Can machines learn?

Rowan Clarke

2020

Abstract

In this dissertation, we go through the meaning of learning and use it to create and justify a model that can be executed computationally. We then go over the methods of teaching the model.

The methodology outlines a path to firstly determine the parameters to use through understanding what they represent, and secondly justify the extent to which a problem can be solved using machine learning.

To understand what the parameters in a neural network mean, we use a combination of algebra and examples of simple and complex problems to devise a method for determining the parameters.

We then walk through the limitations to a model, which take into consideration the model parameters - making a correlation to the previous investigation - and overcoming these through the use of simple optimisations, which prove useful in different types of data. We then discuss the use of decomposition to architect a complex model, rather than assuming the neural network will find a pattern.

Taking all these factors into consideration, we produce and model our own optimisations for our own problem, which leads to the current research of present applications and future applications of machine learning.

We then conclude our understanding by justifying what factors are required to state that machines can learn.

Contents

1 Introduction

Machine learning and artificial intelligence is a topic that is rising in popularity. It is commonly regarded as a complex process that is difficult to understand, and thus has inspired me to understand the process of machine learning. This dissertation should arrive at a simple conclusion that justifies the extent of the usage of machine learning, through an intuitive understanding of machine learning.

However, we cannot simply jump to the topic of machine learning, as we would therefore be assuming that it can be classified as learning itself. If we were to say that machines can learn, we would need to understand what learning is, and how it can be used to produce a computational model.

To find the extent to which any problem can be solved using machine learning, we would have to understand how to determine a suitable model given our problem and find limitations in the model. Then we can find optimisations to overcome these limitations.

Using this information, we walk through an optimisation to our own problem, justifying the ability to create and model all problems. This then follows nicely into the current and future uses of machine learning, where we discuss how machine learning has already been implemented into certain markets and how this will evolve given the current research into the area.

2 Learning

2.1 Inspiration from Human Learning

Our brains process information through neurons, which combine many electrical inputs, called dendrites, into one output, the axon. These neurons are connected via synapses, which chemically transmit information via neurotransmitters. [1]

The amount of neurotransmitter released will change how strong the connection is. If a connection is stronger from one neuron than any other neuron, the activation is mostly dependent on that neuron.

By connecting multiple neurons together, we allow for more complex operations. Our cerebellum - the part of the brain that controls language, motor function, and cognitive learning - consists of roughly 10^{11} neurons, each connected to approximately 10^4 other neurons.[2]

2.2 Hebbian Theory

To digitalise this biological structure of the brain into a machine is simply impossible on any piece of hardware - as we would have to simulate the position and activation of 10^{15} connections - so we need a mathematical model of the brain. The Hebbian Theory attempts to model this behaviour.

The Hebbian Theory suggests that if two neurons in space activate in phase - at the same time - the connection between them should strengthen. If in opposite phase, the connection should become weaker. No correlation between the activations should not affect the connection.[3, p. 70]

Hebbian learning tries to replicate learning in animals such that

$$w_{ij} = x_i x_j \tag{1}$$

where w_{ij} is the weighted synapse between neuron i and neuron j, and x_i is the value at neuron i, at a given time in space. Linking this with the biological neuron, x_i is the axon of i - as with j - and w_{ij} is the strength of the synapse connecting the axon of i to the dendrite of j.
We can rectify Equation 1 to

$$\Delta w_{ij} = x_i x_j \tag{2}$$

such that w_{ij} changes based on its current state so that previous inputs are not obsolete.

To put this into an example, let's say that a neuron activates due to a visual stimulus, and shortly after a neuron activates due to pain. The connection will strengthen between them as they occur in phase.[3, p. 63] This would lead you to associating the visual stimuli to pain. This psychology is called 'association psychology,' and its premise is that learning is the association of ideas over time, which in turn creates an idea.[4]

3 Perceptron Model

3.1 Artificial Neural Networks

The Hebbian Theory allows a computer to distinguish similarities in the features between inputs. This is because, over time, a neuron that represents a feature will be associated with that feature in the input. However, we do not know what these features are, only that they are common within inputs. We, therefore, need another model that takes these neurons and decides what the features represent. For this to work, we need a desired output for each input, telling the model which features are important to which input, to determine the correct output. This is what the perceptron model attempts to do.

3.1.1 Neuron Model

Consider a neuron y. There are many neurons connected to y, $x_1, x_2, \cdots, x_I$. The axon from x_i, $\mathrm{A}(x_i)$, is connected to the i^{th} dendrite of y, $\mathrm{D}_i(y)$, via a weighted synapse w_i. Therefore $\mathrm{D}_i(y) = w_i\mathrm{A}(x_i)$. The dendrites of y are combined to give the axon of y, $\mathrm{A}(y)$.[5, p. 84]

The axon of y can be simply modelled as

$$\begin{aligned} \mathrm{A}(y) &= \sum_{i=1}^{I} \mathrm{D}_i(y) \\ &= \sum_{i=1}^{I} w_i \mathrm{A}(x_i) \end{aligned} \tag{3}$$

If the axon of x_i is deactivated, $x_i = 0$, the dendrite of x_i will always be zero, thus losing information. We must therefore add a bias, so our function does not need to be proportional.

$$\mathrm{A}(y) = b + \sum_{i=1}^{I} w_i \mathrm{A}(x_i) \tag{4}$$

Since we are dealing with just the axons of x_i and y, we can simplify Equation 4 as

$$y = b + \sum_{i=1}^{I} w_i x_i \tag{5}$$

3.1.2 Network Architecture

x_i and y are neurons, which can be modeled as nodes

Figure 1: model of x neurons and y neuron

These neurons are connected via synapses, which can be modeled as a line connecting the neurons.

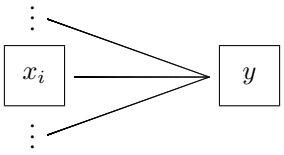


Figure 2: model of x neurons connected to y neuron with synapses

We can add more neurons, y_j, from the same input neurons, x_i, using different synapses. This will increase the complexity of the neural network, allowing for more flexibility.

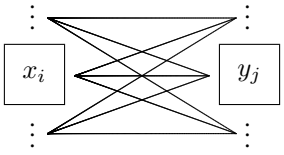


Figure 3: model of x neurons to y neurons with synapses

In a neural network, there are input, hidden, and output layers, where a layer contains neurons that activate at the same time at a specific depth within the network. [5, p. 87]

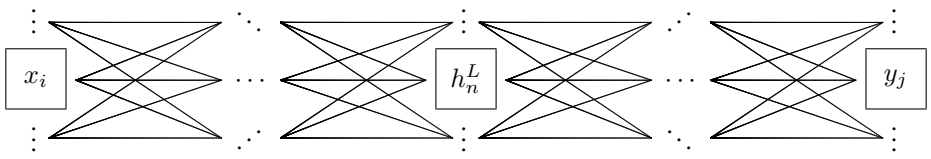


Figure 4: model of x input neurons connected to h^L neurons connected to y output neurons

More layers allow for more complex functions. We will discuss how to correctly determine the complexity in Section 5, and we will discuss the implications of not using this method in Section 6.

3.2 Gradient Descent

So far we have discussed how information is propagated forward through the network to produce an output, however, there is no way we can change the weights of each synapse to achieve a different output, and thus learn.

Gradient Descent attempts to minimise the cost, ∇C, of a neural network, which is a measure of how well it maps the target data. There are many ways of doing this, but for our simple feedforward architecture, the most common method is backpropagation.

3.2.1 Backpropagation

Backpropagation aims to change the weights of a neural network in the most efficient way such that the output of the neural network fits closer to the desired output, given a specified input.

Consider the following neural network.

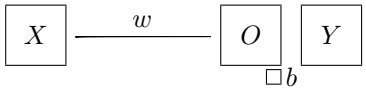


Figure 5: model of neuron X connected to neuron O with a weighted synapse w

We can describe the how close the output, O, is to the target, Y, with the cost, C, which is equal to the square of the difference of O and Y. We square this value so that the cost is positive, and it is continuously differentiable, which will prove very useful later on.[5, p. 313]

$$C = (O - Y)^2$$

We can make a graph of how w affects C, so we can find what weight gives the minimum cost, and therefore the closest output to our target.

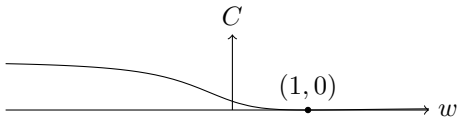


Figure 6: A graph of C against w

We know that where $w = 1$, we achieve a cost of 0, so the output is equal to the input. However, we do not want to iterate through all possible weights in a neural network to find the minimum cost,

as that is very computationally expensive. Instead, we should start at a random w, say -1, and take small steps downhill, hence gradient descent.

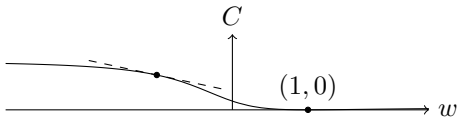


Figure 7: A graph of C against w showing the gradient at $w = -1$

We can decrease the cost, C, by repeatedly changing w proportionally to the negative gradient at w, or

$$\Delta w = -\alpha \frac{\partial C}{\partial w} \tag{6}$$

where α is the learning rate, which is essentially how big of a step in the given direction you take - too small, and it will take too long finding the minima - too large, and you will overshoot the minima, almost rocking back and forth.

Now, all we need to do is find the gradient.

3.2.2 Using the Chain Rule

Using Figure 5, we can express C using the following functions

$$\begin{aligned} C &= (O - Y)^2 \\ O &= \sigma(z) \\ z &= wX + b \end{aligned} \tag{7}$$

Using the chain rule, we can equate

$$\begin{aligned} \frac{\partial C}{\partial w} &= \frac{\partial C}{\partial O} \frac{\partial O}{\partial z} \frac{\partial z}{\partial w} \\ &= 2(O - Y)\sigma'(z)X \end{aligned} \tag{8}$$

Equation 8 only works for Figure 5, so we need a more general model.

It is important to take notice of the function $\sigma()$. This is what is known as the activation function, and limits the range of the value of a neuron. In this example, and everywhere else in this dissertation, we use the activation function sigmoid, and its range is denoted as $0 < \sigma(x) < 1$.

Since we are dealing with multiple neurons, weights, and layers, we need a syntax to index these: x_i^L represents the i^{th} neuron in layer L; Y_i^L represents the target for x_i^L; n_L represents the number of neurons in layer L; w_{ij}^L represents the weight of the synapse connected to the i^{th} neuron in layer L, from the j^{th} neuron in layer $L - 1$.

Consider Figure 8

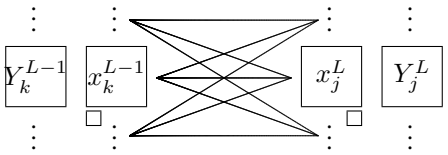


Figure 8: model of neurons x^{L-1} in layer $L - 1$ with targets Y^{L-1} connected to neurons x^L in layer L with targets Y^L.

The cost from multiple outputs is the sum of each cost of the output to the corresponding output.

$$\begin{aligned} C &= \sum_{j=0}^{n_L-1} (x_j^L - Y_j^L)^2 \\ x_j^L &= \sigma(z_j^L) \\ z_j^L &= b_j^L + \sum_{k=0}^{n_{L-1}-1} w_{jk}^L x_k^{L-1} \end{aligned} \tag{9}$$

We can reevaluate the gradient again using the chain rule.

$$\begin{aligned} \Delta w_{jk}^L &= -\alpha \frac{\partial C}{\partial w_{jk}^L} \\ \frac{\partial C}{\partial w_{jk}^L} &= \frac{\partial C}{\partial x_j^L} \frac{\partial x_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{jk}^L} \\ &= 2(x_j^L - Y_j^L)\sigma'(z_j^L) x_k^{L-1} \end{aligned} \tag{10}$$

To find out what b_j^L should be we can use the chain rule against b_j^L.

$$\begin{aligned} \Delta b_j^L &= -\alpha \frac{\partial C}{\partial b_j^L} \\ \frac{\partial C}{\partial b_j^L} &= \frac{\partial C}{\partial x_j^L} \frac{\partial x_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial b_j^L} \\ &= 2(x_j^L - Y_j^L)\sigma'(z_j^L) \end{aligned} \tag{11}$$

To find out what x_k^{L-1} should be, Y_k^{L-1}, we can use the chain rule against x_k^{L-1}. We use Y as a buffer so that we can keep the original x.

$$\begin{aligned} Y_k^{L-1} &= x_k^{L-1} - \alpha \frac{\partial C}{\partial x_k^{L-1}} \\ \frac{\partial C}{\partial x_k^{L-1}} &= \frac{\partial C}{\partial x_j^L} \frac{\partial x_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial x_k^{L-1}} \\ &= 2(x_j^L - Y_j^L)\sigma'(z_j^L) w_{jk}^L \end{aligned} \tag{12}$$

We can then repeat this for each layer going backwards, until all the weights have been rectified to decrease the cost in the most efficient way possible, ∇C. If we repeat this process for each $\{X_i, Y_i\}$ in training set T, we would have trained a function to fit to T. [5, p. 365]

To show what this process is doing, the image in Appendix A.1 shows how the cost is being reduced in the most efficient way.

4 Methodology

4.1 Aim

Using our knowledge of modelling the human brain, learning through the perceptron model, and back-propagation, we aim to evaluate and justify the extent to which any problem can be taught effectively to a machine such that it is able to give accurate results.

4.2 Methods

In order to do this, we must devise a method to measure each of the aspects of a neural network that directly affect the accuracy of the results. Once we have achieved this, we should determine and justify a method for avoiding these aspects for all problems, and then prove it through the use of outlining an optimisation. This should then be met with real-world observations to prove the usefulness of machine learning.

4.3 Reliability

The sources I cite are very well known and have been used in academic papers.

[1] Gordon M. Shepard is a pioneer in the area of neuroscience. He introduced the olfactory system, and used this as a method for modelling dendrites in the brain amongst other areas. His education consists of a MD at Harvard in 1959, and a DPhil at Oxford in 1962.

"By any measure this work is a classic...It will undoubtedly take its place as one of the most significant and comprehensive commentaries of our time on structure and function of nervous tissue."–Electrocephalography and Clinical Neurophysiology

[2] The experiment consisted of "five elderly men", and their method produced accurate results.

[3] Donald O. Hebb is a psychologist who specialises in understanding human learning. He had recieved 3 honorary doctorates in 1961, 1965, and 1975.

The Organisation of Behavior is seen as Donald Hebb's most affluential piece of work modelling our view on neuropsychology.

[4] "The text is very strong with respect to presenting competing theories of learning phenomena. It is one of the best I've read in this regard."–Thomas J. Faulkenberry

[5] "This introductory neural network text targets senior undergraduates, first-year graduate students, and computing professionals who are exploring artificial neural networks."–Stan J. Thomas

5 Parameters

5.1 Learning Paradigms

A learning paradigm specifies what the general architecture of the model is. For example, if we only have inputs to a neural network with no target data, we cannot train a model with outputs. As this determines the very core of the design of the model, we call this a paradigm.

Consider the different types of learning paradigms, supervised, unsupervised, and reinforcement learning.

5.1.1 Supervised Learning

Supervised learning is where the machine is tasked to fit target data. The training set T can be expressed as

$$T = \{X_1, Y_1\}, \{X_2, Y_2\}, \cdots, \{X_I, Y_I\}$$

where X_i is an input, Y_i is the desired outcome of X_i, and I is the length of the training set T. The machine propagates back through the neural network to adjust the synapses in order to fit closer to the target data.

An example of a supervised learning model would be the perceptron model.

5.1.2 Unsupervised Learning

Unsupervised learning is where the machine alters the synapses in the neural network solely based upon the inputs. This is typically used to classify data or cluster data.

An example of an unsupervised model is a K-means clustering problem.

5.1.3 Reinforced Learning

Reinforcement learning is where the machine is given an environment in which the outputs control, which are then rectified through the environment feedback. For example, if a chess AI made a move, the way we give feedback on the move is through an environment, not target data.

We can also have a separate neural network act as the environment, in which we call this an adversarial network; the neural network act as each others environment.

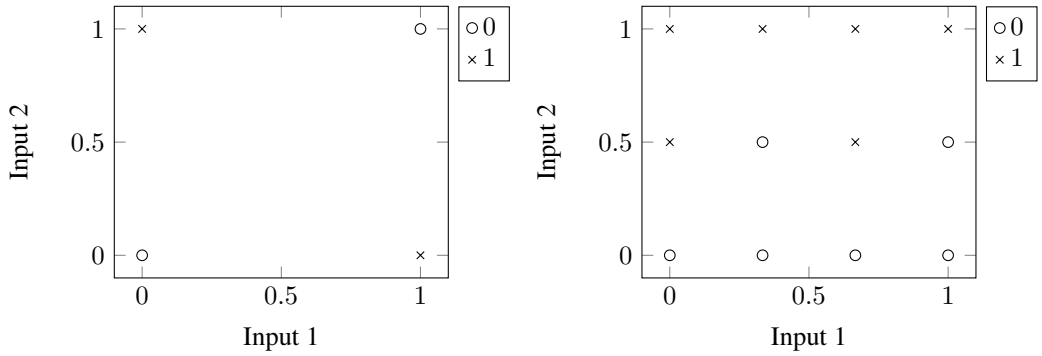
5.2 Determining the Model Parameters

To determine what model parameters we should use, i.e. the number of hidden layer and neurons in each layer, we must understand intuitively what a neuron represents and what layers represent.

A neuron is a function of the neurons in the previous layer. If our function is the product of a training parameter w and a dependent variable x, we can treat this as a line wx. As we can also train a bias b, we result in the equation $wx + b$. If we have multiple neurons in the previous layer, we can treat this as a line through multiple dimensions $w_1x_1 + w_2x_2 + \cdots + b$. In other words, a neuron can classify a linear relationship from the previous layer.

A layer can now represent the building and accumulating of features, as if our linear relationship takes in linear functions, we achieve a non-linear relationship. As an example, the output should determine the representation of shapes, which classifies the relationship from the previous layer of the representation of lines.

This is to say that if we are able to identify lines in the first layer, we can identify lines in those lines in the next layer, producing shapes and non-linear equations. Consider the datasets below



The one on the left can be described as a combination of two lines. As the combination is linear, we only need one layer for the lines. This gives us the model of $2 \to 2 \to 1$.

The one on the right is more complicated. There are three lines that can be used to classify the different sections in the dataset, but they cannot be combined linearly, as they intersect twice. Therefore, we have the model of $2 \to 3 \to 2 \to 1$.

This is backed up by the successfulness of each model architecture for the simple model and complex model in Appendix A.3.

5.3 Determining the Hyperparameters

The hyperparameters are parameters that you specify before the main computation of the model. This primarily includes the learning rate. The learning rate details how much the weight is changed based on a given gradient. This parameter was discussed earlier in Section 3.2.1, however, it is important to understand how to determine its value.

If your problem is complex, then having too large of a learning parameter could result in your program not finding a solution. This is because it is skipping the points at which the cost is the lowest, and thus we are losing information about the gradient. It can be thought of as though we are randomly trying points until they reach a gradient of 0.

If your problem is simple, then having too small of a learning parameter means that it will take a long time to reach zero cost. This is because when the gradient gets smaller, which will happen with simple examples, the weights change less and less. However, if you use too high of a learning parameter, you risk not finding the shallow gradient, meaning you cycle back and forth between values.

6 Limitations

6.1 Model Limitations

6.1.1 Inaccuracies

If a problem is too complex, the computer may not be able to find a solution. It will therefore resolve a minimum that is equally bad at each input, producing an inaccurate result. This could be due to the simplicity of the model. This decreases the flexibility of the neural network to find patterns. This can be decreased through using a suitable model complexity or giving the machine an incentive to find the correct pattern through the use of decomposition and optimisations, or more data such that the pattern is more easily recognised. [6]

6.1.2 Overfitting

If a neural network has too many parameters or rather more than is needed, it may be able to memorise the input instead of learning a pattern. This gives the illusion that it has worked with the given training

data, but when it is given test data, it will not produce accurate results. This is because there are many different solutions to an overly complex model, each with a different pattern. As this cannot, therefore, attribute a single pattern to the model, we do not classify it as learning, but rather memorising. The effect of this can be reduced through simplifying the model to a suitable level or influencing the model to fit to a specific pattern using optimisations and decomposition, or more data such that there is only one pattern that achieves a zero cost.

6.2 Physical Limitations

6.2.1 Data

In supervised learning, you must provide target data to each input. If this data is not sanitised or large enough, the network will find erroneous patterns and produce an incorrect output.[6]

Furthermore, the data you use as input to the neural network must be related to the output, and no relevant data should be left out; the computer doesn't understand the situation, and you cannot assume it does. If you do not know what fields may influence the output, use as much information as possible, increasing only the input layer only marginally increases the computation.

This isn't so much of a problem in corporations that manage multiple customers and branches, as they can easily sample data.

6.2.2 Hardware

If your model is too complex, especially if it is more complex than it should be, it could take a long time to find a solution. This is because it would have to perform more calculations per epoch, and more epochs to find the solution. If you limit the time and computation of training the model, you risk limiting the number of items in your dataset, and you see the same problems that we earlier discussed in Section 6.2.1.

To reduce computation, you can simplify the model, however, if this is not possible, you could implement optimisations and use multiple neural networks that each work on a given simpler task to decompose the problem.

Again, in large industries, the hardware is abundant, and it usually isn't seen as a limitation.

6.3 Legal and Ethical Concerns

There are times when the use of machine learning can produce legal, ethical and moral dilemmas. Concerns include data harvesting, liability, discrimination and profiling, and job displacement. We must take these scenarios into consideration to justify whether a problem can be solved under the law.

6.3.1 Data Harvesting

Data Harvesting is where a large corporation surveys and polls information about the clients they serve. This is necessary such that the model that they train will correctly select the targeted advertisement. This has privacy concerns for the client, as some information may be sensitive. The Data Protection Act prohibits companies from storing data about an individual that is not necessarily required for the purpose that the company sells, without their consent. Therefore, companies that do store a lot of information about you, such as Google, require you to consent to their terms. So, although it is legal to use data harvesting, is it moral to have your product only accessible through surrendering your privacy?

6.3.2 Liability

Liability is a risk when the output dictates a decision that can have life-threatening consequences. These include autonomous vehicles; if the machine makes a moral decision or mistake, who is re-

sponsible? Would it decide to swerve out the way to avoid hurting the group of people, but hurt someone who did nothing wrong or vice versa?

6.3.3 Discrimination and Profiling

Profiling is where the machine uses personal information as input to dictate the output. This could be dangerous if it determines the opportunity for a service. As an example, if it is used in qualifying loans or credit; is it ethical to use race or gender as inputs when estimating return rates when there is no credit history? It is clear to see that this can cause a discrimination issue.[6]

6.3.4 Job Displacement

The most likely threat to our daily lives that machine learning imposes is the use of it in the workforce. Tasks that have a pattern with large amounts of data can be aided significantly with the use of machine learning, decreasing job income and security.

This may not seem to be a large issue as it only affects certain jobs, but when these jobs are likely the entry point for apprenticeships and work experience, the employability for those new to the workforce suffers.

6.4 Conclusion

To overcome both model and physical limitations, we require optimisations to decrease computation. The optimisations will have to be specific to the problem itself, as it would otherwise defer from the model we initially have. We could also use decomposition to incentivise the neural network to find more appropriate solutions, where the parts of the total model deal with a simpler problem.

7 Optimisations

We may want our machine to learn more complex problems, where the patterns to learn are more obscure. We could use the perceptron model and increase the complexity of the model, but this is not very effective as this will drastically increase the computation. Instead we want to find solutions and optimisations to certain types of data that cut down on the number of neurons we use. There are many ways we can optimise for certain types of inputs, but we will look specifically at an unsupervised model, spatial data, and sequential data.

7.1 Auto-encoders

Auto-encoders are unsupervised neural networks, as they do not require target data, however, they still have target data to meet. They encode (compress) the input data into a latent vector, z, and then decode the latent vector into an output that should resemble the input.

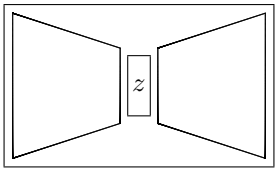


Figure 9: Autoencoder block, AE

If you replace X_i with a function of X_i, $f(X_i)$, you can train an inverse function as the auto-encoder.

They train an inverse function f^{-1} from a given function f.

Figure 10: $\tilde{X}_i$ is the decoded output that resembles the input.

$$X = \{X_1, X_2, \cdots, X_I\}$$

There are some common functions that lead to useful outcomes. Where $f(x) = x$, the auto-encoder acts like a compressor, making it an effective lossy compression technique to reduce broad-band. Where $f(x)$ is a noise filter, the auto-encoder acts as a denoiser, making it faster to render fragment shaders as you do not have to calculate every pixel.

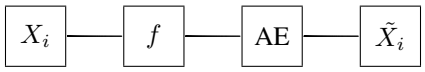


Figure 11: Using a function, we can produce new inputs for our auto-encoder.

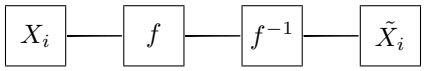


Figure 12: The auto-encoder then behaves as an inverse of the input function.

7.2 Convolutional Neural Networks

A convolutional neural network uses a variety of methods to reduce the computation on spatial data, i.e. where the surroundings of the data are important. One method we can use are convolutions, which are layers that find patterns in context of the spatial data easier.[7, p. 7]

This means that the network requires less neurons, which reduces the risk of overfitting and also saves computation.

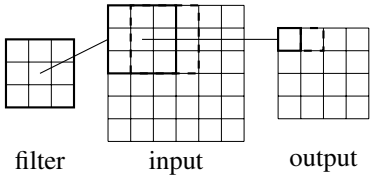


Figure 13: Output is the convolution of the input given the filter.

The output from a convolution is a matrix where each node corresponds to the sum of encapsulated nodes multiplied by the filter's respective node.

$$\mathbf{O}_{ij} = \sum_{x=-1}^{1} \sum_{y=-1}^{1} \mathbf{I}_{(i+x+1)(j+y+1)} \times \mathbf{F}_{(x+1)(y+1)} \tag{13}$$

or simply

$$\begin{aligned} \mathbf{O}_{ij} &= \sum_{x=0}^{2} \sum_{y=0}^{2} \mathbf{I}_{(i+x)(j+y)} \times \mathbf{F}_{xy} \\ &= \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} \mathbf{I}_{(i+x)(j+y)} \times \mathbf{F}_{xy} \end{aligned} \tag{14}$$

To back-propagate through convolutions, we will use a similar method to what we used in Section 3.2.2.

$$\mathbf{O} = v(\mathbf{I}, \mathbf{F}) \tag{15}$$

where v is a function representing a convolution.

Therefore we can decrease the cost C, ∇C, using the chain rule for the filter,

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{F}} &= \frac{\partial C}{\partial \mathbf{O}} \frac{\partial \mathbf{O}}{\partial \mathbf{F}} \\ \frac{\partial C}{\partial \mathbf{F}_{xy}} &= \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \frac{\partial C}{\partial \mathbf{O}_{ij}} \frac{\partial \mathbf{O}_{ij}}{\partial \mathbf{F}_{xy}} \\ &= \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \frac{\partial C}{\partial \mathbf{O}_{ij}} \mathbf{I} - \mathbf{1}_{(i+x)(j+y)} \\ \frac{\partial C}{\partial \mathbf{F}} &= v(\mathbf{I}, \frac{\partial C}{\partial \mathbf{O}}) \end{aligned} \tag{16}$$

and also for the image

$$
\begin{aligned}
\frac{\partial C}{\partial \mathbf{I}} &= \frac{\partial C}{\partial \mathbf{O}} \frac{\partial \mathbf{O}}{\partial \mathbf{I}} \\
\frac{\partial C}{\partial \mathbf{I}_{ab}} &= \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \frac{\partial C}{\partial \mathbf{O}_{ij}} \frac{\partial \mathbf{O}_{ij}}{\partial \mathbf{I}_{ab}} \\
&= \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} \frac{\partial C}{\partial \mathbf{O}_{(a-x)(b-y)}} \mathbf{F}_{xy} \\
\frac{\partial C}{\partial \mathbf{I}} &= V(\frac{\partial C}{\partial \mathbf{O}}, R_{180^\circ}\mathbf{F})
\end{aligned}
\tag{17}
$$

where V is a full convolution. A full convolution is a convolution that allows for a padding of the output, as $a - x$ or $b - y$ could be outside the range of output.[8]

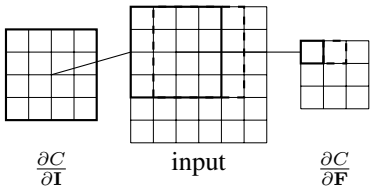


Figure 14: $\frac{\partial C}{\partial \mathbf{F}}$ is the output of a convolution with the filter $\frac{\partial C}{\partial \mathbf{I}}$ on the input.

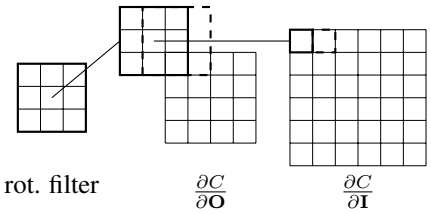


Figure 15: $\frac{\partial C}{\partial \mathbf{I}}$ is the output of the full convolution $\frac{\partial C}{\partial \mathbf{O}}$ given a filter rotated by 180°

By using multiple convolutional layers in series, you teach the neural network to decompose features of the images, such as lines and shapes, which helps a feed-forward neural network to decrease the cost. For example, the first layer may pick out edges, the next lines, and then shapes.

We can also use pooling, which is a method of finding only the important information. It works similarly to a convolution, except the filter is constant, and uses a function instead of a summation, such as returning the maximum. This helps condense the output from convolutions.

In conclusion, we can use convolutions and pooling to help and optimise feed-forward networks to find patterns in spatial data. This is because the trainable parameters are only concerned in data surrounding a node.

Moreover, this concludes that if we can differentiate a function to propagate forward and calculate a cost, we are able to adjust the parameters of the function to decrease the cost.

7.3 Recurrent Neural Networks

Recurrent neural networks are supervised neural networks that allow you to analyse sequential data. They work by feeding the current state into itself, which allows you to predict states of a system in the future based on the present.

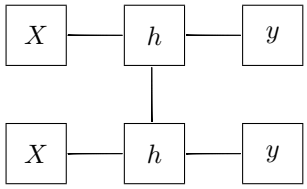


Figure 16: A recurrent neural network feeding the present state into the next.

Back-propagating through this architecture is quite complex. It requires you to back-propagate through each state, which therefore implies that the more states you have, the more operations you would have to do. You would also have to store each of the inputs and outputs for each state.[9]

Aside from this minor difference, the algebra of back-propagation is the same.

8 Modelling Optimisations

Now that we have seen how to model a neural network such that the machine can perform gradient descent on relevant data, we can create our own models.

8.1 Variational Auto-encoder

A variational auto-encoder attempts to use a latent vector, z, that is a sample from a probability distribution of a variable. Therefore, the encoder, $q_\phi(z|X)$ must be a distribution. As a probability distribution is defined as $N(\mu, \sigma^2)$, we can sample the latent vector from a mean vector and a standard deviation vector.

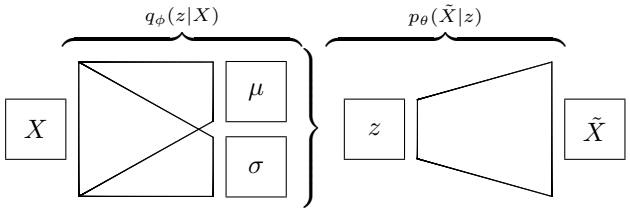


Figure 17: The varaitional auto-encoder with the probabilistic encoder q_ϕ and decoder p_θ.

Sampling from a distribution allows the variable to represent a feature, as small changes in a feature should have a similar output. However, since this give a greater reconstruction loss than a regular auto-encoder due to the restriction, the standard deviation, σ, will become 0, such that it acts as a regular auto-encoder. Therefore, we must compare the sample distribution, $N(\mu, \sigma^2)$, to a distribution of $\sigma = 1$, $N(0, 1)$, and make sure they are roughly equal. This function is called the Kullback–Leibler divergence, $D_{KL}(\alpha||\beta)$, and returns 0 when the distributions α, β are equal.

As we cannot back-propagate through a sample of a distribution, we have to take out the sampled distribution into a normal variable, and then apply the standard deviation and mean to achieve a result. This will then give us a method of back-propagating through the variables. This essentially gives us $z = \sigma x + \mu$, where $x \sim N(0, 1)$.

9 Uses of Machine Learning

9.1 Present Uses

With growing data sets and dispensable hardware, machine learning is becoming increasingly more relevant in today's industry. Primary focuses are on detecting patterns in user's search history for advertisement, face recognition for social media sites, commuting predictions for traffic, surveillance footage for reporting crimes, and translation from and to different languages. Any laborious and extensive task that seems simple can probably be and is automated using machine learning.

"It is difficult to think of a major industry that AI will not transform. This includes healthcare, education, transportation, retail, communications, and agriculture. There are surprisingly clear paths for AI to make a big difference in all of these industries."–Andrew Ng

9.2 Future Uses

Machine learning has already in place for simple tasks for decades, however, only recently has hardware become so cheap and accessible that we can now use machine learning to accelerate complex algorithms such as fluid simulation, where the output doesn't have to be exactly accurate, as long as it is reasonable to the human eye. Another example would be in DLSS for ray trace rendering. It samples some of the rays to produce a low-quality noisy image, passes it through a network and produces a high-quality image.

Recently, there has been a scientific breakthrough in the field of biological engineering due to advancements in machine learning. Google's DeepMind software "AlphaFold 2", has been able to accurately predict the folding of protein structures, a problem that has been around for 50 years. Researchers believe it will enable us to understand viruses, create drugs, and design vaccines.

10 Conclusion

10.1 Summary

We have seen the correlation between machine learning and human learning, modelling learning through the perceptron, understanding how learning occurs through back-propagation, and how we can expand this knowledge to create specific architectures and models.

With these ideas, we were able to find limitations to given problems, and what is required for a machine to learn. Once we have arrived at a clear model for solving problems, we then discussed the ethics and legality of doing so.

10.2 Main Conclusion

It is clear to see that with a model that does not have any assumptions, enough data, and a model complexity that mirrors the complexity of the problem, we can teach a machine any problem through decomposition and our own understanding, to influence the machine to learn a pattern.

It should therefore be considered if it is legal or ethically possible, which under certain conditions, produces moral concerns.

10.3 Recommendations for Further Work

There are numerous areas where further research was necessary to produce a clear argument for how more complex architectures are designed and used. This includes researching and designing different activations, different methods of back-propagation, and complex problems that require decomposition and decomposition of multiple neural architectures.

References

[1] G.M. Shepard. *The Synaptic Organization of the Brain*. Oxford University Press, 1974.

[2] B.B. Anderson. A quantitative study of the human cerebellum with unbiased stereological techniques. *J Comp Neurol*, 1992.

[3] D.O. Hebb. *The Organisation of Behavior*. New York: Wiley & Sons, 1949.

[4] S.B. Klein. *Learning: Principles and Applications*. SAGE Publishing, 2018.

[5] M.T. Hagan. *Neural Network Design*. Boston: PWS Publishing, 1996.

[6] Andrew Ng. What artificial intelligence can and can't do right now. *Harvard Business Review*, 2016.

[7] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *ArXiv e-prints*, 11 2015.

[8] Jefkine. Backpropagation in convolutional neural networks. *Jefkine*, 11 2016.

[9] Ilya Sutskever. Training recurrent neural networks. *University of Toronto: Graduate Department of Computer Science*, 2013.

Glossary

activation A non-linear function that limits the range of a neuron.. 8

axon The result of a neuron once passed through an activation function. 5, 6

back-propagate A means to repeatedly move information backward through functions while updating parameters. 16

cost How well the output of a neural network represents the target data. 7–9, 16, 17

decomposition Breaking a problem down into simpler problems that emphasises and persuades a solution to a neural network. 12, 20

dendrite The axon from the previous neurons when multiplied with a weighted synapse. 5, 6, 10

epoch One full cycle through each element in the dataset.. 13

hidden A layer that does not interact with the user. 7, 11

layer A group of neurons that represent the same functional complexity. 7–9, 11–13, 16, 17

linear An indefinite line through space. 11, 12

matrix A 2 dimensional list of entities, such as numbers, that can be indexed at i, j with the syntax m_{ij}. 16

model A mathematical representation of a problem. 4–8, 10–12, 14, 18, 20

neuron A single node that represents a linear function of the neurons in the previous layer. 5–8, 11, 14, 16

non-linear A combination of lines. 11

perceptron An algorithm that specifies how information is propagated. 5, 10, 11, 14, 20

propagate A means to repeatedly move information forward through functions. 18

synapse The weighted connection between neurons. 5–8, 11

vector A list of entities, such as numbers, that can be indexed at i with the syntax v_i. 14, 18

A Results

A.1 Gradient Descent

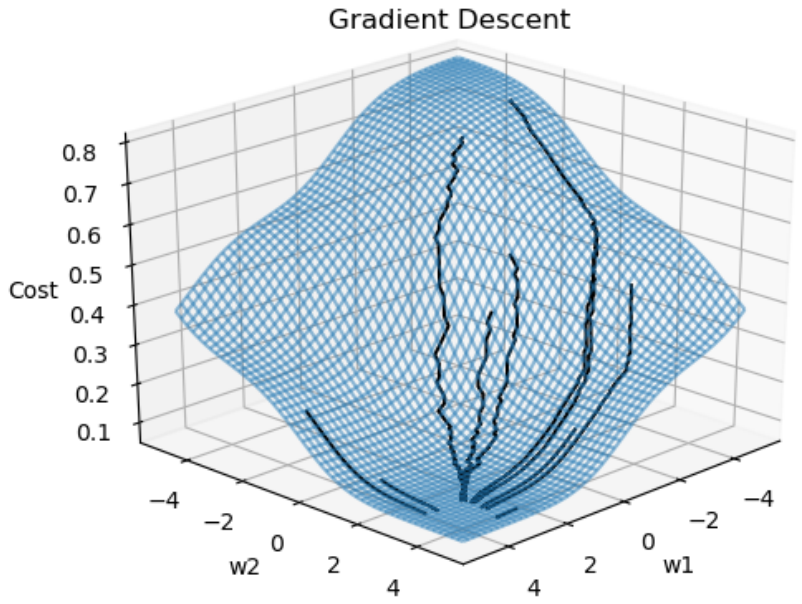


Figure 18: We can see from our own example, that the cost is being reduced in the most efficient way, ∇C by adjusting the weights.

A.2 Training Model

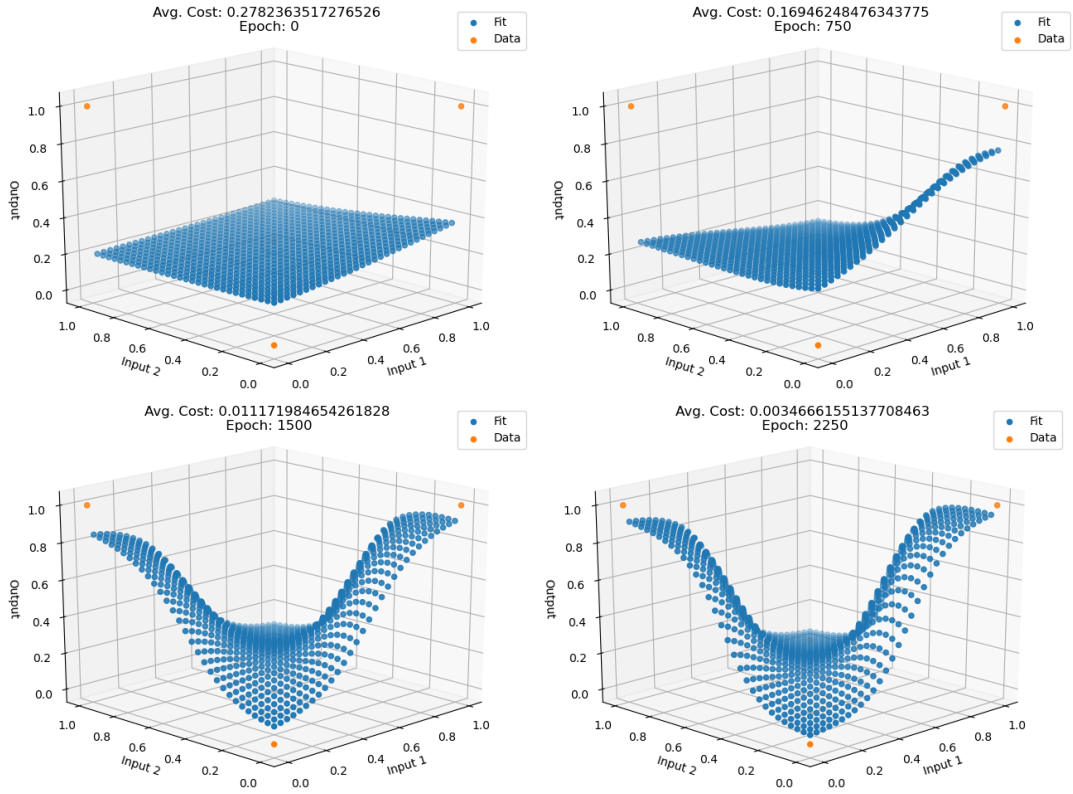


Figure 19: The data the model uses is an XOR function. With each epoch iteration of 750, we can see how our function (fit) gets closer to the actual results (data), decreasing the cost.

A.3 Model Parameters

A.3.1 Simple Model

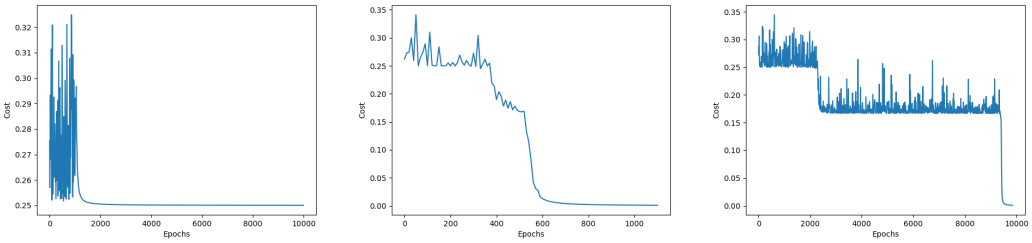


Figure 20: Left model: 2-1; Middle model: 2-2-1; Right model: 2-3-2-1;

A.3.2 Complex Model

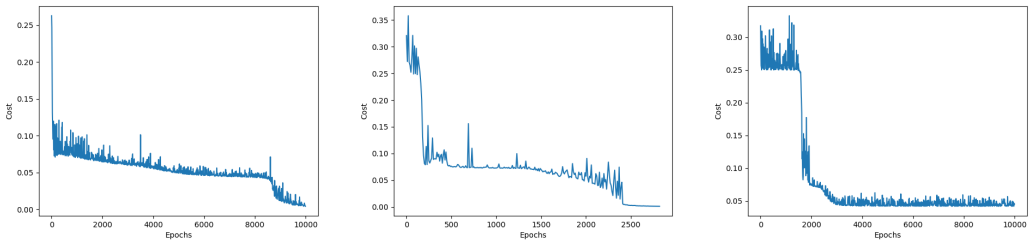


Figure 21: Left model: 2-2-1; Middle model: 2-3-2-1; Right model: 2-8-4-2-1;

The left model of 2-2-1 results in a cost of approximately zero. There are many reasons that could allow this to happen. Firstly, two hidden neurons implies that there are two linear lines, however, that is clearly not possible given the data. Therefore there must be a type of rounding error that allows more complexity in the neuron. Secondly, since we are using a sigmoid activation function, it limits the range to $0 < x < 1$. Therefore, the combination of these lines do not have to be non-linear, as our activation is not.