Open Source Neural Network Model

A Major Qualifying Project Report
submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

by: Paula L Rudy
Project Advisor: Professor Michael A. Gennert
Date: April 27th, 2017

Abstract

This Major Qualifying Project is an open source supervised machine learning neural network software development kit (SDK) utilizing a stochastic gradient descent with cross-entropy error backpropagation algorithm, implemented in Java. This SDK provides a human-readable implementation of a commonly used method of machine learning, thereby supplying a transparent learning tool for students.

The SKD was demonstrated by providing functional examples of networks designed for classification. Examples were provided ranging in complexity from the simplest most transparent example of learning, to a real world example of classification based on color.

Acknowledgements

I would like to express the deepest appreciation and admiration to my advisor, Professor Michael A. Gennert, whose guidance and generous patience made this project possible.

Table of Contents

Abstract.....i
Acknowledgements.....ii
Introduction.....1
Social Implications.....2

Applications.....2
Implications.....2

Background.....4
Methodology.....8
Implementation.....10
Results.....12
Conclusions.....18

Table: Comparison of Learning Times.....18

Future Work.....19
Appendix A: UML diagram of class structure.....21
Appendix B: Table of Test Results.....22
References.....23

Introduction

Machine learning with neural networks is currently a hot topic in modern artificial intelligence. This approach to computer learning is being used to make many modern marvels possible. These applications include general image recognition (such as Amazon's Rekognition API[1]), handwriting recognition[2], and even diagnosing cancer[3]. Despite their widespread use and an abundance of reference material, there remains a lack of fully transparent and easy to follow implementations of these concepts[4]][5]. This can make learning the mathematical techniques and how to implement them difficult. Commonly used frameworks are difficult to use to learn the basics of implementation, either because they are too abstract and high level (such as Weka[6]), or too complicated and involved (such as TensorFlow[7]). Courses, papers, and online resources (such as blogs and forums) are valuable sources of instruction, however the high complexity and connectivity of these networks can be difficult to describe in a two dimensional medium.

This SDK is designed to fill that gap. It provides an open source, easily readable, well documented, and simple implementation of the most commonly used concepts in the application of neural networks. The SDK was implemented in Java, the current most commonly used language[8].

Social Implications

Applications

Machine learning has widespread applications that will bring about advances that will touch all aspects of everyday life. This technology is poised to benefit human endeavors in a myriad of ways. Artificial neural networks in particular have applications wherever human-level (or better) pattern recognition is needed.

This could include functions as varied as authentication of artwork, to medical [3] and scientific research (data mining in particular[9]), to even teaching (from recognizing plagiarism[10], to identifying areas of weakness in a student[11]). Law enforcement is an area poised to exploit these new abilities as well, from implementing functions such as flagging suspicious purchases[12] and online search patterns[13], to making suspect profiles and identifying crimes by the same perpetrator over large databases of police reports[14]. This could even include the possibility of using the wide network of available video surveillance[15] to identify suspicious and possibly criminal behavior[16], and to identify possible witnesses to crimes[17].

Implications

Naturalistic language processing[18], and facial[19] and emotional recognition[20] are areas already benefiting from machine learning. These advancements, combined with our expanded reliance on computational assistance in our everyday lives, could very well be the start of increasingly personal and emotional connections to our technological assistants.

Despite the myriad of applications of this technology and the benefits soon to come, this new stage in the computer revolution is not without its problems. Overfitting in particular is an insidious trap, often exposing gaps in our training data or even our own erroneous[21], prejudicial[22], or subconscious biases[23]. Systems employing machine learning are not magically free from making errors. “More efficient” is not the same thing as being able to be left unsupervised,

especially for tasks on which human lives depend. Perhaps the most daunting and pressing problem is that the capabilities of modern technology is already out-pacing our ability to legislate (an issue known since 1816[24]). This is especially pertinent to modern machine learning. Law enforcement and national security are poised to gain significantly from the various forms of machine learning assisted pattern recognition[25](e.g. face recognition[26][27]). These gains will bring with them a large number of ethical questions that must be addressed on the national (and international) stage[28]. Legal questions about privacy and legislation are not the only ethical issues at stake. As computational learning models grow to more closely resemble biological neurology in size, complexity, function and structure, we are left with profound questions on the precise nature of consciousness and the ethics of employing such systems for our benefit.

The effects of such technology on human social behavior also raises some interesting and important questions, not just about the changes in our personal relationships and communication, but also social structure and interactions on a national scale. Will our increased reliance on technology to communicate cause greater social distance[28] or work to close it[29] ? While technology already facilitates human social interaction[30], will such technologies grow to participate itself[31]?

These issues (and others not mentioned here) bring into sharp focus the need for increased societal understanding of machine learning. This understanding must include the mechanics of how these systems function and not just its abilities. Given the massive leap in ability that machine learning is soon to enable, unequal distribution of these abilities and knowledge of these technologies would likely drive enormous increases in inequality. If kept only in the hands of a governing body or large organizations, this technology could easily be used to enforce conformity and prevent dissidence[32][33].

The far reaching implications of this new wave of machine learning paints a clear portrait of the strong need for public education with tools focused on

expanding the understanding of the abilities, function, and implications of the use of these technologies.

Background

Machine learning is a field of computer science that concentrates on algorithms that can be learned by building models from supplied data without the model being explicitly supplied by programming. Machine learning can take many forms. The form focused on in this SDK is using a convolutional neural network for classification using supervised stochastic gradient descent with a cross entropy error function.

Supervised learning, as referred to here, occurs when the algorithm is supplied with labeled data. Each training input example has a known corresponding expected output. The algorithm is expected to create a function that maps each input to its corresponding output, and in doing so, creates a means of predicting the correct output for any new input. One of the most common tasks employing supervised learning is classification. This involves sorting input into classes based on certain attributes (e.g. “is this object red, green, or yellow?”). Unsupervised learning, in contrast, occurs when the algorithm must create its own classifiers without being given the expected outputs (i.e. the algorithm is given “unlabeled” data).

Backpropagation is the method of training used in this SDK. This approach consists of three stages:

1) Input is propagated forward through the network.

2) An error measurement is taken over the newly calculated output of the network.

3) The error is then propagated backwards through the network, determining the approximate contribution each weight contributes to the error, and updating each weight accordingly, thereby reducing the error rate.

Figure 1: Backpropagation Pseudocode:

```
BEGIN
FOR each training input
     Feed the input through the network
     Compute average cross-entropy error over entire learning set
          (given new out values)
     FOR each layer in network (working backwards)
          IF current layer != a maxpooling layer
               FOR each filter in current layer
                    FOR each weight in current filter
                         Compute partial derivative of cross-entropy error
                              (with respect to current weight)
                         SET current weight to:
                                current weight
                                - [learning rate
                                    * partial derivative of cross-entropy error
                                      (with respect to current weight)]
                    ENDFOR
                    Compute partial derivative of cross-entropy error
                          (with respect to bias for current filter)
                    SET bias for current filter to:
                          bias for current filter
                          - [learning rate
                              * partial derivative of cross-entropy error
                                (with respect to bias for current filter)]
               ENDFOR
          ENDIF
     ENDFOR
ENDFOR
END
```

Methods used to determine the contribution of each weight to the error for that input, and to determine the direction in which to update that specific weight, are referred to as the “optimization function” for that form of backpropagation. Gradient descent is the optimization function used in this SDK. In this technique, the partial derivative of the error with respect to a given weight is used to update the weight in question.

Cross entropy error, used here in this SDK, is an error function especially suited for gradient descent backpropagation for classification tasks.

Equation 1: Average Cross Entropy error for an Artificial Neural Network

$$C = -\frac{1}{x}\sum_{i}^{m}\sum_{j}^{n}\left[e_{ij}\ln a_{ij} + (1 - e_{ij})\ln(1 - a_{ij})\right]$$

Let:

m *= the number of training examples*

n *= the number of output values of the network*

(e_{ij} , a_{ij}) *= the expected (e) and actual (a) values of a single output*

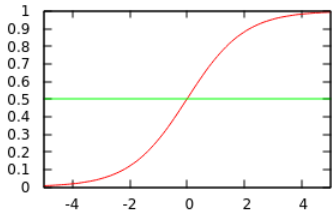
x *= the total number of [expected, actual] tuples*

*(e.g. This would be m*n if the number of outputs for the network is constant over all training inputs)*

Due to the curvature of the sigmoid function (which is employed in this SDK as an activation function), learning can be quite slow (i.e. the partial derivative with respect to a single weight is small) not just when the weight is nearing the correct value, but also when that weight is wildly wrong.

Equation 2: The Sigmoid Activation Function

$$S(t) = \frac{1}{1 + e^{-t}}$$



Let:

t *= the raw output value of a single neuron*

e *= Eular's number*

S(t) *= the output value of that neuron after the activation function is applied*

Cross entropy error works to mitigate this effect by putting more emphasis on results being overall close to the desired output rather than emphasizing wrong answers.

In stochastic gradient descent, instead of computing the gradient over the entire learning input, the gradient is computed for a single input randomly selected from the training information provided. This is significantly simpler and faster than computing the gradient over the entire set of learning input for each backwards (i.e. backpropagation) pass, and also helps to mitigate the detrimental effects of local minima in the gradient function.

Artificial neural network refers to a network mapping of input to output in a manner loosely inspired by biological neural anatomy. In this model, the input passes through one or more “layers” of artificial neurons. Each neuron is connected to many others, and, in passing the input through the network, each neuron may inhibit or enhance the propagation of that signal and the signals of other neurons. These networks are characterized by:

- The **topology** of the network: the map of connections in the network and any rules that govern the formation of connections.
- The **activation function(s)**: how the weighted input is transformed into output at each neuron.

- The **weights** of the network: parameters that are updated during learning and that are then applied when an activation function is calculated.

Topology of artificial neural networks is highly varied. The focus of this SDK is an implementation of a feedforward network (i.e. the graph of the connections of the network can be said to be directed and acyclic). Feedforward neural networks are most commonly organized into layers, in which neurons in one layer take as input the output of one or more neurons in the layer (or layers) preceding it. In this way, computation of the neurons in each layer can be done in parallel, as the output of each neuron in one layer is independent of all other neurons in its layer.

A convolutional artificial neural network (CNN) is a type of artificial neural network inspired by the organization of an animal visual cortex. Layers of CNNs contain neurons organized by not just a two dimensional grid, but also a third dimension of ***depth***, which refers to the number of neurons looking at the same region of input. Layers of CNNs are further characterized by ***stride***, which is the unit amount the receptive field is moved in both the x and y direction over the input before calculating the next result. CNNs contain multiple types of layers:

- **Fully connected** layers where a single neuron in a layer is connected to all the neurons of previous layer.
- **Locally connected** layers, where neurons in a layer are only connected to small region of the layer before it. This small region of input is called the "*receptive field*".
- **Pooling** layers, which implement a type of nonlinear downsampling. "*Maxpooling*" is the most common form, where a neuron is given as input the maximum value in the set of the output of all the neurons in the previous layer it is connected to.
    - **Convolutional** layers, which are characterized by shared weights. ***Filters*** (aka "*kernels*") are a section/pattern of input replicated across

the entire input field. In convolutional layers, each filter has same parameters (*weights* and *biases*) across the entire input field (i.e. the weights are shared).

Methodology

The project was split into three phases: planning, implementation, and testing. Planning included general as well as specific research. General research was required as the author knew little about machine learning when the project commenced. Implementation included three major sections: implementing the network structure and functions to feed the input through a network, implementing the activation functions necessary to limit input to acceptable ranges, and finally implementing the training functionality. The final phase of the project included testing, not just for correct implementation of the mechanics of the network, but also to provide and demonstrate a working implementation of a functional network.

Research during the planning phase commenced with basic concepts necessary to understand artificial neural networks. Research then focused on commonly used industrial applications of neural networks. At this point, a topology was chosen for the main focus of this SDK. Finally, a focused and complete research into the minutia of the mechanics and mathematics of this topology was carried out.

Implementation of the basic network structure was then commenced. This provided a foundation on which the functionality of learning could be built, as well as a means of solidifying the author's understanding of the process of how input moves through the network. It is at this point that the activation functions (both sigmoid and softmax) were implemented. After the foundation of the basic functionality of the network was completed, work on the learning functionality was begun. This functionality necessitated a thorough and complete understanding of the mathematics behind the learning method, and required tests to ensure proper function. A partial redesign of the network structure to include locations in which to store information (such as connections between neurons) was required at this point.

Although testing was done throughout the implementation process, the final stage of the project focused on tests exclusively. This stage focused on verifying learning was taking place as well as providing working examples of functional networks.

Implementation

The SDK implements learning through a back propagation model using stochastic gradient descent with a cross entropy loss function. A Unified Modeling Language (UML) diagram of the system is available in appendix A.

The entire learning process is orchestrated through a main learning function. This function moves backwards through the layers of the network iteratively updating all weights using the values calculated by feeding a random entry in the learning set through the network (thus implementing the stochastic mode of learning). For each weight, the partial derivative of the weight with respect to the error is calculated with the help of three recursive functions. Each of these functions is designed to handle a separate instance of the calculation: the partial derivatives for the cells of the network, for the weights of the network, and for biases of the network. After the partial derivative is calculated, this derivative value is stored at that location. Future calculations can then call upon these values for their own calculations. This allows for faster learning over the whole network by preventing extraneous repetitive calculations. The weight or bias in question is then updated, and the previous value of that weight or bias is stored for later use during the partial derivative calculations for other weights or biases yet to be updated for this input.

Once the entire network is updated, the learning rate is updated as well using the newly calculated cross entropy error rate over the entire input set. Learning then continues for each of the entries in the input set in turn, in a random order. This constitutes a single iteration of the learning function.

The network allows for four types of layers, including max pooling, convolutional, locally connected and fully connected layers. Layers can be assembled in any order and number the user requires, providing the last layer of a network is a fully connected layer. This limits the complexity of expected values, and allows for easier storage, access, and comparison of actual and expected values.

Two types of activation functions are supplied, including a softmax and sigmoid activation function. The softmax activation function is applied to the output of a network when the output neurons are specified to be dependent. The sigmoid activation function is applied to the output of each neuron, and to the output of the network when the output neurons are specified to be independent.

Input functions are provided for three types of images stored in jpg form: images can be processed as grayscale, HSV (Hue, Saturation, and Value), or RGB (Red, Green, and Blue). Input is automatically scaled to fit the dimensions of the first layer of the network, including depth. The input is processed in grayscale if the first layer has a depth of 1, and in HSV if the layer has a depth of 3.

Results

Network feedforward and backpropagation time varies widely based on size of the network, types of layers utilized, number of iterations specified, and number of learning pairs provided. Learning speed proved to be a limiting factor for all but the simplest of network layouts and tasks. This limited the demonstration of the network significantly. Demonstration of the network is further limited by its effectiveness based on both the ease of a user to understand and follow the function that the network is approximating, and how this applies to each weight within the network. Eleven demonstrations of learning were provided in the form of JUnit test cases. Ten simple learning tests are supplied involving progressively more complicated networks identifying areas of interest in the input provided (see figures 2.i through 2.x). One sample network demonstrating the creation and training of a network for simple color classification based on hue values in an HSV image is also provided.

The following figures depict average cross-entropy over the entire learning set (axis Y) vs iteration for each test (a single iteration refers to a single forward and backwards pass for each input in the training set). Each test is trained and evaluated with the same training and test sets. See also Appendix B: Table of Test Results.

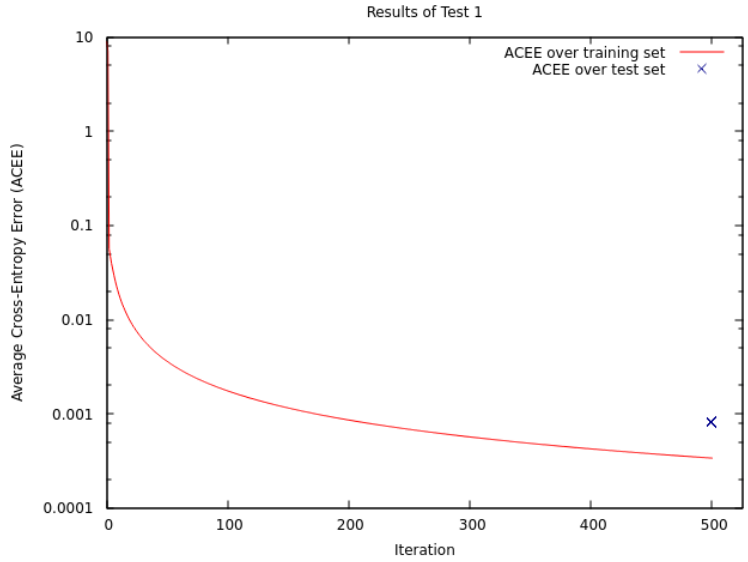


Figure 2.i:
Average Cross Entropy Error (ACEE) vs iteration for the first test. This is the simplest possible demonstration of learning. The network for this test consists of a single layer of width and height of 2 neurons each and depth of 1 neuron. This is a fully connected layer, with a single output. This network is trained to focus on the top left corner of a black and white input image, outputting 1 if this area is black, and 0 if it is white. Output is subjected to a sigmoid activation function.

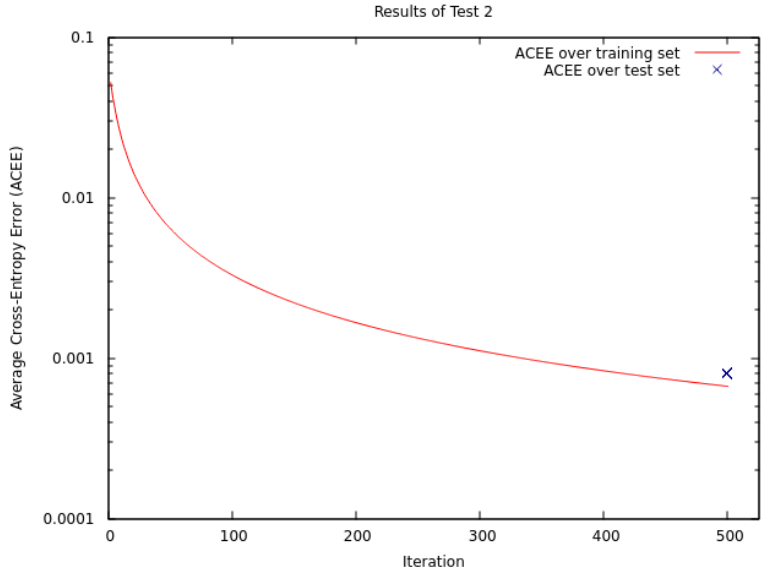


Figure 2.ii:
Average Cross Entropy Error (ACEE) vs iteration for the second test. In this demonstration, the network consists of a single fully connected layer of size 3x3x1 neurons. Input is a black and white image, as before. Output is two independent values subjected to a sigmoid activation function. This network is trained to focus on two areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output value.

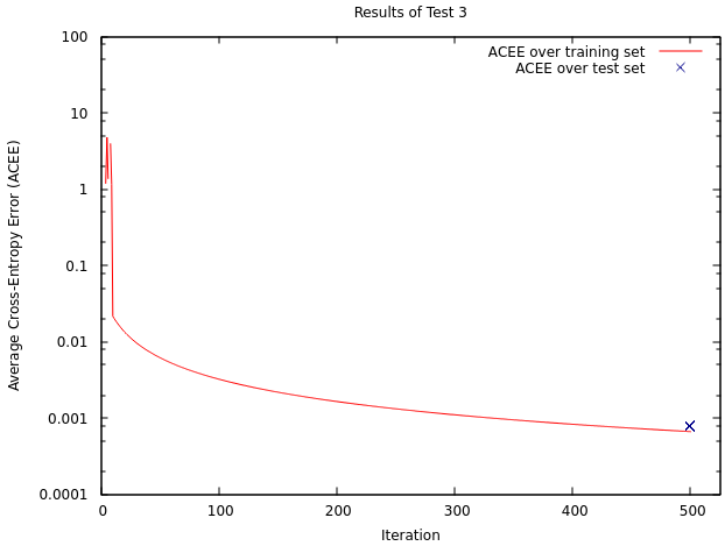


Figure 2.iii:
Average Cross Entropy Error (ACEE) vs iteration for the third test. For this test, the network consists of two layers. Input is a black and white image, as before. The first layer is a maxpooling layer of width and height 4, with 4 pooling filters (of size 2x2, with stride 2). The second layer is a fully connected layer of size 2x2. As in the second demonstration, output is two independent values subjected to a sigmoid activation function. This network is trained to focus on 2 areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output.

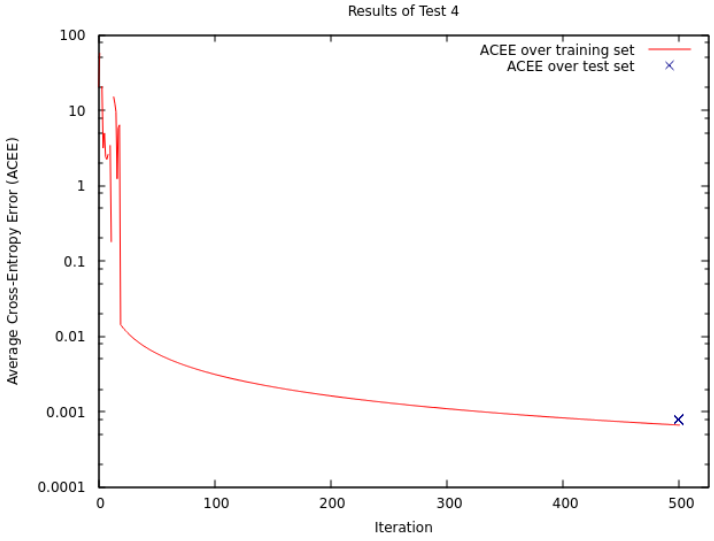


Figure 2.iv:
Average Cross Entropy Error (ACEE) vs iteration for the fourth test. In this test, the network consists of three layers. Input is a black and white image, as before. The first layer is a maxpooling layer of size 8x8, with 16 pooling filters (of size 2x2, with stride 2). The second layer is a maxpooling layer of width and height 4, with 4 pooling filters (of size 2x2, with stride 2). The final layer is a fully connected layer of size 2x2. As in the second and third tests, output is two independent values subjected to a sigmoid activation function, and this network is trained to focus on two areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output value.

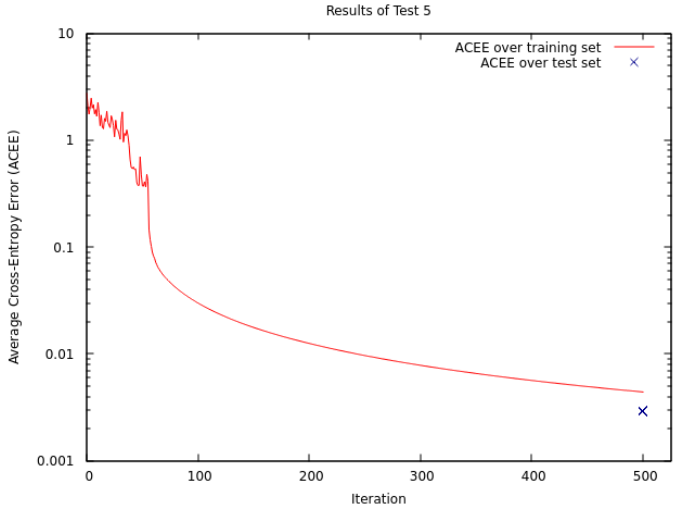


Figure 2.v:
Average Cross Entropy Error (ACEE) vs iteration for the fifth test. For this test, the network consists of three layers. Input is a black and white image, as before. The first layer is a locally connected layer of width and height 8, with 16 local filters (of size 2x2, with stride 2). The second layer is a maxpooling layer of width and height 4, with 4 pooling filters (of size 2x2, with stride 2). The third and final layer is a fully connected layer of size 2x2. As before, output is two independent values subjected to a sigmoid activation function, and this network is trained to focus on two areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output.

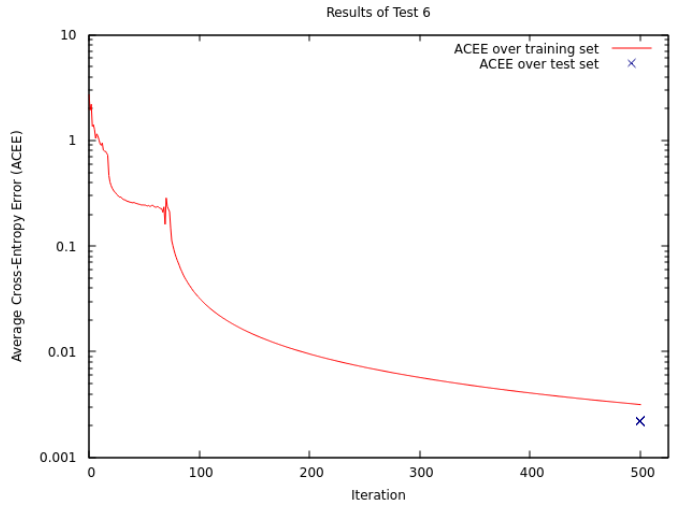


Figure 2.vi:
Average Cross Entropy Error (ACEE) vs iteration for the sixth test. Here, we have a two network. Input is a black and white image, as before. The first layer is a locally connected layer of size 4x4, with 4 local filters (of size 2x2, with stride 2). The second layer is a fully connected layer of size 2x2. Output is 2 independent values subjected to a sigmoid activation function, and the network is trained to focus on two areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output.

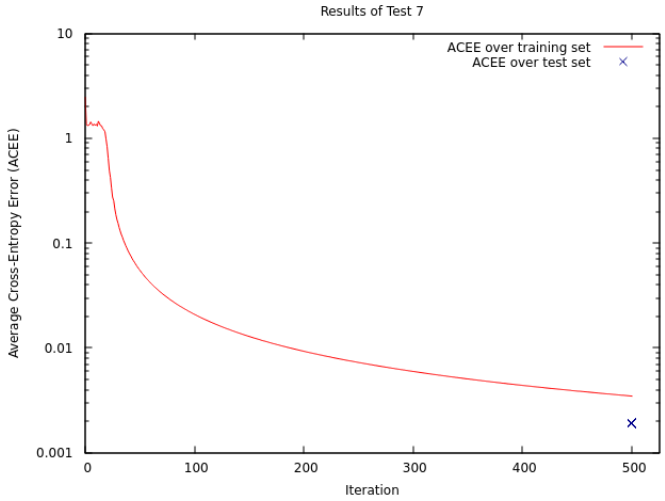


Figure 2.vii:
Average Cross Entropy Error (ACEE) vs iteration for the seventh test. In this test, the network consists of four layers. Input is a black and white image, as before. The first layer is a convolutional layer of width and height 8, with a single filter of size 1x1. The second layer is a maxpooling layer layer of width and height 8, with 16 pooling filters (of size 2x2, with stride 2). The third layer is a locally connected layer of width and height 4, with 4 local filters (of size 2x2, with stride 2). The fourth and final layer is a fully connected layer of size 2x2. As before, output is two independent values subjected to a sigmoid activation function, and this network is trained to focus on two areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output.

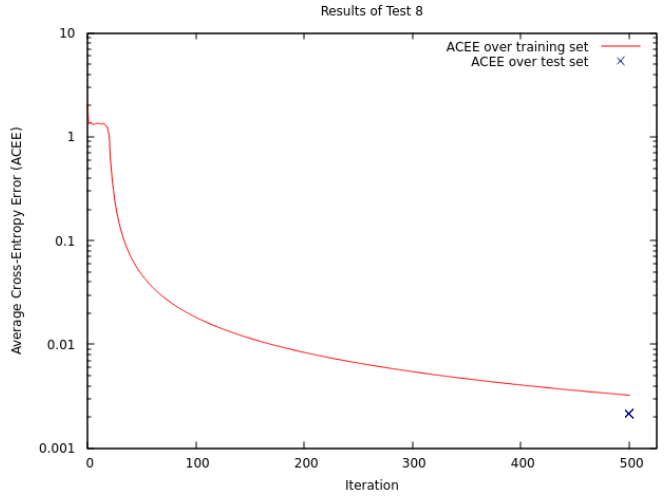


Figure 2.viii:
Average Cross Entropy Error (ACEE) vs iteration for the eighth test. In this demonstration, the network consists of three layers. Input is a black and white image, as before. The first layer is a locally connected layer of width and height 4, with 4 local filters (of size 2x2, with stride 2). The second layer is a convolutional layer of width and height 2, with a single 1x1 filter. The third and final layer is a fully connected layer of size 2x2. Again, output is two independent values subjected to a sigmoid activation function, and this network is trained to focus on two areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output value.

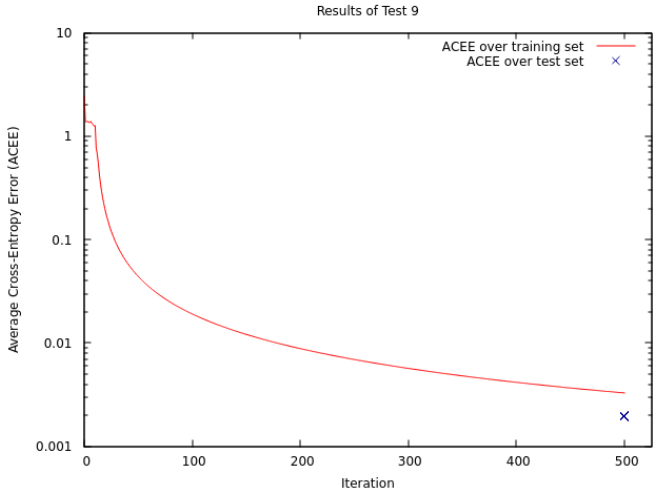


Figure 2.ix:

Average Cross Entropy Error (ACEE) vs iteration for the ninth test. For this test, the network consists of three layers. Input is a black and white image, as before. The first layer is a convolutional layer of width and height 4, with a single filter of size 1x1. The second layer is a locally connected layer of width and height 4, with 4 local filters (of size 2x2, with stride 2). The third and final layer is a fully connected layer of size 2x2. As before, output is two independent values subjected to a sigmoid activation function, and this network is trained to focus on two areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output value.

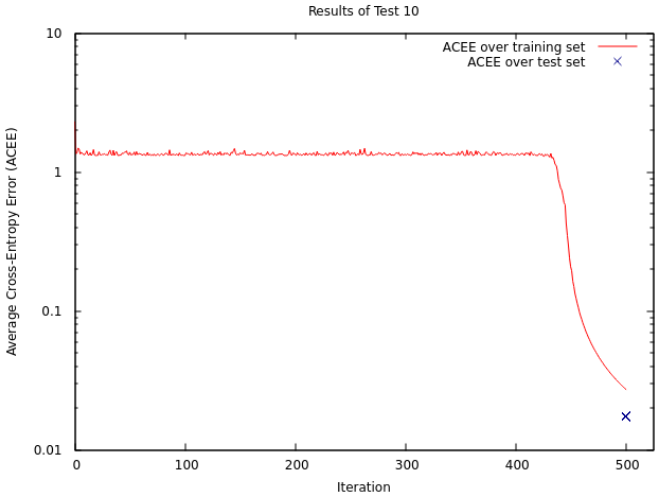


Figure 2.x:

Average Cross Entropy Error (ACEE) vs iteration for the tenth test. In this last test, the network consists of four layers. Input is a black and white image. The first layer is a locally connected layer of width and height 4, with 16 local filters of size 1x1. The second layer is a convolutional layer of width and height 4, with a single filter of size 1x1. The third layer is a locally connected layer of width and height 4, with 4 local filters of size 2x2 (with a stride of 2). The fourth and final layer is a fully connected layer of size 2x2. Again, output is two independent values subjected to a sigmoid activation function, and this network is trained to focus on two areas. The first area of focus is the top left corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the first output value. The second area of focus is the bottom right corner, and the network is trained to output 1 if this area is black, and 0 if it is white for the second output value.

Conclusions

Learning speed proved to be the greatest issue in implementation. Unfortunately, because of the goal of human readability, opportunities for optimization are limited at best. Furthermore, limits on speed have strongly hindered the experimentation needed to produce functional examples. This can be seen in the table below (table 1: comparison of learning times), as even the simplest possible demonstration requires multiple seconds of computation. Concerns regarding speed remain a clear area of focus strongly recommended for future work.

Table: Comparison of Learning Times

All networks received the same 12 training images (3x3 pixels in greyscale JPEG format).
For a more complete table of the data gathered on the test networks, please refer to Appendix B.

Test #	**Network Structure**	**Learning Times** (per 500 iterations)
1	**1 layer:** Fully connected, size 2x2x1 **1 value out**	7.527 seconds
2	**1 layer:** Fully connected, size 3x3x1 **2 independent values out**	8.81 seconds
3	**Layer 1:** Maxpool, size 4x4x1, filter size 2x2x1, stride 2 **Layer 2:** Fully connected, size 2x2x1 **2 independent values out**	9.21 seconds
4	**Layer 1:** Maxpool, size 8x8x1, filter size 2x2x1, stride 2 **Layer 2:** Maxpool, size 4x4x1, filter size 2x2x1, stride 2 **Layer 3:** Fully connected, size 2x2x1 **2 independent values out**	9.532 seconds
5	**Layer 1:** Locally connected, size 8x8x1, filter size 2x2x1, stride 2 **Layer 2:** Maxpool, size 4x4x1, filter size 2x2x1, stride 2 **Layer 3:** Fully connected, size 2x2x1 **2 independent values out**	11.938 seconds
6	**Layer 1:** Locally connected, size 4x4x1, filter size 2x2x1, stride 2 **Layer 2:** Fully connected, size 2x2x1 **2 independent values out**	9.501 seconds
7	**Layer 1:** Convolutional, size 8x8x1, filter size 1x1x1, 1 filter, stride 1 **Layer 2:** Maxpool, size 8x8x1, filter size 2x2x1, stride 2 **Layer 3:** Locally connected, size 4x4x1, filter size 2x2x1, stride 2 **Layer 4:** Fully connected, size 2x2x1 **2 independent values out**	15.036 seconds
8	**Layer 1:** Locally connected, size 4x4x1, filter size 2x2x1, stride 2 **Layer 2:** Convolutional, size 2x2x1, filter size 1x1x1, 1 filter, stride 1 **Layer 3:** Fully connected, size 2x2x1 **2 independent values out**	10.362 seconds
9	**Layer 1:** Convolutional, size 4x4x1, filter size 1x1x1, 1 filter, stride 1 **Layer 2:** Locally connected, size 4x4x1, filter size 2x2x1, stride 2 **Layer 3:** Fully connected, size 2x2x1 **2 independent values out**	10.827 seconds
10	**Layer 1:** Locally connected, size 4x4x1, filter size 1x1x1, stride 1 **Layer 2:** Convoltional, size 4x4x1, filter size 1x1x1, 1 filter, stride 1 **Layer 3:** Locally connected, size 4x4x1, filter size 2x2x1, stride 2 **Layer 4:** Fully connected, size 2x2x1 **2 independent values out**	13.819 seconds

Future Work

Optimization, extending functionality, and further experimentation are clear areas of future focus. Although optimization without sacrificing transparency has proven difficult, speed in learning has shown to be a major impediment to demonstrating all but the simplest examples of real-world functionality. Further research focused into this area may prove fruitful.

Now that basic functionality is established, speed in learning could also be addressed by adding finer control of the learning procedure. A graphical interface for displaying real time learning progress would be helpful by providing the user a broader view of the process. The ability to suspend learning in order to alter parameters, values, or other inputs would also be of utility. Equipping the user with the ability to speed learning through an optimized opaque learning process that can transfer control to the current, more transparent (but significantly slower) function at request could also prove a satisfactory compromise. This could potentially allow the implementation of greater speed through optimization without sacrificing necessary transparency. Such a function could be extended further by allowing learning to be passed to other systems at request, such as to other frameworks or even specialized hardware over a network.

Expanding the collection of provided sample networks is another worthwhile area of future focus. Providing larger, more complicated networks would furnish the user with examples more relevant to real-world applications. This feature could further be extended by providing the user with pre-trained examples of networks in use today. Facebook's Deepface[34] network is one such potential example. Features such as sample networks trained for real-world applications such as face recognition is of significant priority, as this would provide tangible evidence of the value of the concepts demonstrated to even the most casual or non-technical user.

Given the wide variety of neural networks possible, extending the functionality of the SDK to encompass a larger section of this diversity is an

important area of future focus. This includes providing the user with the means to model a wider range of network topologies such as recurrent networks, or networks with bypass neurons. Additional activation functions and a greater variety in the types of learning employed (such as unsupervised, semi-supervised, or reinforcement learning) would also increase the diversity of implementable networks.

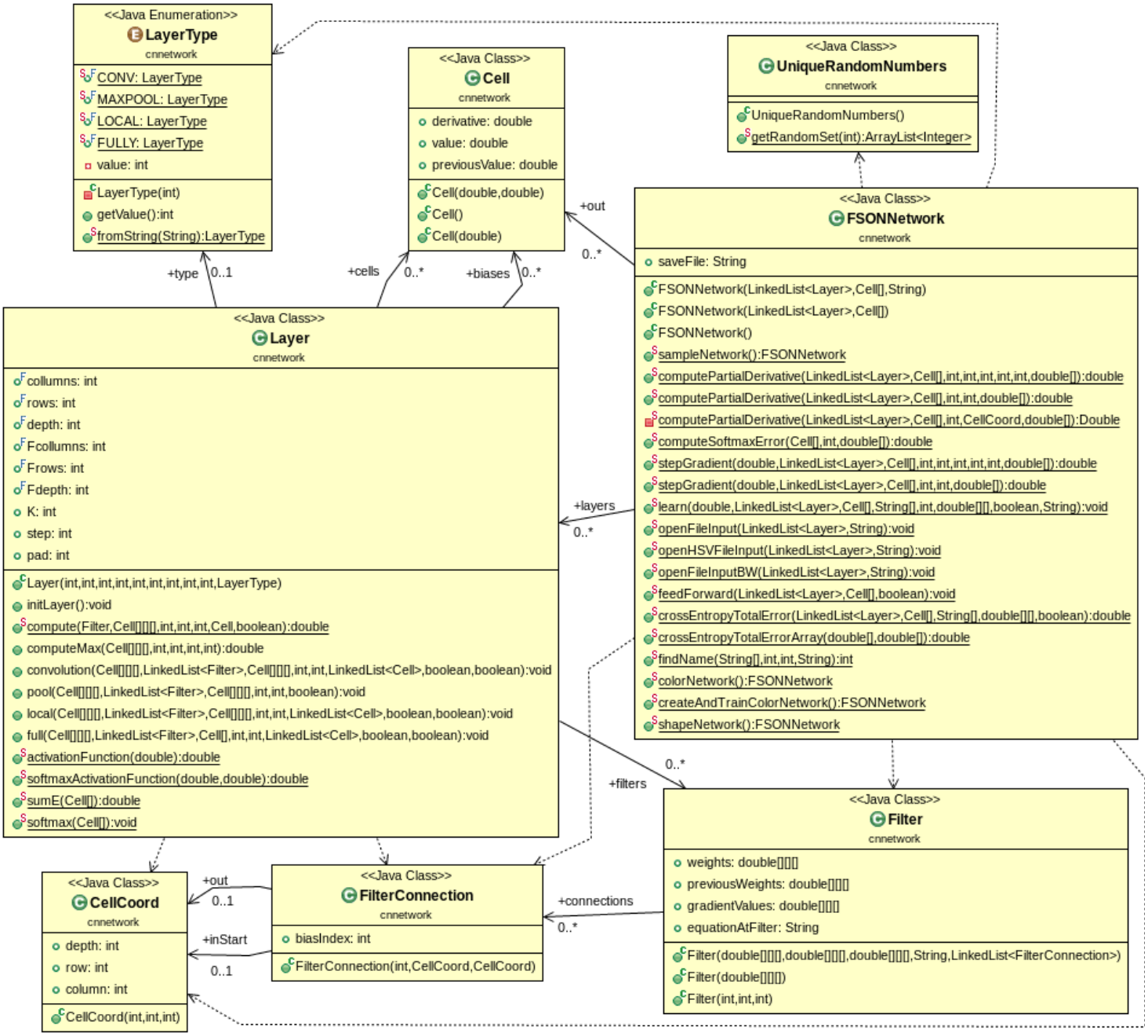
Considering the need for a large amount of data in many applications of artificial neural networks, tools to generate and/or process a greater array of data is of special importance. Providing the user with the ability to mine Internet services (such as Facebook[34]) and/or employ provided resources (such as ImageNet[35] or Labeled Faces in the Wild[17]) would extend functionality greatly.

Given that a set of tools is only as popular as it is useful, ensuring ease of development and use of the SDK is an important endeavor. Included in this area is ensuring up to date cross platform compatibility, conducting user case studies, and maintaining and refining documentation.

Finally, another gainful area of future pursuit is employing the SDK in experimentation in both novel areas of research (such as facial recognition countermeasures) and in comparison to other networks and artificial neural networking tools (in areas such as accuracy and speed).

Appendix A: UML diagram of class structure

Using OMG's UML 2.0 specification[36]. Generated using ObjectAid UML Explorer[37].



Appendix B: Table of Test Results

All input to the networks are given in 3x3 pixel greyscale JPEG format.
Layers are described in the format:

LayerNumber(LayerType, LayerSize, FilterSize, NumberOfFilters, Stride).

The abbreviation “ACEE” refers to average cross-entropy learning error.

Test	**Network Structure**	**Final ACEE** (learning set: 12 samples)	**Final ACEE** (testing set: 5 samples)	**Difference**	**Learning Times** (500 iterations)
1	**1 layer:** L0(Fully, 2x2x1, 2x2x1, 1, 1) **1 value out**	3.380332943491144 x 10^{-4}	8.112796902862001 x 10^{-4}	- 4.732463959370857 x 10^{-4}	7.527 seconds
2	**1 layer:** L0(Fully, 3x3x1, 3x3x1, 2, 1) **2 independent values out**	6.682129709456311 x 10^{-4}	8.018555651347573 x 10^{-4}	- 1.336425941891262 x 10^{-4}	8.81 seconds
3	**2 layers:** L0(Maxpool, 4x4x1, 2x2x1, 4, 2) L1(Fully, 2x2x1, 2x2x1, 2, 1) **2 independent values out**	6.654695449340511 x 10^{-4}	7.985354834221745 x 10^{-4}	- 1.330659384881234 x 10^{-4}	9.21 seconds
4	**3 layers:** L0(Maxpool, 8x8x1, 2x2x1, 16, 2) L1(Maxpool, 4x4x1, 2x2x1, 4, 2) L2(Fully, 2x2x1, 2x2x1, 2, 1) **2 independent values out**	6.644954315139742 x 10^{-4}	7.949182306283689 x 10^{-4}	- 1.304227991143947 x 10^{-4}	9.532 seconds
5	**3 layers:** L0(Local, 8x8x1, 2x2x1, 16, 2) L1(Maxpool, 4x4x1, 2x2x1, 4, 2) L2(Fully, 2x2x1, 2x2x1, 2, 1) **2 independent values out**	4.407808852343283 x 10^{-3}	2.951897022807179 x 10^{-3}	+ 1.455911829536104 x 10^{-3}	11.938 seconds
6	**2 layers:** L0(Local, 4x4x1, 2x2x1, 4, 2) L1(Fully, 2x2x1, 2x2x1, 2, 1) **2 independent values out**	3.1616334224599785 x 10^{-3}	2.2005509730627613 x 10^{-3}	+ 9.610824493972172 x 10^{-4}	9.501 seconds
7	**4 layers:** L0(Convolution, 8x8x1, 1x1x1, 1, 1) L1(Maxpool, 8x8x1, 2x2x1, 16, 2) L2(Local, 4x4x1, 2x2x1, 4, 2) L3(Fully, 2x2x1, 2x2x1, 2, 1) **2 independent values out**	3.4731167370997793 x 10^{-3}	1.9023890812716916 x 10^{-3}	+ 1.5707276558280877 x 10^{-3}	15.036 seconds
8	**3 layers:** L0(Local, 4x4x1, 2x2x1, 4, 2) L1(Convolution, 2x2x1, 1x1x1, 1, 1) L2(Fully, 2x2x1, 2x2x1, 2, 1) **2 independent values out**	3.240792599740226 x 10^{-3}	2.1625152947452437 x 10^{-3}	+ 1.0782773049949823 x 10^{-3}	10.362 seconds
9	**3 layers:** L0(Convolution, 4x4x1, 1x1x1, 1, 1) L1(Local, 4x4x1, 2x2x1, 4, 2) L2(Fully, 2x2x1, 2x2x1, 2, 1) **2 independent values out**	3.286399310456619 x 10^{-3}	1.963023510234371 x 10^{-3}	+ 1.323375800222248 x 10^{-3}	10.827 seconds
10	**4 layers:** L0(Local, 4x4x1, 1x1x1, 16, 1) L1(Convolution, 4x4x1, 1x1x1, 1, 1) L2(Local, 4x4x1, 2x2x1, 4, 2) L3(Fully, 2x2x1, 2x2x1, 2, 1) **2 independent values out**	0.02740031243767276	0.01741656400424064	+ 9.98374843343212 x 10^{-3}	13.819 seconds

References

[1] “Amazon Rekognition – Deep learning-based image analysis,” *Amazon Web Services, Inc*. [Online]. Available: https://aws.amazon.com/rekognition/. [Accessed: 23-Apr-2017].

[2] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, “Dropout Improves Recurrent Neural Networks for Handwriting Recognition,” arXiv:1312.4569v2 [cs.CV], Mar. 2014.

[3] S. Doyle-Lindrud, “Watson Will See You Now: A Supercomputer to Help Clinicians Make Informed Treatment Decisions,” *Clinical Journal of Oncology Nursing*, vol. 19, no. 1, pp. 31–32, 01-Feb-2015.

[4] S. Nadella, “Microsoft’s CEO Explores How Humans and A.I. Can Solve Society’s Challenges—Together,” *Slate Magazine*, 28-Jun-2016. [Online]. Available: http://www.slate.com/articles/technology/future_tense/2016/06/microsoft_ceo_satya_nadella_humans_and_a_i_can_work_together_to_solve_society.html. [Accessed: 23-Apr-2017].

[5] C. Metz, “AI Is Transforming Google Search. The Rest of the Web Is Next,” *Wired*, 04-Feb-2016. [Online]. Available: https://www.wired.com/2016/02/ai-is-changing-the-technology-behind-google-searches/. [Accessed: 23-Apr-2017].

[6] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[7] Abadi, Martín, et al. "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symposium on Operating Systems Design and Implementation,* Savannah, Georgia, 2016.

[8] “Tiobe index 2017,” *TIOBE software.* [Online]. Available: https://www.tiobe.com/tiobe-index/. [Accessed: 23-Apr-2017].

[9] F. Stahl and I. Jordanov, “An overview of the use of neural networks for data mining tasks,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 193–208, 2012.

[10] S. Engels, V. Lakshmanan, and M. Craig, “Plagiarism detection using feature-based neural networks,” *ACM SIGCSE Bulletin*, vol. 39, no. 1, pp. 34–38, Mar-2007. doi: 10.1145/1227504.1227324

[11] Oancea, Bogdan, et al. “Predicting students’ results in higher education using neural networks,” in *Applied Information and Communication Technologies,* Jelgava, Latvia, 2013.

[12] Brabazon, Anthony, et al. “Identifying online credit card fraud using Artificial Immune Systems,” in *IEEE Congress on Evolutionary Computation*, Barcelona, Spain, 2010.

[13] N. Collier, “Uncovering text mining: A survey of current work on web-based epidemic intelligence,” *Global Public Health*, vol. 7, no. 7, pp. 731–749, Aug. 2012.

[14] Office of Justice Programs and L. J. Kangas, *Artificial Neural Network System for Classification of Offenders in Murder and Rape Cases*. NCJ: 190984, Grant Number: 97-IJ-CX-K007

[15] “World biggest online cameras directory,” *Insecam*. [Online]. Available: https://www.insecam.org/. [Accessed: 23-Apr-2017].

[16] “Movidius Strikes Deal with Hikvision to Bring Artificial Intelligence to Intelligent Cameras,” *Movidius*, 24-Oct-2016. [Online]. Available: https://www.movidius.com/news/movidius-strikes-deal-with-hikvision-to-bring-artificial-intelligence-to-intelligent-cameras. [Accessed: 23-Apr-2017].

[17] E. Learned-Miller, G. B. Huang, A. RoyChowdhury, H. Li, and G. Hua, “Labeled Faces in the Wild: A Survey,” in *Advances in Face Detection and Facial Image Analysis*, Springer, pp. 189–248. ISBN 978-3-319-25958-1

[18] G. Yoav, “A Primer on Neural Network Models for Natural Language Processing,” *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, Nov. 2016. Availible: https://www.jair.org/media/4992/live-4992-9623-jair.pdf [Accessed: 23-Apr-2017].

[19] M. M. Kasar, D. Bhattacharyya, and T.-hoon Kim, “Face Recognition Using Neural Network: A Review,” *International Journal of Security and Its Applications*, vol. 10, no. 3, pp. 81–100, Mar. 2016.

[20] G. U. Kharat and S. V. Dudul, “Emotion Recognition from Facial Expression Using Neural Networks,” *Human-Computer Systems Interaction Advances in Soft Computing*, vol. 60, pp. 207–219, 2009.

[21] A. Das, H. Agrawal, L. Zitnick, D. Parikh, and D. Batra, “Human Attention in Visual Question Answering: Do Humans and Deep Networks look at the same regions?” In *Proc. 2016 Conf. Empirical Methods in Natural Language Processing*, Austin, Texas, 2016. arXiv:1606.03556

[22] S. Levin, “A beauty contest was judged by AI and the robots didn't like dark skin,” *The Guardian*, 08-Sep-2016. [Online]. Available: https://www.theguardian.com/technology/2016/sep/08/artificial-intelligence-beauty-contest-doesnt-like-black-people. [Accessed: 23-Apr-2017].

[23] A. Caliskan, J. J. Bryson, and A. Narayanan, “Semantics derived automatically from language corpora contain human-like biases,” *Science*, vol. 356, no. 6334, pp. 183–186, Apr. 2017. arXiv:1608.07187v2

[24] T. Jefferson, “Letter to Samuel Kercheval,” *Teaching American History*, 12-Jun-1816. [Online]. Available: http://teachingamericanhistory.org/library/document/letter-to-samuel-kercheval/. [Accessed: 23-Apr-2017].

[25] J. Mena, *Machine learning forensics for law enforcement, security, and intelligence*. Boca Raton, FL: CRC Press, 2011.

[26] “Long-Time Fugitive Neil Stammer Captured,” *FBI.gov*, 12-Aug-2014. [Online]. Available: https://www.fbi.gov/news/stories/long-time-fugitive-neil-stammer-captured. [Accessed: 23-Apr-2017].

[27] L. Best-Rowden and A. K. Jain, “A longitudinal study of automatic face recognition,” In *2015 International Conf. on Biometrics (ICB)*, Phuket, Thailand, May 2015. DOI: 10.1109/ICB.2015.7139087

[28] M. Wagner, “The Dehumanization of International Humanitarian Law: Legal, Ethical, and Political Implications of Autonomous Weapon Systems,” *Vanderbilt Journal of Transnational Law*, vol. 47, 2014.

[29] J. E. Perry-Smith, “Social Yet Creative: The Role Of Social Relationships In Facilitating Individual Creativity.,” *Academy of Management Journal*, vol. 49, no. 1, pp. 85–101, Feb. 2006.

[30] D. J. Pauleen and P. Yoong, “Facilitating virtual team relationships via Internet and conventional communication channels,” *Internet Research*, vol. 11, no. 3, pp. 190–202, 2001.

[31] T. W. Bickmore, “Relational agents: effecting change through human-computer relationships,” thesis, 2003. [Online]. Availible: http://www.ccis.northeastern.edu/home/bickmore/bickmore-thesis.pdf. [Accessed: 23-Apr-2017].

[32] Mitrou, Lilian, et al. "Social media profiling: A Panopticon or Omniopticon tool?." *Proc. of the 6th Conference of the Surveillance Studies Network*. Barcelona, Spain, 2014.

[33] D. W. Nickerson and T. Rogers, “Political Campaigns and Big Data,” *Journal of Economic Perspectives*, vol. 28, no. 2, pp. 51–73, 2014.

[34] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” In *2014 IEEE Conf. on Computer Vision and Pattern Recognition*, Columbus, Ohio, 2014.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision,* vol. 115, no. 3, pp. 211–252, Nov. 2015. arXiv:1409.0575v3

[36] “Documents associated with UML® Version 2.0,” *UML*, 04-Jul-2005. [Online]. Available: http://www.omg.org/spec/UML/2.0/. [Accessed: 23-Apr-2017].

[37] “ObjectAid UML Explorer,” *ObjectAid*. ObjectAid LLC, 20-Apr-2017.