SINN: Shepard Interpolation Neural Networks

Phillip Williams(✉)

Faculty of Engineering, University of Ottawa, Ottawa, Canada
Pwill044@uottawa.ca

Abstract. A novel feed forward Neural Network architecture is proposed based on Shepard Interpolation. Shepard Interpolation is a method for approximating multi-dimensional functions with known coordinate-value pairs [4]. In a Shepard Interpolation Neural Network (SINN), weights and biases are deterministically initiated to non-zero values. Furthermore, Shepard networks maintain a similar accuracy as traditional Neural Networks with a reduction in memory footprint and number of hyper parameters such as number of layers, layer sizes and activation functions. Shepard Interpolation Networks greatly reduce the complexity of Neural Networks, improving performance while maintaining accuracy. The accuracy of Shepard Networks is evaluated on the MNIST digit recognition task. The proposed architecture is compared to LeCun et al. original work on Neural networks [9].

1 Introduction

Artificial Neural Networks are a biologically inspired class of Machine Learning algorithms. They function by simulating an interconnected network of units called "neurons" [1]. The network learns by adjusting the sensitivity of each neuron to its various inputs, allowing the Neural Network to learn various behaviors [1].

However, Neural Networks need not be biologically inspired. In fact, Neural Networks can be modeled as a non-linear transformation of one vector space to another, more specifically transforming a vector of arbitrary size into a different vector of arbitrary size [2].

$$\mathbb{R}^n \rightarrow \mathbb{R}^m$$

where $\mathbb{R}^k$ is the set of real numbers in k dimensions.

In the case of a Neural Network with only one output neuron, the input vector can be thought of as coordinates in an N dimensional space with the output representing the value of a function at that point. The value calculated by a single output Neural Network models a hyper-surface in N dimensions. Thus a network with multiple outputs can be considered as a collection of distinct hyper-surfaces existing in the same vector space.

There are several benefits to using Shepard Interpolation as a basis for Neural Networks. Firstly, Shepard Networks possess only one trainable hidden layer and can be thought of as a shallow network; the approach proposed allows for an arbitrary level of nonlinearities using only one trainable hidden layer, eliminating problems associated

G. Bebis et al. (Eds.): ISVC 2016, Part II, LNCS 10073, pp. 349–358, 2016.
DOI: 10.1007/978-3-319-50832-0_34

with training deep Neural Networks. Shepard Networks are extensible. Each output represents a hyper-surface in n dimensions, while the neurons in the network represent known coordinate-value pairs shared across the output surfaces. Consequently, neurons can be created from particular data points to deform the surfaced over a narrow range of input values and increase the overall accuracy of the model.

2 Related Works

There are several common Neural Network architectures used in research. Often existing architectures are modified and refined rather than novel architectures being proposed. As a consequence of this there is an absence of works relating to Shepard Interpolation applied directly to Neural Networks. A search revealed only one Neural Network paper relating to Shepard Interpolation [3].

2.1 Shepard Convolutional Neural Networks

In the paper by Rimmy SJ Ren [3], a Shepard Layer is proposed as an addition to convolutional Neural Networks [3]. The Shepard Interpolation method is used to augment kernels for tasks such as inpainting. Convolutional Neural Networks have been used for tasks such as filling in missing pixels in images or removing noise from photos; traditional Convolutional Neural Networks however are poorly adapted to certain types of low-level image processing.

The method proposed in the paper by Ren et al. utilises Shepard Interpolation to provide more powerful kernels allowing increased precision when calculating the color of a pixel based on its neighbors.

3 Shepard Interpolation Neural Network

If the outputs of a Neural Networks are imagined as modelling the topography of hyper-surfaces, the problem of initiating Neural Networks, as well as, the problem of choosing hyper-parameters such as activation functions and number of hidden layers can be solved using multivariate analysis techniques.

The technique outlined in this paper uses a method know as Shepard (or inverse weighing) Interpolation to dynamically populate a Neural Networks hidden layer. Shepard Interpolation is a generalisation of Lagrange Polynomials [4]; it allows for the numerical approximation of a multivariate function in N dimensions that passes through a set of known coordinates $(x_1, x_2, \ldots, x_n)$ each having an associated value u [4]:

$$\left\{\left(x_{1,1}, x_{2,1}, \ldots, u_{n,1}\right), \ldots, \left(x_{1,m}, x_{2,m}, \ldots, u_{n,m}\right)\right\} x_{i,j}, u_{i,j} \epsilon \mathbb{R} \tag{1}$$

Shepard Interpolation is geometric in nature. The Interpolation formula represents a deformable surface that passes through a set of known nodes [6]. In the proposed architecture, the deformation of the surface and the positions of the nodes are learned through Gradient Descent.

As an example, the one-dimensional form of Shepard Interpolation is as follows: (Fig. 1)

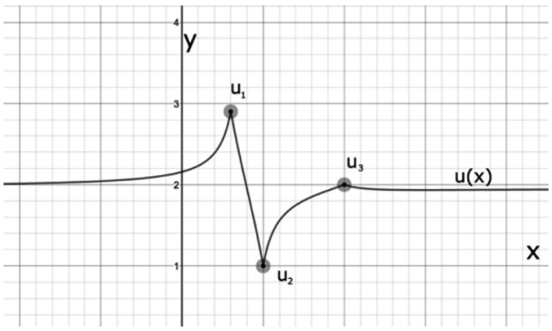

Fig. 1. One-dimensional Shepard Interpolation, the three highlighted points represent the three known data points, the line represents the deformable surface approximating the function. In a Shepard Interpolation Neural Network, the three identified points (u_1, u_2, u_3) would represent the Inverse neurons, with the x and y values representing the input and output to the network respectively.

$$u(x) = \begin{cases} \frac{\sum_{i=1}^{N} w_i(x)u_i}{\sum_{i=1}^{N} w_i(x)} \\ u_i \, if \, d(x, x_i) = 0 \, for \, some \, i \end{cases}. \tag{2}$$

$$w_i(x) = \frac{1}{d(x, y_i)^p}. \tag{3}$$

In the formulation above, the exponent p is a parameter representing the curvature of the interpolation function, while the function $d(x, y)$ is defined as being a mathematical metric [4]. In the physical sense a metric is a function defining the distance between two points, in this case x and y. In the formal definition, a metric is a function satisfying four conditions [5]:

$$d(x, y) \geq 0. \quad \text{Non-negativity} \tag{4}$$

$$d(x, y) = 0 \Leftrightarrow x = y. \quad \text{Identity of indiscernible} \tag{5}$$

$$d(x, y) = d(y, x). \quad \text{Symmetry} \tag{6}$$

$$d(x, y) \leq d(x, z) + d(z, y). \quad \text{Triangle inequality}. \tag{7}$$

In the case of the Shepard Neural Networks, a Parametric Linear Rectifier function is chosen as the distance metric. The equation for the Parametric Linear Rectifier is:

$$\mathrm{P(x, y)} = \max\left(\alpha(x-y),\ -\alpha(x-y)\right). \tag{8}$$

Since the Parametric Linear Rectifier function calculates the difference between the two real inputs x and y, it can be rewritten as a single variable function:

$$P(x, y) = \mathrm{P(x-y)} = \mathrm{P(z)} = \max\left(\alpha z, -\alpha z\right). \tag{9}$$

The expression for the Shepard Interpolation can easily be generalized to higher dimensions [6]. To extend Shepard Interpolation to N dimensions, a distance metric in N dimensions is needed. Since the function P(z) is a metric, a summation P(z) for multiple values of z is also a distance metric. By consequence the multivariate distance function can be written:

$$d(z_1, z_2 \ldots, z_n) = \sum\nolimits_{i=1}^{n} P(z_i). \tag{10}$$

Where $\vec{x} = (x_1, x_2, ..x_n)$ and $\vec{y} = (y_1, y_2, \ldots, y_n)$ are two distinct points in $\mathbb{R}^N$, $\vec{z} = \vec{x} - \vec{y} = (z_1, z_2 \ldots, z_n)$ and $d(z_1, z_2 \ldots, z_n)$ is a measure of the distance between the two points $\vec{x}$ and $\vec{y}$.

The formulation for the general case of Shepard Interpolation can be written:

$$u\left(\vec{z}\right) = \begin{cases} \frac{\sum_{i=1}^{N} w_i\left(\vec{z}\right)u_i}{\sum_{j=1}^{N} w_j\left(\vec{z}\right)} \\ u_i \; if \; d\left(\vec{z}\right) = 0 \, for \; some \; i \end{cases}. \tag{11}$$

$$w_j\left(\vec{z}\right) = \frac{1}{\left[\sum_{i=1}^{n} P(z_i)\right]^p}. \tag{12}$$

3.1 Parametric Linear Rectifier Activation Function

In the context of a Neural Network, an activation function can be defined as:

$$\phi(b + \sum w_i x_i) \tag{13}$$

There is a way to formulate the distance metric $P(x-y)$ as a sub-case of the definition of an activation function. If the Parametric Linear Rectifier activation function is singly connected, in that it only takes one input value, it can be written that:

$$\phi(z) = d(z). \tag{14}$$

It is then obtained from Eq. 9 and the definition of an activation function that:

$$wx + b = x - y. \tag{15}$$

The final activation function is then obtained to be:

$$\phi(\mathrm{z}) = \max\left(\alpha(wx + b), -\alpha(wx + b)\right). \tag{16}$$

By supposing $w = 1$ and $b = -y$, a neuron with a single input can be deterministically created for a known input value y. The neuron has a guaranteed output of 0 when the input is equal to y and a positive output everywhere else. By changing the values of α, b or w, the slope and offset of the output can be tuned.

In the Shepard Interpolation Neural Network architecture, these neurons are named "Metric neurons" and form the first hidden layer of the network. Each Metric neuron calculates a distance between the encoded position and the given input along a single dimension. The Metric neurons are singly connected in their input as well as their output. More specifically, the output of any one Metric neuron is only fed forward to a single node in the following layer.

3.2 Inverse Activation Function

In the Shepard Interpolation Neural Network, information is encoded in coordinate-value pairs. In the previous section, singly connected metric neurons calculate a distance metric between a known point and a given point along a single axis. The next layer in the Shepard Interpolation Neural Network is the Inverse layer, comprised of "Inverse neurons". Each Inverse neuron represents a known point in N dimensions, with each Metric node to which it is connected representing its position along the i^{th} dimension.

In more practical terms, each Inverse neuron represents a w_j term in Eqs. 11 and 12. The activation function of the Inverse neuron is as follows:

$$w_i\left(\bar{z}\right) = \frac{1}{\left[\sum \phi(z_i)\right]^p}. \tag{17}$$

When initiating an Inverse neuron, N Metric neurons are created from the given initiation vector. The Metric neurons are then connected to the Inverse neuron which in turn is added to the network. By creating Inverse neurons, more terms are appended to the Shepard Interpolation formula (Eq. 11) and the overall accuracy of the network is improved as more inflection points are added to the output surface, allowing for a better curve fit of the desired function.

3.3 Normalization and Shepard Layer

The remaining layers in the network are the Normalization Layer and the Shepard layer. Equation 11 can be factored in the following form:

$$u\left(\vec{z}\right) = \sum u_i \frac{w_i(\vec{z})}{\sum_{i=1}^{N} w_i(\vec{z})}. \tag{18}$$

The term $\frac{w_i(\vec{z})}{\sum_{i=1}^{N} w_i(\vec{z})}$ is a normalization of the output of the Inverse layer. By consequence, once the output of the Inverse layer has been calculated, it can be normalized to form $\vec{w'} = (w'_1, w'_2, \ldots, w'_n)$. The Normalization layer represents the normalization operation on the output of the Inverse layer.

The normalized vector $\vec{w'}$ and the Shepard weights $\vec{u} = (u_1, u_2, \ldots, u_n)$ can be substituted into Eq. 14 to form the final activation function of the Shepard Layer:

$$u(\vec{z}) = \sum u_i w'_i. \tag{19}$$

The Shepard neurons take as an input the normalized output of the Inverse layer, and calculate a weighted sum to obtain the final output of the network. The initiation of the Shepard neurons is deterministic, with the values of u_i representing the height of the surface at the known coordinates encoded in the corresponding Inverse neuron.

3.4 Proposed Architecture

Shepard interpolation can be formed by the composition of several activation functions. The architecture of the Neural Network follows from the activation functions above: (Fig. 2)

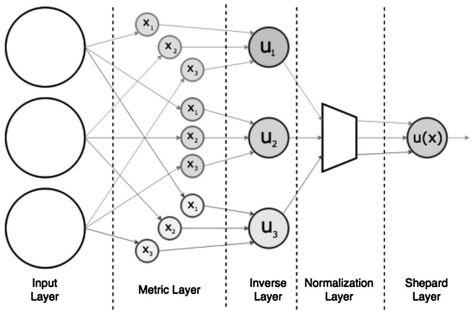

Fig. 2. This diagram represents Fig. 1. Encoded in Shepard Interpolation Neural Network form. Each Inverse node "owns" a set of Metric nodes. After the Normalization layer the Shepard nodes are fully connected to the output of the Inverse layer.

In theory, each term of the Shepard Interpolation represents a known point on a hyper-surface. In order to increase the accuracy of the approximation, more terms can be appended to the existing summation when more data is made available.

Furthermore, the curvature of the interpolation can be adjusted to more accurately represent the given data by learning the α through Gradient Descent or by changing the p hyper parameter in Eqs. 16 and 17.

In practice, each Inverse neuron represents a known coordinate while the weights of each neuron in the Shepard layer represent the value of the hyper-surfaces at the known coordinates. The Metric nodes represent the mathematical metrics used in the interpolation function. The Neural Network learns the topology of the surface in the Shepard layer. More interestingly however is that the Metric nodes each learn their own unique distance function to represent the distance between an input vector and the known "node" in the Shepard surface.

The Neural Network is initiated with all hidden layers' empty. As labelled data is made known, the network can encode data by generating metric neurons and adding weights to the Shepard neurons, in order guarantee the output at a given input (see Sect. 3.6 Distributed Node Initiation). Furthermore, the size of the network can be constrained to a certain size, after which the network can continue increasing its' accuracy through Gradient Descent.

3.5 Gradient Descent in a Shepard Network

The structure of a Shepard Neural Network is summarized by the following equations:

Inverse node:

$$\phi(x) = \max\left(\alpha(wx+b), -\alpha(wx+b)\right) \tag{20}$$

Transfer node:

$$w_i\left(\vec{x}\right) = \frac{1}{[\sum \phi(x_i)]^p} \tag{21}$$

$$\vec{x} = (x_1, x_2, \ldots, x_n) \tag{22}$$

$$p = 2 \tag{23}$$

Normalization node:

$$\vec{w}\,' = \left(\frac{w_1}{\sum w_i}, \frac{w_2}{\sum w_i}, \ldots, \frac{w_n}{\sum w_i}\right) \tag{24}$$

$$\vec{w} = (w_1, w_2, \ldots, w_n) \tag{25}$$

Shepard Node:

$$u(\vec{x}) = \sum u_i * w_i^{'} \tag{26}$$

$$\bar{u} = (u_1, u_2, \ldots, u_n), \textit{Shepard node weights} \tag{27}$$

$$\vec{w'} = (w_1', w_2', \ldots, w_n') \tag{28}$$

The only nodes that update their parameters during Gradient Descent are the Shepard nodes, which update their weights, and the metric nodes, which update the alpha value and the weight value. To update the parameters with Gradient Descent, Back Propagation was used [7].

3.6 Distributed Node Initiation

During experimentation, it was empirically discovered that initial performance was best when the initial nodes were initiated with a variety of different values across the outputs of the network. In essence, when neurons are initiated, the encoded data point represents either a positive or negative classification for the output classes. When the network is initiated with little variety of output classifications, the initial performance is very poor. To compensate for this, the hidden layer is populated by neurons in such a manner that each class receives an equal variety of data, allowing for a more accurate generalization.

4 Experiments

The network architecture is evaluated by analyzing accuracy and efficiency. The accuracy is measured by performing optical character recognition on the MNIST dataset. The efficiency is analyzed by comparing the total number of learned parameters in the Shepard network versus traditional fully connected Neural Networks.

The MNIST dataset is a collection of 60,000 training images and 10,000 validation images. The images are provided as a list of vectors of dimensionality 785. The first element of the vector represents the digit from 0 to 9 in the image while the last 784 elements represent the pixel values of the image of the digit in question. The images are represented by a 28×28 matrix of integers from 0 to 255, representing a grayscale photo of handwritten digits [8].

4.1 Experiment Setup

The Neural Networks were trained using Batch Gradient Descent, Gradient Momentum as well as Learning Rate Annealing. After each epoch of training the accuracy of the model was validated on the validation dataset. If the accuracy had decreased from the previous epoch, the learning rates were multiplied by a constant value smaller than one, to allow for smaller changes to the model in following epochs, otherwise the learning rates remained constant.

All neurons were initiated before training using the Distributed Node Initiation. The training of the Neural Networks was automated using the Nelder-Mead algorithm.

4.2 Simplex Search to Optimize Performance

The Nelder-Mead algorithm (or simplex search) is a derivative-free multivariate optimization algorithm [9]. A Neural Network can be imagined as a multivariate function with the input parameters being the learning rates, with the output being the accuracy of the classification on the MNIST validation set.

The Nelder-Mead search was used to automatically identify optimal learning rates, allowing for more extensive experimentation.

The Nelder-Mead algorithm functions by exploiting the concept of a simplex in N dimensions. Each vertex represents the set of input parameters and the corresponding output. The algorithm will subsequently modify the coordinates by a multiplicity of operations including reflexion, contraction and expansion. The result is that the simplex iteratively makes its way up the surface of the function until it finds a maximum value, in this case the maximum accuracy attainable by the Neural Network.

4.3 Results

The Shepard Interpolation Neural Network attained an accuracy of 95.0% for classification on the MNIST dataset using only 75 nodes. Each of the nodes with n inputs has $2n$ learnable parameters while the output nodes have m learnable parameters where m is the number of neurons in the hidden layer. The total number of tunable parameters in this Neural Network is thus:

$$output * hidden + hidden * (2 * input + 1) = 118\,425\, parameters$$

While the total number of tunable parameters in a fully connected Neural Network of comparable accuracy is $n + 1$ for each neuron, where n is the number of inputs to the neuron. To calculate the total number of parameters for a 300 Neuron Neural network is:

$$(n + 1) * hidden + (hidden + 1) * output = 238\,510\, parameters$$

By simply using a different architecture, the memory footprint of the Neural Network is reduced by 50% for the same accuracy. On top of this, the vast majority of activation functions in the network are Parametric Rectifiers, which are many times less expensive than the common Sigmoid activation function [10]. Furthermore, the different modifications to the Neural Network each boost its overall accuracy.

Experimental results for Shepard Interpolation Neural Networks

Neural network	MNIST accuracy
Initiated all nodes on single digit	8–10%
Random initiation	∼40%
Distributed initiation	63.0%
Gradient descent+Distributed Initiation	94.91%
Gradient descent+Softmax+Distributed initiation	95.0%
500 neuron SINN	96.19%

5 Conclusion

Shepard Interpolation combined with intelligent node initiation can achieve similar accuracy as conventional fully connected Neural Networks with a significant reduction of memory footprint and overall computational costs.

The Shepard architecture is compatible with all of the current Neural Network methodologies while providing the possibility of more compact and efficient learning models.

References

1. Haykin, S.: Neural networks: a comprehensive foundation. Neural Netw. **2**(2004) (2004)
2. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks **2**(5), 359–366 (1989)
3. Ren, J.S.J., et al.: Shepard convolutional neural networks. In: Advances in Neural Information Processing Systems (2015)
4. Shepard, D.: A two-dimensional interpolation function for irregularly-spaced data. In: Proceedings of the 1968 23rd ACM National Conference. ACM (1968)
5. "Metric (mathematics)": Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc., Web Date accessed 10 August 2016. https://en.wikipedia.org/wiki/Metric_(mathematics). Date last updated (21 August 2016)
6. Gordon, W.J., Wixom, J.A.: Shepard's method of "metric interpolation" to bivariate and multivariate interpolation. Math. Comput. **32**(141), 253–264 (1978)
7. Nielsen, M.A.: Neural Networks and Deep Learning. Determination Press (2015)
8. LeCun, Y., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86** (11), 2278–2324 (1998)
9. Singer, S., Nelder, J.: Nelder-Mead algorithm. Scholarpedia **4**(7), 2928 (2009)
10. Glorot, X., Antoine, B., Yoshua, B.: Deep sparse rectifier neural networks. Aistats **15**(106), 275 (2011)

View-Based 3D Objects Recognition with Expectation Propagation Learning

Adrien Bertrand[1], Faisal R. Al-Osaimi[2], and Nizar Bouguila[1(✉)]

[1] Concordia University, Montreal, QC, Canada
{ad_bert,nizar.bouguila}@concordia.ca
[2] Department of Computer Engineering, College of Computer Systems, Umm Al-Qura University, Makkah, Saudi Arabia
frosaimi@uqu.edu.sa

Abstract. In this paper, we develop an expectation propagation learning framework for the inverted Dirichlet (ID) and Dirichlet mixture models. The main goal is to implement an algorithm to recognize 3D objects. Those objects are in our case from a view-based 3D models database that we have assembled. Following specific rules determined by analyzing the results of our tests, we have been able to get promising recognition rates. Experimental results are presented with different object classes by comparing recognition rates and confidence levels according to different tuning parameters.

1 Introduction

For quite some time, creating systems being able to detect and recognize (classify) objects, has been a very popular subject of research, as it goes well along with many fields such as computer vision and pattern recognition. It is even more the case nowadays thanks to a great increase in computing power. There are several types of probabilistic classifiers often used. Such classifiers are capable to predict with a certain probability the class to which a given object should belong. Mixture models, in particular, have been widely used, are efficient, and attractive in terms of ease of implementation and flexibility. Their success comes from their effectiveness in modeling large classes of natural measurements using a small set of parameters. An important step when considering mixture models is the choice of the per-components distributions. The Gaussian distribution has been widely adopted [1]. But, it has been shown to be limited in several real-life applications. Once a distribution is chosen according to the nature of data in hand, the next step is the learning of parameters form the available data. There are several learning frameworks available, although the most common one is expectation-maximization (EM). Despite the fact that it is very sensitive to initialization, it is an easily implementable approach and generally provides acceptable results. However, in the recent years, there has been an upsurge of research done towards more accurate mixture models for real-life applications and learning frameworks that may be more adapted for them. A recent trend is to use finite Dirichlet-based mixtures [2–4]. Recent studies have indeed shown

G. Bebis et al. (Eds.): ISVC 2016, Part II, LNCS 10073, pp. 359–369, 2016.
DOI: 10.1007/978-3-319-50832-0_35