Neural Networks

How a neural network learns to transform data

Feranmi Dere

Tom Rocks Maths Essay Competition 2024

Introduction

Neural networks are a type of modelling used in Artificial Intelligence and Machine Learning – mostly used in deep learning – and this specifically refers to the use of artificial neural networks with multiple hidden layers, the meaning of which will be clarified later in this essay.

Machine learning is the concept of algorithms or software that learns without explicit programming. It can be split into two branches – supervised learning and unsupervised learning. Supervised learning refers to problems that include an outcome or output – most referred to as labels, and supervised learning models aim to capture the relationship between the inputs and the output(s), and later predict new outputs based on unseen data. On the other hand, unsupervised learning has no outputs or outcomes, and the models aim to capture patterns in the data. Supervised learning falls into two categories: regression (continuous outputs) and classification (distinct or categorical outputs).

This technique has been loosely modelled after the organisation of neurons in a brain, made up of connected nodes – referred to as artificial neurons – which perform certain mathematical functions and calculations to model patterns in data.

The main function of most machine learning and AI models are to recognise patterns in data – and for some models – to predict outcomes based on new data. We will now explore how this is done with neural networks in the case of trying to learn mathematical functions.

1 Introducing the neural network

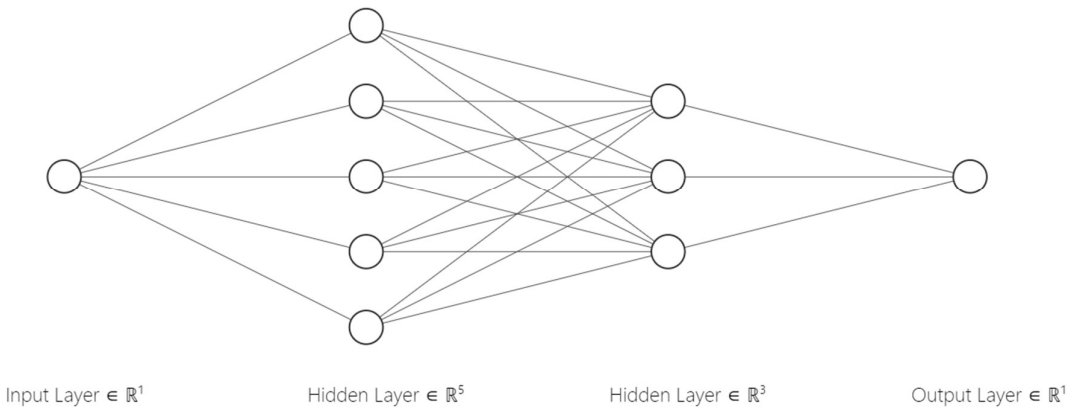

Figure 1

Figure 1 shows an artificial neural network. As seen above, the neurons are arranged in layers. The layers in between the input and output layers are called hidden layers, and these layers are known as dense layers as each neuron in the layer takes an input from all the neurons in the previous layer. This will be important later in forming an equation for the functions applied to a layer's input.

1.1 Activation, weights, and bias

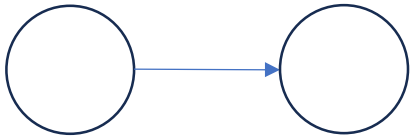

Figure 2

Figure 2 shows a neural network with one input node and one output node.

Each neuron performs two operations on the input it receives – a weighted sum of inputs and an activation function. The output is then passed onto the next node in the network – this is shown by the arrow.

Let us assume that we have an input variable x. The neuron first multiplies this input by a weight and adds a bias value. I will use the variable r to denote the result of a neurone's calculations to reserve y for the network's output.

$$r = wx + b$$

where w is the weight, and b is the bias.

This is immediately recognisable as a linear regression function. However, we cannot effectively model real-world data with only linear relationships, therefore, we commonly implement a non-linear function to improve our modelling. This is the activation function. While some linear activation functions do exist, they are generally ineffective at modelling complex relationships. We will use two examples of non-linear activation functions: Rectified Linear Unit (or ReLU), and the sigmoid activation function.

The ReLU function is defined as:

$$ReLU(x) = \max(0, x)$$

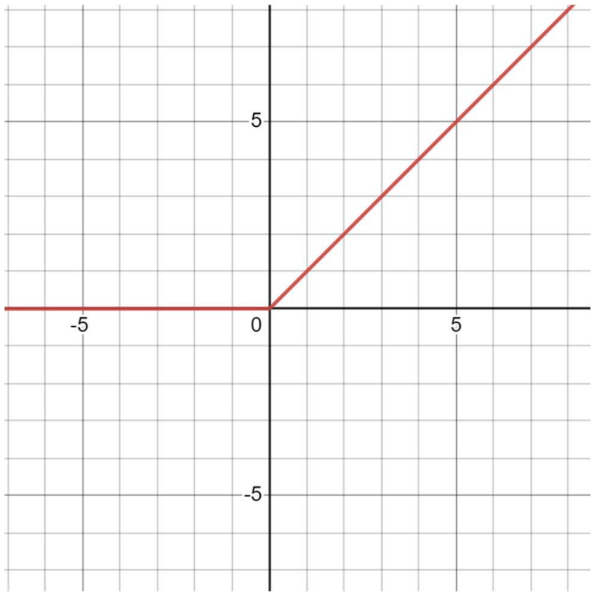

Figure 3

Figure 3 shows the ReLU function on a graph. It deactivates the neuron for negative values – or values below a given threshold.

The sigmoid activation function is defined as:

$$sigmoid(x) = \frac{1}{1 - \exp(-x)}$$

$$\exp(x) = e^x$$

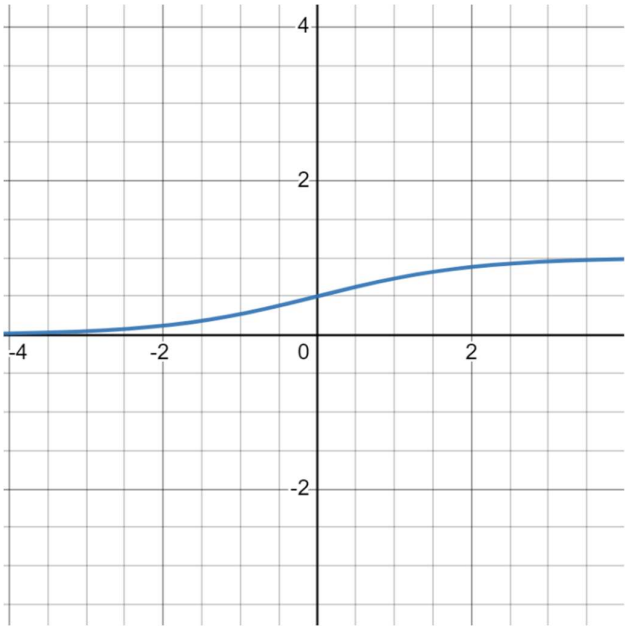

Figure 4

This is also known as the squashing function – it turns them into outputs ranging from 0 to 1 – with more positive values producing outputs towards 1 and more negative values producing outputs towards 0. This means that it is useful for classification and probability prediction tasks.

Now that we have examined some examples of non-linear activation functions, we can amend our original equation for the neuron's calculations.

$$r = a(wx + b)$$

where a is the activation function.

This seems simple right now, but we rarely deal with single variables, so let's investigate how the neuron works when taking an input with multiple variables.

1.2 Multivariate linear regression

When dealing with inputs that have multiple variables, the neuron moves from using a univariate linear regression to a multivariate linear regression.

$$r = a(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b)$$

$$w \in \mathbb{R}, n \in \mathbb{N}$$

$$r = a(\sum_{i=1}^{n} w_ix_i + b)$$

This is also known as a weighted sum, or linear combination, of the inputs. We can more compactly express the above equation by introducing linear algebra and matrices into our working. We will now rewrite the multiple values of w and x into single vectors.

$$\boldsymbol{x} = [x_1, x_2, \dots, x_n] \in \mathbb{R}^n$$

$$\boldsymbol{w} = [w_1, w_2, \dots, w_n] \in \mathbb{R}^n$$

Our equation now becomes:

$$r = a(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

with the dot in between **w** and **x** showing a linear combination.

We have only covered a neural network with layers containing only one neuron each. Now, we will consider a much more common neural network architecture where layers may have multiple neurons.

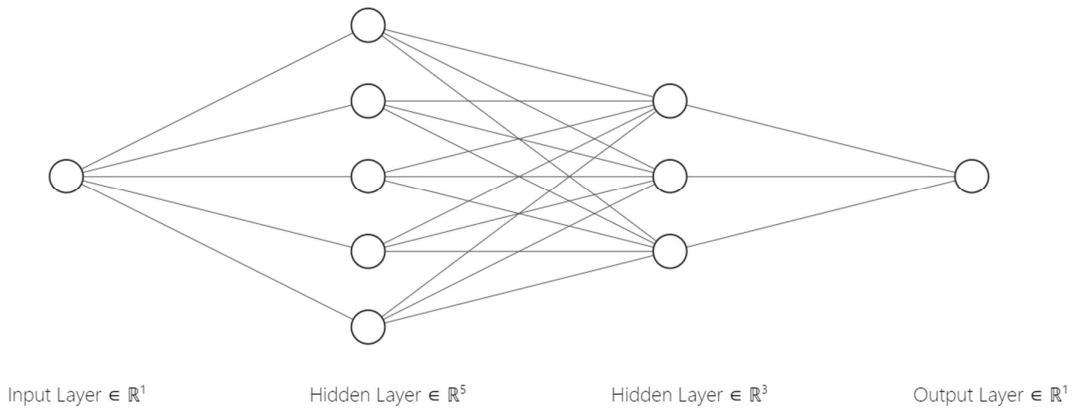

Figure 1 repeated

We will now apply the previous equation to an entire layer. We will take the second hidden layer, or the third layer of the network in Figure 1. Each neuron in the layer will have five weights – one for each neuron's output in the previous layer, and one bias value. Therefore, we can represent the layer's weights and biases as a matrix and vector respectively.

$$\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5]$$

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,5} \\ \vdots & \ddots & \vdots \\ w_{3,1} & \cdots & w_{3,5} \end{bmatrix}$$

$$\boldsymbol{b} = [b_1, b_2, b_3]$$

Each row of the weights matrix $\boldsymbol{W}$ represents the row vector of weights for a corresponding neuron in the second hidden layer. Now, we can use a single equation to represent all of the mathematical operations in the layer.

In this case, the vector **r** will have 3 values – one for each neuron's output.

$$\boldsymbol{r} = a(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

We have now shown how neurons and layers in a neural network process inputs to form outputs – however we have not yet discovered how they facilitate learning and pattern recognition in data. The weights and biases used in each layer are key to this – and they can be referred to as parameters. Parameters are generally variables in a model that are learned from data and updated during the training of a model. Next, we will explore a concept called gradient descent to understand how neural networks learn and find the best weights and biases to model the data.

2 Optimization and loss

The way in which neural networks decide what values to use for its weights and biases is called optimization. There are various optimization algorithms, and we will specifically examine the (stochastic) gradient descent algorithm.

2.1 Loss

Loss, or error, is a simple concept – how far the prediction is from the true value. This differs between regression and binary classification (true or false) problems. An example of a regression loss metric is mean squared error (MSE), defined as:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

where $\hat{y}_i$ refers to the prediction of a y value from the network's output, and y_i refers to the true value of y for that data point.

An example of a classification loss metric is log loss (also known as binary cross-entropy), which measures the models performance by indicating how close the prediction probability is to the true value. It is defined as:

$$L = -\frac{1}{n}\sum_{i=1}^{n} y_i \log\big(p(y_i)\big) + (1 - y_i)\log\big(1 - p(y_i)\big)$$

where $p(y_i)$ denotes the predicted probability of the outcome being true, therefore $1 - p(y_i)$ is also the predicted probability of the outcome being false.

It is important for us to examine how the graphs look for each of these loss functions because it will allow us to perform the next step of the weight optimization process.

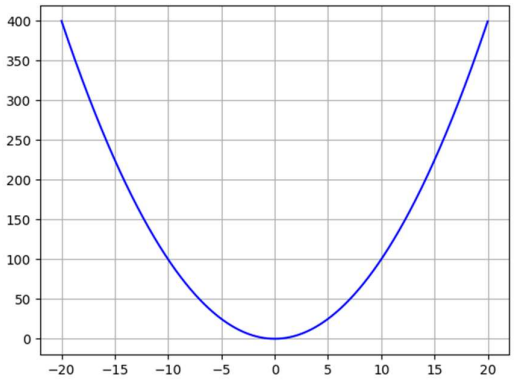

Figure 4.1

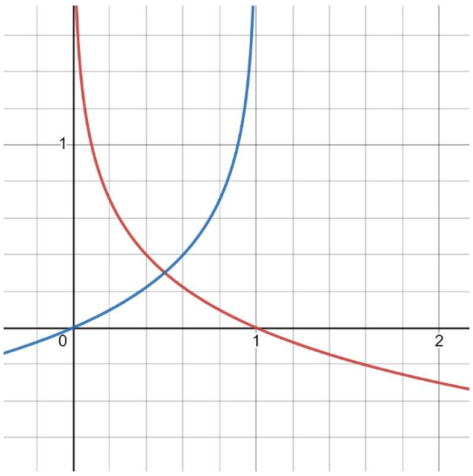

Figure 4.2

Figure 4 shows the two graphs for the loss functions – with mean squared error on the left and log loss on the right. Both functions are differentiable, which is important in the process of optimizing weights through gradient descent. It is important to note that both functions have minimum values that can be reached. They can also be described as convex functions.

2.2 Gradient Descent

The gradient descent algorithm works by calculating the derivative of the loss function with respect to the parameters of the neural network, and then iteratively updating the weights until they are optimised. This happens during the training of a neural network. The loss function is considered with the parameters as an input - $L(\theta)$ where θ refers to the parameters (weights and biases) of the model. By calculating a gradient – the model can change the weights to approach a local minimum of the loss function. This is where the model performs best, and the weights are optimised.

The gradient function is referred to as:

$$\nabla L(\theta)$$

A learning rate is used to control how much the weights are changed to minimise loss. If the learning rate is too high, the algorithm may not converge to the minimum. On the other hand, if the learning rate is too low, the algorithm may take too long to converge. The equation below represents how the weights are updated after each iteration of the gradient descent algorithm.

$$\theta_{new} = \theta - \eta \nabla L(\theta)$$

where η denotes the learning rate.

Once the weights are optimised, they are used to predict an outcome given a new input value.

Conclusion

The neural network is a powerful model that can be applied to many real-world problems – including but not limited to computer vision, natural language processing and stock price forecasting. We have investigated a simple feedforward network, however there are many more types of networks that including convolutional neural networks and recurrent neural networks that have unlocked new possibilities for the world of AI and machine learning.