of experts like PFNN. We compare the performance of an MLP and several GFNN models with a 2D grid. Table 1 shows the evaluated model configurations, all of which have approximately 10 000 parameters. As can be seen, even with the added parameter interpolation cost, GFNN quickly becomes less computationally expensive than MLP for larger grid sizes with smaller experts. ReLU activation (Nair & Hinton, 2010) is used for the hidden layers and linear activation in the output. Input and output data is normalised with respect to the mean and standard deviation of the training data. For the GFNN models, the non-periodic functions p_1 and p_2 that determine the point in the grid to activate are derived from the input x and y values, applying the sigmoid function to their normalised values. The grids are therefore directly correlated with the domain of the functions, and each expert will be specialised in a particular subspace of the problem.

Rosenbrock

Model	Median	Mean	SD
MLP	145.1	266.5	368.3
3×3	7.7	20.6	50.6
5×5	4.5	15.9	30.9
7×7	13.5	43.6	120.0

Ackley

Model	Median	Mean	SD
MLP	1.723	1.776	1.083
3×3	0.489	0.581	0.402
5×5	0.492	0.689	0.573
7×7	0.411	0.615	0.492

Ackley (small region)

Model	Median	Mean	SD
MLP	0.393	0.484	0.228
3×3	0.028	0.068	0.151
5×5	0.018	0.052	0.110
7×7	0.020	0.057	0.106

Table 2. Absolute error statistics of the evaluated models for each problem.

Every model was trained on each problem to minimise the mean squared error at the output using Adam optimisation (Kingma & Ba, 2015). The training ran for 100 000 steps on batches of 32 examples per step with a fixed learning rate of 0.001. For the sake of simplicity, and to reduce the amount of confounding factors affecting the results, no regularisation mechanisms were used. Results of the experiments can be found in table 2. In general, GFNN models clearly surpass the MLP in all problems, even though 5×5 and 7×7 models require fewer floating point operations. The mean and median errors are reduced between 60% and 95% for GFNN, and standard deviation is also significantly decreased in all cases. It is interesting to note that different grid sizes are better suited for different problems. This shows the potential of the GFNN grid size as a hyperparameter to the model. The function reconstructions in fig. 3 offer a more intuitive interpretation of the results. Even in the more difficult case of Ackley, GFNN models can give a closer approximation of the overall shape of the function. In the case of Rosenbrock and Ackley (small region), the MLP reconstructions are very poor in comparison with the fairly accurate GFNN models.

In order to evaluate the scalability of the models, we ran another set of experiments over a revised version of the Ackley problem. We take now 1 200 training samples and 300 testing samples, which should allow the models to capture the high-frequency detail of the function, and train more complex configurations with around 60 000 parameters to increase the capabilities of the models, as shown in

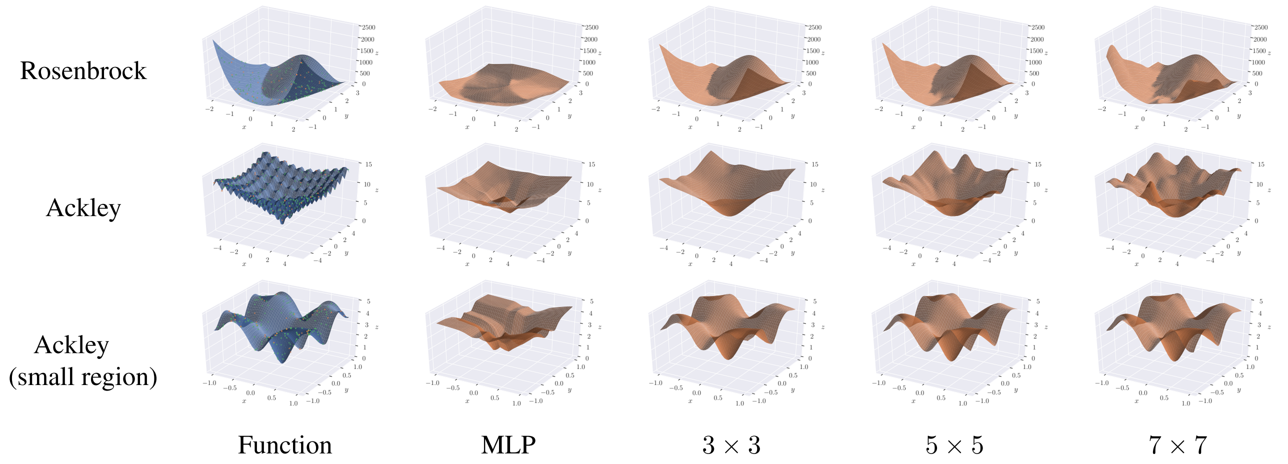
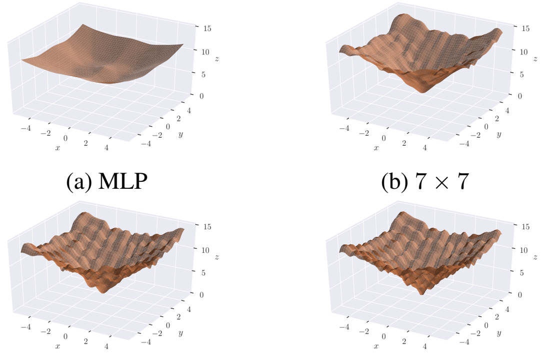


Figure 3. Evaluation problems and reconstructions. Green points in the problem functions represent training data, and orange points represent evaluation data.



(a) MLP

(b) 7 × 7

(c) 10 × 10

(d) 14 × 14

Figure 4. Reconstructions of the Ackley function for the revised problem.

Grid size	Hidden layers	Parameters	Flops
1 × 1 (MLP)	250, 250	63 751	190 752
7 × 7	33, 33	61 495	46 402
10 × 10	22, 22	59 500	22 004
14 × 14	15, 15	58 996	11 140

Table 3. Evaluated models for the revised Ackley problem. All models have two input units, one output unit and two hidden layers with the specified number of neurons.

Model	Median	Mean	SD
MLP	1.556	1.597	0.918
7 × 7	0.303	0.380	0.486
10 × 10	0.194	0.289	0.264
14 × 14	0.148	0.225	0.233
Interp. (1 200 samples)	0.174	0.248	0.238

Table 4. Absolute error statistics for the revised Ackley problem. Last row shows the error statistics using linear interpolation over the training data.

table 3. The design of GFNN allows us to adapt the grid size to the problem, so we can use even larger grids with smaller experts for this case. This also means that the number of floating point operations of the GFNN models is even smaller than before with respect to the MLP. All models are trained in the same manner as before. The results in table 4 show that, while the new MLP only reduces the mean error by about 10% with respect to its previous configuration, the GFNN models improve the best previous result by more than 60%. It also shows a progression in the accuracy as the grid size increases, in accordance with the high local variance of the problem. These results are less practical, since the size of the training dataset is big enough to get good estimations by simply interpolating the available data; nevertheless, GFNN models can even surpass that result, in stark comparison with the MLP. The function reconstructions in fig. 4 reveal that, while the MLP is barely any closer to the actual function shape than before, the GFNN models are now capable of reproducing most of the detail in the function surface accurately.

These experiments suggest that not only can GFNN models learn complex functions more effectively with a lower computational cost, but also they scale better when the parameter budget is increased, thanks to the flexible grid design.

6. Case Study: Quadruped Locomotion

The capabilities of GFNN have been further evaluated in the context of a real-world 3D animation problem: quadruped locomotion synthesis as studied by Zhang et al. (2018) in their discussion of mode-adaptive neural networks (MANN). This is an example of a problem where PFNN (Holden et al., 2017), which proposed a one-dimensional structure of experts specialised in the bipedal locomotion phase, cannot be directly used, due to the inherent complexities of quadruped locomotion patterns. Instead, the MANN model uses an unstructured pool of experts activated according to a second neural network, thus acquiring a specialisation defined by the training process itself. We compare this model with GFNN to demonstrate the benefits of a structured grid of experts and the flexibility of our approach.

The goal of this problem is to predict the pose that a quadruped character will adopt in the next frame given the current pose and additional control information (namely past and future trajectory). We use the same dataset and input and output encoding presented by Zhang et al., changing only the model doing the prediction. We define two non-periodic specialisation dimensions for our models: the velocity of the character and the angle of the future trajectory. This means our GFNN model will contain a 2D grid of experts specialised in different combinations of character speed and control direction. Refer to the supplementary material for more details on the grid dimension definitions and data preparation.

We trained a 3×3 GFNN model with a base architecture of two hidden layers with 512 units and ReLU activation, similar to the expert architecture in the original MANN model. Following the discussion in section 4 on inference approximation, we test different model approximations using linear parameter interpolation with a varying number of precalculated values in the grid. This offers a level of control over the cost of evaluating the model, which is critical in the context of real-time animation. These models are compared against one MANN model with eight experts, as proposed by Zhang et al., and another with nine experts, which matches the number of experts (and thus the number of parameters) in our GFNN model. Table 5 summarises the evaluated models. Note that all linear approximation models have the same inference cost, as the number of experts that participate in each individual evaluation is independent of the grid size. In the case of MANN, all experts are always involved in the inference. Linear approximations accelerate the evaluation of the slower cubic interpolation model to make it more than 30% faster than both MANN models.

Model	MFLOPS	Time (ms)	Size (MiB)
GFNN			
Cubic 3×3	12.12	7.91	18.9
Linear 3×3	3.31	3.89	18.9
Linear 5×5	3.31	3.85	52.5
Linear 7×7	3.31	3.87	102.9
Linear 9×9	3.31	3.82	170.2
MANN			
8 experts	9.37	5.67	16.8
9 experts	10.47	6.13	18.9

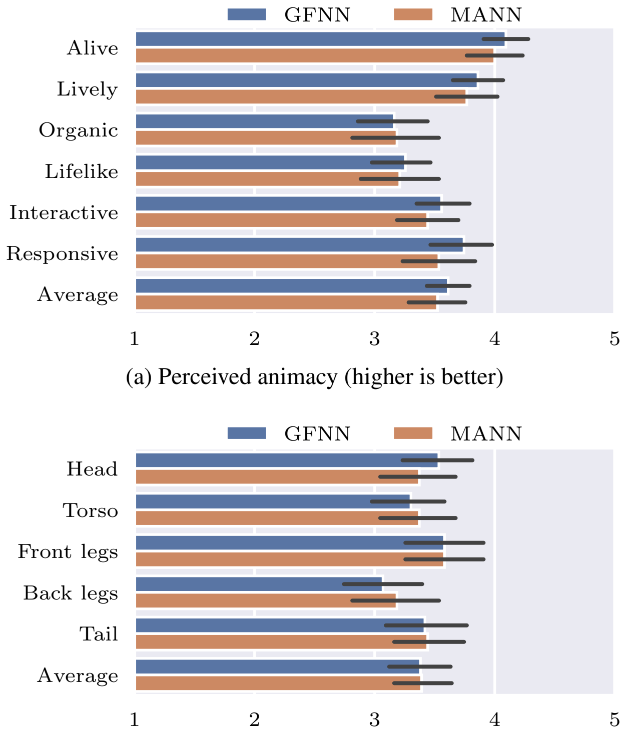
Table 5. Evaluated model configurations for the quadruped locomotion problem. All models use a base architecture with two hidden layers with 512 units each. Grid size of all GFNN models is 3×3. Evaluation time measured on an Intel Core i7-7700K CPU running at 4.20 GHz.

All models are evaluated over an identical scenario where the character is instructed to follow a given trajectory at a certain speed. We first evaluate the accuracy of the models adhering to the requested control commands, measuring the deviation from the marked path and the difference between the requested and actual speed of the character. Table 6 shows these results. While all models exhibit comparable behaviour in terms of trajectory accuracy, GFNN models are clearly superior at matching the requested control speed. Linear approximations have a slightly lower accuracy than the cubic interpolation model, but are still significantly better than MANN, demonstrating the potential of the accuracy, memory and speed trade-off in inference approximation. We believe this stark difference between GFNN and MANN is due to the fact that GFNN has explicitly defined experts for the different speeds of the character, which results in a much more reliable behaviour than the emergent expertise on which MANN relies. It is also worth noting that, seemingly for the same reason, MANN models were also far more sensitive to training initialisation.

Finally, in order to evaluate the visual quality of the results, we conducted a user study showing the animated character traversing the same evaluation path. In this case we limited the models to the MANN with 8 experts, corresponding to the originally proposed model, and the GFNN linearly interpolated on a 7×7 grid, which offers a reasonable balance between quality and size while being significantly faster than MANN. A total of 43 participants were shown videos of the character driven by each of the models and asked to complete a questionnaire about their impressions. This included questions from a set of scales defined in the "Animacy" section of the Godspeed questionnaire (Bartneck et al., 2009), as well as a scale measuring the perceived realism for each part of the character. The results of the questionnaire are shown in fig. 5. As can be seen, there is no significant difference in visual quality as perceived by the participants. Thus, our model delivers comparable quality with a more predictable behaviour and more accurate control, all while reducing the computational cost of the evaluation. A more extensive discussion of this case study and results can be found in the supplementary material.

	Trajectory (cm)		
	Median	Mean	SD
GFNN			
Cubic 3 × 3	5.7	8.8	8.7
Linear 3 × 3	5.0	10.6	11.6
Linear 5 × 5	6.5	11.9	11.5
Linear 7 × 7	6.0	10.3	10.4
Linear 9 × 9	6.2	9.5	9.4
MANN			
8 experts	6.9	9.1	7.9
9 experts	7.7	10.0	7.8
	Speed (cm/s)		
	Median	Mean	SD
GFNN			
Cubic 3 × 3	22.5	32.2	29.9
Linear 3 × 3	23.9	45.0	51.8
Linear 5 × 5	21.1	42.0	48.0
Linear 7 × 7	21.2	39.3	43.9
Linear 9 × 9	21.5	37.0	40.0
MANN			
8 experts	79.0	90.9	66.0
9 experts	47.0	65.3	53.9

Table 6. Error statistics for the evaluated quadruped locomotion models.



(a) Perceived animacy (higher is better)

(b) Perceived realism across body parts (higher is better)

Figure 5. Results of the quadruped animation study. Bar size represents mean value, black lines are 95% confidence intervals.

7. Discussion

We have introduced GFNN as a new kind of neural network architecture, that uses a structured grid of expert parameterisations to learn local patterns that are integrated in a single, continuous model. Our evaluation shows that GFNN can significantly surpass the performance of regular MLP models, with very promising results both in terms of accuracy and inference complexity, and it also offers important benefits over a state-of-the-art model like MANN due to its flexibility and predictability.

As described in our model definition, GFNN can be used to define a grid of arbitrary dimension, making it a particularly flexible architecture adaptable to any domain where the space of the problem can be meaningfully broken down into subspaces. In spite of the complexity introduced by the cubic spline parameter interpolation, we have shown that GFNN can actually be faster to evaluate with the appropriate grid and expert sizes. It also offers different approximate inference methods that permit trade-off between memory and accuracy on the one hand and inference time on the other. This makes it a viable candidate for real-time applications and other resource-constrained contexts.

It is worth noting that GFNN only defines how the parameters of a given neural network model are computed, but it does not restrict the shape of that model. That is, it is entirely possible to build convolutional or recurrent networks that use grid-functioned parameters as described here. Understanding the utility of GFNN in these and other contexts is an open question deserving of its own analysis in future research.

The encouraging results yielded by GFNN in the problems studied here open the door to a broader exploration of the possibilities of this architecture in other domains. Although the origins of the model, as well as the presented case study, lie in the area of computer animation, we believe fields such as spatial reasoning or implicit representation, among others, could take great advantage of our proposal. Future study of these applications will shed more light on the impact of the hyperparameters that GFNN introduces (grid shape and expert size) and the domains for which it holds promise.

Acknowledgements

This work was funded and supported by the Centre for Digital Entertainment at the University of Bath (EPSRC award EP/L016540/1) and Ninja Theory Ltd.

References

Ackley, D. H. *Stochastic Iterated Genetic Hillclimbing*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1987.

Bartneck, C., Kulić, D., Croft, E., and Zoghbi, S. Measurement Instruments for the Anthropomorphism, Animacy, Likeability, Perceived Intelligence, and Perceived Safety of Robots. *International Journal of Social Robotics*, 1(1): 71–81, 2009. ISSN 1875-4805.

Breiman, L. Bagging predictors. *Machine Learning*, 24(2): 123–140, 1996. ISSN 1573-0565.

Catmull, E. and Rom, R. A Class of Local Interpolating Splines. In *Computer Aided Geometric Design*, pp. 317–326. Academic Press, 1974. ISBN 978-0-12-079050-0.

Drucker, H. Improving Regressors using Boosting Techniques. In *Proceedings of the 14th International Conference on Machine Learning (ICML '97)*, pp. 107–115. Morgan Kaufmann, 1997. ISBN 978-1-55860-486-5.

Freund, Y. and Schapire, R. E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55 (1):119–139, 1997. ISSN 0022-0000.

Holden, D., Komura, T., and Saito, J. Phase-Functioned Neural Networks for Character Control. *ACM Transactions on Graphics*, 36(4):42:1–42:13, 2017. ISSN 0730-0301.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, 2015.

Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014.

Liu, R., Lehman, J., Molino, P., Petroski Such, F., Frank, E., Sergeev, A., and Yosinski, J. An intriguing failing of convolutional neural networks and the CoordConv solution. In *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018.

Nair, V. and Hinton, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML '10)*, pp. 807–814. Omnipress, 2010. ISBN 978-1-60558-907-7.

Peng, S., Niemeyer, M., Mescheder, L., Pollefeys, M., and Geiger, A. Convolutional Occupancy Networks. In *Proceedings of the 16th European Conference on Computer Vision (ECCV 2020)*, pp. 523–540, Cham, 2020. Springer International Publishing.

Rosenbrock, H. H. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 1960. ISSN 0010-4620, 1460-2067.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit Neural Representations with Periodic Activation Functions. In *Advances in Neural Information Processing Systems 33*. Curran Associates, Inc., 2020.

Starke, S., Zhang, H., Komura, T., and Saito, J. Neural State Machine for Character-scene Interactions. *ACM Transactions on Graphics*, 38(6):209:1–209:14, 2019. ISSN 0730-0301.

Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., and Fidler, S. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. *arXiv:2101.10994 [cs]*, 2021.

Zeiler, M. D. and Fergus, R. Visualizing and Understanding Convolutional Networks. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T. (eds.), *Proceedings of the 13th European Conference on Computer Vision (ECCV 2014)*, Lecture Notes in Computer Science, pp. 818–833. Springer, 2014. ISBN 978-3-319-10590-1.

Zhang, H., Starke, S., Komura, T., and Saito, J. Mode-Adaptive Neural Networks for Quadruped Motion Control. *ACM Transactions on Graphics*, 37(4):145:1–145:11, 2018. ISSN 0730-0301.