Neural-Networks

December 1, 2020

B00139079@mytudublin.ie
Technological University Dublin
Ireland

Abstract

This paper discusses how neural-networks work, showing how the outputs are produced through the process of forward propagation and how the model is trained using the back-propagation method. Then, the paper illustrates autoencoder neural networks. A practical application of neural-networks is analysed and discussed to show in practice strengths and limits of the algorithms, discussed in the final conclusions.

keywords: Neural-networks, backpropagation, image recognition

1 Introduction

Neural networks (NN) is an algorithm usually referred to as artificial intelligence, whose main characteristics is that they start with no knowledge of their assigned task [1], and little by little learn the rules through different attempts and errors. The algorithm is based on the ability of the human brain to recognize patterns. The main idea behind is that after facing a sufficient amount of specific instances [1], the algorithm will finally learn the general underlying rules: e.g. in the field of image recognition, the algorithm can learn to identify a specific image by analysing several examples of it and using the results to identify such image in other pictures. The algorithm does this without any previous knowledge of the features of that specific image. After several trials, it becomes able to create identifying characteristics from the previous examples that it has processed.
Neural networks became popular relatively recently [1], despite the fact they have been around since 1943, thanks to the implementation of technological improvements which increased the model's competitiveness [1]. The algorithm showed to reach almost human performances in many fields [1], such as: image recognition, speech recognition, medical diagnosis, self-driving cars, etc.

2 Neural-Networks (NN)

2.1 Neuron's Output

In biology, a neural network is something inside the human brain [2]. It is a network of neurons and synapses that recognises patterns after being triggered by chemical impulse. The principle

used in machine learning is similar to this biological model [2]. A NN can be split in multiple layers composed by neurons. The neurons in the different layers create connection among each other, similarly to the synapses in the human brain [2], from here the name artificial neural network (ANN). A neural-network can have a different number of layers, considering a neural-network with the structure as in figure 1, we can identify: an input layer, an hidden layer and an output layer.

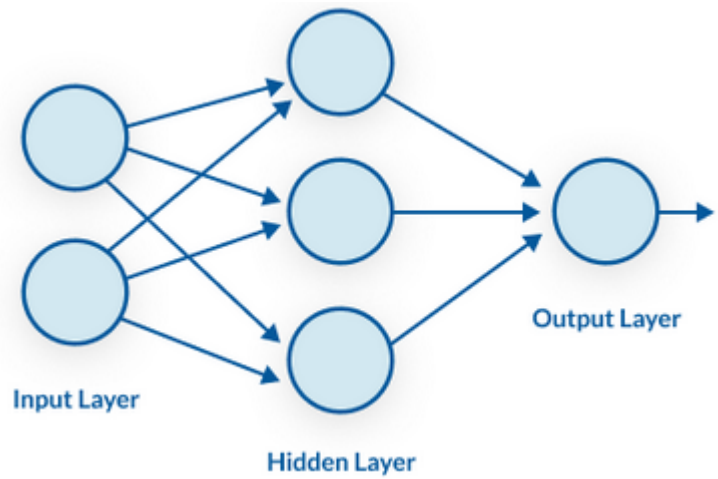

Figure 1. A simple neural network, https://missinglink.ai/guides/neural-network-concepts.

The different layers can have different number of neurons, which are the units interconnected to each other. The data are plugged into the input layer and the outcome gets out from the output layer, as a value. The final outcome is a result of the intermediate steps in the hidden layers, where the initial data are processed and transferred to the adjacent neurons. In the input layer each single neuron represents a single column within the data that will pass through the model. The neurons are then connected to every single neuron in the next layer, called hidden layer. The connections between layers transfer the output from the previous neuron, as input, to the receiving neuron. The transferred output is the result coming from the connections where the input is processed by the *weights* and *bias* parameters. The result of each neuron is calculated as showed in equation below:

$$y = b + \sum_{j=1}^{n} w_j x_j \tag{1}$$

- y is the neuron's output
- x_j is the input from the previous j-th neuron
- b is the neuron's bias
- w_j is the weight of the connection

As showed before, the connections are weighted and the weights represent how much important a connection is to have a correct final output [2]. The weights show that some connections are more important than others [2]. To explain it, we can think about the example of recognizing a clothes from a picture. The input is the picture of the clothes, which can have different characteristics: silhouette, size, colour, etc. Not all of them have the same importance to recognize the clothes, the colour might help to understand the material of the clothes but it is not as important as the silhouette to understand what kind of clothes is [2]. Hence, the weight associated is different. The bias is an additional parameter added to the weighted sum. This parameter helps in shifting the result to better fit the data and consequently to increase the flexibility of the model [3].

The result coming from equation (1) is then transferred to an *activation function* which determines the output of that specific neuron. The activation functions convert a straight line in a non-linear curve. Their outputs are defined between a lower limit and upper limit, and so it will be the neuron's output. There are different type of activation function, such as:

- *Sigmoid function*, has a typical S shape. An example of sigmoid function is the logistic function, for which the higher the input is the closer to 1 the output, and the smaller the input is the closer to 0 the output. So in this case 0 is the lower limit and 1 the upper limit, as showed in figure 2.

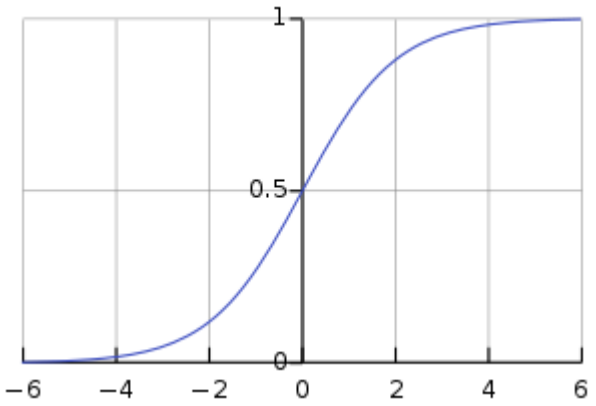

Figure 2. Logistic curve, Wikipedia.

- *Rectified Linear Units (ReLU)*, it transforms the input in the maximum between 0 and the input itself. So, if the input is negative the output is 0. The function has the shape as in figure 3 and it can be defined by the following formula: $f(x) = max(0, x)$

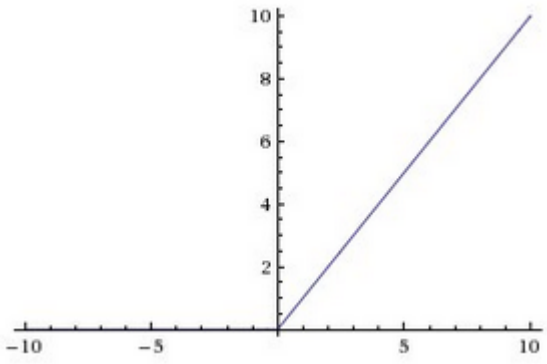

Figure 3. Rectified Linear Units, Kaggle.

In the human brain some neurons are activated by some input and others neurons are not activated. The same happens in an ANN: when using a sigmoid function, the closer to 1 is the neuron's output the more activated that neuron is, the closer to 0 the less activated it is. When using a ReLu, the more positive the neuron's output is the more activated the neuron is, the closer to 0 the less activated it is.
So, the whole process that happens in a neuron can be summarized as in figure 4. The sum of the weighted inputs is added to the bias, the result is fed into an activation function which gives the neuron output.

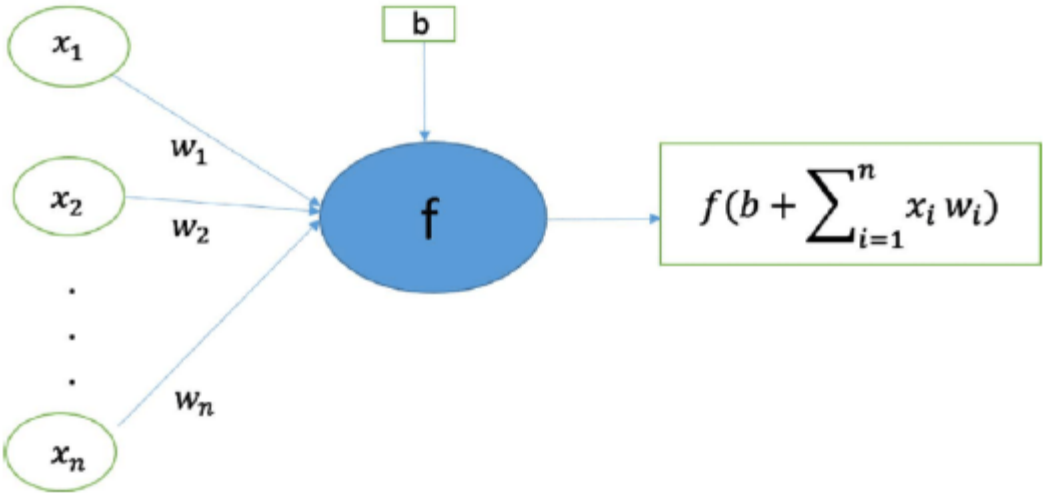

Figure 4. Output calculation for a single neuron [4].

2.2 Training the model

The model is initialized using random numbers as weights, after feeding the input of the network, the model produces the output which will be different from the actual output. The function that quantifies the error between the model output and the actual output is the *cost function*.
A single pass of the data through the model is called *epoch*, after a single epoch the same data will pass over and over through multiple epochs. At the end of each epoch the model will accumulate all the single error for each input. The aim of training the model is to adjust the weights in order to minimise the cost function. To minimise this function we can use an optimiser like Stochastic Descent Gradient (SDG), which aim will be to minimise a given cost function. So SDG will assign the weights in order to have a cost function as close to 0 as possible. The way the gradient minimises this error is calculating the cost function derivative with respect to each of the weights previously set

$$SDG = \frac{d(cost)}{d(weight)}$$

After computing the SDG, this value is multiplied by a *learning rate*, λ, which is a number between 0 and 1. Then, a single weight is updated as showed in the following formula:

$$w_j^{k+1} = w_j^k + \lambda * SDG$$

where:

- k is the number of epoch
- j is the j-th neuron
- w_j^k is the vector that contains all the weights obtained from the previous epoch for the j-th neuron

The previous formula is applied for every single weight in the network, at each epoch.

2.3 Stochastic Descent Gradient (SDG)

The computed gradient will be different for every weight, as calculated with respect to each weight. The updated weights will be closer and closer to their optimised value while SDG works to minimise the cost function. The cost function can be expressed as a combination of functions, if we assume the minimum squared error as cost function, for the all network it is expressed as follows

$$C = \sum_{j=1}^{n} (y_j - a_j^{(L)})^2 \tag{2}$$

where

- $a_j^{(L)}$ is the activation output in the output layer L for the j-th neuron
- y_j is the actual output of the j-th neuron

Hence, for a single neuron it will be:

$$C_j = \left(y_j - a_j^{(L)}\right)^2$$

As y_j is a constant, the cost function can be expressed just as a function of the activation output

$$C_j\left(a_j^{(L)}\right)$$

In turn, we have seen that the activation output of the j-th neuron in the output layer L is an activation function of the input for the j-th neuron, z_j

$$a_j^{(L)} = g^{(L)}\left(z_j^{(L)}\right) \tag{3}$$

The input, for the j-th neuron in the L layer, can be expressed as a function of all the weights connected to this neuron from the previous layer, $w_j^{(L)}$

$$z_j^{(L)}\left(w_j^{(L)}\right)$$

Hence, the cost of a specific neuron is a function of the activation output for that neuron, which in turn is a function of the input to that neuron, and this input is a function of the weights that connect the neurons in the previous layer to this neuron. As expressed in the following formula:

$$C_j = C_j\left(a_j^{(L)}\left(z_j^{(L)}\left(w_j^{(L)}\right)\right)\right)$$

The total cost C is the sum of the costs of each output neurons

$$C = \sum_{j=1}^{n} C_j$$

Hence, the total cost of the network is a composition of multiple functions. According to the *chain rule* the cost's derivative with respect to the weight is the product of the multiples functions

derivatives:

$$\frac{\partial C}{\partial w_j^{(L)}} = \left(\frac{\partial C}{\partial a_j^{(L)}}\right)\left(\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\right)\left(\frac{\partial z_j^{(L)}}{\partial w_j^{(L)}}\right)$$

We can look at each term individually, according to (2) the first term can be expressed as:

$$\frac{\partial C}{\partial a_j^{(L)}} = \frac{\partial}{\partial a_j^{(L)}}\left(\sum_{j=1}^{n}(y_j - a_j^{(L)})^2\right)$$

And calculating the derivative the final result will be:

$$\frac{\partial C}{\partial a_j^{(L)}} = 2\left(y_j - a_j^{(L)}\right)$$

This means that SDG depends on the difference between actual output and model output. If the model output is smaller than actual output, $y_j - a_j^{(L)} > 0$, then the difference is a positive number and the weight will be increased. Viceversa, if the model output is greater than the actual output the difference is a negative number and the weight will be decreased.
As showed in (3), $a_j^{(L)}$ is a function of z (the neuron's output), so the second term can be expressed as:

$$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = \frac{\partial}{\partial z_j^{(L)}}\left(g^{(L)}\left(z_j^{(L)}\right)\right) = g^{'(L)}\left(z_j^{(L)}\right)$$

The third term, we know is equal to the weighted sum of the inputs from the previous layer. Expanding the sum and taking the derivative of each weight, there will be only one term that contains w_j, so the final result will be the input value from the previous layer:

$$\frac{\partial z_j^{(L)}}{\partial w_j^{(L)}} = a^{(L-1)}$$

To summarize, according to the chain rule the SDG is the product of the derivatives of the functions that compose the cost function. Solving this derivatives, we get that SDG depends on:

- The difference between the actual output and model output
- The derivative of the activation function, which depends on the chosen activation
- The input from the previous layer

If a neuron is located in a layer that's not directly connected to the output layer, we can't use the same approach we used before. Because the cost function is not a direct function of the activation outputs anymore.
This is also visible from the network in figure 5, the activation output from the second hidden layer is located in the output layer which is directly connected to the cost function. However, the activation output of the neurons in the first hidden layer is located in the second hidden layer, which is not directly connected to the cost function.
If we want to update the weights in the first hidden layer, we can't calculate the difference from actual output and model output, because we don't know the actual output in the hidden layers.

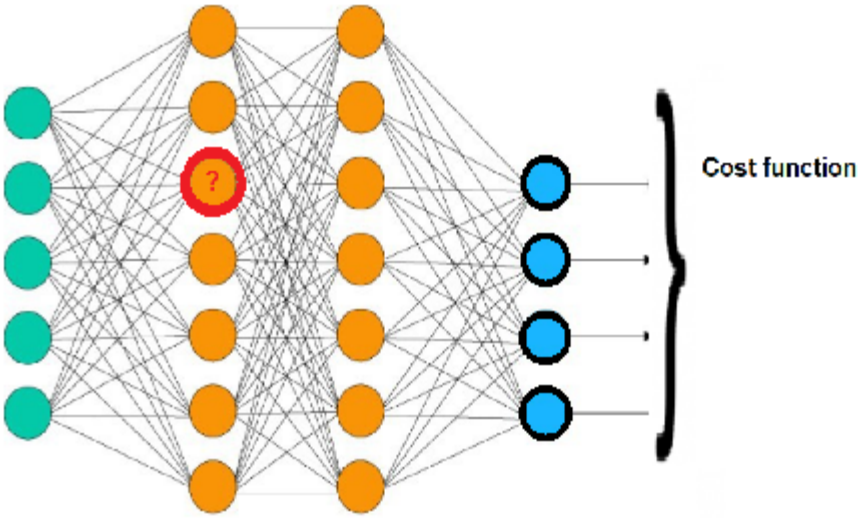

Figure 5. Neural network with multiple hidden layers, modified from https://www.kdnuggets.com/2019/02/neural-networks-intuition.html.

To calculate the error of a neuron in the first hidden layer, we have to calculate the error of the neurons in the second hidden layer as weighted sum of the weights at each connection multiplied by the errors from the output layer, figure 6.

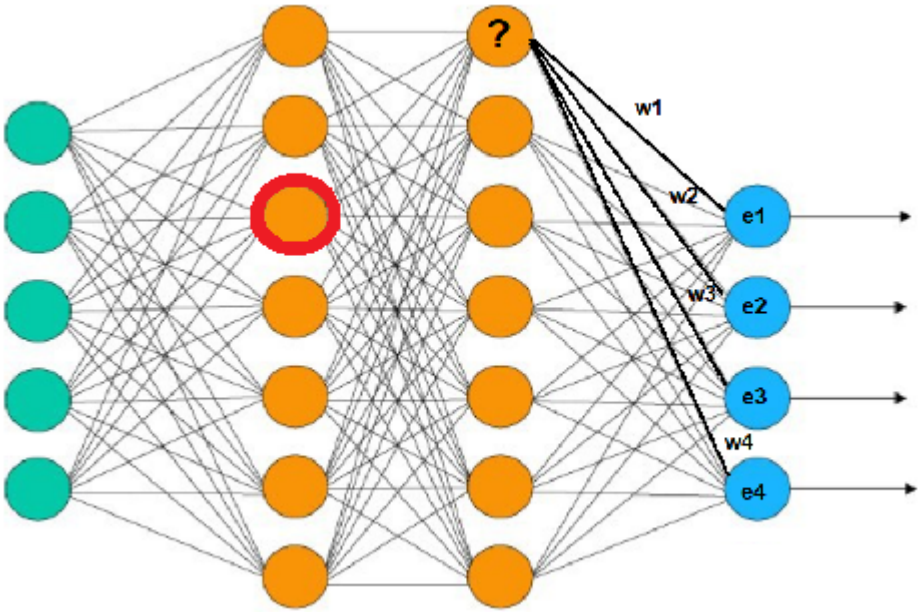

Figure 6. Error calculation in the second hidden layer, modified from https://www.kdnuggets.com/2019/02/neural-networks-intuition.html.

The new calculated errors in the second hidden layer are then multiplied by the corresponding weights in the connections with the first hidden layer, in order to calculate the error of the neuron in the first layer, figure 7. This is how the model is trained, using the SDG, the weights are every time updated to minimise the errors. This method, called Backpropagation, moves backward in order to minimise the cost function.

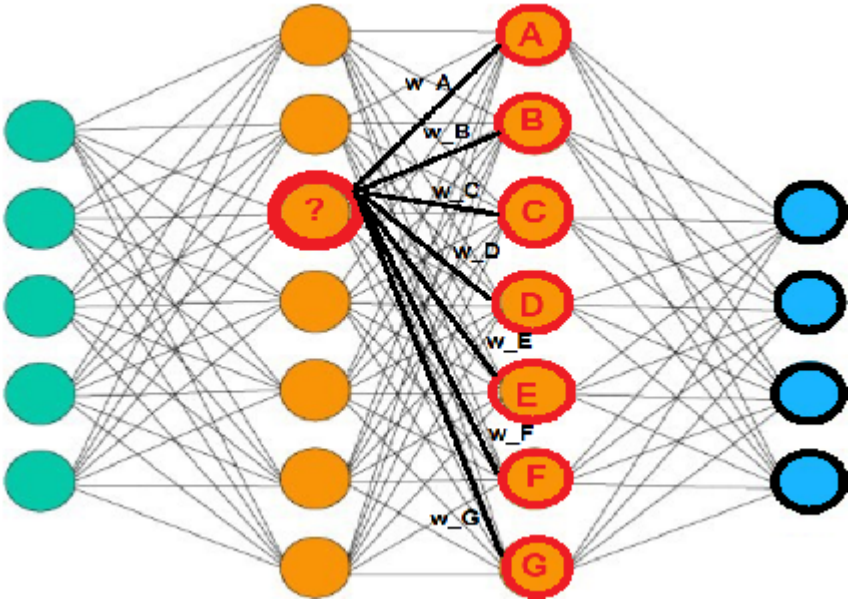

Figure 7. Error calculation in the first hidden layer, modified from https://www.kdnuggets.com/2019/02/neural-networks-intuition.html.

3 Autoencoder Neural Networks

Autoencoders are one of the most popular models in Deep Learning [5]. They learn to convert any data to code automatically. They have an encoder and a decoder which are trained together during the training phase as if they were a single model [5]. After the training, these models are separated and used to compress and decompress the data. The purpose of the autoencoder model is to compress the given data and regenerate it with as little loss as possible as seen in Figure 8.

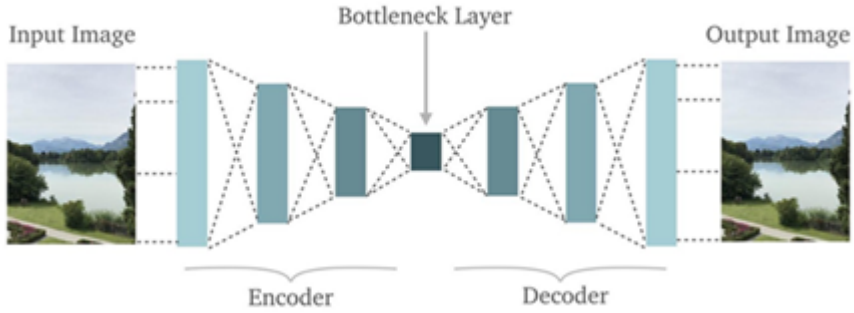

Figure 8. Autoencoder model [6]

The generated code is the output of the middle layer which is called “bottleneck layer” and it determines the size of the code [3].

3.1 Architecture of the Autoencoder model

A simple auto-encoder is a multi-layer perceptron (MLP) which consists of an input layer, an output layer, and one or more hidden layers connected to each other. The architecture is showed in Figure 9. The output layer has the same number of neurons as the input layer in order to rebuild the inputs. The aim is to minimize the difference between input and output rather than predicting the Y outputs corresponding to the given X inputs. So that they are unsupervised learning models

as they do not require labeled entries.

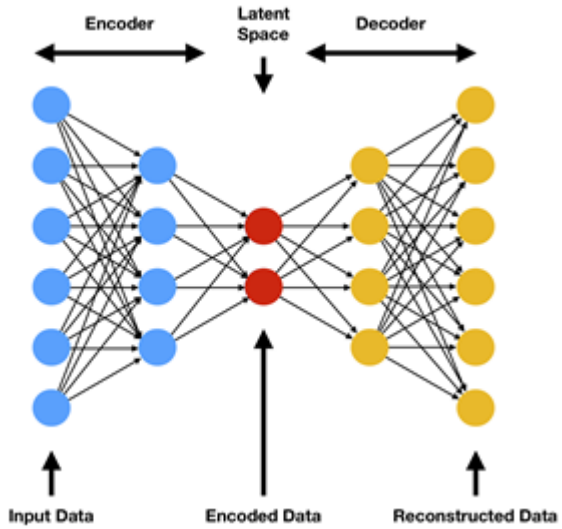

Figure 9.Autoencoder architecture [8]

An autoencoder model consists of three parts: an encoder, a bottleneck and a decoder. We represent the transition function, a function that indicates the initial input state and the final output state, of the encoder with σ and ψ is the transition function of the decoder.

$$\sigma : X \to F$$
$$\psi : F \to X'$$

Encoder which is shown with blue neurons in Figure 9, takes the d-dimensional input x $\in R^d$ = X and converts it to p-dimensional latent representation (encoded data in the latent space of the middle layer) h $\in R^p$ = F.

$$h = \sigma(Wx + b)$$

Here, h is the code in other words latent representation, σ is the activation function which can be sigmoid or rectified linear unit. W stands for weight matrix and b is for bias. We initialize W and b randomly and learn them iteratively with back-propagation.
Decoder which is shown with yellow neurons in Figure 9, converts h to the reconstructed output X', which has same shape with input X.

$$X' = \sigma'(W'h + b')$$

Here σ', W', b' are unrelated with the corresponding encoder notations σ, W, b. Objective is to minimize the reconstruction error and can be interpreted as follows by using squared errors.

$$L(X, X') = ||X - X'||^2 = ||X - \sigma'(W'(\sigma(Wx + b)) + b')||^2$$

3.2 Application areas

The autoencoder neural networks have different field of application,such as:
- Feature Extraction: Autoencoders learn to keep the most valuable information in a lower dimension while compressing the data [7].
- Denoising: It can be trained for noise removal by giving noisy data as input. For example, if we add noise synthetically to the data set and calculate the difference between the noiseless data, the model will learn to remove the noise [5].
- Image colorization: The process of converting black and white images to color images. If we have both black and white and colored versions of each picture an autoencoder model can learn the relationship between these two and colorize the black and white images [9].
- Generating new data: New data can be produced by using Variational Autoencoder (VAE), a type of Autoencoder. VAEs learn a certain probabilistic distribution on the bottleneck layer and new data can be generated from this distribution [10].

4 Handwritten Digits Recognition with Artificial Neural Network

In [11] is presented how ANN can be applied to image recognition, particularly to handwritten digits recognition. The following application can be used in different contexts such as computerized bank check numbers reading [11].
The practical case has been applied to a dataset extracted from MNIST database, 42.000 samples has been extracted to perform the experiment. They are then divided as follows: 28.000 digits images has been used for training and 14.000 to perform the test.
The aim of implementing an ANN is to recognize handwritten digits from 0 to 9. The images have low resolution, 28-by-28 pixels in grayscale, figure 8. Every single image represents a number.

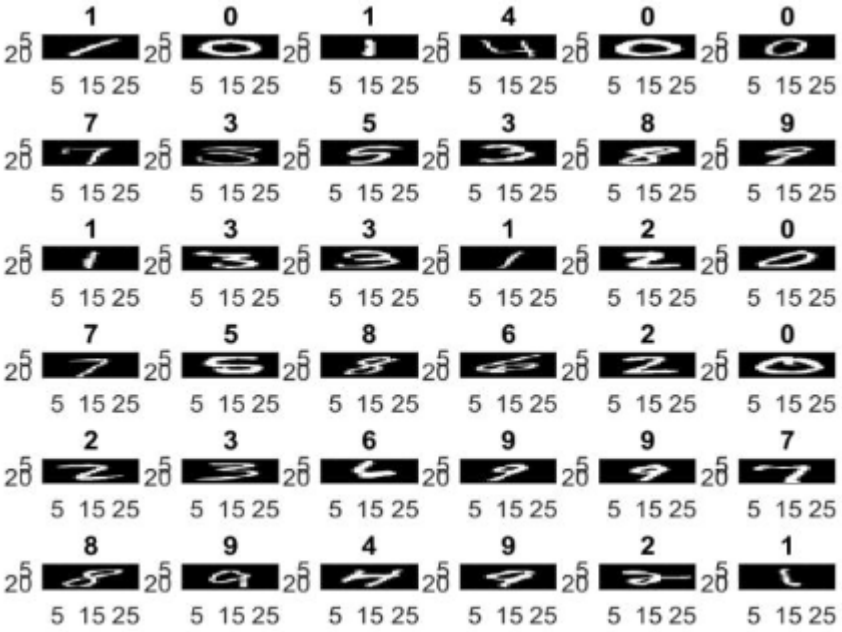

Figure 8. Sample digits from MNIST datset [11]

The raw pixels are usually used as features when working with images [11]. No feature engineering is required as it is difficult to extract features from images, text and shapes [11].
In order to design the ANN, two paremeters have been defined. One parameter represents the

number of classes (10 in our case, number from 0 to 9), the other parameter shows the classier about the utilized features. The ANN is structured as follows: an input layer with 784 input neurons (which is the result of 28x28 pixels), an hidden layer with 100 neurons and an output layer with 10 output neurons, which are the classes of handwritten digits samples.

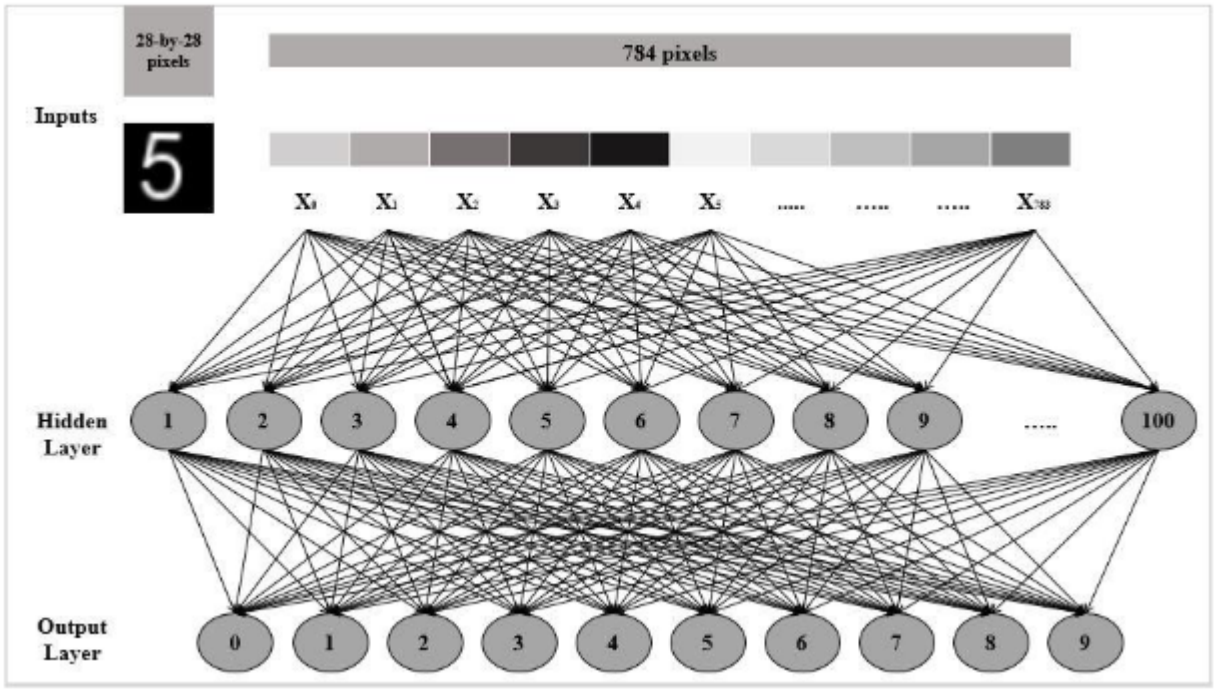

Figure 9. ANN's structure [11]

The training samples are randomly split as follows: 25.200 for training (90%), 1.400 for validation (5%) and 1.400 for testing (5%). In figure 10 it is showed the training performance, we can see that the errors are minimized at every epoch, and at epoch 107 it is registered the best validation performance with an error equal to 0.01331. Thisprevents the network from overfitting (excessive learning, which results in memorizing training samples and decreasing success in test samples).

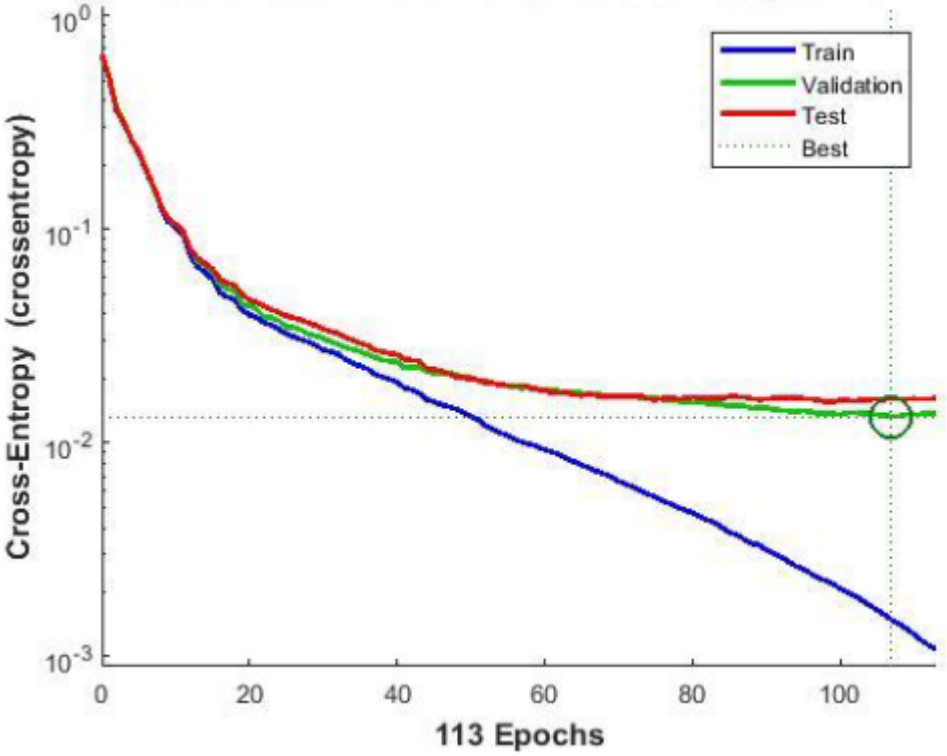

Figure 10. Training performance [11]

After training the model, testing can be applied. The format of the output matrix is the result of combining 28.000-by-10 binary matrix, as showed in figure 11.

Rows	Handwritten digits									
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	**1**	0	0	0	0	0	0	0	0	0
1	0	**1**	0	0	0	0	0	0	0	0
2	0	0	**1**	0	0	0	0	0	0	0
3	0	0	0	**1**	0	0	0	0	0	0
4	0	0	0	0	**1**	0	0	0	0	0
5	0	0	0	0	0	**1**	0	0	0	0
6	0	0	0	0	0	0	**1**	0	0	0
7	0	0	0	0	0	0	0	**1**	0	0
8	0	0	0	0	0	0	0	0	**1**	0
9	0	0	0	0	0	0	0	0	0	**1**

Figure 11. Neural network output format [11]

The number zero is the binary output (1 0 0 0 0 0 0 0 0 0), the number one is the binary output (0 1 0 0 0 0 0 0 0 0), and so on. The test is applied on the 14.000 samples originally assigned for testing. The performance are illustrated in figure 12, the rows show the output of the neural network and columns show the actual class of the digit.

The red squares represent the wrong classification and the green squares represent the correct classification according to output and target class. For example, the first cell on the upper left corner shows the number of samples which has actual label 1 and also predicted as 1. Cell to its right there is 0, it means there are no samples which predicted as 1 but actual label is 2. Seventh column in the first row shows there are 2 samples which have actual label 7 but predicted as 1. There is a 6 in the third cell of in the fifth row, this means 6 samples were predicted 5 incorrectly while they have label 3. There is a 4 in the third row eighth cell, this means 4 samples were predicted 3

while they have label 8. Last two observations imply that the most confusing recognitions are 5-3 and 3-8 for the neural network.
In the last column the green ratio shows what percentage of samples predicted to be 1 is actually 1. Red ratio shows what percentage of samples predicted to be 1 but actually have another label. In the last row the green ratio shows what percentage of samples that are actually 1 were predicted as 1 correctly and red ratio shows what percentage of samples that are actually 1 were predicted as another class incorrectly. The diagonal of this matrix shows true predicted samples. In the bottom right corner, the green percentage in the blue cell (99.60%) shows the accuracy of the model and the red percentage shows the error rate.

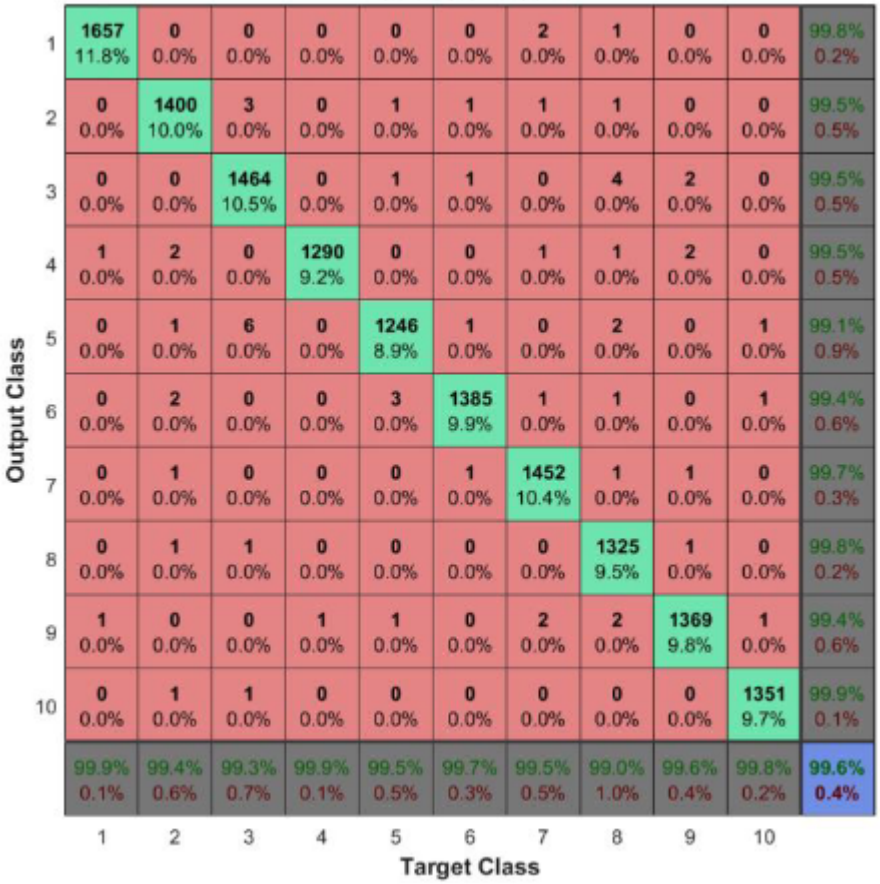

Figure 12. Test confusion matrix [11]

The result can be compared with other classifier methods for digits and image recognition, which use features extraction methods too. The paper doesn't show them into deep, but it presents the final results showed in figure 13. We can notice that the ANN's results have the highest accuracy and we can conclude that ANN classifies better than SVM. Given the high level of accuracy, we could assume that the training and validation dataset were well representative of the actual data we wanted to predict.

Features extraction method	Classifier	Accuracy (%)
R-HOG	SVM	95.64
HOG	SVM	81.00
HOG	SVM	83.60
HOG	ANN	97.33
CCH	SVM	98.48
CNN	CNN+SVM	94.40
Image Pixels	ANN	99.60

Figure 13. Performance between proposed and other existing methods [11]

5 Conclusions

In the previous section was showed that ANN can provide good results in predicting handwritten digits. Further studies may be focused on trying to predict more complex and different from each other images, to see if the model keeps having an acceptable lavel of accuracy. Based on what we have showed in sections 2.1, 2.2. and 2.3, we could conclude that it is not always easy to implement an ANN because of its "black box" nature. Furthermore, the model strongly depends on data quality. This influences the expensiveness of the computation, requiring long time for training in some cases.
However, this algorithm has the advantage of predicting non-linear phenomena accuratively. Furthermore, as mentioned in section 2.1, it is able to recognise the most important features assigning higher weights in the network.

References

[1] Bogdan Wilamowski. *Neural network architectures and learning algorithms*. Industrial Electronics Magazine (2010).

[2] Mikko Mäkelä. *Understanding, Building and Analyzing Basic Convolutional Neural Networks*. Metropolia University of Applied Sciences, Bachelor's Thesis (2019).

[3] Hu Wei. *Towards a Real Quantum Neuron* (2018).

[4] Hung Dao. *Image classification using convolutional neural networks* Oulu University of Applied Sciences, Bachelor's Thesis, (2020).

[5] Goodfellow, Ian. *Deep learning* Vol. 1. No. 2. Cambridge: MIT press, 2016.

[6] *https://github.com/iperov/DeepFaceLab/issues/636*

[7] Kramer, Mark A. *Nonlinear principal component analysis using autoassociative neural networks.* AIChE journal 37.2 (1991).

[8] *https://www.compthree.com/blog/autoencoder/*

[9] *https://www.geeksforgeeks.org/colorization-autoencoders-using-keras/*

[10] Diederik P., Max Welling *Auto-encoding variational bayes* arXiv preprint arXiv:1312.6114 (2013)

[11] Kh Tohidul Islam, Ram Gopal Raj, Ghulam Mujtaba, Henry Friday Nweke. *Handwritten Digits Recognition with Artificial Neural Network* International Conference on Engineering Technologies and Technopreneurship (September 2017)