Spring 5-18-2015

Using Neural Networks for Image Classification

Tim Kang
San Jose State University

Using Neural Networks for Image Classification

San Jose State University, CS298 Spring 2015

Author: Tim Kang (timothykang.x@gmail.com), San Jose State University
Advisor: Robert Chun (robert.chun@sjsu.edu), San Jose State University
Committee: Thomas Austin (thomas.austin@sjsu.edu), San Jose State University
Committee: Thomas Howell (thomas.howell@sjsu.edu), San Jose State University

Abstract

This paper will focus on applying neural network machine learning methods to images for the purpose of automatic detection and classification. The main advantage of using neural network methods in this project is its adeptness at fitting non-linear data and its ability to work as an unsupervised algorithm. The algorithms will be run on common, publically available datasets, namely the MNIST and CIFAR-10, so that our results will be easily reproducible.

Table of Contents

Introduction

Computer vision is a problem that has existed for a long time. In this paper, we will be focusing on the task of classification of computer images into different preset categories. This specific part of computer vision has many diverse real world applications, ranging from video games to self-driving cars. However, it also has been traditionally very difficult to pull off successfully, due to the enormous amount of different factors (camera angles, lighting, color balance, resolution etc.) that go into creating an image.

We will be focusing on using artificial neural networks for image classification. While artificial neural networks are some of the oldest machine learning algorithms in existence, they have not been widely used in the field of computer vision. More recent improvements in the methods of training artificial neural networks have made them worth looking into once again for the task of image classification.

Background

Short History of Artificial Neural Networks

Artificial neural networks were designed to be modeled after the structure of the brain. They were first devised in 1943 by researchers Warren McCulloch and Walter Pitts [1].

Back then, the model was initially called threshold logic, which branched into two different approaches: one inspired more by biological processes and one focused on artificial intelligence applications.

Although artificial neural networks initially saw a lot of research and development, their popularity soon declined and research slowed because of technical limitations. The computational intensity of artificial neural networks was too complicated for the computers at the time. Computers at the time did not have sufficient computational power and would take too long to train neural networks. As a result, other machine learning techniques became more popular and artificial neural networks were mostly neglected.

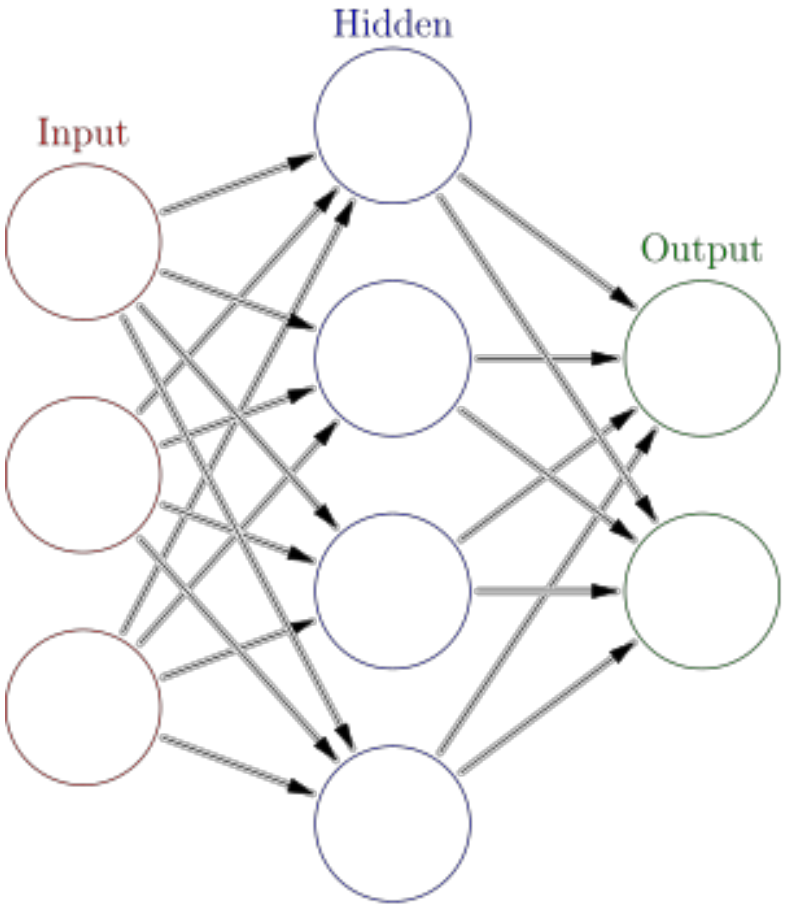
However, one important algorithm related to artificial neural networks was developed during this time -- backpropagation, discovered by Paul Werbos [2]. Backpropagation is a way of training artificial neural networks by attempting to minimize the errors. This algorithm allowed scientists to train artificial networks much more quickly.

Artificial neural networks became popular once again in the late 2000s when companies like Google and Facebook showed the advantages of using machine learning techniques on big datasets collected from everyday users. These algorithms are nowadays mostly used for deep learning, which is an area of machine learning that tries

to model relationships that are more complicated -- for example, non-linear relationships.

Quick Explanation of Artificial Neural Networks

Each artificial neural network consists of many hidden layers. Each hidden layer in the artificial neural network consists of multiple nodes. Each node is linked to other nodes using incoming and outgoing connections. Each of the connections can have a different, adjustable weight. Data is passed through these many hidden layers and the output is eventually interpreted as different results.



In this example diagram, there are three input nodes, shown in red. Each input node represents a parameter from the dataset being used. Ideally the data from the dataset would be preprocessed and normalized before being put into the input nodes.

There is only one hidden layer in this example and it is represented by the nodes in blue. This hidden layer has four nodes in it. Some artificial neural networks have more than one hidden layer.

The output layer in this example diagram is shown in green and has two nodes. The connections between all the nodes (represented by black arrows in this diagram) are weighted differently during the training process.

A Little Regarding Image Classification

Image recognition and classification is a problem that has been around for a long time and has many real world applications. Police can use image recognition and classification to help identify suspects in security footage. Banks can use it to help sort out checks. More recently, Google has been using it in their self-driving car program.

Traditionally, a lot of different machine learning algorithms have been utilized for image classification, including template matching, support vector machines, k-NN, and hidden Markov models. Image classification remains one of the most difficult problems in machine learning, even today.

Related Work

In academia, there is related work done by Professor Andrew Ng at Stanford University, Professor Geoffrey Hinton at University of Toronto, Professor Yann LeCun at New York University, and Professor Michael Jordan at UC Berkeley. Much of their work deals with applying artificial neural networks or other machine learning algorithms. Following is a

sampling of a few papers from the large body of work that is available. These papers are all fairly recent and are more geared towards specific applications of machine learning and artificial neural networks. There is also a lot of research involving machine learning and artificial neural networks going on in industry, most specifically at Google and Facebook, which we will talk about briefly.

Deep Learning with COTS HPC Systems

This is a collaboration between Andrew Ng's Stanford research group and NVIDIA [3]. One of the main problems facing machine learning today is training the systems. As data sets get larger and larger, more and more computing power is required to train the models. In fact, artificial neural networks are especially hard to train.

The paper presents a more inexpensive alternative to training machine learning models by using COTS HPC (community off-the-shelf high performance computing) hardware. COTS HPC hardware refers to computer hardware that can be bought at your typical computer hardware store: things like Intel or AMD CPUs. In this case, the COTS HPC used were NVIDIA GPUs.

Their setup was of 16 servers. Each server had two quad-core CPUs, four NVIDIA GTX 680 GPUs and the FDR Infiniband adapter (for low latency communication). They chose this configuration specifically to balance the number of GPUs with CPUs, citing past

examples where too many GPUs overwhelmed the systems through I/O, cooling, CPU compute, and power issues.

Their software was written in C++ and built on top of the previous MVAPICH MPI framework, chosen to make communication between different processes easier. The code also includes GPU code written in NVIDIA's NVI CUDA language.

In the end, they were able to train deep learning systems (including neural networks) with over 11 billion parameters, which is several times what other people were able to do before.

Learning New Facts from Knowledge Bases with Neural Tensor Networks and Semantic Word Vectors

This is another paper from Andrew Ng's Stanford research group, this time focused on using neural networks to try to extract data and insights from unannotated text [4]. It focuses on lexical databases like WordNet and Yago. These databases store information about English words, specifically definition and usage, and also provide information about the relationship between different English words. They are commonly used in aritificial intelligence and text processing research.

The paper uses a specific type of neural network called a neural tensor network. This type of neural network is used because it is easier to adapt to words and their relationships. A big advantage of using this type of model is that it can relate two inputs directly. The models were trained using gradients.

Using the WordNet data set, the trained model was asked to classify whether an arbitrary triplet of entities and relations is true or not. False examples were created by purposely messing up existing known triplets by switching entities and relations. This model was able to achieve 75.8% accuracy, which is much better than what other researchers were able to achieve before, with similarity based models (66.7%) and Hadamard based models (71.9%).

Convolutional Recursive Deep Learning for 3D Object Classification

This is yet another paper from Andrew Ng's Stanford research group. This time the paper is about using neural networks to classify images of objects, specifically focusing on RGB-D images [5]. RGB-D images are images that also have depth information in addition to the typical color information included in images. A good, everyday example of a device that captures RGB-D information is the Kinect sensor created by Microsoft for the Xbox One and the Xbox 360.

The paper uses a type of neural network called the convolutional-recursive neural network (CNN-RNN) for learning and classifying RGB-D images. This actually consists of two different networks. The convolutional neural network is first trained in an unsupervised manner by clustering the images. This is then used to create the filters for the CNNs. The resulting features are then fed into the RNNs, which classify the images.

The data set used in this paper was the RGB-D Object Dataset from the University of Washington, organized by Kevin Lai. The paper was successful in classification of the objects in the Lai dataset, outperforming most previous attempts made using the same dataset.

A Fast Algorithm for Deep Belief Nets

This paper is from Geoffrey Hinton's group at the University of Toronto and deals with using the concept of "complementary priors" to both speed up and improve neural networks that have a large number of hidden layers [6]. This is done by creating a neural network with the top two layers as undirected associated memory and the rest of the hidden layers as an acyclic graph. There are a few advantages of doing this, most notably that it allows the neural network to find a decent set of parameters much more rapidly.

The paper uses the commonly referenced MNIST handwritten digits database to test out this new way of creating a neural network. Another advantage gained from using the MNIST is that there has already been a lot of research published using the MNIST so that it would be easy for the researchers to find other methods to compare against. The results this paper got after running against the MNIST database were favorable to other results obtained by using the more common feed forward neural networks.

While the initial results were good, the paper also outlines several problems that could limit the power of this particular method. For example, it treats non binary images as probabilities, which won't work for natural images.

Reducing the Dimensionality of Data with Neural Networks

This is a publication by Geoff Hinton originally appearing in the Science Magazine in July 2006 and deals with the problem of using neural networks to reduce the dimensionality of data [7]. The problem of dimensionality reduction has been traditionally tackled using methods like principal components analysis (PCA). Principal components analysis basically looks for the greatest variance in the data set and can be done with algorithms like singular value decomposition.

This paper discusses using a type of neural network called an "auto encoder" as an alternative to using principal components analysis. An auto encoder is a multilayer

neural network that can take high dimensional data, encode it into low dimensional format and also has the ability to try to reconstruct the original high dimensional data using files in the low dimensional format.

The paper tests the trained auto encoder on several different data sets. One of the data sets used was a custom randomly generated set of curves in the two dimensional space. For this data set, the auto encoder was able to produce much better reconstructions than PCA. The MNIST digits dataset, Olivetti face data set, and a data set of documents were also used. Once again, the auto encoder was able to perform better than PCA.

To Recognize Shapes, First Learn to Generate Images

This paper is from Geoffrey Hinton at the University of Toronto and the Canadian Institute for Advanced Research and deals with the problem of training multilayer neural networks [8]. It is an overview of the many different strategies that are used to train multilayer neural networks today.

First it discusses "five strategies for learning neural networks," which include denial, evolution, procrastination, calculus, and generative. Out of these five strategies, the most significant ones are strategies four and five. Calculus includes the strategy of

backpropagation, which has been independently discovered by multiple researchers. Generative includes the "wake-sleep" algorithm.

The rest of the paper goes over many common ways of training a multilayer neural network including "learning feature detectors with no supervision, learning one layer of feature detectors (with restricted Boltzmann machines), a greedy algorithm for learning multiple hidden layers, using backpropagation for discrimininative fine tuning, and using contrastive wake-sleep for generative fine tuning."

Learning Multiple Layers of Representation

This is a paper by Geoffrey Hinton that deals with the problem of training multilayer neural networks [9]. Training multilayer neural networks has been done mostly using the backpropagation algorithm. Backpropagation is also the first computationally feasible algorithm that can be used to train multiple layers. However, backpropagation also has several limitations, including requiring labeled data and becoming slow when used on neural networks with excessive amounts of layers.

This paper proposes using generative models to solve that problem. Neural networks can be run "bottom-up" in order to make recognition models or "top-down" to make generative connections. Running "top-down" through a neural network with stochastic neurons will result in creating an input vector. The paper suggests training models by tweaking the weights on the top level connections and trying to achieve the maximum similarity with the original training data, with the reasoning being that the higher level features will be able to affect the outcome more than the lower level features. According to the paper, the key to making this work is the restricted Boltzmann machine (RBM).

The paper tests its trained models on two different data sets, the MNIST handwritten digits data set and also a data set of sequential images. In the end, Hinton attributes the success of this model to three different factors: using a generative model instead of

attempting to classify, using restricted Boltzmann machines to learn one layer at a time, and having a separate fine tuning stage.

Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting

This paper is by Yann LeCun from the Courant Institute at New York University and deals with object recognition [10]. Recognition of objects using just shape information, without accounting for pose, lighting, and backgrounds, is a problem that has not been dealt with frequently in the field of computer vision.

The paper uses the NORB data set, which is a large data set with 97,200 image pairs of 50 objects that belong to five different categories, specifically human figures, animals, airplanes, trucks and cars. This NORB data set was used to generate other data sets where the images vary in location, scale, brightness, etc.

A wide range of different machine learning methods were used to test the images from the data sets, including linear classifier, K-NN (nearest neighbor with Euclidean distance), pairwise SVM (support vector machines with a Gaussian kernel), and convolutional neural networks. For the most part, convolutional neural networks ended up with good performance compared to the other methods, except surprisingly on the jittered-clustered data set. Many of the other methods also ran into significant troubles

because of limitations in CPU power and time and could not be trained reasonably on many of the data sets tested.

Scaling Algorithms Towards AI

This is a collaboration between Yoshua Bengio of the University of Montreal and Yann LeCun of New York University and deals with the long term problem of training algorithms to work with artificial intelligence [11]. The paper discusses many of the common limitations found when working with artificial intelligence, mainly “shallow architecture” and “local estimators.”

The paper does not attempt to find a general learning method, saying that such a task is “doomed to failure.” Instead, they attempt to look for learning methods for specific tasks, reasoning that finding out these methods will bring them closer to creating an artificially intelligent agent.

The paper then goes into more detail on the advantages and disadvantages of different algorithm setups, for example deep versus shallow. It also compares various algorithms like support vector machines and convolutional neural networks against each other, using data sets like the MNIST handwritten digits data set and the NORB data set.

Comparing SVM and Convolutional Networks for Epileptic Seizure Prediction from Intracranial EEG

This is a paper from Yann LeCun of New York University that focuses on using machine learning methods like support vector machines and convolutional neural networks in order to try to predict when epileptic seizures will occur [12]. Epilepsy is a neural disease that affects around one to two percent of the world population and causes its victims to have seizures occassionally. There has been a lot of other research done on trying to predict when these seizures will occur, but almost none using modern machine learning methods.

The paper uses data gathered from an electroencephalagraphy machine, which records the local voltage of millions of brain cells at a time. Current methods of seizure prediction suffer from a tradeoff between being able to predict the seizures and avoiding false alarms when predicting the seizures. The most common approach currently used is binary classification, which is susceptible to these problems. The machine learning algorithms used by this paper can mitigate these problems because of their ability to clasify non linear features in a high dimensional feature space.

The paper then uses MATLAB to implement the support vector machines and convolutional neural networks used. The results were highly successful, especially for

the convolutional neural networks, which were able to achieve no false alarms on all of the patients except for one.

Google

Google has been focusing much more on its machine learning and data science departments. Almost every product at Google uses some sort of machine learning or data science. For example, Google Adsense uses data science to better target ads towards customers and Picasa uses machine learning to recognize faces in images.

One of the more interesting Google products using machine learning and data science is DeepMind. DeepMind Technologies was a tech startup based in London that was acquired by Google near the beginning of 2014 [13]. Their goal is to “combine the best techniques from machine learning and systems neuroscience to build powerful general-purpose learning algorithms.”
DeepMind has, in fact, trained a neural network to play video games, including classics like Pong and Space Invaders [42].

Facebook

Like Google, Facebook has also been focusing a lot on machine learning and data science. This is because, as a company relying heavily on advertising for revenue, and

as a company with huge amounts of user personal data, machine learning and data science will allow them to target their ads better and get more revenue.

The main Facebook operation is currently based in New York City. In 2014, Facebook hired New York University professor and famed neural network researcher Yann LeCun to help head and lead this operation.

Proposed Approach

Machine Learning Libaries

We considered many different machine learning libraries, including:

- Torch7
- Theano / PyLearn
- Caffe

Torch7 is a machine learning library that is being developed at New York University, the Idiap Research Institute, and NEC Laboratories America [14]. According to their description:

> Torch7 is a scientific computing framework with wide support for machine learning algorithms. It is easy to use and provides a very efficient implementation, thanks to an easy and fast scripting language, LuaJIT, and an underlying C implementation.

Theano is a machine learning library being developed mainly at the University of Montreal [15]. It is a Python library and is more of a general purpose computer algebra system library with a emphasis on matrix operations.

Caffe is a machine learning library being developed at UC Berkeley [16]. According to their description:

> Caffe is a deep learning framework developed with cleanliness, readability, and speed in mind. It was created by Yangqing Jia during his PhD at UC Berkeley, and is in active development by the Berkeley Vision and Learning Center (BVLC) and by community contributors. Caffe is released under the BSD 2-Clause license.

It is important to note that all three libraries have a focus on the deep learning aspect of machine learning, but are also configurable in many different ways and can support many other algorithms.

After some preliminary evaluation, we decided on using the Torch7 machine learning library. This library was chosen out of the different machine learning frameworks that support artificial neural networks because of many reasons. Speed wise, it is a lot faster than the alternatives. It also supports interfacing with C and CUDA code easily. Finally, out of the frameworks considered, at this point in time, it is the most commonly used in industry. Both Facebook and Google have teams that are using Torch7 for machine learning research.

Data Sets

We also considered many different image datasets, including:

- Caltech 101
- PASCAL VOC
- MNIST Digits
- Flower classification data set
- Stanford Dogs
- Animals with Attributes
- Cifar10

When considering the different image data sets, we took into consideration the size of the data set, the content of the images, and the format the data in the data set is

presented. We wanted a data set that already had the images well formatted and would be easy to work with using our machine learning libraries.

The Caltech 101 dataset [17] is, according to their description:

> Pictures of objects belonging to 101 categories. About 40 to 800 images per category. Most categories have about 50 images. Collected in September 2003 by Fei-Fei Li, Marco Andreetto, and Marc 'Aurelio Ranzato. The size of each image is roughly 300 x 200 pixels.

This dataset of images also contains outline annotations of the different objects shown, which possibly could come in useful later.

The PASCAL VOC dataset [18] is from the 2009 challenge run by the PASCAL2 Network of Excellence on Pattern Analysis, Statistical Modelling, and Computational Learning and funded by the EU.

According to their website, PASCAL VOC contains twenty categories of everyday objects:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep

- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

The MNIST digits dataset [25] is a large collection of images of handwritten digits. They have a training set of 60,000 examples and a testing set of 10,000 examples. The images have already been centered and normalized and the whole dataset is a smaller subset of a larger data set available from the NIST (National Institute of Standards and Technology).

The Flower Classification dataset is from the Visual Geometry Group at the University of Oxford. There are actually two versions of the dataset, one with 17 categories and one with 102 categories.

The flowers are common flowers seen in the United Kingdom [19].

Stanford Dogs dataset [20] is a dataset from Stanford University. According to their website:

> The Stanford Dogs dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization.

It contains 20,580 images of dogs sorted into 120 different categories with class labels and bounding boxes.

Animals with Attributes [21] is a dataset from the Max Planck Institute for Biological Cybernetics, which is located in Tübingen, Baden-Württemberg, Germany. According to their description:

> This dataset provides a plattform to benchmark transfer-learning algorithms, in particular attribute base classification. It consists of 30475 images of 50 animal classes with six pre-extracted feature representations for each image. The animal classes are aligned with Osherson's classical class/attribute matrix, thereby providing 85 numeric attribute values for each class. Using the shared attributes, it is possible to transfer information between different classes.

Cifar10 [36] is a dataset put together by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton from the University of Toronto. It is a smaller subset of the larger 80 million tiny images dataset, but with the advantage of having everything labeled. There are 50000 training images and 10 different categories -- airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.

After looking over all the different data sets, it was decided that we will be mainly using the MNIST digits dataset. A lot of prior research has been done using this dataset so we

can easily compare results to tests that others have run before. We will also be running additional tests using the Cifar10 dataset due to the excellent support it has with our other tools.

Hardware

All of this will be run on a computer running Ubuntu Linux 14.04 Trusty Tahr with the following specifications:

- AMD Phenom II X3 720
- 4GB RAM
- Nvidia Geforce 750 Ti
- 500GB 7200 rpm HDD

The Nvidia Geforce 750 Ti graphics card can be especially useful because Torch7 is also coded to include support for the Nvidia CUDA library. Nvidia CUDA allows programs to use the massively parallel computing power of a graphics card. The rest of the hardware was chosen simply just because it was readily available.

Implementation

Overview

The implementation of our neural network requires many different steps.

The first step required is dataset preparation. Even for those that are commonly used, datasets come in many different formats. It is often necessary to write a few short scripts that will take in the examples from the dataset and then format them properly for the machine learning tools that will be used.

It is also often necessary to do what is known as feature engineering. Examples from datasets can have too many features. Running a training algorithm on a dataset with too many features can cause the algorithm to become confused and produce subpar results. Therefore, it is necessary to pick out which features to keep and which to remove (or to give less weight to). This can be done manually by hand or using an algorithm like PCA (principal components analysis).

Finally, even after successfully running the algorithm on a dataset, it may be helpful to tweak some parameters and re-run the algorithm.

MNIST Digits and Cifar10 Details

As mentioned above, the MNIST Digits is a dataset of handwritten digits. There are 60,000 different handwritten digit files available in this particular dataset, designed to be used for training the selected algorithm. There are also 10,000 digit files from the dataset designed to be available for testing out the algorithm after it has been trained [25].

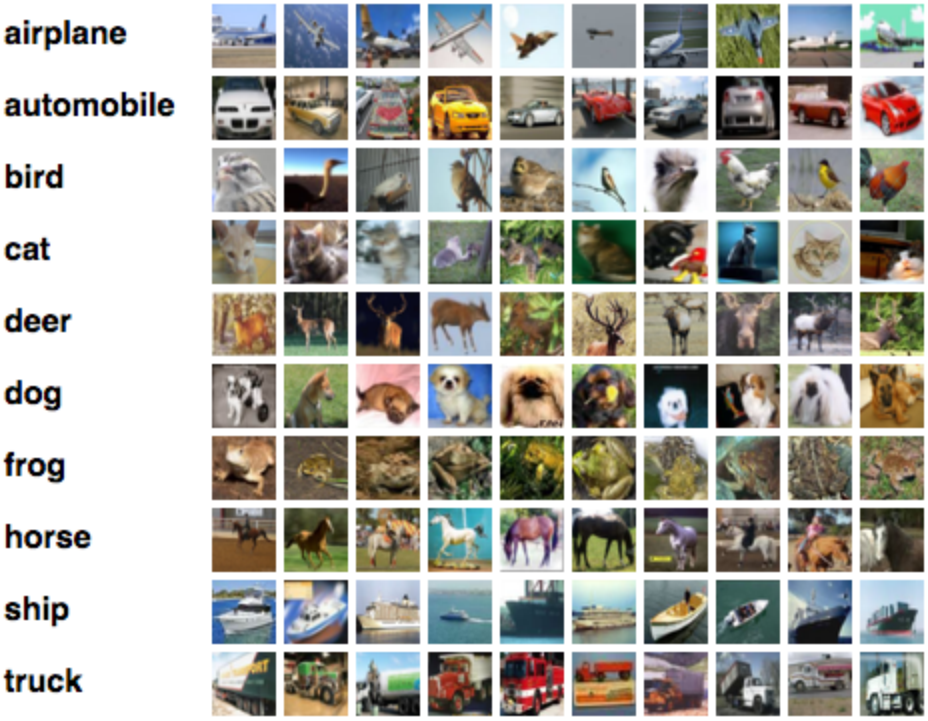
It is available online from the NYU Courant Institute and is a subset of a larger NIST (National Institute of Standards and Technology) dataset of handwritten digits. The original NIST dataset consists of many “special databases,” which are handwritten digits collected from different sources and are organized into groups called “Special Digits.” The MNIST dataset uses digits from the NIST Special Digits 1 and NIST Special Digits 3. The digits from Special Digits 1 are from Census Bureau employees while the digits from Special Digits 3 were collected from high school students. MNIST uses an even mixture of 30,000 digits from Special Digits 1 and 30,000 digits from Special Digits 3 for the training set and an even mixture of 5,000 digits from each for the testing set.

The digit files are images of the Arabic numerals 0 to 9. Each image is 28 by 28 pixels and is normalized so that the numerals fit while also keeping the same aspect ratio. The images have also been centered to fit into the 28 by 28 pixel area. The following is a sampling of images from the MNIST dataset:

Cifar10 and Cifar100 are datasets from the University of Toronto. These datasets are a subset of the 80 million tiny images data set, with the advantage of having everything labeled.

The Cifar10 dataset contains 60,000 images that are sorted into 10 different classes while the Cifar100 contains 60,000 images that are sorted into 100 different classes. We will be using the Cifar10 dataset for our experiments.

The images in the dataset are 32x32 color images. They are categorized into 10 different classes -- airplane, automobile, bird, cat, dog, deer, frog, horse, ship and truck. The classes contain no overlap with each other -- for example if something is labeled as a car it will not be labeled as a truck. Following is a sampling of images from the dataset:



Torch7 Details

As mentioned above, Torch7 is an open-source machine learning library being developed primarily at New York University. It uses the Lua scripting language as its default language of choice, although it also allows snippets of C code to be inserted as well as interfacing with NVIDIA CUDA, when speed is especially important.

The Lua programming language is an unusual choice by the developers of Torch7. According to their website [26], Lua describes itself as:

> Lua is a powerful, fast, lightweight, embeddable scripting language.
>
> Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

It was developed at the Pontifical Catholic University of Rio de Janeiro in Brazil by three computer scientists: Luiz Henrique de Figueiredo, Roberto Ierusalimschy, and Waldemar Celes [26] who were part of the Tecgraf (Computer Graphics Technology Group) at the time. Lua was developed during a period of time when Brazil had enacted many trade barriers, especially in regards to technology. As a result, Lua was created almost from scratch and has many strange quirks. For example, it is customary for Lua array indices to start at 1 instead of the standard 0 used in most other programming languages.

Although it has been used by many large companies, including Adobe, Bombardier, Disney, Electronic Arts, Intel, LucasArts, Microsoft, NASA, Olivetti and Philips [26], its usage in the general programming community remains quite low.

The TIOBE Index is a ranking of programming language popularity that is maintained by TIOBE Software [27]. While it is not an exact measurement by any means, it is a good way to get a rough estimate of a particular programming language's popularity with the community. According to the TIOBE Index for January 2015, Lua is ranked 31st in popularity being used in about 0.649% of all programming applications, even behind old languages such as Ada and Pascal.

Torch7 uses the LuaJIT compiler for most general purposes [28]. LuaJIT is a open source Lua compiler that aims to provide a JIT (just-in-time) compiler for the Lua language. Many other languages like Java also use just-in-time compilation for the compiler. The advantage of just-in-time compilation is that it allows code to be executed more quickly than code than is interpreted.

Torch7 also allows for the use of LuaRocks, which is an open source package management system for Lua [29]. Programs can be bundled together in the form of a package called a "LuaRock." Many of the core Torch7 packages are hosted at LuaRocks and can be installed easily from the command line. The following command

is an example of how LuaRocks can be used to install a package called "somepackage."

```
$ luarocks install somepackage
```

There is also a custom command line interpreter included with the default Torch7 install. This can be accessed through the "th" command from the terminal, once Torch7 is installed and all the PATH settings are configured correctly. This custom command line interpreter is called TREPL, which stands for "torch read eval print loop." TREPL has several advantages over the default Lua one because it has many extra features designed to make working with Torch7 Lua code easier, such as tab-completion and history. This is an example of the "th" command, taken from the Torch7 website [14]:

```
$ th

  ______                __   |  Torch7
 /_  __/__  ________/ /   |  Scientific computing for Lua.
  / / / _ \/ __/ __/ _ \  |
 /_/  \___/_/  \__/_//_/  |  https://github.com/torch
                          |  http://torch.ch

th> torch.Tensor{1,2,3}
 1
 2
 3
[torch.DoubleTensor of dimension 3]

th>
```

Out of the numerous LuaRocks available through the package management system, an especially important one for this project is “dp.” This is a library designed to facilitate the process of using Torch7 for deep learning. “dp” was developed by Nicholas Leonard while he was a graduate student working in the LISA lab under the supervision of Yoshua Bengio and Aaron Courville [30].

It describes itself on its homepage as a “high-level framework that abstracts away common usage patterns of the nn and torch7 package such as loading datasets and early stopping” with “hyperparameter optimization facilities for sampling and running experiments from the command-line or prior hyper-parameter distributions” and “facilites for storing and analysing hyperpameters and results using a PostgreSQL database backend which facilitates distributing experiments over different machines.”

Data Preparation & Feature Engineering

Both the MNIST digits dataset and the Cifar10 dataset do not come in a standard image format. They come in their own special formats designed for storing vectors and multidimensional matrices. Usually when working with these type of datasets, one is required to write a small program to parse the special format. However, the dp Luarocks module (which is designed to eliminate common repetitive tasks) makes this unnecessary because it already includes a small amount of code to facilitate the loading of the data from many common datasets (including MNIST and Cifar10).

The dp Luarocks module also contains a few methods to help preprocess and standardize the data. This is accomplished using the `dp.Standardize` method, which subtracts the mean and then divides by the standard deviation. While the MNIST dataset is already formatted nicely for the most part, we apply it anyway using the common code pattern show below:

```
mydata = dp.Mnist{ input_preprocess = dp.Standardize()}
```

This gets rid of any anomalies and is good practice in general when doing machine learning. The Cifar10 dataset does not work well with `dp.Standardize`, so we leave it out when running tests on Cifar10.

Running the Test

The first step in running the test is creating a set of parameters that will be used for the test. Storing all these parameters in a table in the form of a Lua variable is a good idea because it lets us keep track of things more easily and also allows us to change parameters as we see fit. The code for that would look something like this:

```
myparams = {
      hiddenunits = 100,
      learningrate = 0.1,
      momentum = 0.9,
      maximumnorm = 1,
```

```
batchsize = 128,
maxtries = 100,
maxiterations = 1000
}
```

These are the basic parameters that we need to supply to Torch7 and dp in order to run our test. Here is an quick explanation of what each of these parameters mean:

- `hiddenunits`: This represents the number of nodes in the hidden layer (shown as the blue-colored nodes in a previous diagram).
- `learningrate`: This variable determintes the learning rate for the neural network. A smaller learning rate makes he system learn in finer increments, but can also drastically increase the time required to train the system.
- `momentum`: The momentum adds part of the previous weight to the current weight. This is done to try to prevent the system from settling on a local minimum when training. A high momentum can make the system train faster, but can also end up overshooting the minimum.
- `maximumnorm`: This is used to determine how much to update the neuron weights.
- `batchsize`: This is the batch size for the training example.
- `maxtries`: This determines when to stop the training process early, if after a certain amount of tries, error has not decreased
- `maxiterations`: This determines the maximum amount of times to iterate overall.

We then need to build a model to represent our neural network. We use the `dp.Neural` class which represents a layer in the neural network. We use the parameters that we set above in myparams. A single layer looks something like this:

```
dp.Neural {
      input_size = mydata:featureSize(),
      output_size = myparams.hiddenunits,
      transfer = nn.Tanh()
}
```

We create several of these layers and combine them together using the `dp.Sequential` module to form our neural network. The `transfer` variable used to set the transfer function. Transfer functions are used to allow for more complexity than what a typical logistic regression function would provide.

Next we set up three propagators: `train`, `valid`, and `test`. These determine how the neural network will train the system and determine what is good and what is bad.

The `train` propagator is an instance of the `dp.Optimizer` class and requires a few parameters to be provided:

- `loss`: This represents the typical machine learning loss function, which is a function that the system wants to minimize

- `visitor`: Here we use some of the parameters we set above in myparms, namely `momentum`, `learningrate` and `maximumnorm`
- `feedback`: This determines how feedback is provided after each iteration of training. We use `dp.Confusion`, which is just a confusion matrix.
- `sampler`: This determines which order to iterate through the dataset. For the train propagator we use `dp.ShuffleSampler` which randomly shuffles the dataset before each iteration.

The `valid` and `test` propagators are instances of the `dp.Evaluator` class and also require a few parameters to be provided:

- `loss`: (same as `train` propagator)
- `feedback`: (same as `train` propagator)
- `sampler`: For the `valid` and `test` propagators, we use a different sampler than the `train` propagator. Instead of `dp.ShuffleSampler`, we use `dp.Sampler` which iterates through the dataset in order.

Finally, we set up the experiment and prepare for training. We use the `dp.Experiment` class which takes in the following parameters:

- `model`: This is set using the model that we set up before using `dp.Neural` to form layers and `dp.Sequential` to combine them into a neural network

- `optimizer`: This is set using the `train` propagator that we created above.
- `validator`: This is set using the `valid` propagator that we created above.
- `tester`: This is set using the test propagator that we created above.
- `observer`: Observer is a feature of dp that listens as the model is being trained and then calls specific functions when certain events occur. Our observer uses `dp.EarlyStopper` which ends the training process early if no additional results are being obtained and `dp.FileLogger` which writes the results to a file
- `max_epoch`: This is the maximum number of iterations that the experiment will go through when training. We set this to `myparams.maxiterations`.

The final step is running the experiment on the datasets, which can be accomplished by the following line of code:

```
myexperiment:run(mydata)
```

Now that we have run the experiment, let us look at the results.

Results

MNIST Results

One of the main advantages of the MNIST dataset is that it is widely used and so there are a lot of previous tests run on the MNIST dataset that we can compare our results to.

Following are a few papers with results run on the MNIST Digits data set. I have decided to use a sampling of results run using many different machine learning methods, which will allow us to make a better comparison and see a wider picture on how our results measure up.

Type of Machine Learning Method	**Lowest Error Rate**	**Highest Error Rate**	**Median Error Rate**	**Citations**
Our NN Results	2.7%	3.8%	3.04%	
Linear classifier	7.6%	12.0%	8.4%	[31][31][31]
K-Nearest Neighbors	0.52%	5.0%	1.33%	[32][31][35]
Non-linear classifiers	3.3%	3.6%	3.3%	[31][31][31]
Support vector machines	0.56%	1.4%	0.8%	[33][25][31]
Neural Nets	0.35%	4.7%	2.45%	[34][31][31]

CIFAR10 Results

The Cifar10 dataset has fewer previous tests run on it when compared to the MNIST dataset, but there are still enough to make a comparison. The table below has a sampling of prior results as well as our own results. When reporting results from

Cifar10, it is the accuracy rate that is commonly reported (unlike MNIST where the error rate is reported). Also note that the images to be classified in Cifar10 are much more complex than the ones in MNIST.

Type of Machine Learning Method	**Accuracy Rate**	**Citation**
Our NN Results	50.46%	
Deeply Supervised Nets	91.78%	[37]
Network in Network	91.2%	[38]
Sum Product Networks	83.96%	[39]
Convolutional Kernel Networks	82.18%	[40]
PCANet	78.67%	[41]

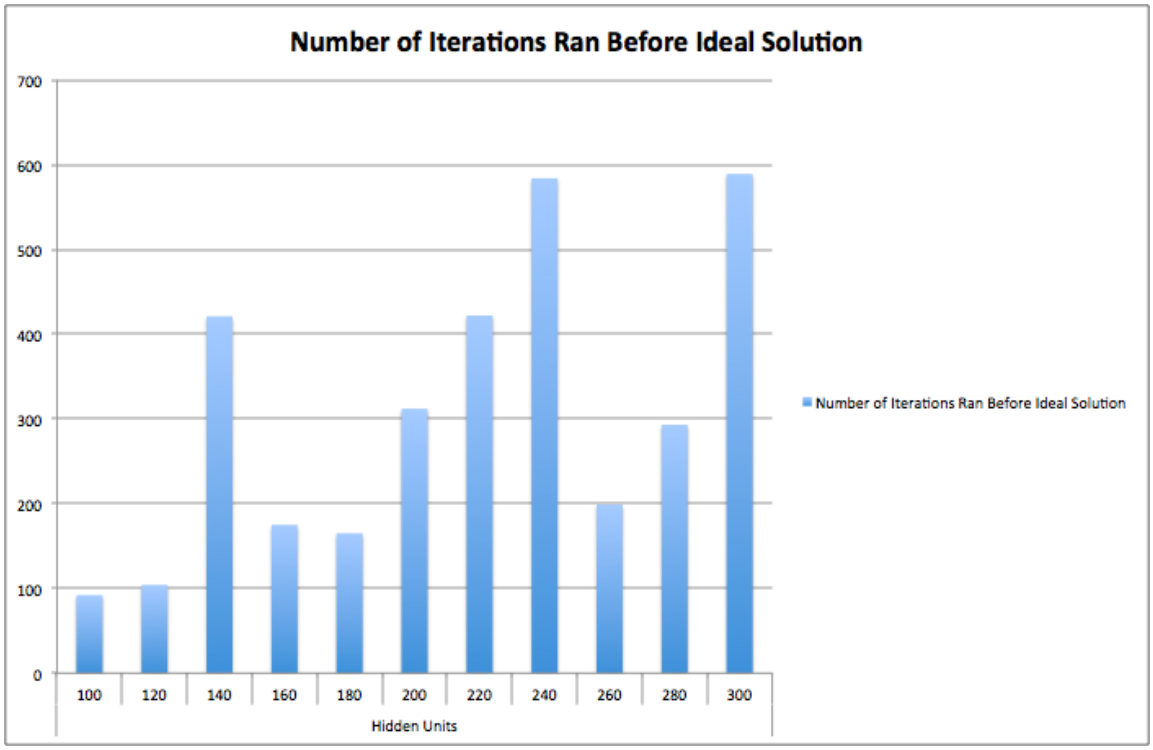
Other Thoughts

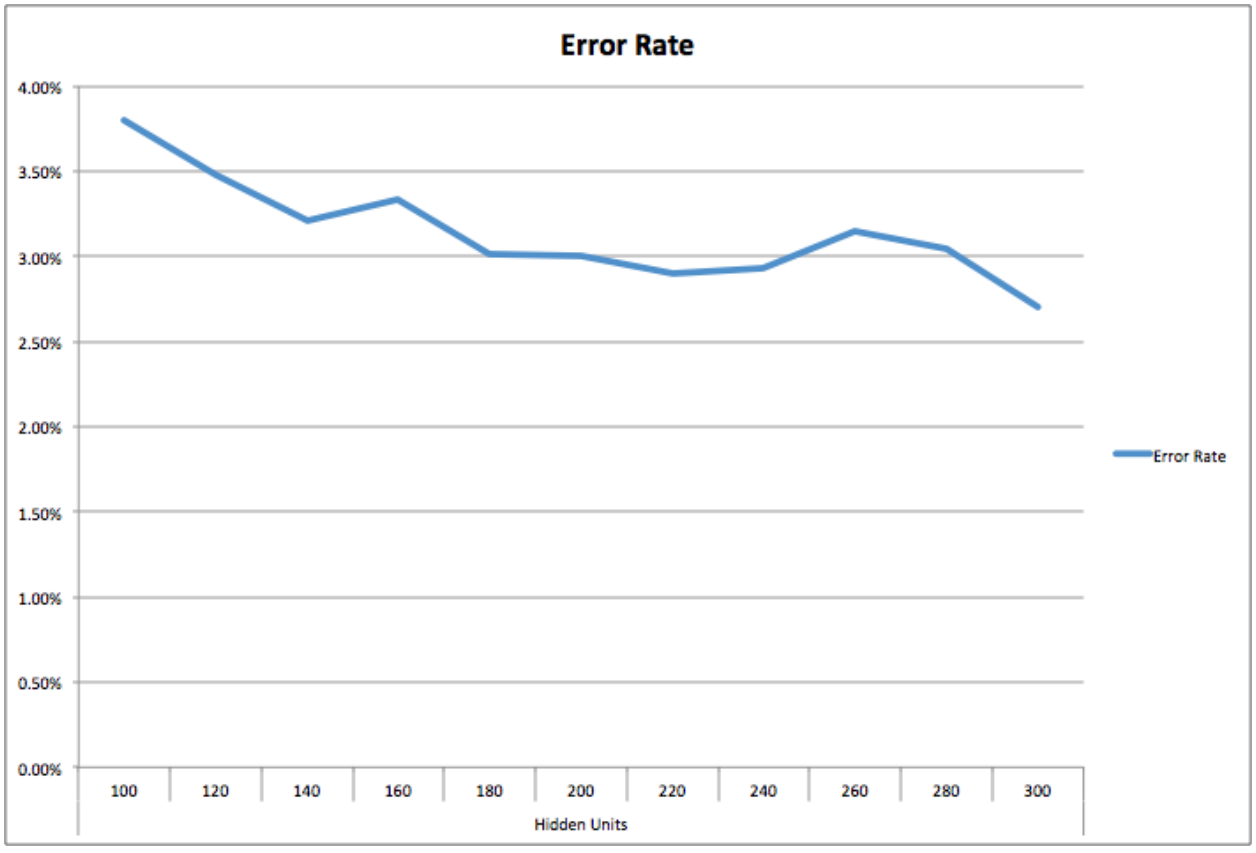
We trained the neural network and ran our test a few times using many different parameters. Our main tests were all set to run for a max of 1000 iterations with a learning rate of 0.1 and to stop early if after 100 iterations, no further progress was made. The models all had one hidden layer with the number of hidden nodes (also sometimes referred to as hidden units) being variable each time the test was run. Following are a few tables and graphs with more detailed information on all the numerous experiments run:

MNIST Table

Dataset	Hidden Units	Number of Iterations Ran Before Ideal Solution	Error Rate
MNIST	100	92	3.80%
MNIST	120	104	3.48%
MNIST	140	421	3.21%
MNIST	160	175	3.33%
MNIST	180	165	3.01%
MNIST	200	312	3.00%
MNIST	220	422	2.90%
MNIST	240	584	2.93%
MNIST	260	199	3.15%
MNIST	280	293	3.04%
MNIST	300	589	2.70%

MNIST Graphs

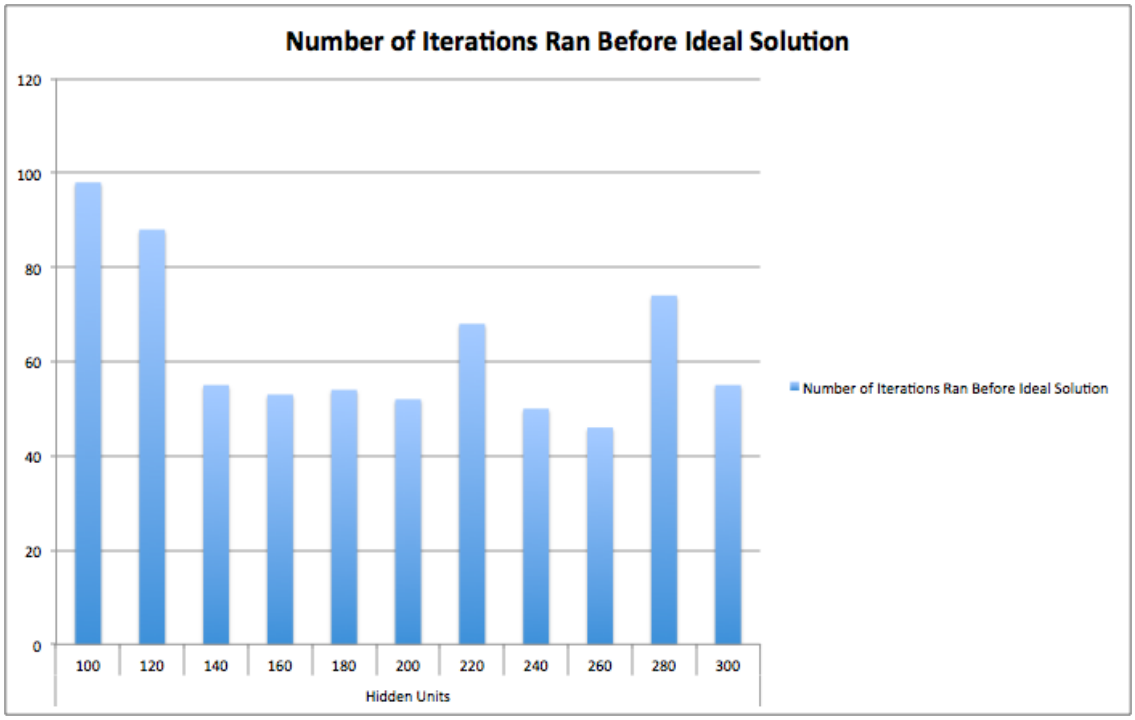


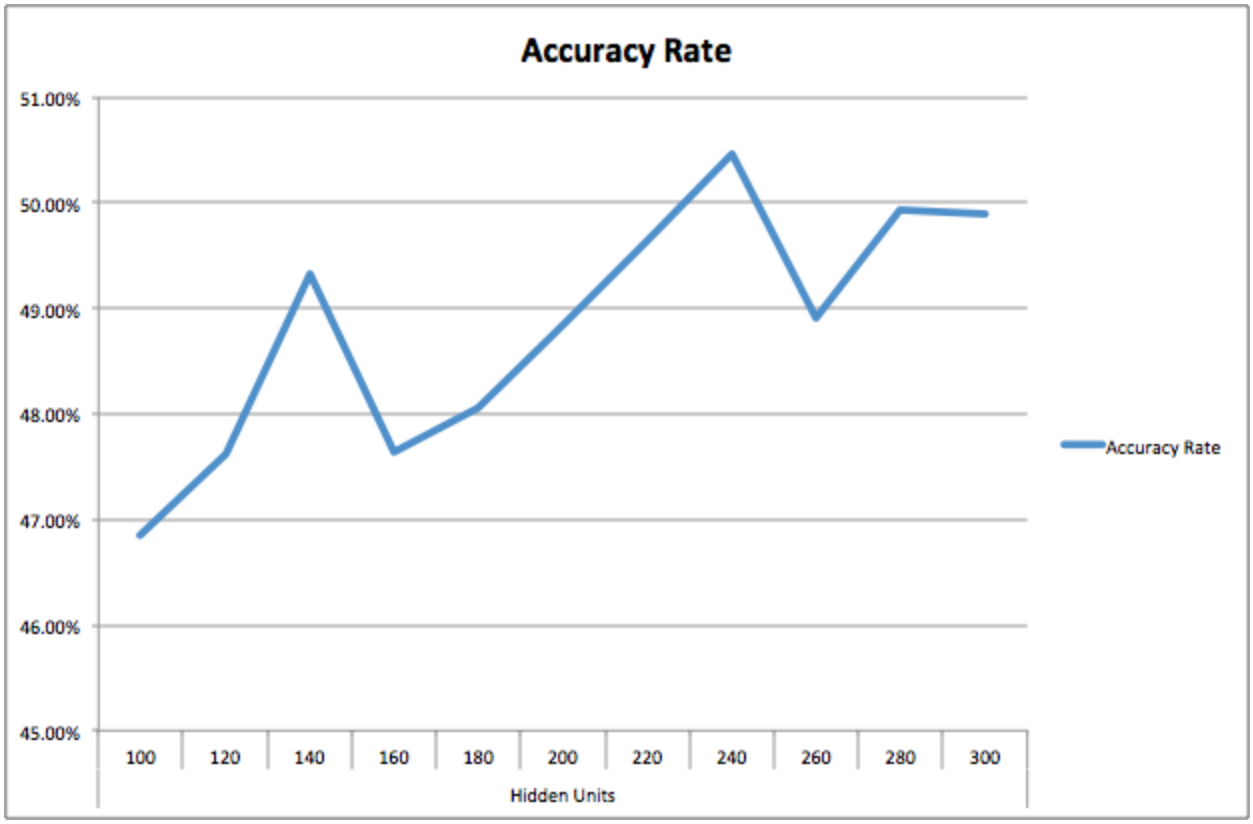


Cifar10 Table

Dataset	Hidden Units	Number of Iterations Ran Before Ideal Solution	Accuracy Rate
Cifar10	100	98	46.85%
Cifar10	120	88	47.62%
Cifar10	140	55	49.32%
Cifar10	160	53	47.63%
Cifar10	180	54	48.06%
Cifar10	200	52	48.84%
Cifar10	220	68	49.65%
Cifar10	240	50	50.46%
Cifar10	260	46	48.91%
Cifar10	280	74	49.93%
Cifar10	300	55	49.90%

Cifar10 Graphs





One disadvantage of neural networks is the long training times. We can use our experiences training MNIST to demonstrate (our experiences with Cifar10 are similar). When we set the number of hidden nodes to 100, each iteration took about six seconds to run and our program went through 193 iterations before deciding to stop early due to lack of progress. The ideal solution was found on the 92nd iteration. We end up getting an error rate of 3.8%. When we set the number of hidden nodes to 300, each iteration took about ten seconds to run and our program went through 589 iterations before deciding to stop early due to lack of progress. The ideal solution was found on the 488th iteration. We end up getting an error rate of 2.7%. We should also note that increasing the number of hidden nodes from 100 to 300 dramatically increased the time it took to train the neural network from around 30 minutes to over 3 hours.

Interestingly enough, the number of iterations required before reaching the ideal solution seems to have no correlation with the number of hidden units. This is because the algorithm chooses a random spot to start walking towards the ideal solution and may sometimes land in a more favorable spot initially. However, in general, more hidden units result in more favorable results, especially for MNIST.

The Cifar10 dataset resulted in much worse results than the MNIST. This can be explained by the much more complex images contained in the dataset. In order to improve results, it may be necessary to increase the number of hidden units, increase

the number of layers in the model, or use a more aggressive learning rate. Unfortunately this is unfeasible with the limited hardware that we have access to.

Overall, these results are somewhat typical of neural networks, which seem to have a large variation in error rate percentage. On the MNIST dataset other results range from 4.7% in a test run by Yann LeCun [31] to 0.35% in a test run by Dan Ciresan [34]. The current best result for Cifar10 [37] also uses a variation on neural networks, showing that neural networks are indeed a good candidate for classification of more complex imagery as well.

Nevertheless, these are pretty favorable results (especially when considering the limited hardware that the test was run on) and aptly demonstrate the potential that neural networks have when solving these sorts of problems.

Conclusion

In conclusion, neural networks are shown to be a viable choice when doing image recognition, especially of handwritten digit images. This is because neural networks are especially useful for solving problems with non-linear solutions, which applies in the case of handwritten digit images, since the hidden units are able to effectively model such problems. We must also note that this is with the caveat of having the necessary

computational hardware and time. Without such resources, results can be subpar, as shown by the Cifar10 tests.

While there are many advantages to using neural networks, there are also a few drawbacks. One drawback to neural networks is in training. Since the system has to go through many iterations during the training phase, this may cause training to take a long time, especially when run on computers using older hardware.

Another drawback to neural networks (although this applies to all machine learning in general) is picking the right parameters (such as number of hidden nodes per layer, learning rate, and number of layers). The right parameters can cause a huge difference in results with a massive decrease in error rate percentage. However, it is difficult to balance and the wrong choices can cause extremely long training times or inaccurate results.

Future work may include running tests on models that have more hidden units and layers as well as using a more aggressive learning rate. To accommodate the large hardware demands that are required to do such work, we may look into running our computations in the cloud. Amazon has the EC2 cloud which may be able to offload a lot of the work. Another possible way to drastically improve hardware performance may be to use the graphics card to help with computations. NVIDIA has the CUDA library

which is excellent at running computations in parallel, which Torch7 actually has some built in support for.

With the rise of tools such as torch7 (and dp) neural networks are now more useful than ever before and will probably be applied to many other problems in the future. This can range from item recommendation at a shopping service like Amazon or even to things like self-driving cars. We live in an era ruled by data and I am excited to see what will come next.

References

[1] Warren S. McCulloch and Walter Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Published 1943. Accessed Oct 2014.

[2] Paul J. Werbos. *Backpropagation Through Time: What It Does and How to Do It.* Published 1990. Accessed Oct 2014.

[3] Adam Coates, Brody Huval, Tao Wang, David J. Wu, Bryan Catanzaro and Andrew Y. Ng. *Deep Learning with COTS HPC Systems.* Published Jul 2013. Accessed Oct 2014.

[4] Danqi Chen, Richard Socher, Christopher D. Manning and Andrew Y. Ng. *Learning New Facts From Knowledge Bases with Neural Tensor Networks and Semantic Word Vectors*. Published Mar 2013. Accessed Oct 2014.

[5] Richard Socher, Brody Huval, Bharath Bhat, Christopher D. Manning and Andrew Y. Ng. *Convolutional-Recursive Deep Learning for 3D Object Classification*. Published 2012. Accessed Oct 2014.

[6] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. *A Fast Learning Algorithm for Deep Belief Nets*. Published 2006. Accessed Oct 2014.

[7] Geoffrey E. Hinton and R.R. Salakhutdinov. *Reducing the Dimensionality of Data with Neural Networks*. Published 2006. Accessed Oct 2014.

[8] Geoffrey E. Hinton. *To Recognize Shapes, First Learn to Generate Images*. Published Oct 2006. Accessed Oct 2014.

[9] Geoffrey E. Hinton. *Learning Multiple Layers of Representation*. Published Oct 2007. Accessed Oct 2014.

[10] Yann LeCun, Fu-Jie Huang, and Leon Bottou. *Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting*. Published CVPR, 2004. Accessed Oct 2014.

[11] Yoshua Bengio and Yann LeCun. *Scaling Learning Algorithms Towards AI*. Published MIT Press, 2007. Accessed Oct 2014.

[12] Piotr W. Mirowski, Yann LeCun, Deepak Madhavan, and Ruben Kuzniecky. *Comparing SVM and Convolutional Networks for Epileptic Seizure Prediction from Intracranial EEG*. Published 2008. Accessed Oct 2014.

[13] Antonio Regalado. *Is Google Cornering the Market on Deep Learning?*. Published MIT Technology Review, Jan 2014. Accessed Oct 2014.

[14] “Torch7.” Accessed Oct 2014. <http://torch.ch/>

[15] “Theano.” Accessed Oct 2014. <http://deeplearning.net/software/theano/>

[16] “Caffe.” Accessed Oct 2014. <http://caffe.berkeleyvision.org/>

[17] “Caltech 101.” Accessed Oct 2014.

<http://www.vision.caltech.edu/Image_Datasets/Caltech101/>

[18] “PASCAL VOC.” Accessed Oct 2014.

<http://pascallin.ecs.soton.ac.uk/challenges/VOC/>

[19] “Oxford Flowers.” Accessed Oct 2014.

<http://www.robots.ox.ac.uk/~vgg/data/flowers/>

[20] “Stanford Dogs.” Accessed Oct 2014.

<http://vision.stanford.edu/aditya86/ImageNetDogs/>

[21] “Animals with Attributes.” Accessed Oct 2014.

<http://attributes.kyb.tuebingen.mpg.de/>

[22] Thomas Serre, Lior Wolf, and Tomaso Poggio. *Object Recognition with Features Inspired by Visual Cortex*. Published 2005. Accessed Nov 2014.

[23] Kristen Grauman and Trevor Darrell. *The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features*. Published ICCV, 2005. Accessed Nov 2014.

[24] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. *Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification*. Published CVPR, 2009. Accessed Nov 2014.

[25] “MNIST Handwritten digit database.” Accessed Dec 2014.

<http://yann.lecun.com/exdb/mnist/>

[26] Robert Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes. *The Evolution of Lua*. Accessed Jan 2015. <http://www.lua.org/doc/hopl.pdf>

[27] “TIOBE Software: TIOBE Index.” Accessed Jan 2015. <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

[28] “LuaJIT.” Accessed Jan 2015. <http://luajit.org/luajit.html>

[29] “LuaRocks.” Accessed Jan 2015. <http://luarocks.org/>

[30] “dp.” Accessed Jan 2015. <http://dp.readthedocs.org/en/latest/index.html>

[31] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. *Gradient-Based Learning Applied to Document Recognition*. Published Nov 1998. Accessed Feb 2015.

[32] Daniel Keysers, Thomas Deselaers, Christian Gollan, and Hermann Nay. *Deformation Models for Image Recognition*. Published 2007. Accessed Feb 2015.

[33] Dennis Decoste, Bernhard Scholkopf. *Training Invariant Support Vector Machines*. Published 2002. Accessed Feb 2015.

[34] Dan Ciresan, Ueli Meier, Luca Gambardella, and Juergen Schmidhuber. *Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition*. Published Mar 2010. Accessed Feb 2015.

[35] “MNIST Nearest Neighbor Results.” Accessed Feb 2015. <http://finmath.uchicago.edu/~wilder/Mnist/>

[36] “CIFAR-10 and CIFAR-100 Datasets.” Accessed April 2015. <http://www.cs.toronto.edu/~kriz/cifar.html>

[37] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, Zhuowen Tu. *Deeply-Supervised Nets.* Published 2014. Accessed April 2015.

[38] Min Lin, Qiang Chen, Shuicheng Yan. *Network In Network.* Published 2013. Accessed April 2015.

[39] Robert Gens, Pedro Domingos. *Discriminative Learning of Sum-Product Networks.* Published 2012. Accessed April 2015.

[40] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, Cordelia Schmid. *Convolutional Kernel Networks.* Published 2014. Accessed April 2015.

[41] Tsung-Han Chan, Kui Jia, Shenghua Gao, Jiwen Lu, Zinan Zeng, Yi Ma. *PCANet: A Simple Deep Learning Baseline for Image Classification.* Published 2014. Accessed April 2015.

[42] “The Last AI Breakthrough DeepMind Made Before Google Bought It For $400m.” Accessed April 2015. <https://medium.com/the-physics-arxiv-blog/the-last-ai-breakthrough-deepmind-made-before-google-bought-it-for-400m-7952031ee5e1>