Artificial Neural Networks and Image Processing

Author: Conor Lorsung
Advisor: Dr. Mike Heroux
May 14, 2018
E-Mail: *cglorsung@csbsju.edu*

CONTENTS

***Abstract*—Artificial Neural Networks (ANNs) and image processing has applications in character recognition, saliency mapping, and medical landmark identification. We survey the history of ANNs and the Widrow-Hoff Algorithm. Next is an examination of the transition from pattern to image processing. We then explain the general architecture of: the neuron model, input layer, hidden layers, and the output layer. Once these have been covered, we will look at recent learning algorithms, and how they work. We then examine practical applications of ANNs, such as: character recognition and medical landmark mapping. Following this technical analysis, we form predictions about the future direction of ANNs. Specifically, we recognize parallel computing and constructive artificial neural networks.**

I. INTRODUCTION

Artificial Neural Networks, while holding a solid position in Computer Science, hold a foundation in Biology. D.O. Hebb showed in his study titled, "The Organization of Behavior", that specific neurons typically grow stronger when given recurring stimuli that are relatively homogeneous. [4] This principal holds true in ANNs today. The neural network learns patterns by virtue of the backpropagation algorithm, which aims to minimize a loss function. As we will see, this idea has allowed us to teach computers skills such as: reading characters, predicting salient areas in images, and pinpointing medical landmarks. However, neural networks require much time to train and their performance can suffer immensely when using large datasets; due to their time complexity. Thus it is necessary to examine ways of enhancing their learning capabilities. This comes in the forms of: parallel computing with CPUs and GPUs, self-constructing neural networks, and specialized hardware. Our state of the field report will explore these topics in chronological order, beginning with their history, and ending with our future predictions.

II. A SURVEY OF ARTIFICIAL NEURAL NETWORKS

A. A Purpose in Pattern Recognition

Neural networks excel in shaping their structure to become familiar with certain types of information and tasks. The mechanics of the neuron network allow each particular node (neuron) to be good at responding to specific input. The implication here being that given enough training data, the neural network should be able to accurately determine between erroneous and valuable data. One of the fundamental adaptive learning algorithms is the *Widrow-Hoff Algorithm (same as Least Mean Square (LMS))*. This algorithm is based on gradient descent, in the sense that it expects the weights to converge

on the optimal weight. It operates by assuming negligible weights at each node and then calculating the mean square error (the difference between wanted and actual outputs). The gradient is taken from this product and used to update the weights in the network. The basic equation is:

$$W_{n+1} = W_n - \mu\nabla\varepsilon[n] \tag{1}$$

Where ε is the mean square error and μ is the convergence coefficient. [21] This idea is the driver behind adaptive learning neural networks.

Widrow and Rodney Winter published their research on this topic in 1988. Now, it implemented multi-layer neural networks for adaptive filtering and adaptive pattern recognition. Their system relied on the notion that in a black-box situation, we can devise knowledge of the system by inputting known inputs and measuring outputs for adjusting weights. This new approach automated the latter part of the process using multiple layers of neurons and backpropagation algorithms. [21] Having a system that can teach itself has great implications for uses in data analysis and pattern recognition.

B. *From Patterns to Image Processing*

Applications specific to image processing include extracting words from pictures, determining image saliency, and predicting movement. Tai-hoon Kim explores the various implementations of artificial neural networks with a focus on their image pattern recognition abilities. For example, "In 1997, Nallasamy Mani and Bala Srinivasan [6] applied artificial neural network approach [sic] for optical character recognition (OCR)." [8] Mani and Srinivasan used an artificial neural network to analyze 32x32 and 60x60 pixel arrays. The vector that is input into the neural network is the combination of the vertical and horizontal array vectors. By first using erosion and dilation techniques (to remove extraneous pixels and then adding pixels back to the letters respectively), Mani and Srinivasan were able to thoroughly pre-process the arrays for pattern recognition. Using a histogram to calculate densities in the arrays, they were then able to feed it into the artificial neural network. It then predicted output strings with accuracy rates of 70% with noisy data and 99% sans noise. A histogram analysis of letters and numbers is illustrated in Fig.1.

Using a neural network, we are able to take the final image analysis (the last row in Fig.1) Pattern recognition aids in determining the correct letter or number, since ideally they will have unique structures in their respective histograms. Although it may not be completely visible to the human eye, a well trained neural network should be able to accurately guess the correct value. The question now becomes, how do these models work? Moving forward, we will take an in-depth look at the neuron model and network structure.

Fig. 1. Sample histogram of visual text analysis. As you can see, the histograms become more "blocky" from top to bottom in the figure. This is due to using lower levels of abstraction in processing the characters. Essentially, the histogram begins to focus only on smaller, more significant pieces of the original histogram. This allows for a more compressed and similarly reliable analysis set for the provided information. (*Source:stackoverflow.com*)

III. Technical Analysis of Neural Networks

A. *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.

*1) **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. [5]

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. [19]

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.

*2) **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.

*a) **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. [20] For the

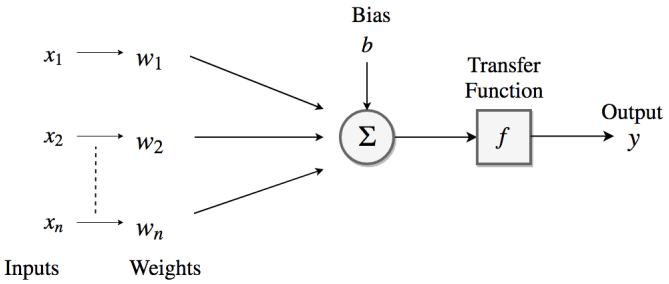

Fig. 2. Model of an artificial neuron. On the left-hand side is the input vector. Each input x is a value from the vector. Each x-value is multiplied by its respective weight, w. After this step, each product is aggregated in the body of the neuron. This is denoted by Σ in the center of the figure. After 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.

entire input layer, we can generalize this with the following equation:

$$v = \vec{X} \cdot \vec{w} \tag{2}$$

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:

$$u = \sum_{i=1}^{n} v_i \tag{3}$$

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.

*b) **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 $f(u)$. [16], [20]

1) *Threshold Function:* With this function, u (3) 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:

$$f(u) = \begin{cases} 0 & 0 > u \\ 1 & u \geq 0 \end{cases} \tag{4}$$

2) *Linear Function:* Here, u (3) is output as is. It does not need to exceed a threshold to be transferred, it is passed proportionally to its weight.

$$f(u) = \begin{cases} 0 & u \leq u_{min} \\ mu + b & u_{min} < u < u_{max} \\ 1 & u \geq u_{max} \end{cases} \tag{5}$$

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:

$$f(u) = \frac{1}{1 + e^{-u}} \tag{6}$$

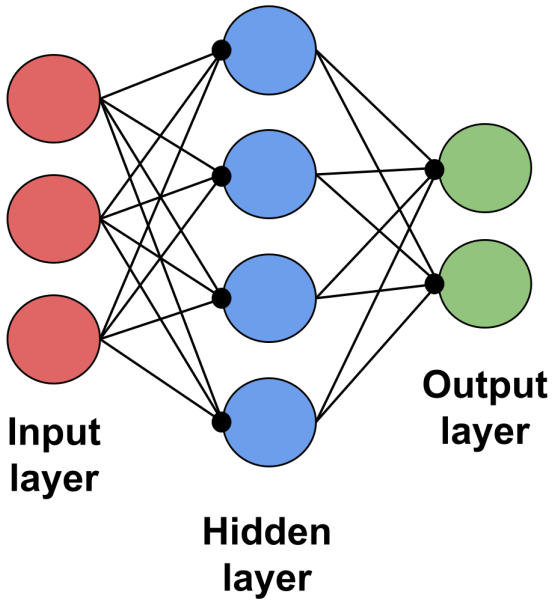

Fig. 3. A simple neural network. The neurons are represented by circles in the diagram. The input layer accepts the input vector, weights the values, and passes the transfer function output to each node in the subsequent layer. The hidden layer performs the same evaluations as the input layer, but is 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.

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

B. 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 [23]:

$$\gamma_i^l = \sum_{j=0}^{N_{l-1}} w_{ij}^l z_j^{l-1} \tag{7}$$

for the variables defined in the following table:

*1) **Input Layer**:* For information on the *input layer*, please see section III-A2a.

III-B. Variables in eq. (7)	
l	Current layer
i	Current neuron
j	Vector value at i_j
w	Weight vector
z	Previous hidden layer response

*2) **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. [3]

*3) **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. [23] 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.

C. 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.

*1) **Feed Forward Networks**:* These types of networks do not automatically learn. [19] Instead, the weights at each node are static and must be chosen using some supervised metric. They use a transfer function still (III-A2b), and are aggregated with eq. (7), 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.

*2) **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. [21] 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.

Backpropagation relies on some metric which can describe the changing behavior of backpropagated values [10]. Fig.3 illustrates an implementation of *gradient descent* which we will cover in the following section. Essentially, we find the

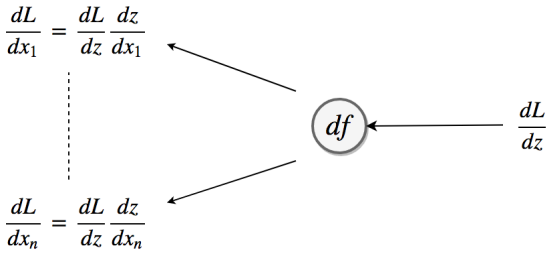

Fig. 4. This figure shows the process of backpropagation on a given neuron. The current neuron receives feedback from the following layer as a gradient with respect to its output z. By applying the chain rule, we can evaluate the gradients of inputs $x_1 \dots x_n$, which are in turn passed as input to the preceding layer.

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 [7], refactored to account for our previous definitions of constituent equations.

To calculate initial output values in the network, we evaluate eq. (7) 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. (7), the error calculation of the output layer may be written as:

$$\delta_{Ni} = \gamma_{Ni}(1 - \gamma_{Ni})(L_{out_i} - \gamma_{Ni}) \tag{8}$$

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:

$$\delta_i^j = \gamma_i^j(1 - \gamma_i^j) \sum_{k=1}^{N_{j+1}} \delta_k^{(j+1)} W_{ki\cdot}^{(j+1)} \tag{9}$$

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:

$$W_{ik}^{j+} = W_{ik}^j + \beta \delta_i^j \gamma_i^j \tag{10}$$

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) [7], Least Mean Square (LMS) [21], or the Widrow-Hoff Algorithm [17], 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 [7]:

$$\delta_i^j = -\frac{\partial E_p}{\partial \gamma_i^j}\frac{\partial \gamma_i^j}{\partial T_i^j}. \tag{11}$$

where E_p is the objective function for training record p, γ is eq. (7), and T is the sum of products in the network layer.

*3) **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.

*a) **Batch Gradient Descent (BGD) [15]**:* 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 III-C3c)

*b) **Stochastic Gradient Descent (SGD) [15]**:* 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.

*c) **Mini-batch Gradient Descent (MBGD) [15]**:* 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	$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

D. 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 III-C3.

*1) **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 III-A2b, this range should be on the interval $[0 \dots 1]$.

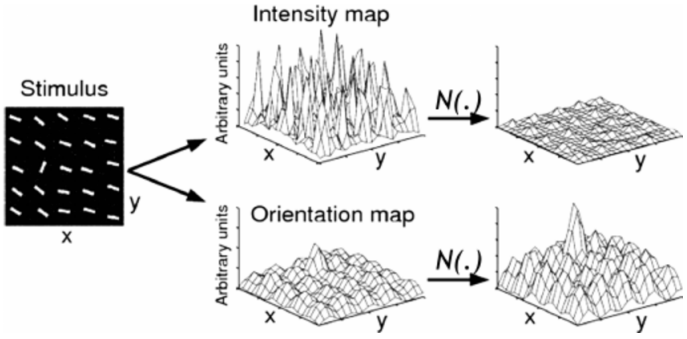

Fig. 5. This figure is borrowed from [6]. It demonstrates the purpose of normalizing the dataset before evaluating it using the Gaussian pyramid and intensity mapping techniques. As we can see, normalization is used to extract notable features from an input image. The specific technique used in determining saliency, provided by L. Itti et al., is demonstrated on the above middle two maps.

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 [6]:

$$I = \frac{(r + g + b)}{3} \tag{12}$$

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. [6] 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.

*2) **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 [2].

*a) **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

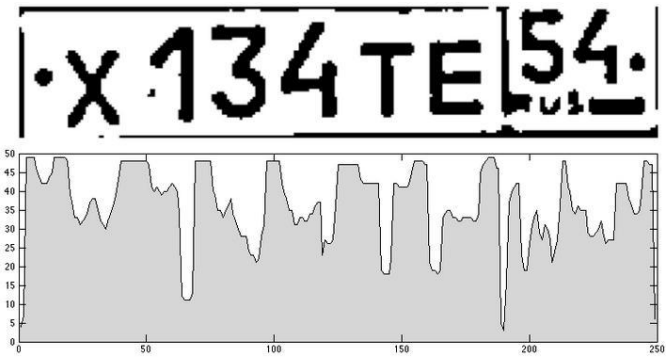

Fig. 6. The above figure shows layers of histograms that have been generated from the text characters in the top row. The graphs in the subsequent rows illustrate abstraction of the generated character histograms from lowest to greatest.

peaks and valleys that allow for the neural network to learn. Each character produces its own unique, and largely consistent, histogram. [2] 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.

*3) **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. [22] 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:

$$Loss = \frac{1}{N_l} \sum_{i=1}^{N_l} w_i \|\tilde{d}_i^g - \tilde{d}_i^p\|_2 \tag{13}$$

where $w_i = e^{-\frac{\|\tilde{d}_i^g\|}{a}}$, a is a scaling coefficient, and N_l is the number of landmarks. [22] 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 III-C3. 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 III-D1 and III-D2. Backpropagation, as we have covered, was used in their research to train the neural networks.

E. Technical Analysis Overview

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.

After examining the structure of ANNs and which problems they are best suited to, we will explore the future of ANNs in areas we have seen to be largely influential.

IV. Future Trends

A. Parallel Computing

Parallel computation is defined as the concurrent execution of processes or calculations. Typically this includes the use of multiple Central Processing Units (CPUs) working together on a single task for quicker evaluation. Additionally, Graphics Processing Units (GPUs) may be used in parallel computing. Depending on the nature of the given process, CPUs and GPUs will perform differently due to their hardware architecture. ANNs rely on the calculation of many vectors, given that the artificial neuron input layers are typically modeled as a vector. This means architectures better suited to matrix multiplication will perform faster than those that are not. We will cover specific cases where each processor excels, the growth of processor technology, and the implications this has for ANNs. First, it is necessary to understand why this is needed.

*1) **ANN Time Complexity**:* The following time complexity is an estimation of ANN weight convergence performance [13]:

$$3\sum_{j<i}|w_{ij}| = O(n^2 \cdot \max_{i,j}|w_{ij}|) \tag{14}$$

where w is the weight value at neuron i of layer j. In this equation, n is the number of bits needed to represent the input. This complexity represents the total number of neuron state changes using an asynchronous update rule. Asynchronous in this regard means each neuron is updated in some sorted order. Parallel computing seeks to minimize this complexity by adding synchrony to the computation. This is achieved by using multiple processors.

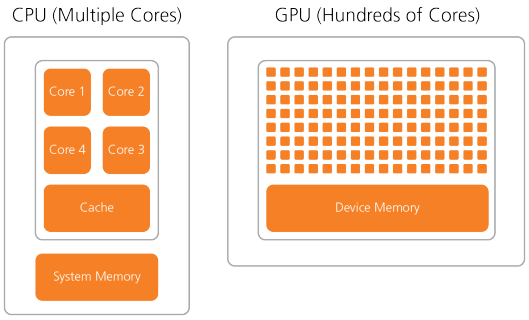

Fig. 7. The figure above illustrates the basic architectural differences between CPUs and GPUs. As shown, CPUs have fewer cores compared to GPUs. This is due to their different use cases: general processing (no strict need for synchrony) and matrix processing (largely asynchronous) respectively.

*2) **General Processor Architecture**:* Parallelism makes use of the cores in a processor and their threading capability. That is, each core may run a separate program since they contain individual processing threads. Taking this into account, we may delegate almost as many tasks to the processor as it has cores. Using this technique, it becomes apparent why parallel computing is typically faster than single core computing. In the case of ANNs and their need for many computations, parallelism becomes ideal since it allows for greater overall computational throughput. One product of increasing core counts is quicker computing.

*3) **The CPU**:* CPUs are the processors in a computer system that carry out arithmetic, logic, and input/output of a computer program. The widely used format of these processors today is the microprocessor: oftentimes a single-die with many transistors to perform logical tasks. According to Moore's Law, the number of semiconductors that can fit onto a single computer chip doubles roughly every two years. [12] With this idea in mind, it is easy to see that computers have become very fast.

*a) **Applications to ANNs**:* The CPU has a smaller number of cores compared to the GPU, but they are generally much faster. [11] This allows for the CPU to process calculations at each neuron quickly, but it is limited by the number of cores the CPU has. This is an example of low throughput. Despite the limited number of cores, Moore's Law still exists and we have seen a growth in the number of both cores and semiconductors in CPUs. [12]

*b) **Future Analysis**:* If Moore's Law persists, we expect to see a growth in the number of high-powered cores and semiconductors in CPU architecture. Despite CPUs not being explicitly engineered for matrix processing, they remain a staple jack-of-all-trades processor that can quickly process smaller tasks. This will provide quicker training times for ANNs and the ability to upscale the amount of neurons and hidden layers.

*4) **The GPU**:* GPUs are good at parallel computing since they have a large number of cores relative to the CPU. This is shown in Fig. 7 As GPUs have many cores, they can handle larger amounts of data than the CPU. Additionally, they can execute tasks synchronously; meaning the GPU will have a large throughput.

*a) **Applications to ANNs**:* Given that the GPU has many cores, it becomes an ideal candidate for matrix multiplication. Generally, this is the underlying purpose of the unit. For example, rasterizing images is done through matrix transformations and arithmetic. Given that ANNs are composed of vectors and matrices, a GPU can be used to evaluate large output of the network quicker than a CPU by computing in parallel.

*b) **Future Analysis**:* Moore's Law also affects GPU architecture. With an increasing number of semiconductors and cores, GPU performance metrics are expected to rise. In addition to the architectural improvements demonstrated in Fig. 8, software that streamlines these processes is being developed. One example of this is the consumer and academic software built by NVIDIA for GPU computation and parallelization. [1] We expect that architecture will continue to advance at the rates is has, thus providing improvements in ANN size and learning, specifically relating to image processing. [12]

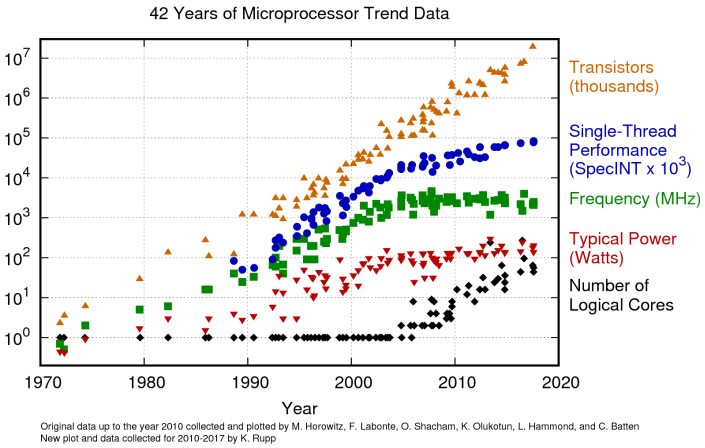

Fig. 8. This figure shows the trends in microprocessor data. It is apparent that Moore's Law (coded by the Transistors category) is still effective. Additionally, we can see an increase in processor cores; adding to the efficacy of parallel computing. Also significant is the single-thread performance, as cores execute programs on their threads. The figure indicates that parallel processing is more viable than ever, given an increase in thread performance and number of cores. We expect the trends to continue with the support of Moore's Law and historical data.

B. Evolutionary ANNs

Evolutionary ANNs are artificial neural networks that learn the data they are processing and form their structure around it. One example of this is the Constructive Artificial Neural Network (CoNN). These networks make use of a *Constructive Algorithm* that generates additional nodes, layers, and connections as needed to best fit the training data. [18] There are six unique motivators for using CoNNs. These are outlined in [18]:

1) **Flexible topology exploration**: The topology of CoNNs can be explored since we know the general construction algorithm and how it generates structures. It will produce different topologies given unique data, thus allowing us to see how they may behave in different problems.
2) **Complexity matching**: Given a difficult task, CoNNs can better match the inherent complexity of the presented information. This is due to their constructive, generational behavior.
3) **Complexity estimation**: Since we know the method behind the CoNN construction, we can generally estimate the performance of the network by comparing it to similar tasks.
4) **Performance tradeoffs**: Given the nature of CoNNs, they need time to learn and train for a given problem set before they can be used. This means there will be a greater time requirement up front while the CoNN alters its structure, but greater time savings after it has optimized itself.
5) **Network-based memory**: CoNNs are constructive, thus meaning they can fit themselves to different problems. This can be new data, but it also means older data can be incorporated into the network to solve more problems in tandem.
6) **Lifelong learning**: CoNNs can learn to solve more than one problem at a time. This opens the door to solutions for larger, more complex problems since they can learn the foundational pieces and apply them to a greater picture. Given their memory, they learn problems and "remember" them too.

Considering the points above, we see how these networks can be useful in applications relating to image processing. Since CoNNs readily adapt to training data and further input data, they can be trained on a wide variety of inputs (within reason). As these systems are further researched, and expanded, we will gain a better understanding of their behavior. Given the current state of research into CoNNs [14], we postulate that generalized models will be developed. As processing power continues to grow (see sections IV-A3 & IV-A4), we can expect these networks to compute, learn, and shape faster. This will give us structural information quicker, as it will become feasible to use larger and more complex networks.

C. ANN Architectural Expansion

As ANNs grow, their computational time complexity does too. This is a problem, since the time complexity given in Eq. (14) grows roughly to the order of n^2. There are other ways to further develop ANNs besides creating more layers.

*1) **Biological Contributions**:* Using *coding schemes* to evaluate the transfer of data could allow for improvements. These could be time-series evaluations of neuron activity in ANNs. *Coding schemes* are currently used to evaluate biological brain performance. [9] By further examining the biological brain, we may uncover valuable information to better construct our virtual networks.

*2) **Specialized Hardware**:* One limitation of growing ANNs will be the need for specialized hardware. Neural networks that may find use with thousands of neurons will require large amounts of power to operate through just one iteration. [9] Therefore it will be necessary to create power-efficient devices that can meet the memory, computational, and speed requirements of larger ANNs.

*3) **Multidimensional Neural Networks**:* While only existing in theory, currently, they would be an improvement upon the ANNs we have today. "The theory is based on the concept of generalizing one-dimensional logic gates to multi-dimensional logic gates and in a similar manner one-dimensional neural networks have been generalized to [Multidimensional Neural Networks] MDNN's." [9] The authors further explain that the MDNN receives input from the states of a hypercube, then figure out the stable states afterwards.

V. Conclusion

After examining where ANNs come from, how they are developed, and where we see progress being made, it becomes clear that these systems are optimal for pattern recognition. Image analysis relies heavily on recognizing recurring patterns since most items typically have a set standard of forms. For example, as we've seen, characters generally have similar forms — this is how we recognize them, after all. Considering this, neural networks seem to be an optimal candidate for image processing. With their learning capabilities through backpropagation and their burgeoning ability to modify their own structure, neural networks have established themselves as a novel solution to many image-reliant problems.

Appendix

This course in particular has taught me how more efficiently learn, and what it means to learn responsibly. It has done so by requiring me to read research papers and articles that I initially found to present a significant challenge. However, as the course progressed, reading and understanding material in this field became easier and more natural.

That said, I could not have tackled this topic without the knowledge other courses here at Saint John's University have given me. *Data Mining*, *Algorithms and Concurrency*, and *Discrete Math* each provided strong foundations from which I could build this course.

1) **Data Mining** gave me the ability to begin thinking like a researcher, and understanding more of the scientific approach to Computer Science. Additionally, the intensity required in that course helped me construct a better work ethic.
2) **Algorithms and Concurrency** helped me understand the algorithms present in the research contained within

this paper. Learning and implementing them was easier than it would have been without learning how to understand terminology and specific pseudo-code in this area of research.

3) **Discrete Math** gave me the fundamental knowledge required to begin understanding the formulae that are at the base of machine learning.

Of other Computer Science skills this class has required me to learn, technical writing and reading is perhaps the most useful. I have read more research papers in this course than I previously had in my life. If I had known this coming into the course, I doubt what confidence I had would have remained. However, this has proved incredibly useful in forwarding my education and ability to learn at a higher level. Moreover, my time management skills and work ethic were greatly improved by this course. All said, this course has greatly improved my critical thinking capabilities and ability to recognize the depth of problems as they arise. It has taught me to be a better student, researcher, and all-around person.

REFERENCES

[1] About cuda. https://developer.nvidia.com/about-cuda.

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

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

[4] D. O. Hebb. *The organization of behavior: a neuropsychological theory*. Science editions ; 275-S; Science editions ; 275-S. John Wiley, New York :, 1964.

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

[6] 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.

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

[8] Tai-hoon Kim. Pattern recognition using artificial neural network: A review. In Samir Kumar Bandyopadhyay, Wael Adi, Tai-hoon Kim, and Yang Xiao, editors, *Information Security and Assurance*, pages 138–148, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[9] Sachin Lakra, T. V. Prasad, and G. Ramakrishna. The future of neural networks. *CoRR*, abs/1209.4855, 2012.

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

[11] Christian de Looper. What's the difference between the cpu and gpu?, Feb 2017. https://www.pcmech.com/article/cpu-vs-gpu/.

[12] Chris A Mack. Fifty years of moore's law. *IEEE Transactions on Semiconductor Manufacturing*, 24(2):202207, May 2011.

[13] Pekka Orponen. Computational complexity of neural networks: A survey. 1, 05 2000.

[14] Beatriz Pérez-Sánchez, Oscar Fontenla-Romero, and Bertha Guijarro-Berdiñas. A review of adaptive online learning for artificial neural networks. *Artif. Intell. Rev.*, 49(2):281–299, February 2018.

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

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

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

[18] Sudhir Sharma and Pravin Chandra. Constructive neural networks: A review. 2, 12 2010.

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

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

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

[22] 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.

[23] 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.