Monmouth College Department of Computer Science and Mathematics

INVESTIGATION OF “DEEP” LEARNING CATALYZED BY TOPICS IN COMPUTER VISION

PROJECT PROPOSAL FOR SENIOR RESEARCH

DEVIN KING
12-5-2016

TABLE OF CONTENTS

1. BACKGROUND AND INTRODUCTION

1.1 Introduction

I have always had a fascination with neural networks; I have been to a few technical presentations discussing the advancements in machine learning and AI, and although these presentations have quite a bit of information about the various applications of artificial intelligence, the lack of precise information about *how* a neural network works disturbs me. To some degree, the extent of specific, technical information given in these presentations amounts to the statement, "we don't really know how these things work…they just do." As a mathematician and computer scientist, this is simply not enough to satisfy my desire to understand the inner workings of these seemingly complex structures. As a venture into the depth of the math and computer science that drives them, I have elected to study neural networks in higher detail for this project.

1.2 Background

Computer vision, as its subject matter is understood now, has been an area of interest in academia since the 1970s. In short, "computer vision is the science that aims to give [a machine the capability to visually sense the world around them]" [1]. Due to the amount of time that the topic has been of interest, there is quite a bit of information about heuristics used to design these networks and criteria (such as "ground truth-ing" [2]) for judgement on efficiency and performance of them; these are the issues are that will be investigated in this project. The roots of this subject lie in the original intent for computer vision to be a topic of machine intelligence and emulation of human recognition. The story referenced in Szeliski's text concern's Marvin Minsky's charge to a student to complete the task of plugging a camera into a computer and having it report what it saw, which, as it is known now, is a much more difficult task than a simple summer's project can overcome. The reconstruction of what is present in the two-dimensional image into three-dimensional information was seen as a "stepping stone" into the perspective of artificial intelligence which concerns recognition. Segmentation of the image into three-dimensional blocks is becoming of a system of recognition in the determination of "objects" in any scene.

Neural networks have not been a practical implementation of computer vision solutions until recently, but now there are applications for networks that exist within the context of computer vision. In particular, the image classification problem and its associated solutions tackle problems that appear in the context of security and identification; for example, the National Institute of Standards and Technology produced a dataset of handwritten characters that can be used to classify postmarked letters and eliminate the human element from the mail forwarding process.[1] The classification problem also appears in the context of security applications; security camera footage requires identification of cars, people, and other elements depending on resolution, and the ability to classify a single image quickly enough allows for the continuous classification on a feed and, as one would expect, the reduction of human interaction in the process. Outside of computer vision, there is an abundance of applications for neural networks, including but certainly not limited to financial, medical, and industrial applications. Neural networks can be used to classify information, extrapolate based on trends in high-dimensional data, and make decisions based on input state information. These applications are all useful in the sense that they show their users valuable information about the data they have collected; for example, an industrial use of neural networks is the determination of appropriate process control for a manufacturing plant. To quote Alyuda systems, "complex physical and chemical processes that may involve the interaction of numerous

[1] https://www.nist.gov/srd/nist-special-database-19

(possibly unknown) mathematical formulas can be modeled heuristically using a neural network" [3].

Competition computer vision architectures for neural networks have been wrestling each other for the best performance, and as a result, there are several standardized datasets available for use by those looking to compare results. Because this data is so accessible, it is reasonable to pursue undergraduate research in the realm of computer vision. The accessibility of this data nixes the requirement for manual data set creation, which would be tedious and unfeasible in the scope of this project. It is unlikely that any comparison will be able to be made to these competition architectures, and as such this project will focus on the differences in locally constructed implementations.

There is also a need to research the mathematics informing the design of a neural network. The abstract (and yet somehow simultaneously definite) mathematics used to structure the neural network forms the core of the investigation; questions about *why* an architecture works better than another delve into the multidimensional calculus that defines the networks. The math gives a way to clearly define the differences between one architecture and another. Simple problems (such as approximating/producing a sine function) are able to lead into the precise details in the construction of a multidimensional architecture (as opposed to the practical aspect of function overcoming justification of function in applications).

2. PROJECT DESCRIPTION

In order to delve into the inner workings of the image classification neural network, it is necessary to pose a research question that facilitates the investigation:

"How do changes in training sets and hyperparameters of an image classification feedforward convolutional neural network affect its training time, memory usage, and percentage of items correctly classified in the final validation set?"

Answering this question involves constructing/pre-selecting a neural network and iteratively:

1. Selecting a hyperparameter to vary
2. Fixing all other parameters
3. Testing the effects of variation on the neural network training time

The engineering process that I have chosen involves constructing the network, training the network, testing and benchmarking the performance, and deploying the network. This project will focus on the first three segments of this design model:

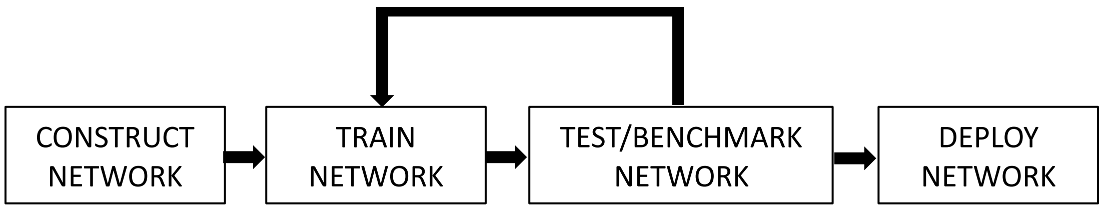


Figure 1: Flow diagram of the engineering paradigm

2.1 Network Architecture

The identification (i.e. classification) of an item (or items, in more complex cases) in a given image is the goal of the image classification problem. This project makes use of the IMAGENET Large Scale Visual Recognition Challenge dataset from the 2015 competition. The network will classify images into one of ten distinct classes; one of either an apple, dog, bagel, microphone, basketball, lobster, saxophone, tie, stove or water bottle. This classification will be done with the ImageNet architecture as well as the LeNet architecture.

2.2 Experimental Training and Testing by Variation of Hyperparameters

The hyperparameters of a neural network affect its performance, so it is useful to choose any one of these hyperparameters and treat it as an independent variable in an experiment on the training performance of the network. The hyperparameters that will be tested in this project are the learning rate eta, the number of epochs, and the training set size:

- Eta (Learning Rate):

  The selection of the learning rate variation must be done on the basis of performance of the network for a single epoch. In order to appropriately a sufficient range of values to be tested, an initial eta must be selected on small scale to determine where a reasonable solution convergence lies in the spectrum of possible eta. Zero is an obvious null choice because a learning rate of zero destroys the gradient descent algorithm. Increasing choices of eta make the adjustment algorithm converge faster, but finding a reasonable starting point to vary. After an appropriate eta has been found, a unit eta adjustment can be used to scale eta down or up. The precise amount of scaling depends on the absolute amount of time that it takes to run a training session

- Number of Training Cycles (Epochs):

  As stated above, the number of epochs that are reasonable to run is dependent on the time that a network takes to run one epoch of training. The most important initial step here is to get the network up and running so that it can be determined what is feasible selection for the number of training cycles that can be completed within the scope of this project.

- Variation of training set size:

  The data set for the test will be divided in varying ratios starting with 80% to 20%, which will be the total training set and final validation set, respectively. The key here is to separate the data so that a part of it is not involved in the training process until the network has been completely trained. The total training set will then be segmented using K-fold cross validation with a k value of ten – i.e. the training set will be divided such that one-tenth of the data will be used as validation during training and the other nine-tenths will be used as adjustment examples. This process will be repeated with ten different segmentations of the training set (10-fold cross-validation), each segment selecting a validation portion that has not been used before. This allows for a representation of the average values of the data points that avoids the issue of single observation data.

The dependent values will be the training time, memory usage, and percentage of items correctly classified in the validation set. The performance of the two architectures will be compared on the basis of hyperparameters – e.g. a parallel table for each architecture will be constructed, with data that shows the variation in performance. Graphs will be constructed showing how these hyperparameters affect the above dependent variables.

2.3 Presentation of Backpropagation Algorithm

A key element of the calculation of the change in the weights in biases of a network based on the cost function is the backpropagation algorithm. A brief description of stochastic gradient descent is given in section 3 of this paper; this forms the basis of the backpropagation algorithm. As a part of this project, the backpropagation algorithm will be presented in the context of a single hidden layer neural network and full derivation of the four "fundamental" backpropagation equations will be given using the MSE function as a representative of the change. Source synthesis will be done to present the fundamentals of the backpropagation algorithm illustratively using graphics in a poster or PowerPoint presentation.

2.4 GPU Effectiveness for Training

2.4.1 GPU Speed-Up Graphs

Part of the training cost for the neural network depends on the performance of the computing unit that is doing the training, whether that be a CPU or a GPU. As such this project will also measure for advantages to using the GPU versus the CPU. All of the tests that are done in Section 2.2 can be repeated using the GPU as the primary compute tool:

- Eta (Learning Rate)
    - Dependent on optimal unit value
- Number of Training Cycles (Epochs)
    - Dependent on original single epoch training time.
- Vary training set size:
    - Varying split of data set with k-fold cross-validation strategy with k value of ten to average results for each segmentation of full training set

The key difference, in this case, is that the GPU garners an increase in the amount of training that can be done in a fixed time T. The two tables that are generated in this case can be compared to the tables made in section 2.2 and the performance increase factor for fixed T can be calculated, showing the advantage or disadvantages to using a GPU as a training tool.

2.4.2 GPU Neural Network Primitives

The training/testing process of neural network construction is very compute intensive; the feedforward process, the gradient calculation for backpropagation, and the process of adjusting weights are the most work-heavy parts of the lifecycle for a neural network. The implementation of these operations is of key interest, and for these reasons, the GPU implementation of key elements of the neural network training and evaluation process will be implemented or studied:

- Implement a single neuron layer on the GPU:

  The neural network is defined as nested function evaluations, where the result of any neuron in any layer *l* is dependent on the layer before, *l-1*. In this nested evaluation, the fundamental operations that are executed are matrix multiplication and vector addition. Matrix multiplication and vector addition are both highly parallelizable and as such lend themselves well to GPU implementation. Moving this fundamental evaluation operation over to the GPU enables the gap in execution time that appears in NVIDIA speedup graphs.

- Isolate key backpropagation operations and implement one of these:

  Cost evaluation and backpropagation of adjusted weights have an opportunity for parallelism that exists due to dot products and vector addition and subtraction. The cost derivative is propagated through the weight and bias matrix and vector using the same operations. It is likely that the actual matrix adjustment is the implementation that will be completed as a part of the project. A full explanation of backpropagation is part of section 2.3 and as such this segment of the project will focus only on the physical operations and implementation rather than the theory.

3. FOUNDATIONS

Computer vision has a multitude of subtopics that vary from the pixel level to the highly abstract recognition. A full deconstruction of all of these topics is beyond the scope of this project, but the fundamentals of the image classification problem, as well as foundational material for neural network construction, will be reproduced here:

3.1 The Image Classification Problem

In the most general sense, the image classification problem can be conceptualized as taking a set of images and deciding what the item(s) in the image are. Mathematically, the problem can be abstracted to the fitting of a complex function that takes as input any given image and returns the class of the image. In the case of the particular classification problem that this project will aim to solve, the output of this function will be a ten-dimensional vector whose elements are all zero except for the designated class element:

$$\boldsymbol{y} = \langle y_{class-1}, y_{class-2}, \dots, y_{class-n} \rangle$$

This classification function is not well defined and as such cannot be computed easily. The solution to the problem is to approximate the using known data points. For this project, the known data points are images whose associated class are known. There are more complex instances of the problem where an image can be stretched or warped and contain multiple instances of different classes, however, the problem that will be solved in the scope of this project is to fit an image classifier function that classifies images containing only one instance a given class.

A formal mathematical restatement of the above:

> Given a set S_t of images whose associated classification vectors are known that represents a classification function $f^*(\boldsymbol{x}) = \boldsymbol{y}$ where $\boldsymbol{x}$ is any image in the domain (i.e. images whose class exists) and $\boldsymbol{y}$ is a classification vector of dimension ten whose elements are predicates (in this case, one or zero) that describe the instance of a class in the image, design an approximation function $f(\boldsymbol{x}) = \boldsymbol{y_a}$ where $\boldsymbol{y_a}$ is an approximation of $\boldsymbol{y}$ for any $\boldsymbol{x}$.

When comparing approximations against established data points for a function, it becomes necessary to describe the differences in the approximation by using the notion of error. The error function (also referenced as the objective, cost, or loss function) is a way to quantify this difference, and in the scope of this project, the error function is the mean-squared error function (MSE). For a pre-classified image set S of size n:

$$E(S_t) = \frac{1}{n}\sum_{1}^{n}\left(f^*(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)\right)^2$$

Equation 1: MSE Function

The element-wise difference in the resultant classification vectors is indicative of the quality of the function fit for a particular example $\boldsymbol{x}$. Thus, the image classification problem can partially be described as the minimization of an error function. It should also be mentioned that the function f for any fixed $\boldsymbol{x}$ can be reparameterized (in this case, w and b are parameters that describe weights and biases in a neural network, which are described in section 3.2). This reparameterization becomes useful in calculating the gradient of the cost function for a particular example $\boldsymbol{x}$, and the gradient is used in the backpropagation algorithm, a concise analytic description of which is a project goal.

3.2 Perceptrons and Neural Networks

The modern neural network is rooted in the McCollugh and Pitts paper about neural activity; these networks are based on the actual neural networks that occur in nature – those in the brain. In their paper, a "logical calculus" is used to describe the unit of operation (i.e. a neuron) in an actual neural network. McCollough and Pitts then use propositional logic to discuss the properties of the relationships between these neurons, and gives the first ability to discuss these relationships in the abstract and as such allows comparison between different networks [4].

These networks were far from the artificial neural networks that are known today, however. Marvin Minsky, in his text on Perceptrons, furthers the work of McCullough and Pitts introduces ideas of computational geometry in the context of the definition of knowledge in computing, pattern recognition, and learning. Abstract algebra and predicate logic facilitate the discussion of these topics, and although they're relevant foundation, they distract from the focus of this project and will not be included here. The text introduces a titular concept that is critical to the conceptualization of a neural network – the perceptron:

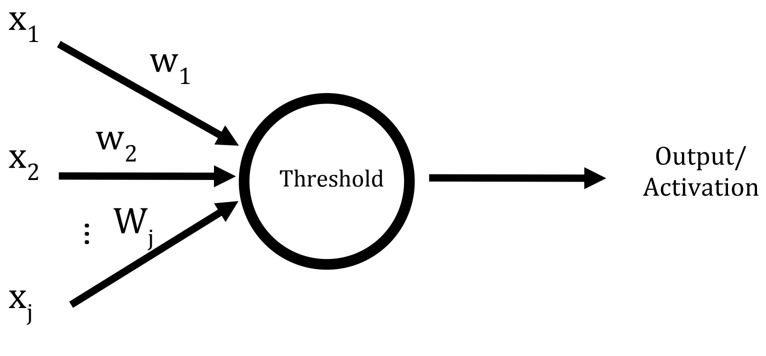


Figure 2: Visual representation of a perceptron

Figure 2 shows the perceptron conceptualized. Visually, it is similar to the neuron that occurs in nature. The predicate logic that is well defined in Minsky's text forms the logical basis for this artificial neuron – all neurons make decisions based on input [5]. Each neuron has a set of input factors and a notion of how much each input factor matters (called a factors and weights). Given a set of weights and a threshold value for a neuron, the neuron behaves as such:

$$\sum_j w_j x_j \equiv \boldsymbol{w} \cdot \boldsymbol{x} \Rightarrow a(\boldsymbol{x}) = \begin{cases} 0 \ if\ \boldsymbol{w} \cdot \boldsymbol{x} \le threshold \\ 1 \ if\ \boldsymbol{w} \cdot \boldsymbol{x} > threshold \end{cases}$$

The threshold value for a neuron determines at what point it decides that a “no” answer should become a “yes” answer. It is customary to check the sign of the activation sub-function rather than its position relative to a threshold, so the negative of the threshold is represented as the $bias\ b$ for any neuron. Algebraically, the perceptron can be represented as a piecewise linear function:

$$a(\boldsymbol{x}) = \begin{cases} 0 \ if\ \boldsymbol{w} \cdot \boldsymbol{x} + b \le 0 \\ 1 \ if\ \boldsymbol{w} \cdot \boldsymbol{x} + b > 0 \end{cases}$$

Equation 2: Algebraic representation of a neuron

The sum of weights associated with factor x is equivalent to the input-vectorized form of the output function. The neuron in this form is a pure “yes” or “no” decision function. It switches with an absolute nature between the two possibilities. The adjustment of weights and biases based on the cost function described in 3.1 will not fare well when quantifying the change in output based on a predicate function. Due to this reason, it becomes necessary to choose a smoothing function that, in this case, is representative of how close the neuron activation is to zero or one. This function is called the “sigmoid” function, and it can be mathematically described as such:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

Equation 3: Sigmoid function

The graph for the sigmoid function shows the continuity between values of zero and one:

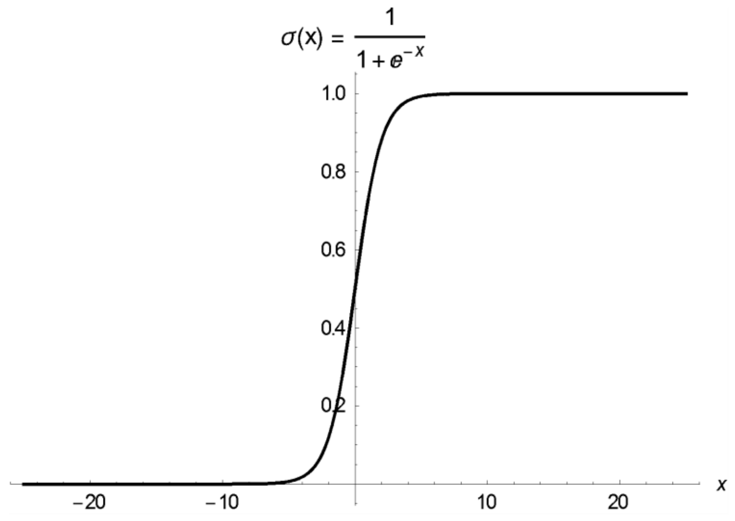


Figure 3: Graph of Sigmoid Function

This smoothing function is applied to the binary output of the neuron:

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

Equation 4: Activation Function with Sigmoid Smoothing

The smoothing applied here allows the determination of how small changes in weights change the output by eliminating the binary nature of the neuron output. This becomes especially important in the backpropagation algorithm, which adjusts the weights and biases as necessary to converge to a function fitting. There are different selections of the sigmoid function that can result in different smoothing, and this choice results in one of the versatilities of the artificial neuron [6].

Neurons on their own are interesting constructions, but their true power is realized in numbers. This is the basis of a neural network, a composition of neuron elements. Neural networks are a conceptual (and as of recent practical) solution to the image classification problem. The structure of a neural network is akin to the fitting of the image classifier function. Abstractly, an image classification neural network can be conceptualized as a black box that takes as an input an image and outputs a classification vector that contains the correct classification. For example, in the figure below, the images, which have been drawn from the UCI Machine Learning Library are fed into the neural network and the classification vector represents the identity of the person in the image:

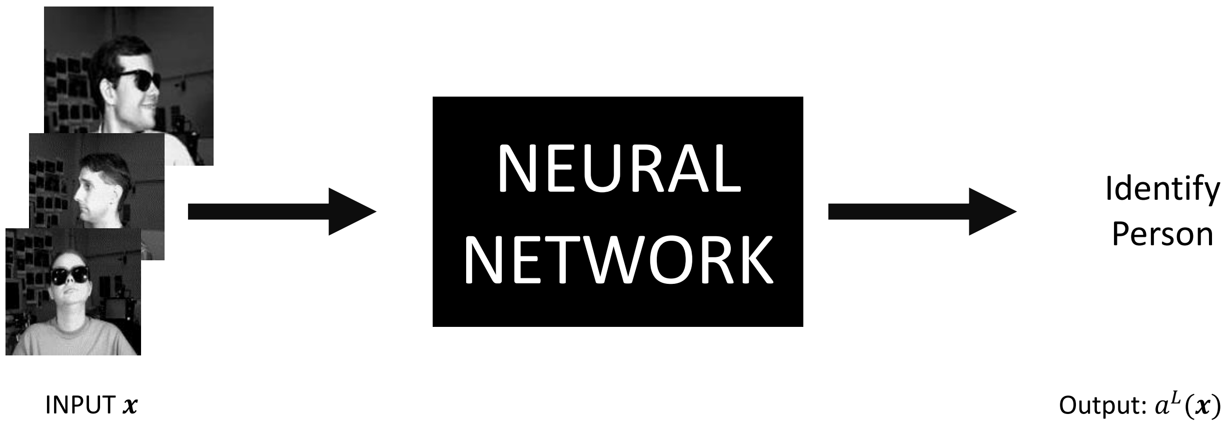


Figure 4: "Black box" neural network

Neural networks are usually conceptualized as a collection of circles and lines indicating the neurons and connections/weights, respectively. The tree of connections can be organized in such a way that each "level" of the tree represents a "layer" in the network:

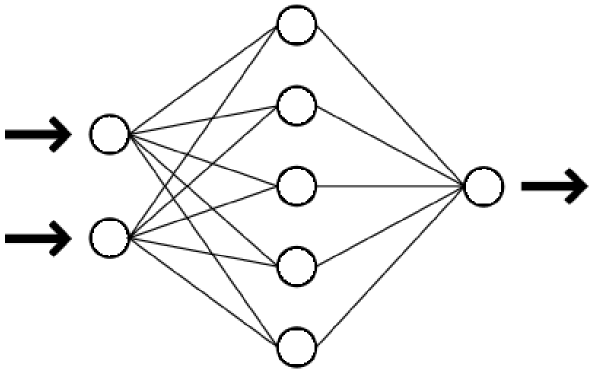


Figure 5: Visual representation of neural network

The flow of information in the network, in this visualization, is from left to right. This flow is known as "feeding forward." The graph-like nature of the neural network is useful in organizing the function of layers of the network. The organization of the neurons requires specification based on layer and neuron in the formulation for the activation of a neuron. Without getting into the nitty-gritty of index specification, the final vectorized form of a neuron is given below, where l is the layer of the network, w is a matrix of weights (vectors for each neuron in the layer) and b is a vector of biases for each neuron:

$$\boldsymbol{a^l = \sigma\left(w^l a^{l-1} + b^l\right)}$$

This recursive definition of the function evaluation of a neural network encapsulates the nature of the neural network's evaluation as well as the detail within a step [6].

3.2.1 Engineering Paradigm

The engineering process that has been chosen for the construction of the network is represented by the following flow diagram:

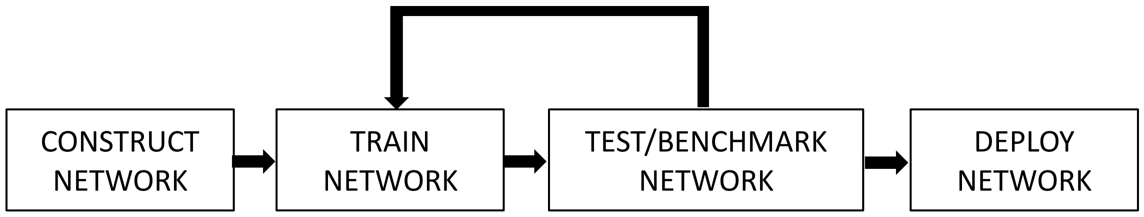
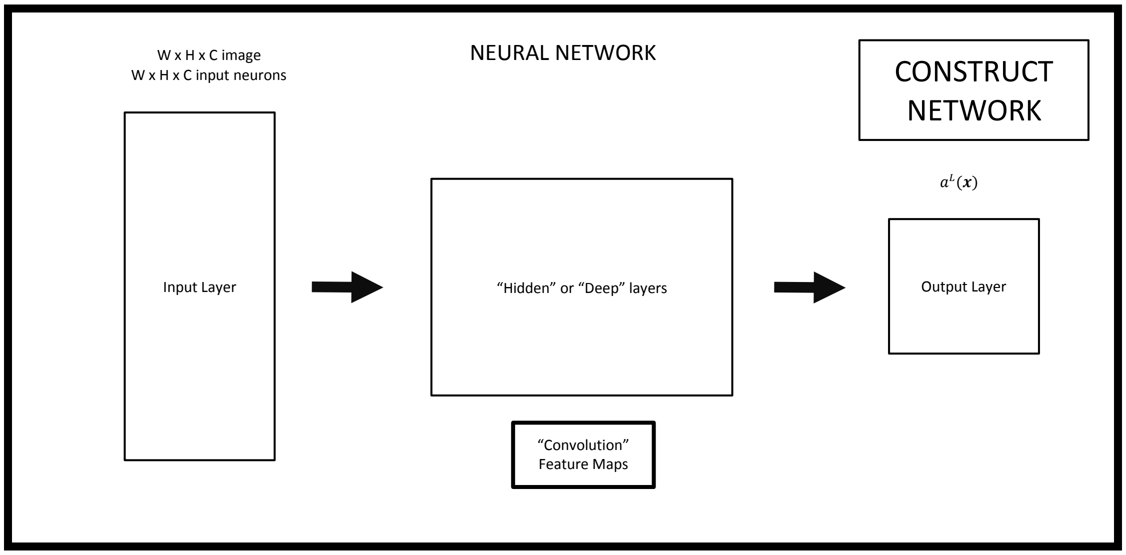


Figure 6: Flow diagram of the engineering paradigm

After a particular problem has been identified as a candidate for solution by neural networks, one can begin using this design paradigm to construct the solution. The engineering design paradigm involves the construction of the network, training of the network, testing and benchmarking of the network; after the cycle of training and testing has been completed, the network can be deployed as a solution.

3.2.1.1 Network Construction

The network construction is a step whose precise actions depend on the application. In the context of this project, this segment of the engineering paradigm will involve the selection of pre-existing network architectures to use as the basis for the construction of the network. Construction of individual layers in the network will involve describing the number of neurons in each layer and deciding whether that layer is a processing layer of a decision layer:



In the networks that will be constructed/tested in this project, the input layer consists of neurons that represent the total dimensionality of the image. The output layer represents the set of possible classes, represented by an output vector as described earlier.

3.2.1.2 Network Training

The second step in the design paradigm is to test the effectiveness of the by running multiple passes on the training set (called epochs in the literature). The method of training that will be used in this project involves splitting the training set into "mini-batches." These mini-batches determine the number of iterations that the network will train with. This distinction can be confusing, so it will be reiterated again: The *epoch* count in network training determines the number of cycles through the training set that is done. The *number of iterations* in a training cycle is the number of individual examples that the network trains on:

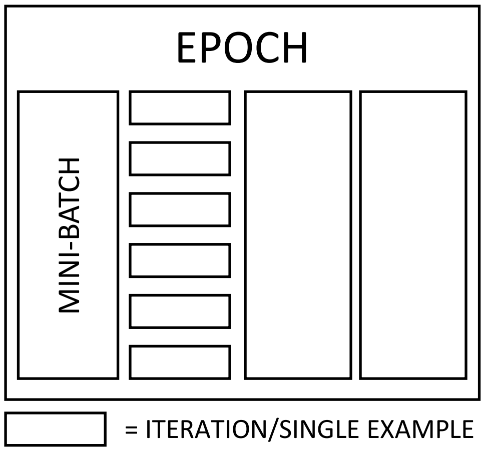


Figure 7: Visualization of Epochs vs. Iterations

It is important that this set is disjoint from the validation set as will be described in the following section. The calculation of a classification is the result of a single feed forward using a single training example, but the training – i.e. the adjustment of weights is done, in this case, using stochastic gradient descent (SGD).

In order to properly adjust the weights in the network, it becomes necessary to discuss how small changes in the parameters of the neural network – i.e. weights and biases – affect the output of the network. While the sigmoid function resolved the issue of the standard perceptron's binary output by smoothing it and made it possible to detect a change in individual neurons, the final output of the neural network must be compared to the correct output for any training example. The cost function, as described earlier, gives a quantifiable measure of the accuracy of the classification. It is useful to restate the cost function using the neural network terms that have been previously described (where a^L is the neural network, or the function composition that produces the activations of the final output layer of the network, f^* is the correct image classifier, and $\boldsymbol{x}_i$ is any pre-classified item in the training set S_t):

$$E(S_t) = \frac{1}{n}\sum_{1}^{n}\left(f^*(\boldsymbol{x}_i) - a^L(\boldsymbol{x}_i)\right)^2$$

In order to describe these small changes, multidimensional calculus becomes useful. This project will use the method of stochastic gradient descent to calculate the change in the output based on small changes in the weights and biases in a network. Consider a multivariate function $F(\boldsymbol{x})$ where $\boldsymbol{x} \in \boldsymbol{V}^n$ with a gradient as such:

$$\nabla F(\boldsymbol{x}) = \left\langle \frac{\partial \boldsymbol{F}}{\partial x_1}, \frac{\partial \boldsymbol{F}}{\partial x_2}, \dots, \frac{\partial \boldsymbol{F}}{\partial x_n} \right\rangle$$

In three dimensions, where one dimension depends on the other two and $\boldsymbol{x} \in \mathbb{R}^2$, the function can be conceptualized as a surface with hills and valleys, and the gradient at any point on the surface can be considered the fastest way to the peak nearest to that point. This is shown below as follows; any directional derivative on a surface can be considered as the dot product of the gradient vector and a unit direction vector $\hat{\boldsymbol{u}}$:

$$D_u F = \nabla F \cdot \hat{\boldsymbol{u}}$$

The dot product of any two vectors can be geometrically conceptualized as the product of the vector lengths and the cosine of the angle between them. As such, the following formulation holds:

$$D_u F = \nabla F \cdot \hat{\boldsymbol{u}} = \|\nabla F\| \|\hat{\boldsymbol{u}}\| \cos\theta = \|\nabla F\| \cos\theta$$

$\cos\theta$ is maximized when the angle is zero, so the optimum direction of ascent is described by the gradient:

$$\hat{\boldsymbol{u}} = \frac{\nabla F}{\|\nabla F\|}$$

As such, the gradient vector, when generalized to n dimensions, can be used to find local maxima in any multidimensional function. This knowledge can be used to form the method of gradient descent by using the negative of the gradient:

$$v \to v' = v - \eta \nabla F$$

Where eta is the rate of descent and v is a vector that iteratively moves toward a local minimum of the curve. This algorithm is useful for adjusting the weights and biases in a neural network (which for any fixed $\boldsymbol{x}$ can be represented as $f(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{b})$) based on the value of the cost function, and a precise yet simplifies description of how this is done analytically is a goal for this project. The stochastic nature of the algorithm comes from the division of the training set into equally sized, *randomly* sampled subsets (mini-batches). The gradient decent algorithm is then performed on these individual subsets until the entire training set has been seen by the training algorithm. The epoch count then determines how many times this process repeats.

3.2.1.3 Network Testing and Benchmarking [7]

In the most basic sense, the performance of the neural network is judged by its ability to classify items that are not in the training set – e.g. things that the network has not seen before. Stuart Russel and Peter Norvig propose that the cycle of training and testing should proceed as such:

1. Collect a large set of examples
2. Divide it into two disjoint sets: the training set and the test [or validation] set
3. Apply the learning algorithm [in this case, SGD] to the training set and produce a hypothesis *h*
4. Measure the percentage of examples correctly classified by *h*
5. Repeat steps 2 to 4 for different sizes of training sets and different randomly selected training sets of each size.

The learning curve can then be generated for parameter adjustments and associated test set performance. The Norvig text mentions an important aspect of performance assessment that is worth mentioning here due to its relevance when related to the approximated image classifier; Peeking at the test data, which is “all too easy” to do. In the example in the text, the correct fitting function is chosen based on its performance on the test set, and as such this information has become a part of the algorithm. Another challenge to overcome is the use of “meaningless regularity” in the construction of the fitting function. This concept is known as *overfitting*.

In order to counter these problems, the method of c*ross-validation* will be used. The entirety of the set with a known classification will be separated into two parts. *K-fold* cross-validation is described as such: For each experiment, up to k, set aside $1/k$ of the data to test on and assesses performance. Average the results, and use the highest performing values as the best approximation function

3.3 Graphics Processing Unit Advantages

Part of the training cost for the neural network depends on the performance of the computing unit that is doing the training, whether that be a CPU or a GPU. It has been established (and is, in fact, a major selling point for GPU developers to the compute market) that GPUs make problems that are well structured for parallelization run blazingly fast. These improvements lend themselves well to the function composition that defines the neural network, and as such, there are GPU implementations of many neural network primitives in NVIDIA’s cuDNN library, which allows for neural network libraries such as Caffe to make implementation of a neural network much simpler.

The cost of these primitives becomes immediately apparent in the function composition that is the basis of the feedforward operation. The primary mathematical structures that are used to represent the weights and biases for any given layer are matrices and vectors, and the actual mathematical operations that complete this process are matrix multiplication and vector addition and subtraction. As stated in section 2.1, these mathematical structures and the operations associated with them present an excellent opportunity for parallelization.

4. IMPLEMENTATION PLAN AND TIMELINE

4.1 Initial Project Constraints

The nature of this project requires that something is known about the total time that a network constructed using the Caffe framework takes to train on one set of data. Getting Caffe up and running and completing a training cycle is, therefore, a necessity in order to determine the scope of the training that can be completed for the data collection. This will be a primary first goal of the project, as the rest of the timeline depends on this constraint.

1. Construct and run a MINST network from the tutorials on the Caffe network to determine how long the construction process takes with the framework.
2. Use the Caffe framework for MINST to determine training cycle time
3. Use above data to restructure timeline to adhere to appropriate constraints:
    a. Is cross-validation feasible?
    b. Hyperparameter variation limits

The following is a presentation of an idealized timeline; however, it should again be stated that the more future planning is done, the higher the number of unknowns (i.e. the further into the timeline one reads, the less defined the timeline is).

4.2 Implementation Timeline

Checkpoint 1:

- Construct bash scripts & C++ files for compiling test data from the IMAGENET archive
    - Create test directory
    - Reference class HTML address list for downloading
    - Pseudorandom Sampling (C++)
    - Disjoint training and validation sets for each class (C++)

Checkpoint 2:

- Boilerplate Caffe Code (Caffe):
    - Initialization of GPU/Caffe and Library
    - Command Line independent variables if needed
    - Image loading and Transfer to GPU memory (blobs)
- Backpropagation Derivation of final two equations in Neilsen text

Checkpoint 3:

- Network Construction (cuDNN and Caffe):
    - Mid-construction of networks with Caffe primitives
- Boilerplate CUDA code for Neuron Construction and Backpropagation Operations
    - Initialization of CUDA error checking
    - Basic Kernel code
    - Timing code using C++ STL chrono library
- Rough-draft of Backpropagation Explanation using example single layer network

Checkpoint 4:

- Network Construction (cuDNN and Caffe):

    - Finish network with Caffe primitives
    - Confirm functionality of network (e.g. data set loading), simplify program parameter adjustment (eliminate recompilation)
    - Output relevant dependent variables:
        - Objective Scores
        - Validation & Test set percentages
- Neuron & Backpropagation Operation Implementation Progress Check

Checkpoint 5:

- Network Testing:
    - Test Network and Vary hyperparameters
        - CPU/GPU monitoring software ("nvida-smi" and "htop")
    - Compile data points for parameter adjustment
        - Vary hyperparameters and record effect on dependent variables
    - Vary test and validation sets using bash script
- Complete Backpropagation Explanation using example single layer network

Checkpoint 6:

- Completed Implementation of Neuron and Backpropagation operations:
    - Timing information for average operation time
- Final Poster and Presentation of Data
    - Pretty data tables
    - Performance Graphs
    - Poster Design(s) complete

4.3 Fall-back Benchmarks

In the event that the major goals of the project are far too ambitious, the following is an emergency list of the minimum required elements:

- Experimental, Empirical test data for ONE neural network architecture on the CPU
    - No k-fold cross-validation for data point averaging
- Experimental, Empirical test data for THE SAME neural network architecture on the GPU and associated speedup factor
- Backpropagation derivation from Neilson
- (GPU implementation of Neuron)

4.4 Implementation Tools

- Caffe Deep Learning Framework:

  The Caffe deep learning framework provides modularity in the construction of neural network layers rather than construction from primitives alone. Without this abstraction, it would be infeasible to fully construct an image classification architecture and run it for this project. The framework also provides a command line interface and a scripting language in python to analyze the configuration files that define the network and run the training.

- NVIDIA CUDA (Compute Unified Device Architecture):

The NVIDIA CUDA allows for the definition of operations and primitives in a language that can interface with the device architecture on a GPU. These primitives and knowledge of the GPU architecture are required create an efficient implementation of basic linear algebra operations. Since these operations form the foundation of neural network feedforward and training, it is important that they be as fast as possible to increase the training time.

- The C++11 Programming Language:

  Since speed and flexibility are primary concerns in this project, the C++ programming language has been selected as the basis of the project. Bash scripts, python scripts, and Caffe definition code will also be used but critical operations will be completed in the C++ language.

5. CONCLUSION

The purpose of this project is to investigate a topic in computer vision using the neural network as a catalyst for this discussion. The image classification problem is an excellent way to start the investigation into the complexities of neural network construction because the problem, although somewhat challenging to specify rigorously, is internally understood and easy to identify with. The modern neural network as a construction is complex and involves the function composition of many different vector and matrix operations. These operations allow activations in any layer to propagate through the network in an act called feeding forward. This feedforward operation is the primitive that performs a classification in the image classification problem.

The adjustment of the fixed values that define the network requires a notion of error and an algorithm called “backpropagation,” which allows the performance on known examples to be compared to the correct values and allows this delta to be propagated back through the network. As the network size increases in complexity, these operations become much more difficult and constrain the training time. In order to combat these issues, the use of graphics processing units (GPUs) is needed. These units of computation, being designed original for vector processing on a massively parallel scale, lend themselves well to the fast calculation of these matrix and vector products.

Given these considerations, this project aims to construct a neural network that will be used to examine the effect of hyperparameter adjustment on the training time and classification performance. These metrics will be recorded and compared to GPU implementations on the same machine to create speedup graphs that accentuate the improvements offered by GPUs. The neural network primitives of matrix multiplication and vector addition and subtraction will be implemented on a GPU to show the speedup over a significantly large dataset. The final deliverables of this project will be data tables showing the relationship between the hyperparameters and the performance metrics, associated speed-up tables for the GPU usage, a poster (or posters) with information about the project implementation details, and a paper or poster describing the backpropagation procedure.

BIBLIOGRAPHY

[1] The British Machine Vision Association (BMVA), "The British Machine Vision Association (BMVA)," The British Machine Vision Association (BMVA), 2016. [Online]. Available: http://www.bmva.org/aims. [Accessed 12 September 2016].

[2] S. Krig, Computer Vision Metrics: Survey, Taxonomy, and Analysis, New York: Apress, 2014.

[3] Alyuda Research Company, "Alyuda | Products & Solutions. Decision Making Software," Alyuda Research Company, 2016. [Online]. Available: http://www.alyuda.com/products/forecaster/neural-network-applications.htm. [Accessed 12 September 2016].

[4] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics,* vol. 5, no. 1, pp. 115-133, 1943.

[5] M. Minsky and S. Papert, Perceptrons, 3rd ed., Cambridge, Massachusetts: The MIT Press, 1988.

[6] M. Neilsen, "Neural Networks and Deep Learning," Determination Press, January 2016. [Online]. Available: http://neuralnetworksanddeeplearning.com/. [Accessed 04 September 2016].

[7] S. Russel and P. Norvig, Artificial Intelligence: A Modern Approach, 2nd ed., New Jersey: Pearson Education, 2003.

[8] T. Nitta, Complex-Valued Neural Networks: Utilizing High-Dimensional Parameters, Hershey, Pennsylvania: IGI Global, 2009.

[9] J. D. Cowan and D. H. Sharp, "Neural Nets and Artificial Intelligence," *Daedalus,* vol. 177, no. 1, pp. 85-121, 1988.

[10] R. Szeliski, Computer Vision: Algorithms and Applications, 2010.

[11] R. Hecht-Nielsen, Neurocomputing, Melo Park, California: Addison-Wesley Publishing Company, 1990.

[12] nVIDIA Corporation, "CUDA Toolkit Documentation," nVIDIA Corporation, 3 February 2017. [Online]. Available: http://docs.nvidia.com/cuda/. [Accessed 12 March 2017].

[13] Y. Jia and E. Shelhamer, "Caffe: Convolutional Architecture for Fast Feature Embedding," Berkeley Artificial Intelligence Research Lab, 2014. [Online]. Available: http://caffe.berkeleyvision.org/. [Accessed November 2016].