Neural Networks

Oliver Temple

March 4, 2022

Abstract

Now commonplace in todays world, neural networks are becoming increasingly popular. This paper will explore the basic concepts behind neural networks, their history, how they are used in the real world and how the most simple neural networks work at a low level. The paper will conclude by exploring how to code a neural network from scratch to recognize hand written digits from 0-9.

Neural networks are complex computational models that have many uses. Every one of the numerous different types of neural networks each has its own pros and cons which suits it for one or another of many possible tasks including computer vision, classification and prediction. The structure of neural networks can vary dramatically for different models, allowing for a wide range of applications. The fluidity of neural network structure allows them to adapt to a wide range of tasks.

Introduction

What is a neural network?

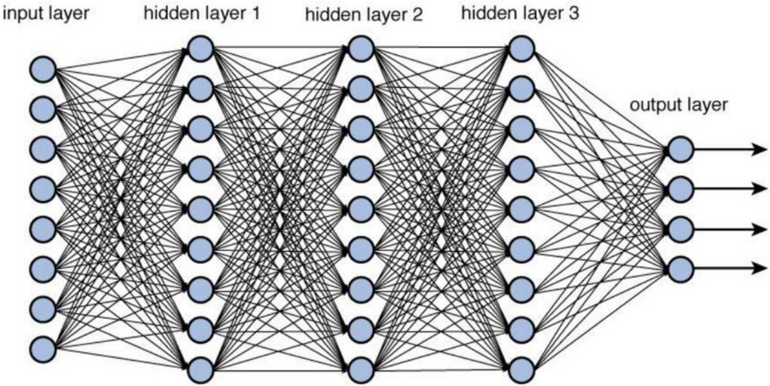


Figure: Neural Network

Introduction

Different types of neural networks

- Perceptron
- Feed Forward Neural Network
- Multi-Layer Perceptron
- Convolutional Neural Network
- Recurrent Neural Network
- Long Short-Term Memory

Introduction

Different types of neural networks

All the different network types described below would of course be implemented in software, as part of a computer program. When modeling a neural network using software, it can have a number of different inputs $x_1, x_2, x_3 ... x_n$, which are assembled into a vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$. Likewise, the weights are also assembled into a vector (or a matrix in the case of more complicated models) $\mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix}$

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2n} \end{bmatrix}$$

Different types of neural networks

Perceptron

Where $\mathbf{y}$ is the output, $\mathbf{W}$ is a vector of the weights, $\mathbf{x}$ is a vector of the inputs, b is the bias and σ is the activation function.

Different types of neural networks

Perceptron

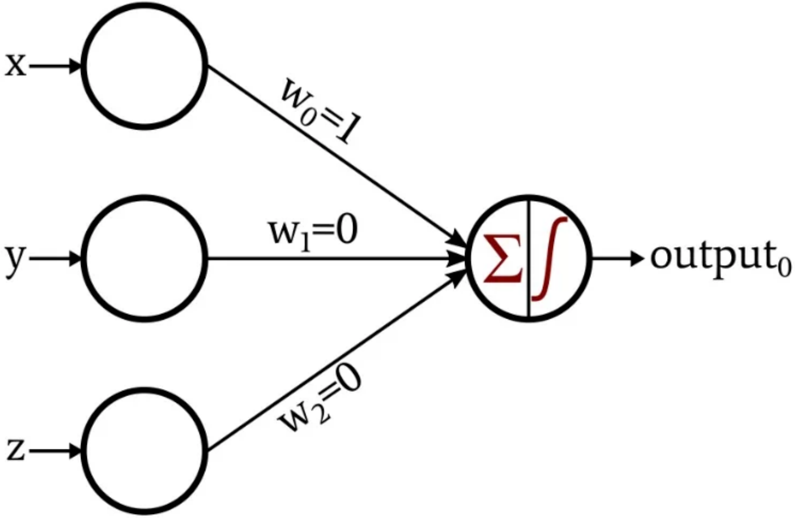


Figure: Perceptron

Different types of neural networks

Feed Forward Neural Network

$$\mathbf{y}_n = \sigma(\mathbf{W}_n \cdot \mathbf{y}_{n-1} + \mathbf{b}_n) \tag{2}$$

Where $\mathbf{y}_n$ is the output vector of the nth layer, $\mathbf{W}_n$ is the weight matrix of the nth layer, $\mathbf{y}_{n-1}$ is the output vector of the previous layer, $\mathbf{b}_n$ is the bias vector of the nth layer, and σ is the activation function.

Different types of neural networks

Convolutional Neural Network

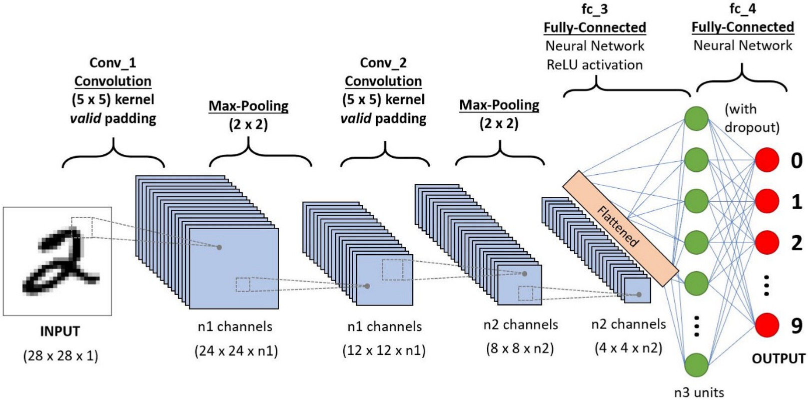


Figure: Convolutional Neural Network

The History of Neural Networks

$$\texttt{Weight Change} = \texttt{Pre-Weight line value} \times \frac{\texttt{Error}}{\texttt{Number of Inputs}} \tag{3}$$

Examples of Bias in AI

PortraitAI Art Generator

Currently, the AI portrait generator has been trained mostly on portraits of people of European ethnicity. We're planning to expand our dataset and fix this in the future. At the time of conceptualizing this AI, authors were not certain it would turn out to work at all. This generator is close to the state-of-the-art in AI at the moment. Sorry for the bias in the meanwhile. Have fun!

Examples of Bias in AI

Twitter Photo Cropping

Our team did test for bias before shipping the model and did not find evidence of racial or gender bias in our testing. But it's clear from these examples that we've got more analysis to do. We'll continue to share what we learn, what actions we take, and will open source our analysis so others can review and replicate.

How Neural Networks Work

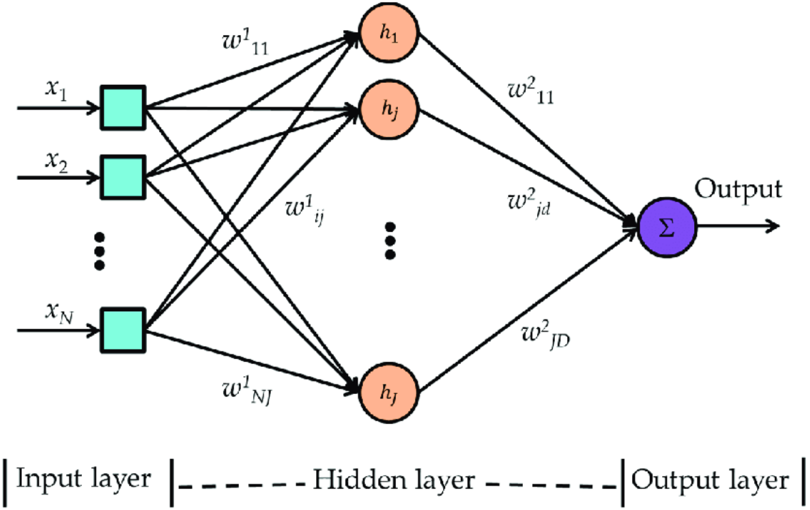


Figure: Multilayer Perceptron

The purpose of forward propagation is to get an output (or prediction) from our neural network. To calculate it we use the vector dot product (Appendix **??**) of the inputs and the weights, and add the bias. The output of the n^{th} layer can be calculated using the formula in the equation **??**.

How Neural Networks Work

Forward Propagation

$$\mathbf{y}_n = \sigma(\mathbf{W}_n \cdot \mathbf{y}_{(n-1)} + \mathbf{b}_n) \quad (4)$$

where $\mathbf{W}_n$ is a matrix of the weights for the n^{th} layer, $\mathbf{y}_{(n-1)}$ is the output vector of the previous layer (the input of the network for the fist layer) and $\mathbf{b}_n$ is a vector of the biases for the n^{th} layer.

How Neural Networks Work

Activation Functions

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU

Activation Functions

Sigmoid

The sigmoid function(Figure **??**) takes any input, x and translates it to a value between 0 and 1. The equation is:

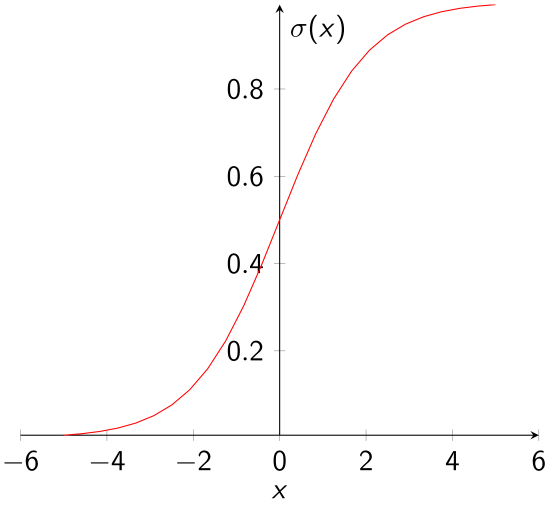
Activation Functions

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

Activation Functions

Sigmoid



Activation Functions

Tanh

Similar to the sigmoid function, the tanh function(Figure **??**) takes any input, x and translates it to a value between -1 and 1. The equation is:

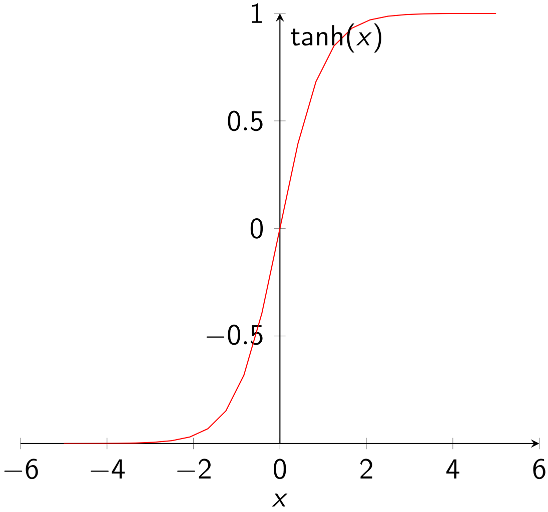
Activation Functions

Tanh

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6}$$

Activation Functions

Tanh



Unlike the sigmoid function and the tanh function, the ReLU function(Figure **??**) takes any input, x and if it is negative, it returns 0, otherwise it returns the input. The equation is:

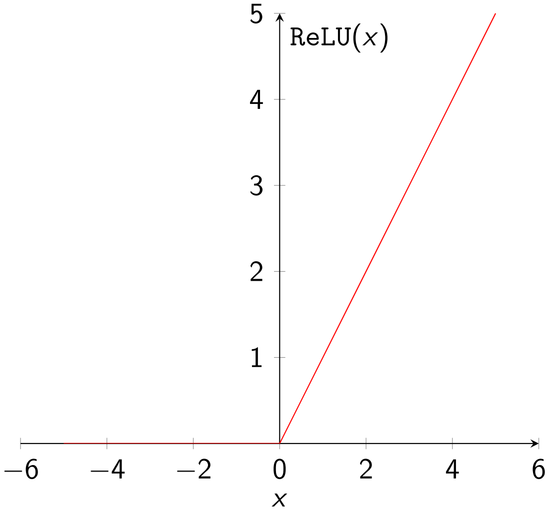
Activation Functions

ReLU

$$\texttt{ReLU}(x) = \max(0, x) \tag{7}$$

Activation Functions

ReLU



The Leaky ReLU function(Figure **??**) takes any input, x and if it is negative, it returns a scaled down input, otherwise it returns the input. The equation is:

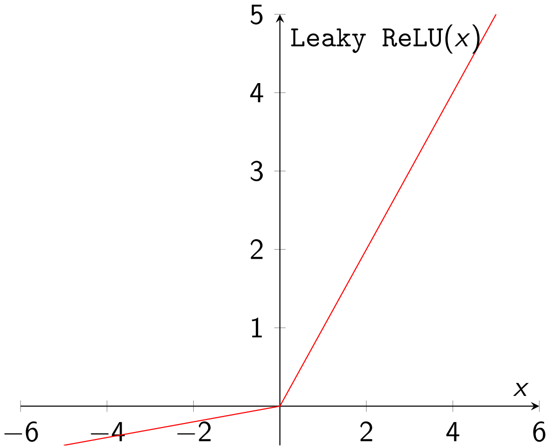
Activation Functions

Leaky ReLU

$$\texttt{Leaky ReLU}(x) = \max(0.1x, x) \tag{8}$$

Activation Functions

Leaky ReLU



- Mean Squared Error (MSE)
- Cross Entropy Loss (or Log Loss)

Loss Function

Mean Squared Error

$$\texttt{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{9}$$

Loss Function

Mean Squared Error

where n is the number of samples we are testing against, y is the desired output of the network, and $\hat{y}$ is the actual output of the network.

Loss Function

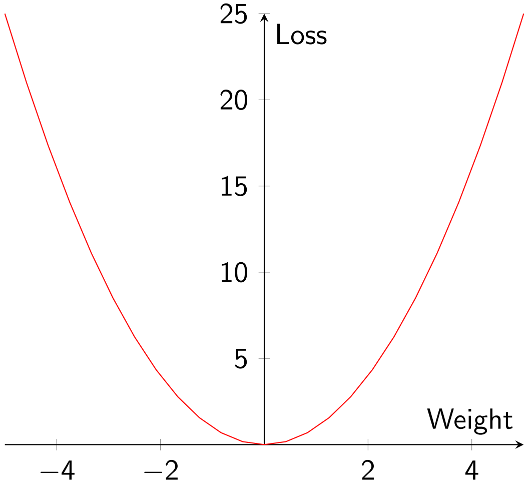
Cross Entropy Loss

$$\texttt{CEL} = -\frac{1}{n}\sum_{i=1}^{n}(y_i \times \log(\hat{y}_i)) \tag{10}$$

where n is the number of samples we are testing against, y is the desired output of the network, and $\hat{y}$ is the actual output of the network.

How Neural Networks Work

Backpropagation



1. Propagate all the values in the input layer through the network (Forward Propagation).
2. Update the weights and biases of the network using the loss function (Back Propagation).
3. Repeat until the accuracy of the network is satisfactory.

Defining the Model

Activation Function

$$\sigma'(x) = x(1 - x) \tag{11}$$

Coding a Neural Network

Training the Model

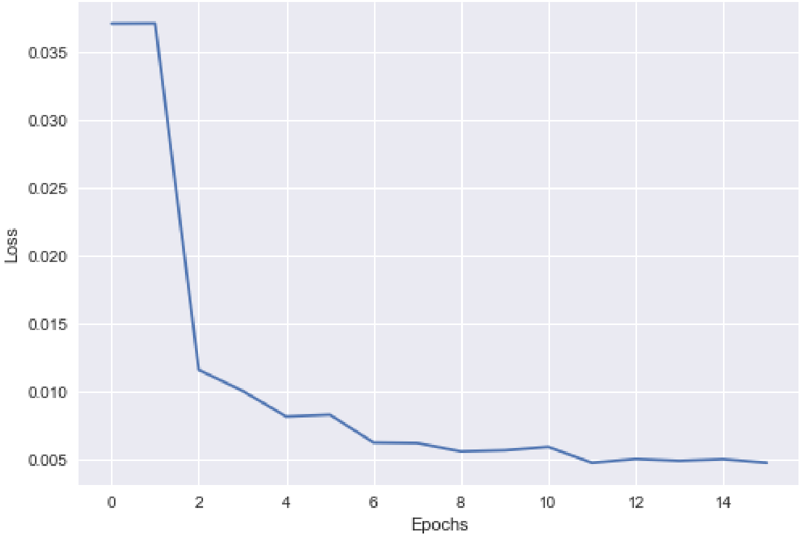


Figure: Loss of neural network over epochs

This phenomenon is known as the curse of dimensionality. Of particular concern is that the number of possible distinct configurations of a set of variables increases exponentially as the number of variables increases

Coding a Neural Network

Testing the Model

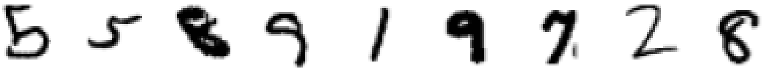


Figure: Sample predictions of neural network

Appendix

Vector Dot Product

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Appendix

Vector Dot Product

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

Appendix

Vector Dot Product

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nn} \end{bmatrix}$$

Appendix

Vector Dot Product

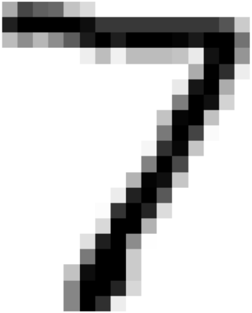
$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Appendix

Vector Dot Product

$$\begin{bmatrix} x_{11}y_1 + x_{12}y_2 + \cdots + x_{1n}y_n \\ x_{21}y_1 + x_{22}y_2 + \cdots + x_{2n}y_n \\ \vdots \\ x_{n1}y_1 + x_{n2}y_2 + \cdots + x_{nn}y_n \end{bmatrix}$$

Appendix
MNIST Dataset



Appendix

MNIST Dataset

output	1.3e-05	3.4e-06	0.0025	0.0025	7.9e-09	0.00033	2.9e-06	**0.99**	0.00062	2.5e-05
prediction	0	1	2	3	4	5	6	**7**	8	9

Appendix

Python 3.9.10 Code

- History of Neural Networks https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html
- Bias in Artificial Intelligence https://www.lexalytics.com/lexablog/bias-in-ai-machine-learning
- Loss Functions https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-mach
- AI generated hypotheses https://www.scientificamerican.com/article/ai-generates-hypotheses-human-scientists-have-not-thoug
- Curse of dimensionality Page 155, Deep Learning