THE INTROSPECTION OF DEEP NEURAL NETWORKS WITHIN A PARAMETRIC MODELLING ENVIRONMENT

Towards Illuminating the Black Box

N. KHEAN,
University of New South Wales, Sydney, Australia
nariddh97@hotmail.com

Abstract. Machine learning has yet to make a significant impact in the field of architecture and design. However, with the combination of artificial neural networks, a biologically inspired machine learning paradigm, and deep learning, a hierarchical subsystem of machine learning, the predictive capabilities of machine learning processes could prove a valuable tool for designers. This potential is the driving factor for many machine learning frameworks within design platforms. However, they fail to address the complexity of machine learning, as well as the inherent knowledge gap between the fields of architecture and computer science. This research proposes a method to lessen that gap, through the creation of a teaching tool, made specifically for architects and designers. Within a parametric modelling environment, this research develops a framework to express the mathematic and programmatic operations of neural networks in a visual scripting language. It will be developed within Grasshopper, and made almost completely of basic Grasshopper components. Every operation within neural networks will be segmented into their most basic expressions and parameterised. This provides an intermediary between machine learning and design, which will hopefully facilitate a greater presence of artificial intelligence within architecture.

Keywords. Machine learning, neural network, framework, supervised learning, parametric modelling environment

1. Introduction: Research Motivations

Machine learning (ML), a branch of artificial intelligence, has been the driving factor for hundreds of computational processes in almost every large company (Biewald, 2016). However, the relevance of ML spreads far beyond that of the major corporations. Transportation (autonomous vehicles and aircraft autopilots), communication (spam filters and email prioritization), and education (plagiarism detection and optical character recognition) are instances where ML is already a significant aspect of mainstream operations (Narula, 2017).

The study of ML already has a long and exciting history (Sardina, 2017). Yet, only recently has the topic become a reasonably mature area of computer science (Sardina, 2017). Among other factors, the increase of speed and efficiency in parallel computing (Kelly, 2014), and the rising accessibility of big data (Kelly, 2014), has provided the perfect foundation for innovation in the field of ML.

However, considering the advancements and successes of ML algorithms, why isn't there a greater presence of ML within design? Only sparingly have ML algorithms been applied in the field of architecture (Phelan, 2016), and where they have held no influence over the design process itself. Yet, ML algorithms have the potential for resolving prediction problems, categorising vast quantities of data, and modelling for optimisation, and this suggests reasonable credence for ML applications in architecture.

Accordingly, this research examines the current ML frameworks within the parametric modelling environment, Grasshopper. This research aims to develop an original framework, made specifically as a teaching tool for newcomers in the field of ML. By utilising an action research methodology, an ML framework will be developed iteratively, with a continuing goal of maintaining an introspective quality, for maximum transparency and increased effectiveness as a teaching tool.

2. Research Aims

The aims of this research are twofold: to develop an ML framework within the parametric modelling environment, Grasshopper; and to do so in a way that is completely transparent for those attempting to comprehend their inner workings.

The overarching goal is to contribute to a greater presence of ML directly in the design process. As such, the initial objective is to simply create a framework, within an environment used by computational designers, and potentially architects. A framework can be described as an abstraction of a computational process, which can be altered to suit a multitude of applications. Thus, the original intention is to develop a tool for designers capable of ML.

The second objective is to combat the pre-existing tool's opaque, internal operations. The current frameworks that exist within Grasshopper, are enclosed components, offering little to no introspective quality. As such, unless the computational designer is also trained in ML, it is difficult to yield sensical results. This research aims to remedy this through the development of a framework that is completely transparent. To newcomers in the field of ML, the combination of the framework's transparency, and the fact that it is built within an environment that they are familiar with, enhances the potential for their understanding. The outcome is an intermediary between ML and design.

3. Research Questions

At the commencement of this research, there are several questions needing exploration. Initially, investigations pertaining to the multitude of ML algorithms and their functions are needing assessment to determine the most appropriate to begin with. What ML algorithms exist, and what are their individual functions and purpose? Furthermore, the question of their functionality leads to studies of the underlying mathematics. How does the algorithm work? And how can one translate these mathematics to computational logic?

Subsequently, this highlights that beyond simply understanding, is the need to effectually convey heuristics through computational operations. How can one explicate information effectively, specifically to a design audience? Using an environment familiar to computational designers, and potentially architects, how is clarity achieved though the visual scripting language?

Furthermore, the iterative research process generates questions about the research process itself. By what method can the developed framework be proved to be accurate and functional? How can one assess the effectiveness of the framework as a teaching tool for computational designers? And to what extent does the framework deliver on clarity and transparency for the algorithm's internal operations?

From these numerous queries, the research project has narrowed the exploration to the following key questions:

1. What knowledge is required to understand the operations of ML algorithms?
2. What strategies can be utilised to effectively teach computational designers about the function of ML algorithms?
3. How can the key processes be implemented within a computational design environment with the utmost transparency?

4. Methodology

The methodology of developing an ML framework within Grasshopper adopts an action research approach that is characterised by iterative progression. “Action research has been described as a reflective and experimental model of inquiry” (MacIsaac, 1996). This cycle, through the conceptualisation of a problem, the action towards its resolution, and an evaluation of that action, drives this research.

This project is further divided into three sections: theoretical research, implementation, and validation.

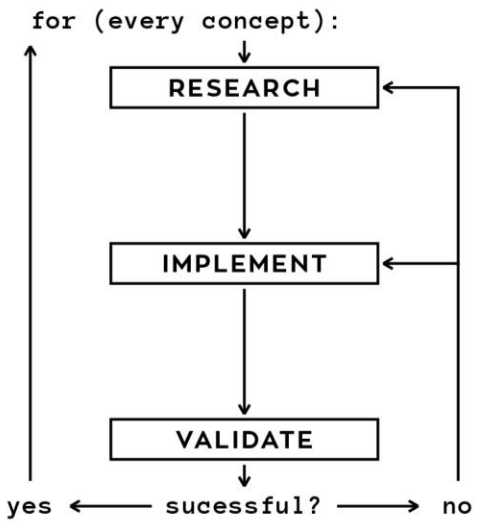

Figure 1. Flowchart of research methodology.

A holistic understanding of the fundamental mathematics of ML algorithms is obtained during the theoretical research phase. This is further segmented into two categories: basic operation, and optimisation and regularisation techniques. The comprehension of the basic operations is imperative as they are the key processes that everything else is built upon. On the other hand, the research of optimisation and regularisation techniques is almost optional. By analysing advancements in the field, these techniques will be judged on their potential to be additionally included in the proposed framework.

Gathering the algebraic rationale behind each concept, the next stage was their implementation. This involves the translation of mathematics into the visual scripting logic of Grasshopper. Furthermore, it is crucial to maintain a connection between the hyperparameters; user-controlled variables that dictate the architecture of the network. This allows for parametrisation, further enhancing the framework's clarity and effectiveness as a teaching tool. Several small tests will then be conducted to assess the robustness of these hyperparameters, and limitations will be set at their functional extents.

Finally, larger tests are needed, following the inclusion of each concept, to validate the accuracy of the implementation. This is achieved through multiple comparisons of error graphs, produced during simulated training. The comparison will be conducted between the Grasshopper framework and a reputable C#-based framework. If the evaluation yields inconsistencies, the research would revaluate the failures of the two prior stages and reiterate.

After the successful implementation of a concept into the Grasshopper environment, the three-stage process restarts. This methodology repeats for every technique researched.

5. Background Research

Architectural historian, Mario Carpo, captured the progression of the first digital turn in architecture (Carpo, 2013). However, almost 25 years since the start of that revolution, Carpo observes a whole new design industry, with unprecedented computational power, favouring "a new kind of science" (Carpo, 2017). Carpo predicts that this second digital turn in architecture is completely different than the first. "The trend as we see it is asymptotically approaching a state of an almost infinite amount of data, recorded, transmitted, and retrieved at almost no cost" (Carpo, 2016). As we approach this limit, computers can perform more and more operations, which can be compared against one another to find the optimal outcome. This, almost trial-and-error approach can eventuate in intuitions about performance, and perhaps recognition of the different patterns that develop through infinite iteration. “Using advanced computation, massive trial-and-error becomes a viable computational strategy. In fact, that’s the best computational strategy, because that’s the only thing computers really do.” (Carpo, 2016). This extraction of intuition and pattern recognition through massive trial-and-error is a computational metaheuristic known as a genetic algorithm, a form of ML. “Designers have been toying with machine thinking and ML for some time” (Carpo, 2017), yet, “toying” is all that is currently happening. However, that is slowly changing.

In a series about ML, Grant Sanderson discusses gradient descent, and how, specifically, neural networks learn (Sanderson, 2017). Artificial neural networks are a biologically inspired ML paradigm, capable of solving complex signal processing or pattern recognition problems. At the start of the series, Sanderson introduces neural networks as the first ML algorithm that beginners should understand. He admits that “[neural networks] are old technology, the kind researched in the eighties and nineties. But you do need to understand it before you can understand more detailed modern variants, and it is clearly capable of solving some interesting problems” (Sanderson, 2017). Consequently, the first algorithm to be implemented within the Grasshopper framework is the neural network.

There have been several instances wherein neural networks were implemented as predictive tools in the field of architectural construction. In 2009, Van Truong Luu and Soo Yong Kim published their paper, *Neural Network Model for Construction Cost Prediction of Apartment Projects in Vietnam* (Luu and Kim, 2009). “Although the proposed model is not validated in a rigorous way,” Luu writes, “the artificial neural network-based model is useful for both practitioners and researchers.” Later, in 2011, Ismaail ElSawy, Hossam Hosny, and Mohammed Adbel Razek published their research, *A Neural Network Model for Construction Projects Site Overhead Cost Estimating in Egypt* (Elsawy et al., 2011). The paper concluded that “an artificial neural network-based model would be a suitable tool”. These are only two instances of research, focused specifically on the application of neural networks, after the design process. However, neural networks have yet to make a significant impact within the design process itself.

In more recent years, there have been attempts to combat this and introduce ML algorithms into design platforms. Looking specifically at the visual programming extension, Grasshopper, an addon for the 3D computer-aided design (CAD) software, Rhino, there have been a total of four plugins that introduce such algorithms into the environment. In 2015, Lorenzo Greco published the first of which, called *Dodo* (Greco, 2015), capable of non-linear and constrained optimisation, and artificial neural networks. In later years, *Crow* (Felbrich, 2016), *Owl* (Zwierzycki, 2017), and *Lunchbox* (Miller, 2017) were released, all capable of various forms of ML. However, as these plugins are within an environment used by computational designers and architects, and unless the user possesses sufficient knowledge in these specific algorithms, it would be incredibly difficult to yield anything more than nonsensical outputs from simply adjusting the input parameters. Furthermore, due to the nature of all plugins, the underlying operations that drive these algorithms are hidden behind sealed components, which produce yet another layer of abstraction between user and comprehension. Finally, the plugins themselves are built from C#-based libraries, namely *AForge.Net* (Kirillov, 2012), the discontinued *NeuronDotNet* (Dinesha, 2013), and *Accord.Net* (Souza et al., 2014), which additional limits the introduction and facilitation of ML processes to designers. This evaluation of the current ML frameworks within Grasshopper was the driving factor for this research.

With the overarching objective of developing a neural network framework from base principles, it was necessary to obtain a comprehensive understanding of the underlying mathematics. In 2015, researcher, Michael Nielsen, published *Neural Networks and Deep Learning*, a “principle-oriented” book, which aimed to convey the core concepts of neural networks, rather than specific coding libraries. Nielsen argues that “while [libraries have] an immediate problem-solving payoff, if you want to understand what’s really going on in neural networks, if you want insights that will still be relevant in years from now, then it’s not enough just to learn some hot library” (Nielsen, 2015). Throughout the book, Nielsen structured his explanations by first verbalising the problem, then progressively reasoning through the mathematics, and finally applying this knowledge upon a common ML problem: handwritten digit recognition. He does so in an interactive manner, through the addition of optional reader activities, and several JavaScript elements that effectively visualise his explanations, only possible, due to the online medium. Having obtained an understanding of the two procedures that allow for the neural network to learn, feedforward and backpropagation, further exploration of how the human can better facilitate that learning ensued.

Through the investigation of optimisation and regularisation strategies, this research utilised the University of Stanford’s Computer Sciences resource, *Convolutional Neural Networks for Visual Recognition* (Karpathy, 2014), as a yardstick for current research within the field of neural networks. This lead to the consideration of the 2014 paper, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* (Srivastava et al., 2014), and the 2005 paper, *Maximum-Margin Matrix Factorisation* (Srebro et al., 2005), as a few techniques which could be implemented within the framework to better facilitate ML.

6. Case Study

This research initially pursued an application-based approach. In collaboration with Architectus™, an architectural firm based in Australia and New Zealand, the objective was to develop a neural network model that could predict the outcomes of current projects. After considerable experimentation, it was found that the quantity of training data paled in comparison to the quantities needed for successful neural networks, forcing a change in direction.

Instead of applying neural networks to a specific problem, this research explored the potential for creating a general ML tool in a computational design environment. After a brief study of the existing tools, it was discovered that this concept had already been developed. Yet, even with these tools, there was still a minuscule presence of ML within design. This lead to the final objective of this research: to create a neural network framework in a design environment, that poses as a teaching resource, to inform computational designers and architects about ML algorithms.

6.1. BASIC OPERATIONS

Neural networks function through the combination of two processes: feedforward and backpropagation. This research explores these procedures, as well as their parametric implementation as a Grasshopper definition.

6.1.1. Feedforward

In neural networks, feedforward describes the process of passing input values, through a hierarchical layering of neurons, to produce an output in the final layer. There are two operations within a neuron: the calculation of a value known as the net, and the activation of that net through a nonlinearity known as an activation function. Ignoring the latter for now, the net calculation can be defined as a weighted sum of inputs, in addition to a bias.

$$net = \sum(inputs \times weights) + bias \tag{1A}$$

$$z_i^l = \sum_j a_j^{l-1} \cdot w_{ij}^l + b_i^l \tag{1B}$$

These weights and biases are values that change over the learning process, and as such are known as learned parameters. Below is the net calculation, as defined in the Grasshopper environment.

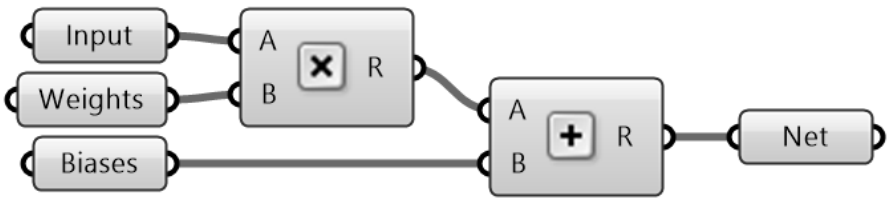

Figure 2. Net calculation of a neuron assuming a singular input.

However, the above definition (figure 2) only contains the calculations within a single neuron. Neural networks are typically made of several layers, containing tens, and at times, hundreds of neurons in each layer. It would be ridiculous to ask the user to repeat this definition for every neuron within the network. So, for the development of a framework that is to be robust enough for any network configuration, this definition had to be reconsidered.

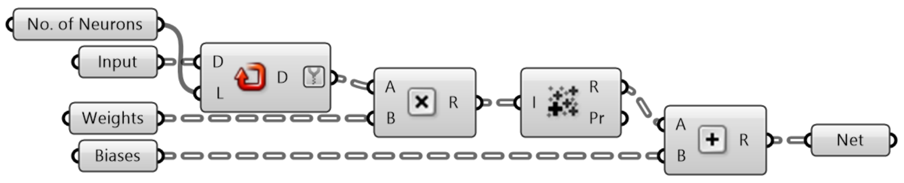

Figure 3. Layer-based net calculation.

Instead of calculating the procedures of a single neuron, the definition in figure 3 utilises data trees to represent every neuron within the layer. However, for every neuron, there is a need replicate each input. To facilitate this repetition, the inclusion of a hyperparameter, a user-controlled variable, dictates the number of times the previous activations are replicated. Testing this definition with simulated data trees proved its ability to successfully handle the net calculations within an entire layer; from a single neuron, to an infinite amount.

The final operation that occurs within a neuron is the activation of the net. This involves passing the net through a nonlinearity known as an activation function. Their purpose is to allow the network to approximate more complex outcomes. There are several types of activation functions, however, this research uses the Sigmoid and ReLU.

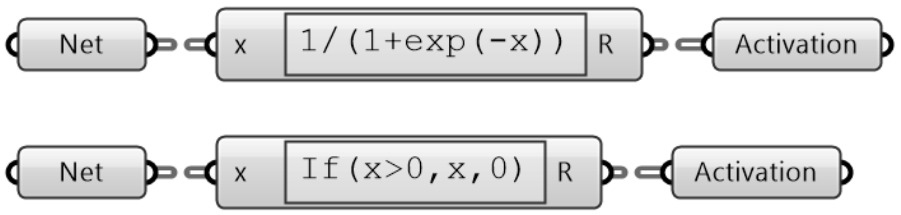

Figure 4. From top to bottom, the Sigmoid and ReLU activation functions.

Adding the activation function component to the net will finalise the operations in feedforward (figure 5).

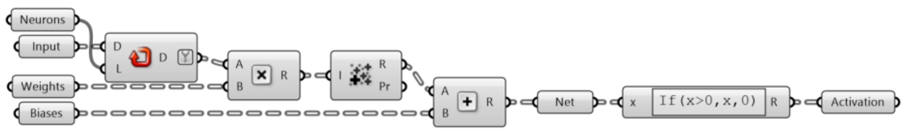

Figure 5. Activation function is applied after the net calculations.

6.1.2. Backpropagation

Executing the process of feedforward alone does not constitute any ML. Actual learning in neural networks happen through an assessment of performance, leading to an adjustment of the learned parameters, in an attempt to improve the output of feedforward. This is known as backpropagation.

Backpropagation is the process by which the individual error contribution of each neuron is calculated and passed backwards through the network. The weights and biases connected to those neurons are adjusted proportional to their error contribution. This adjustment of learned parameters is how the machine learns.

The error contribution for the output neurons is dependent upon the activation function used in that layer, however, below is the generalised form.

$$output\ error\ contribution = (output - target) \times activation\ derivative\ of\ net \tag{2A}$$

$$\delta_i^L = \left(a_i^L - y_i\right) \odot \sigma'\left(z_i^L\right) \tag{2B}$$

Assuming that the output layer is using the Sigmoid activation function, below is the Grasshopper definition for calculating the individual error contribution of the output neurons.

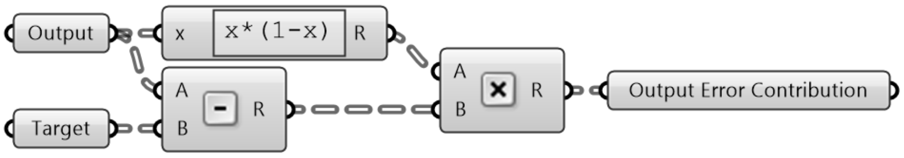

Figure 6. The output error contribution with a Sigmoid layer.

The adjustment of the learned parameters requires the introduction of another hyperparameter, the learning rate. The learning rate determines the scale of the adjustment. Below are the mathematics behind updating the weights and biases of a neural network, as well as the corresponding Grasshopper definition.

$$updated\ weight\ =\ previous\ weight - learning\ rate \times error\ contribution \times output\ of\ previous\ layer \tag{3A}$$

$$^{+}w_{ij}^L = w_{ij}^L - \eta \cdot \delta_i^L \cdot a_j^{L-1} \tag{3B}$$

$$updated\ bias = previous\ bias - learning\ rate \times error\ contribution \tag{4A}$$

$$^{+}b_i^L = b_i^L - \eta \cdot \delta_i^L \tag{4B}$$

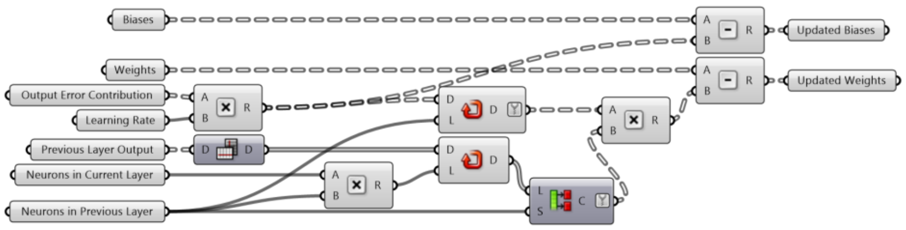

Figure 7. Updating the weights and biases.

6.1.3. Network Error Calculation

The error, also known as the cost or the loss, is a value attributed to the performance of the entire network. It is determined by the network prediction against the target output. This target output is attached to the training data, which is why neural networks are a supervised ML algorithm. The lower the error, the more accurately the network is modelling the training data.

It is important to note that the total error differs from the individual error contribution of neurons. The individual error contribution is a judgement on each neuron's performance, whereas the total network error is a less nuanced indication of how the entire network is performing.

There are several different error functions, however this study exclusively uses the mean squared error function, also known as the quadratic cost function.

$$error = \frac{1}{2}\sum(target - output)^2 \tag{5A}$$

$$E = \frac{1}{2n}\sum(y - a^L)^2 \tag{5B}$$

Below is the implementation of the mean squared error function within Grasshopper.

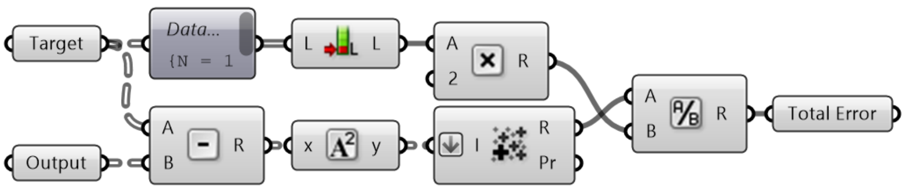

Figure 8. Mean squared error function.

6.1.4. Deep Learning

“The earliest artificial neural networks lacked hidden layers”, writes Frank Wilczek. “Their output was, therefore, a relatively simple function of their input” (Wilczek, 2011). Hidden layers introduce the concept of deep learning to neural networks, by simply stacking the process of feedforward.

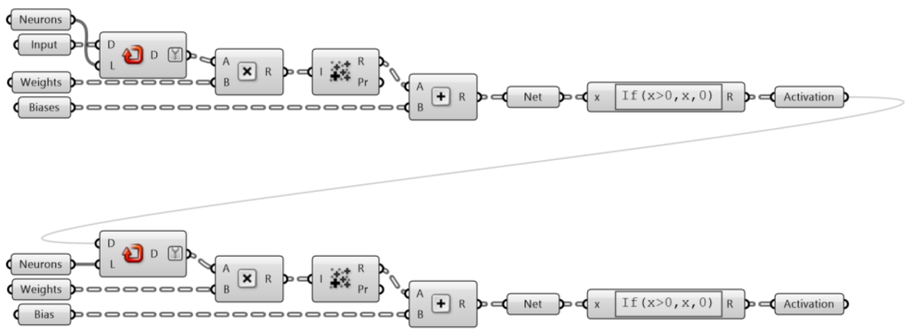

Figure 9. Stacking two feedforward operations to facilitate deep learning.

However, the process of backpropagation becomes slightly more complex. This is because there are no explicit target outputs for the hidden layers. As such, the hidden layers rely on the error of the layer after it, in conjunction to their dependence on the activation function.

$$hidden\ error\ contribution = \sum(error\ contribution\ of\ net\ layer \times weights) \times activation\ derivative\ of\ net \qquad (6A)$$

$$\delta_j^l = \sum_i \left(\left(w_{ij}^{l+1}\right)^T \cdot \delta_i^{l+1} \right) \odot \sigma'\left(z_j^l\right) \qquad (6B)$$

Below is the Grasshopper definition for the calculation of a hidden layer’s error contribution, assuming that the activation function used in the layer is the ReLU activation.

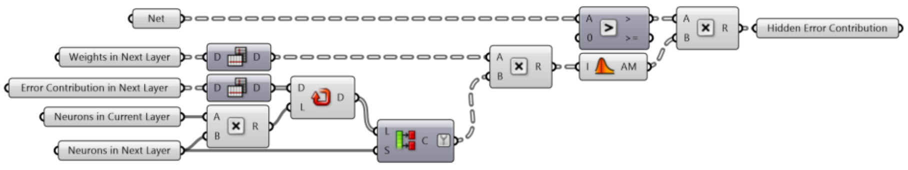

Figure 10. The hidden error contribution with a ReLU layer.

6.1.5. Batch and Mini-Batch Gradient Descent

Thus far, every Grasshopper definition was created under the assumption that the input neurons would contain only one value. This is known as stochastic gradient descent: teaching the neural network by showing it one training data point at a time, and letting it adjust its learned parameters after each data point.

Stochastic gradient descent is a great starting point from an educational perspective, as it is the simplest to understand and implement. However, in practice, there are several downsides. Calculating the error of a single data point, which could be an outlier or noise, will cause unwanted adjustments.

A method used to combat this is known as mini-batch gradient descent. Instead of inputting a single data point at a time, mini-batch learning involves splitting up the pool of training data into a determined number of mini-batches, and training the network upon every data point within the batch. If the number of mini-batches is set to one, the network will calculate the error of all training examples at once, which is known as full-batch learning.

Full-batch gradient descent has the reputation of being an accurate, but slow learning method. With mini-batch learning, however, a degree of accuracy is maintained through the averaging of multiple data points, alongside the benefits associated with stochastic learning, such as the speed and some friendly noise.

By providing the user control over the number of batches, this research developed a Grasshopper definition that is robust enough to handle both stochastic, and full-batch gradient descent, as well as any number of mini-batches in between.

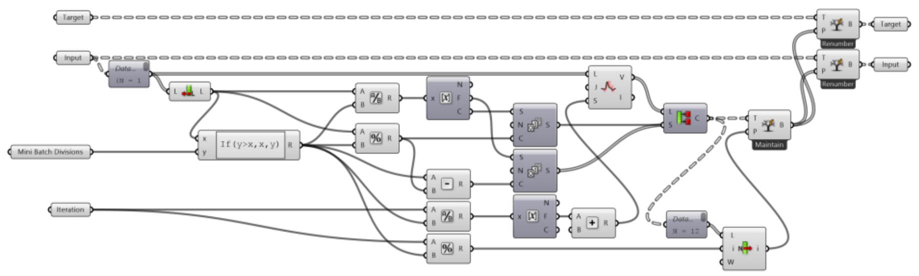

Figure 11. Division of input and target data points according to mini batch hyperparameter.

However, all the previous definitions only operate with stochastic gradient descent. As such, a revision of each component in the framework was needed.

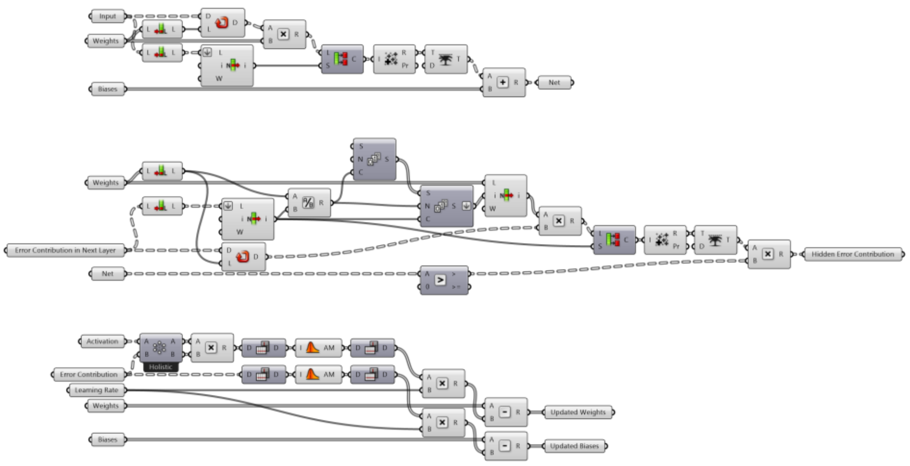

Figure 12. From top to bottom, revised definitions for the net calculations, the hidden error contribution, and the updating of weights and biases.

6.2. OPTIMISATION AND REGULARISATION TECHNIQUES

Deep learning is a great technique to better the complexity and capabilities of neural networks. However, it also increases the potential for a model to overfit. Overfitting is a statistical concept whereby a model, such as a neural network, mimics the training data too closely, and fails to derive the trend. Overfitting is detrimental as the neural network's accuracy will slowly deteriorate the moment overfitting starts.

Below are the four optimisation and regularisation methods implemented within the developed framework, to better the process by which the machine learns.

6.2.1. Exponential Learning Rate Decay

Using too small a learning rate will cause the network to take a ridiculous amount of time to train, however, using a learning rate too large will cause the gradient descent algorithm to struggle to settle in a local minimum.

A method to circumnavigate the effort needed to find the perfect learning rate is to decay the learning rate over time. This allows for rapid adjustments at the start of learning, which then changes to a more refined search later on.

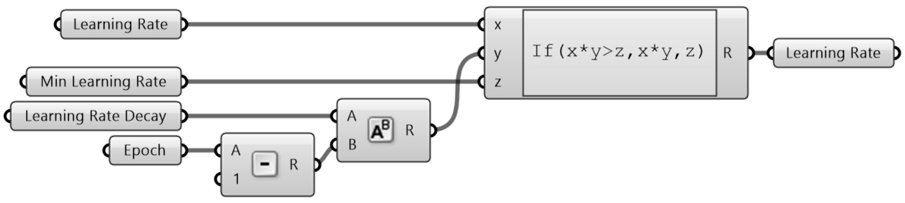

Figure 13. Exponentially decaying the learning rate.

6.2.2. Momentum Learning

Gradient descent alone can become a slow process. Especially when nearing a minimum, the gradients will inevitably decrease in magnitude, subsequently lowering the parameter adjustment. A technique used to speed up similar consecutive adjustments is known as momentum learning. Momentum learning stores the previous adjustments to the weights and biases, to slightly influence the current adjustment. This speeds up consecutive, similar adjustments, but also causes the adjustments to overshoot the optima.

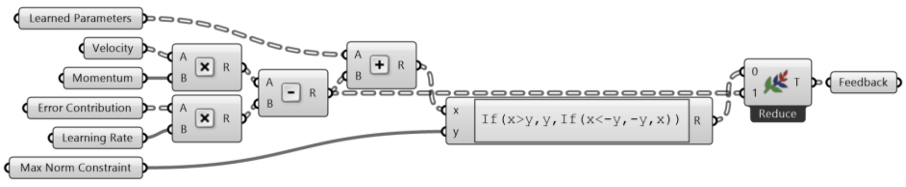

Figure 14. Momentum learning and max norm constraint.

6.2.3. Max Norm Constraint

On occasion, if the learning rate is too high, an update to the learned parameter has the potential to leap to an incredibly large magnitude. A form of regularisation is to enforce an absolute upper bound on these parameters. This technique is called the max norm constraint (Srebro et al., 2005).

6.2.4. Dropout

The larger a neural network, the more likely the network will overfit. A method to overcome this is through bootstrap aggregating (Breiman, 1994). This involves training several networks, and averaging the learned parameters. However, as this is mostly needed for larger networks, this is a computationally taxing procedure. A technique for addressing this problem is called dropout (Srivastava et al., 2014).

Dropout involves the random selection of a percentage of neurons within a layer, and setting their activation to zero for this training cycle. The random selection changes every iteration. "This prevents units from co-adapting too much" (Srivastava et al., 2014).

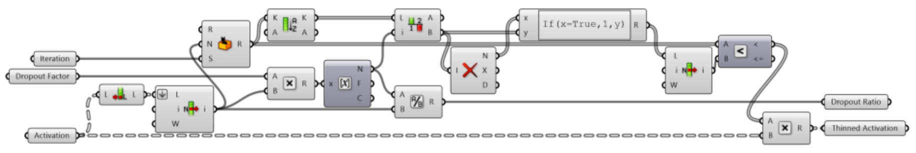

Figure 15. Dropout applied after every activation.

6.3. DEVELOPED FRAMEWORK

Combining the definitions of feedforward and backpropagation, as well as the aforementioned optimisation and regularisation techniques, all connected with a mechanism to overcome Grasshopper's recursive loop avoidance check, produces the deep neural network framework.

Figure 16. Final neural network framework (see Appendix B for a higher resolution).

7. Significance of Research

ML algorithms have proven to be incredibly effective at problem-solving in numerous fields of research and industry. They have been at the forefront of innovations such as image recognition, feedback control of actuated prosthetics, and cancerous cell classification. The successes of these advancements have inspired the use of ML algorithms on the peripheries of architecture and design.

The utilisation of neural networks as a model to predict the construction cost of apartment buildings (Luu and Kim, 2009), is an instance wherein ML algorithms were successfully implemented in the field of architectural construction. Looking at the design process itself, there have been projects that produced ML plugins inside parametric modelling environments, in the hopes of a greater ML presence within design.

This research has added an alternative pathway for the implementation of ML algorithms in a design environment. Instead of focusing on a specific application-based problem, the research has developed a framework which is flexible to a multitude of applications. This has the potential to increase the occurrence of smarter, more efficient design, prompted by artificial intelligences in the future of architecture.

In parallel, the research outcomes were developed as a teaching tool aimed at computational designers and architects. The entirely transparent nature of the framework provides for an enquiry-based learning approach (Hutchings, 2006). Newcomers to ML, who have prior knowledge of the Grasshopper environment, can utilise this tool to comprehend the inner workings of neural networks in a hands-on fashion.

Lastly, there have been a substantial quantity of research papers developing methods to improve how neural networks learn. Over time, these methods can be easily added to the framework, due to its unambiguous and robust development. This has the potential to not only enhance the process by which the machine learns, but also add to the volume of neural network content available for designers to learn.

8. Evaluation of Research Project

The aim of this research was to develop an ML framework, within a parametric modelling environment, in a completely transparent and unambiguous fashion. The framework is a method to facilitate a greater ML presence within design, by informing computational designers and architects about the operations within ML algorithms.

The goal of developing an ML framework, that includes a multitude of algorithms created from their base mathematical principles, was slightly beyond the scope of this research. The developed outcome was comprised of only one ML algorithm: the neural network. However, “you need to understand [neural networks] before you can understand more detailed modern variants” (Sanderson, 2017). Given further time, other ML algorithms, such as convolutional neural networks, or recurrent neural networks, could be implemented, furthering the framework’s potential for application, as well as its breadth for education.

Retrospectively evaluating the outcome reveals its robustness. A user can effectively apply the framework, and expect a reliable result. However, in comparison to existing tools, such as Owl (Zwierzycki, 2017) or Dodo (Greco, 2015), the framework will inherently underperform in terms of speed. Yet, the framework offers something that no plugin can: introspection.

The framework was created with clarity and transparency in mind. Everything within a neural network was broken into their smallest operation, and explicitly constructed within the Grasshopper environment. Further, every concept was divided into separate, colour-coded groups and labelled accordingly. It does well in maintaining a logical progression, by comprehensively labelling and organising the flow of data. By creating an interactive, spatial learning tool, the framework facilitates a greater potential for comprehension by designers and architects, ultimately fulfilling the research objectives.

9. Conclusion

Mario Carpo suggests a design industry “where prediction can be based on sheer information retrieval, and form finding by simulation and optimisation can replace deduction from mathematical formulas” (Carpo, 2017). Through the development of an original ML framework, made specifically as a teaching tool for designers, a step has been taken towards Carpo’s prediction.

Progression towards more ML-driven design starts at an operational understanding of artificial neural networks. Through experimentation within a familiar environment, and investigation into functions that have been fractured into their base mathematical and programmatic operations, computational designers can gain this understanding. Throughout this research project, a teaching tool to facilitate such understanding was developed.

If this research were to be taken further, more complex ML algorithms could potentially be implemented within the Grasshopper environment. This not only provides a more comprehensive ML framework, but also increases the scope of teaching that this tool can provide.

Finally, this framework is not limited to architects and designers. Anyone interested in ML can utilise this tool. Perhaps the upcoming influx of intelligence algorithms will shape more than we anticipate.

Acknowledgements

This research was supported by the faculty of the Built Environment at UNSW, BIM Consulting and Architectus.

References

Biewald, L.: 2016. “How Real Businesses are Using ML”. Available from: TechCrunch <https://techcrunch.com/2016/03/19/how-real-businesses-are-using-machine-learning/> (accessed 02 August 2017).

Breiman, L.: 1994. “Bagging Predictors”. Available from: Department of Statistics University of California <https://www.stat.berkeley.edu/~breiman/bagging.pdf> (accessed 13 August 2017).

Brownlee, J.: 2017. “A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size”. Available from: ML Mastery <https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/> (accessed 23 July 2017).

Carpo, M.: 2013. “The Digital Turn in Architecture 1992 – 2012”. Available from: Google Books <https://books.google.com.au/books/about/The_Digital_Turn_in_Architecture_1992_20.html?id=sc9B3mxLCUcC&redir_esc=y> (accessed 15 August 2017).

Carpo, M.: 2016. “ACADIA 2016 Keynote – Mario Carpo”. Available from: Vimeo <https://vimeo.com/210622365> (accessed 15 August 2017).

Carpo, M.: 2017. “The Second Digital Turn: Design Beyond Intelligence”. Available from: The MIT Press <https://mitpress.mit.edu/books/second-digital-turn> (accessed 15 August 2017).

Dinesha, V.: 2013. “NeuronDotNet”. Available from: GitHub <https://github.com/trarck/NeuronDotNet> (accessed 26 August 2017).

ElSawy, I., Hosny, H. and Razek, M. A.: 2011. “A Neural Network Model for Construction Projects Site Overhead Cost Estimating in Egypt”. Available from: Cornell University Library <https://arxiv.org/abs/1106.1570> (accessed 25 July 2017).

Felbrich, B.: 2016. “Crow”. Available from: Food 4 Rhino <http://www.food4rhino.com/app/crow-artificial-neural-networks> (accessed 26 August 2017).

Greco, L.: 2015. “Dodo”. Available from: Food 4 Rhino <http://www.food4rhino.com/app/dodo> (accessed 26 August 2017).

Hutchings, B.: 2006. “Principles of Enquiry-Based Learning”. Available from: Centre for Excellence in Enquiry-Based Learning <http://www.ceebl.manchester.ac.uk/resources/papers/ceeblgr002.pdf> (accessed 02 November 2017).

Karpathy, A.: 2014. “Convolutional Neural Networks for Visual Recognition”. Available from: Github <http://cs231n.github.io/> (accessed 10 August 2017).

Kelly, K.: 2014. “The Three Breakthroughs that have Finally Unleased AI on the World”. Available from: Wired <https://www.wired.com/2014/10/future-of-artificial-intelligence/> (accessed 02 August 2017).

Kirillov, A.: 2012. “AForge.NET”. Available from: AForge.NET <http://www.aforgenet.com/> (accessed 26 August 2017).

Luu, V. T. and Kim, S. Y.: 2009. “Neural network Model for Construction Cost Prediction of Apartment Projects in Vietnam”. Available from: Research Gate <https://www.researchgate.net/publication/264113646_Neural_Network_Model_for_Construction_Cost_Prediction_of_Apartment_Projects_in_Vietnam> (accessed 25 July 2017).

MacIsaac, D.: 1996. “An Introduction to Action Research”. Available at: Buffalo State University <http://physicsed.buffalostate.edu/danowner/actionrsch.html> (accessed 02 November 2017).

Miller, N.: 2017. “Lunchbox”. Available from: Food 4 Rhino <http://www.food4rhino.com/app/lunchbox> (accessed 26 August 2017).

Narula, G.: 2017. “Everyday Examples of Artificial Intelligence and ML”. Available from: Tech Emergence <https://www.techemergence.com/everyday-examples-of-ai/> (accessed 02 August 2017).

Nielsen, M. A.: 2015. “What this Book is About”. Available at: Neural Networks and Deep Learning <http://neuralnetworksanddeeplearning.com/about.html> (accessed 12 July 2017).

Phelan, N.: 2016. “Designing with ML”. Available from: We Work <https://www.wework.com/blog/posts/designing-with-machine-learning> (accessed 02 August 2017).

Sanderson, G.: 2017. “Gradient Descent, How Neural Networks Learn”. Available at: Youtube <https://www.youtube.com/watch?v=IHZwWFHWa-w> (accessed 16 October 2017).

Sardina, S.: 2017. “Artificial Intelligence”. Available from: RMIT University <http://www1.rmit.edu.au/courses/004123> (accessed 02 August 2017).

Souza, C., Kirillov, A. and Catalano, D.: 2014. “Accord.NET”. Available from: Accord.NET <http://accord-framework.net/> (accessed 26 August 2017).

Srebro, N., Rennie, J. D. M. and Jaakkola, T. S.: 2005. “Maximum-Margin Matrix Factorization”. Available from: University of Chicago <http://ttic.uchicago.edu/~nati/Publications/MMMFnips04.pdf> (accessed 17 October 2017).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: 2014. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. Available from: University of Toronto <https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf> (accessed 13 August 2017).

Wilczek, F.: 2011. “Hidden Layers”. Available from: Edge https://www.edge.org/response-detail/10351 (accessed 3 August 2017).

Zwierzycki, M.: 2017. “Owl”. Available from: Food 4 Rhino <http://www.food4rhino.com/app/owl> (accessed 26 August 2017).

Appendix A: Equations

A.1. NET

$$z_i^l = \sum_j a_j^{l-1} \cdot w_{ij}^l + b_i^l$$

A.2. ACTIVATION

$$a_i^l = \sigma\left(z_i^l\right)$$

A.3. NETWORK ERROR

$$C = \frac{1}{2n} \sum_x \left(y(x) - a^L(x)\right)^2$$

A.4. ERROR CONTRIBUTION OF OUTPUT NEURONS

$$\delta_i^L = \left(a_i^L - y_i\right) \odot \sigma' \left(z_i^L\right)$$

A.5. ERROR CONTRIBUTION OF HIDDEN NEURONS

$$\delta_j^l = \sum_i \left(\left(w_{ij}^{l+1}\right)^T \cdot \delta_i^{l+1}\right) \odot \sigma' \left(z_j^l\right)$$

A.6. PARTIAL DERIVATIVE OF ERROR WITH RESPECT TO WEIGHT

$$\frac{\partial C}{\partial w_{ij}^l} = a_j^{l-1} \cdot \delta_i^l$$

A.7. PARTIAL DERIVATIVE OF ERROR WITH RESPECT TO BIAS

$$\frac{\partial C}{\partial b_i^l} = \delta_i^l$$

A.8. EXPONENTIAL LEARNING RATE DECAY

$${}^{+}\eta = \lambda \cdot \eta$$

A.9. MOMENTUM

$$^{+}v_{ij}^{l} = \mu \cdot v_{ij}^{l} - \eta \cdot \frac{\partial C}{\partial w_{ij}^{l}}$$

$$^{+}v_{i}^{l} = \mu \cdot v_{i}^{l} - \eta \cdot \frac{\partial C}{\partial b_{i}^{l}}$$

A.10. UPDATING WEIGHT

$$^{+}w_{ij}^{l} = w_{ij}^{l} + v_{ij}^{l}$$

A.11. UPDATING BIAS

$$^{+}b_{i}^{l} = b_{i}^{l} + v_{i}^{l}$$

A.12. MAX NORM CONSTRAINT

$$|w| \leq c$$

A.13. DROPOUT

$$r_{j}^{l} \approx Bernoulli(p)$$

$$\hat{a}_{j}^{l}(x) = r_{j}^{l} \cdot \sigma\left(a_{j}^{l}(x)\right)$$

$$z_{i}^{l+1}(x) = \sum_{j} \hat{a}_{j}^{l}(x) \cdot w_{ij}^{l+1} + b_{i}^{l+1}$$