Neural Networks & Image Processing
A Technical Analysis

Conor Lorsung

***Abstract*—Neural networks and image processing has applications in text recognition, saliency mapping, and resolution enhancement. We cover the foundations of the artificial neuron model and different techniques for implementing them. This includes: the input layer, the aggregation function, and the transfer functions. We explain three different transfer functions: threshold, linear, and sigmoid. Afterwards, we examine how these neurons can form a network. In the network, we cover input layers, hidden layers, and output layers, comparing and contrasting the purposes of each. Next we look at a technique used for teaching the network: *backpropagation*. We conclude with a practical examination of neural network implementations such as histogram character recognition and saliency mapping.**

I. Introduction

The artificial neuron has a simple construction: an input vector, a weight vector, an aggregation function, and a transfer function. Combining artificial neurons into layers gives us an *Artificial Neural Network* (ANN). These systems are excellent at recognizing patterns within data sets due to their ability to learn based on selected input. The learning itself is done using one of a few techniques. We will look at feed-forward networks and the backpropagation algorithm. Once the ANN has been given ample training data and time to learn, it can be used to glean information from test data. We will look at the applications of ANN's to saliency mapping, optical character recognition, and medical landmark identification. First, we must build an understanding of the neuron itself.

II. The Neuron Model

Neurons serve two primary functions: receiving input from neurons and, if a threshold is exceeded, producing output as input for other neurons.

A. Biological Foundation

In biology, the neuron is a small cell that exists within the brain. It is composed of four primary components: dendrites, a cell body, the axon, and the terminal bulb. Neurons process electrical signals from other neurons by receiving impulses in its dendrites, which then pass the impulse forward into the cell body. Here, if the charge is sufficiently strong, it acts as a catalyst and triggers a subsequent impulse. This travels along the axon, and into the terminal bulb where it is passed to the following connected neurons. If the charge is not sufficient, the neuron remains inert and awaits another impulse. [3]

A notable feature of neurons is that those in close proximity to one another tend to fire together. That is, these neurons can be said to "learn" from the behavior of others nearby. [10]

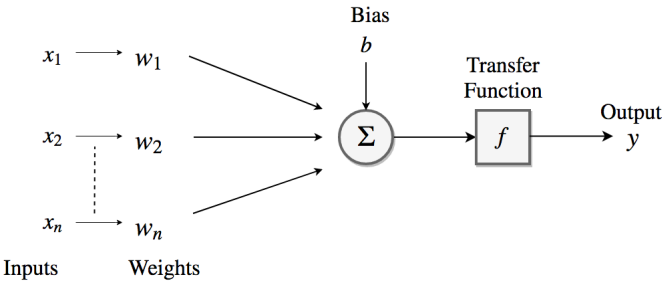

Fig. 1. Model of an artificial neuron. On the left-hand side is the input vector. Each input x is a value from the vector. Each x-value is multiplied by its respective weight, w. After this step, each product is aggregated in the body of the neuron. This is denoted by Σ in the center of the figure. After the values have been aggregated, they are referenced against the transfer function, f. Once the value has been processed in the transfer function, the neuron generates and passes along output y to the following network layer.

An intuitive way to think about the general behavior of neurons is thinking about them like transistors. They receive input current, and once a threshold is met, they release current.

B. Artificial Neurons

The artificial neuron is named after the biological neuron. This is because they share similar structures. Artificial neurons are composed of the following: input nodes (dendrites), weights for each input, a summation function to aggregate inputs (impulses), and a transfer function (to check for ample "charge"). We will further explain each component in the subsequent sections; beginning with the input layer and weights.

*1) **The Input Layer**:* Input for a neural network is typically a vector. Input can also be a matrix, if the dimensions of the input are compatible with the weights. Effectively, the weights will be a vector within the input layer. That is to say, the neuron receives input $\vec{X}$ and already contains $\vec{w}$. These take their respective forms:

$$\vec{X} = \begin{bmatrix} x_1 \dots x_n \end{bmatrix} \quad \text{and} \quad \vec{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

At this stage, each input value is weighted with respect to the weight value at the receiving input node. [11] For the entire input layer, we can generalize this with the following equation:

$$v = \vec{X} \cdot \vec{w} \tag{1}$$

where $\vec{X}$ is our input vector and $\vec{w}$ is our weight vector. Once this has been completed, we want to aggregate the products in the resulting vector. This can be generalized by:

$$u = \sum_{i=1}^{n} v_i \tag{2}$$

where $n = |v|$ and u is the sum of all elements. Once the values have been aggregated into u, this value is passed forward to the transfer function.

*2) **Transfer Functions**:* Now we must consider *when* exactly the neuron should fire. There are three common transfer functions we will cover: threshold, linear, and sigmoid functions. In each case, the value which is passed along is the result of $f(u)$. [8], [11]

1) *Threshold Function:* With this function, u (2) has to exceed a threshold T, which in common application, is typically initialized as a random number within a predefined interval. In my research, we have generally found this interval to be $[0 \dots 1]$. As the neural network learns, this threshold may be modified. The function is as follows:

$$f(u) = \begin{cases} 0 & 0 > u \\ 1 & u \geq 0 \end{cases} \tag{3}$$

2) *Linear Function:* Here, u (2) is output as is. It does not need to exceed a threshold to be transferred, it is passed proportionally to its weight.

$$f(u) = \begin{cases} 0 & u \leq u_{min} \\ mu + b & u_{min} < u < u_{max} \\ 1 & u \geq u_{max} \end{cases} \tag{4}$$

3) *Sigmoid Function:* This function gets its name from its telltale "S" shape. A common sigmoid function is the *logistic function*, which is defined as:

$$f(u) = \frac{1}{1 + e^{-u}} \tag{5}$$

After the transfer function, the artificial neuron model is complete. What must be done now is to piece together layers of these neurons.

III. The Network

There are multiple components in a neural network: the input layer, hidden layers, and the output layer. In the prior section, we examined the construction of the input layer. Essentially it is a vector of artificial neurons which take input, weight it, and evaluate it through its transfer function. Then it is passed inward, to the hidden layers. These are similarly composed of neurons but do not have any direct means of interaction, hence the prefix "hidden". In turn, these hidden layers forward their processed information to the output layer. This system can intuitively be visualized by Fig.2. We will cover more about the specific purpose of each. First, we will generalize the process of weighting sums in a multi-layer network with the equation below [14]:

$$\gamma_i^l = \sum_{j=0}^{N_{l-1}} w_{ij}^l z_j^{l-1} \tag{6}$$

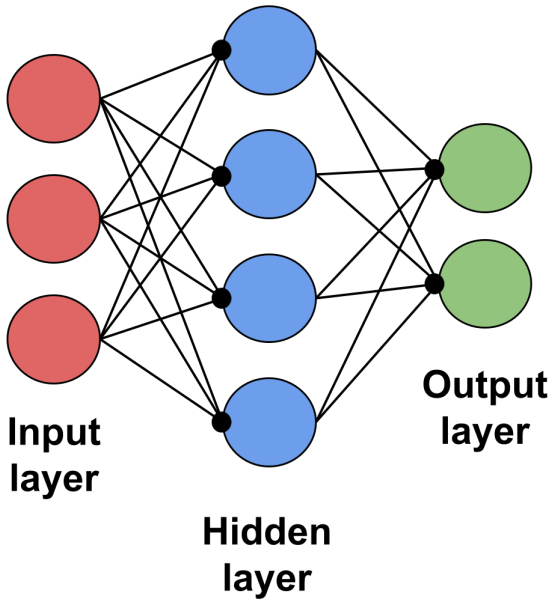

Fig. 2. A simple neural network. The neurons are represented by circles in the diagram. The input layer accepts the input vector, weights the values, and passes the transfer function output to each node in the subsequent layer. The hidden layer performs the same evaluations as the input layer, but is directly connected to the output layer. It is possible for multiple hidden layers to exist. The output layer is where we receive the final values and may evaluate them against our expected output in the following layer.

for the variables defined in the following table:

III. Variables in eq. (6)	
l	Current layer
i	Current neuron
j	Vector value at i_j
w	Weight vector
z	Previous hidden layer response

A. Input Layer

For information on the *input layer*, please see section II-B1.

B. Hidden Layers

The hidden layers provide more weights, thus providing greater evaluation of the inputs. This allows for better pattern recognition, since the weights are modified according to the error delta in the output layer. The information from the final hidden layer is passed into the output layer, which is where we evaluate the values. [2]

C. Output Layer

The output layer is where we directly interface with the results of the neural network. This final layer can be a vector, and is usually a one dimensional vector equivalent in size to

the number of rows within our training data matrix. [14] That is, if:

$$L_{in} = \begin{bmatrix} x_{0,0} & \dots & x_{n,0} \\ \vdots & \ddots & \vdots \\ x_{0,n} & \dots & x_{n,n} \end{bmatrix} \text{ then, } L_{out} = \begin{bmatrix} f(L_{in_1\cdot}) \\ \vdots \\ f(L_{in_n\cdot}) \end{bmatrix}$$

where $n\cdot$ is the respective row from L_{in}. It is here that our learning algorithms gather input to calculate errors.

IV. Calculated Learning

The neural network is unique, since it teaches itself as it processes iterations during the learning phase. There are numerous algorithms for learning in a neural network, but we will primarily focus on the *feed forward networks* and the *backpropagation* algorithm. We have already covered some of the equations in feed forward networks, but we will now look into its exact behavior. Second, we will examine the backpropagation learning algorithm.

A. Feed Forward Networks

These types of networks do not automatically learn. [10] Instead, the weights at each node are static and must be chosen using some supervised metric. They use a transfer function still (II-B2), and are aggregated with eq. (6), but they do not adjust their weights. Essentially, the objective purpose of each neuron in this network is to pass forward a value. This type of learning is typically referred to as supervised learning.

B. The Backpropagation Algorithm

One difficulty in teaching a neural network is that we do not necessarily know the optimal and expected values for the hidden layers. [12] Thus, neural networks with multiple layers will be undoubtedly tough to train by hand. This is the purpose of backpropagation. This algorithm provides the means by which neural networks may learn in an unsupervised setting. It operates in reverse compared to the neuron in Fig.1. This is demonstrated in Fig.3.

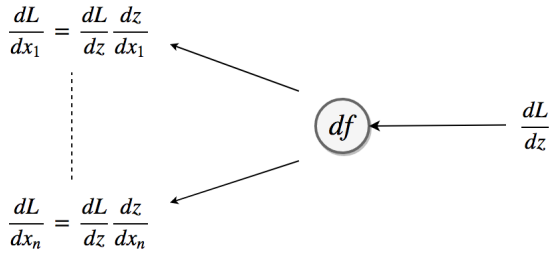

Fig. 3. This figure shows the process of backpropagation on a given neuron. The current neuron receives feedback from the following layer as a gradient with respect to its output z. By applying the chain rule, we can evaluate the gradients of inputs $x_1 \dots x_n$, which are in turn passed as input to the preceding layer.

Backpropagation relies on some metric which can describe the changing behavior of backpropagated values [6]. Fig.3 illustrates an implementation of *gradient descent* which we will cover in the following section. Essentially, we find the gradient (rate of change) of the errors in the current layer with respect to the prior output of this same layer.

To formalize the process of backpropagation, I will reference the backpropagation equations provided by Kiyoshi Kawaguchi [5], refactored to account for our previous definitions of constituent equations.

To calculate initial output values in the network, we evaluate eq. (6) for each neuron and layer. Next, we must calculate the errors of the network. This is a two-fold process where we calculate the *Output layer* and then the *Hidden layers*. After evaluating eq. (6), the error calculation of the output layer may be written as:

$$\delta_{Ni} = \gamma_{Ni}(1 - \gamma_{Ni})(L_{out_i} - \gamma_{Ni}) \tag{7}$$

where N is the number of layers in the network and i is a particular value in the current layer. To calculate the errors of the hidden layers, we then aggregate the product of the errors and their weights at each node. We then evaluate the product of this aggregation and the pass-forward output of the neuron. We can generalize this process with the following equation:

$$\delta_i^j = \gamma_i^j(1 - \gamma_i^j) \sum_{k=1}^{N_{j+1}} \delta_k^{(j+1)} W_{ki\cdot}^{(j+1)} \tag{8}$$

where j denotes the current layer and i is the i^{th} neuron in the layer. Considering this, δ_k^j and $W_{ki\cdot}^{j+1}$ are the error value and the weights of each neuron k in the network layer j, respectively.

Adjusting weights in the network is the next step in backpropagation. A general form for this is given below:

$$W_{ik}^{j+} = W_{ik}^j + \beta \delta_i^j \gamma_i^j \tag{9}$$

with j being the current layer, i being the neuron, and k being an element of neuron i's weight vector. Also here, β is a weight adjustment factor in the interval $[0 \dots 1]$.

This process is repeated for every training record (vector) in the neural network's input. Once this process is completed, we need to decide on an objective function which we will use to evaluate the change in errors. There are a few rules we could choose: Root Mean Square (RMS) [5], Least Mean Square (LMS) [12], or the Widrow-Hoff Algorithm [9], to name a few. In each algorithm, we will want to maximize or minimize their objective functions. This is dependent upon each algorithm, so for brevity we will not cover their inner workings. A general form for finding an error value for neurons in the network is the following [5]:

$$\delta_i^j = -\frac{\partial E_p}{\partial \gamma_i^j} \frac{\partial \gamma_i^j}{\partial T_i^j}. \tag{10}$$

where E_p is the objective function for training record p, γ is eq. (6), and T is the sum of products in the network layer.

C. The Gradient Descent Algorithm

Gradient descent is a popular algorithm to perform optimization and certainly one of the most popular algorithms used to optimize neural networks. We will cover three common

implementations of gradient descent, but will not cover the optimization algorithms used to optimize gradient descent.

1) ***Batch Gradient Descent (BGD) [7]:*** This particular variant is also referred to as vanilla gradient descent. Its purpose is to compute the gradient of a given cost function to optimize a parameter, θ. This algorithm is slow, considering it needs to calculate the gradients for the entire dataset to perform a single update. (Row 1 in table IV-C3)

2) ***Stochastic Gradient Descent (SGD) [7]:*** This variant performs an update for every training record x^i and every classifier y^i. Therefore it is much faster than BGD. One problem with SGD is that it fluctuates. This provides the chance for SGD to find new local minima or the global minimum, but it will have trouble settling into one position. When we decrease the learning rate, this algorithm converges to minima much like BGD.

3) ***Mini-batch Gradient Descent (MBGD) [7]:*** MBGD combines the techniques from both prior algorithms and performs updates for each "mini-batch" containing n training records.

Gradient Descent Variants	
BGD	$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$
SGD	$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$
MBGD	$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

V. Visual Processing — Image Analysis

Processing images and determining areas of interest is a difficult task for us to define; as it is a broad topic with various facets. Therefore, we will focus on a few applications that use neural networks for the purpose of image analysis. Specifically, our focus will be on: saliency mapping, character recognition, and medical landmark identification. As we mentioned previously, one popular algorithm is the *gradient descent* algorithm, which was covered in section IV-C.

A. Saliency Mapping

Saliency is the metric that determines areas of interest in an image or scene. That is, a location with high saliency has a high chance of being the first area in an image that our eyes look to. Performing this technique requires us to teach a neural network *how* to see a picture and then evaluate areas that stand apart from the rest of the image. The term *normalization* refers to processing the raw data we have into a predefined and understood range of numbers. In concordance with our outlined transfer functions in section II-B2, this range should be on the interval $[0 \dots 1]$.

L. Itti, et al. demonstrate how we may normalize an image for use in a neural network. Their goal was to create a system to determine image saliency. It works by normalizing the input data, in this case an image, and generating an intensity map of this input. This is obtained from [4]:

$$I = \frac{(r + g + b)}{3} \tag{11}$$

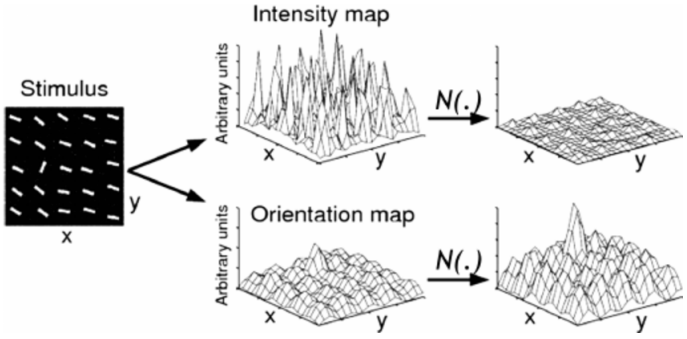

Fig. 4. This figure is borrowed from [4]. It demonstrates the purpose of normalizing the dataset before evaluating it using the Gaussian pyramid and intensity mapping techniques. As we can see, normalization is used to extract notable features from an input image. The specific technique used in determining saliency, provided by L. Itti et al., is demonstrated on the above middle two maps.

where r, g, and b are the red, green, and blue color channels of the input image. The resulting intensity map is then used to generate a Gaussian pyramid $I(\sigma)$, where $\sigma \in [0 \dots 8]$ is the scale. [4] This normalization technique is visualized in Fig.4.

The Gaussian pyramid is a matrix containing values corresponding to the intensity of the values in the input. Once the pyramid map has been normalized, we are able to input each row as a vector for the neural network. This will allow for the network to identify areas of high intensity with regards to other regions of high intensity. Gradient descent is behind the network determining the most salient portions of an image. It is optimal for this purpose since it will identify areas of high contrast in the input.

B. Character Recognition

There are many reasons why we may want a computer to read characters from an image. This could be for license plate scanning to automated document digitization. One popular technique for processing character-containing images with neural networks is to generate histograms of the image contents [1].

1) ***Histograms:*** Histograms are graphs that show the relative intensity of vertical regions within an image. For an example, see Fig.5. Intuitively, histograms can be thought of as sand hills. If the characters in the top row suddenly became sand, they would collapse and leave peaks where more sand (more feature density) has collected and valleys otherwise.

The purpose of generating histograms is normalization also. It allows for characters in formatted images to be extracted with relative ease (Bhushan et al. were able to achieve 98.70% recognition rate). Much like normalization in saliency mapping, the process here generates consistent figures with peaks and valleys that allow for the neural network to learn. Each character produces its own unique, and largely consistent, histogram. [1] This allows us to, similar to that in saliency mapping, use the rows of values from the generated histograms as input vectors for the neural network. Thus, we are able to train the network to identify characters in images.

Fig. 5. The above figure shows layers of histograms that have been generated from the text characters in the top row. The graphs in the subsequent rows illustrate abstraction of the generated character histograms from lowest to greatest.

C. *Medical Landmark Identification*

Neural networks and image processing can be applied to medicine in a unique way. With enough training information, neural networks can be used to determine suspect areas in patient scans. Different forms of scans include: magnetic resonance imaging (MRI) scans, computerized tomography (CT) scans, and x-ray scans. Each scan produces some image of a patient while attempting to show some form of interesting data. Not every scan is labeled well or correctly, thus leading to misclassification of scans and diagnoses.

Zhang et al. explain their neural network system in their 2017 publication. [13] The purpose of their neural network is to solve the issue surrounding automated anatomical landmark identification: the lack of imaging data. To do this, they had to develop a system that could perform two tasks other systems had been taking on individually. These tasks are: generating image patches to examine and determining landmark locations. With these tasks, Zhang et al. proposed their own loss-measure objective function. This is defined as:

$$Loss = \frac{1}{N_l} \sum_{i=1}^{N_l} w_i \|\tilde{d}_i^g - \tilde{d}_i^p\|_2 \tag{12}$$

where $w_i = e^{-\frac{\|\tilde{d_j^g}\|}{a}}$, a is a scaling coefficient, and N_l is the number of landmarks. [13] The implementation here identifies patches of wanted features in the image and weights them based on distance to their nearest landmark. As for *how* this is implemented in the neural network, the loss function described above would become the backpropagation optimization parameter θ mentioned in section IV-C. Considering the above, we can now see how a neural network can be trained to examine medical images. After defining the cost measure $Loss$, inputting images for the network is the same process described in V-A and V-B. Backpropagation, as we have covered, was used in their research to train the neural networks.

VI. Conclusion

Image processing using neural networks is reducible to two distinct problems: setting up the neural network and training it to analyze images. Each neuron has the capability of using different transfer functions, thus requiring us to choose one for the task at hand. The three most common functions are: Threshold, Linear, and Sigmoid.

Building the network requires us to create matrices containing the artificial neurons. The network must contain an input layer and an output layer. Typically, networks will also contain one or more hidden layers (as a way to enhance performance of the network). Once it has been constructed, it is necessary to decide the learning model: feed forward or backpropagation.

The learning model will influence the overall behavior of the network, thus making it essential to choosing an algorithm that will provide us with good, intelligible results. We have looked at the gradient descent algorithm as a means to optimize learning with respect to a cost function. Within this, we have also defined three variants of the algorithm — each with a different behavior.

Analyzing images comes in some distinct forms: saliency mapping, character recognition, and landmark identification. The three described topics have shown crucial variances in the initialization and construction of image perceptrons (neural networks). L. Itti et al. showed the importance of normalization prior to inputting data into the network. With character recognition we saw the importance of generating readable input data. Moreover, we have showed the requirement for consistency within the input data for neural networks. Lastly, our examination of medical landmark information demonstrated the flexibility of neural network systems as Zhang et al. implemented their own cost function and loss measure.

References

[1] Bharat Bhushan, Simranjot Singh, and Ruchi Singla. *License Plate Recognition System using Neural Networks and Multithresholding Technique*, volume 84. 2013.

[2] Josef Burger. A basic introduction to neural networks.

[3] Queensland Brain Institute. How do neurons work?, 2018.

[4] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *Ieee Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

[5] Kiyoshi Kawaguchi. *A MULTITHREADED SOFTWARE MODEL FOR BACKPROPAGATION NEURAL NETWORK APPLICATIONS*. Thesis, 2000.

[6] Fei-Fei Li, Justin Johnson, and Serena Yeung. Cs231n: Convolutional neural networks for visual recognition, 2018.

[7] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[8] Saed Sayad. Artificial neural network, 2018.

[9] Rob Schapire and Frank Xiao. Cos 511: Theoretical machine learning. Note sheet, 2013.

[10] Vidushi Sharma, Sachin Rai, and Anurag Dev. A comprehensive study of artificial neural networks. 2012.

[11] Christos Stergiou and Dimitrios Siganos. Neural networks.

[12] B. Widrow and R. Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *Computer*, 21(3):25–39, March 1988.

[13] J. Zhang, M. Liu, and D. Shen. Detecting anatomical landmarks from limited medical imaging data using two-stage task-oriented deep neural networks. *IEEE Transactions on Image Processing*, 26(10):4753–4764, 2017.

[14] Q. J. Zhang and K. C. Gupta. *Neural Networks for RF and Microwave Design (Book + Neuromodeler Disk)*. Artech House, Inc., Norwood, MA, USA, 1st edition, 2000.