Cmput 399 Project Report

Touqir Sajed, Tristan Meleshko, Zhaorui Chen
University of Alberta
{touqir, tmeleshk, zhaorui} @ualberta.ca

***Abstract* – Along the way to greater machine learning and data analysis accuracy, we are faced with great barriers. A limitation of most of the classical machine learning techniques is that increasing the data samples doesn't necessarily contribute to higher prediction accuracy. A second limitation of these methods is that they require a great deal of feature extraction and preprocessing to achieve significant results, and thus require a deeper domain knowledge in order to achieve better accuracy. The major advantage of neural networks is that they can learn feature extraction on their own and if they are deep enough they usually perform better with large datasets. In this paper, we examine the performance of Multilayer Perceptrons (MLP) and Convolutional Neural Networks (CNN) on the MNIST data. We discuss the advantages and disadvantages of each architecture and technique, as well as discuss some reasons for their relative performance. Specifically, we examine the differences of including convolutional layers in deep networks and the impact dropout has on final accuracy. As well, we test different architectures on three different machine learning toolboxes: MATLAB's DeepLearnToolbox, Python's Theano/Lasagne library, and Lua's Torch 7 library.**

1. Introduction

With the rise of data in todays digital era comes the high demand of data analysis with greater accuracies in prediction tasks. Unfortunately after a certain point, increasing the data samples doesn't necessarily contribute to higher prediction accuracy using classical machine learning techniques. Secondly, for many of these methods, feature extraction becomes necessary for those methods, requiring a deep domain knowledge and highly specialized architectures. Neural networks ofter an elegant solution to both of these problems, but for a long time, taking full advantage of its general function approximation property has been hard, leading to poorer accuracies than the state of the art methods of the times. However, their capability of learning feature extractions has been one of the key motivators for research over the years. With the advent of the first successful deep learning techniques on 2006 by Hinton et al.[5], training neural networks of more than 2 hidden layers has become much easier and has boosted its function approximation capacity. Since then, they have become the state-of-the-art algorithm for many Machine Learning tasks.

The main problem with training neural networks with more than 2 hidden layers was that the error function landscape would be plagued with local minimas, resulting in poor accuracies. However, shallow networks are limited in that the space of functions that can be approximated is rather quite limited. Modern learning techniques, called "deep learning," don't necessarily decrease the number of local minimas, but instead make the network more resilient to them, allowing it to converge to more optimal solution with lower error.

A solid explanation with good mathematical reasoning hasn't been established yet. However, experiments have shown that the architectures discover hierarchical compositional relationships inside the data which resembles how humans learn and specially how the human visual recognition works.

One the key techniques of deep learning is that training of each layer is first done separately. In the case of unsupervised training, each hidden layer is trained in an unsupervised fashion, such as by using auto-encoders, and then the whole architecture is further trained as a single unit. The unsupervised learning part is often called the pre-training stage and all the other layers' weights are kept fixed during this phase. In the case of greedy learning, the weights of a particular layer is trained in a supervised fashion with the output layer's target set to the data labels. After a layer is trained, another layer is added between it and the output layer and the process continues. Again, the weights are updated one layer at a time. Then the architecture is trained as a single unit just like before. This final stage is called fine tuning. For an introduction to deep learning techniques, read the report by Touqir Sajed[6]. For a more in-depth introduction - [7]. The first convolutional neural network was developed by Yann LeCun in the late 80s, named LeNet. LeNet was the state-of-the-art algorithm at handwritten digit recognition of its time and has been used in US postal code service for

handwritten zip code recognition. For a thorough discussion, read [3]. For image classification benchmarks, the MNIST digit database has been used since the late 90s as the testing data [4]. The database contains 60,000 training images and 10,000 testing images. So far, the state-of-the-art result for MNIST digit classification has been achieved by Li Wan et al [8] which is 0.21 error rate using deep CNN and DropConnect as the regularization method.

2. Choice of method

From the late 2000s until today, deep convolutional neural networks has been the state-of-the-art method for visual recognition tasks, in some cases achieving human level performance. This mainly motivated us to try out deep CNNs and get a feel and experience on the whole process of training, testing and evaluation with different parameter settings. We also wanted to see how its performance compares to the more conventional multilayer perceptron. The other major motivation behind choosing these two methods is that they automatically learn the features and this allows us to circumvent the tedious feature extraction phase. It has been found that usually a dropout of 0.5 works best. We wanted to find out how changing the dropout parameter would affect the performance.

In the case of image datasets, the idea is that a deep convolutional neural network first learns features at the lowest levels, such as edges and corners. In the next layer, it learns to compose these edges to form shapes. Then, it combines these shapes to form larger feature grouping, and so on and so forth, until it can effectively classify objects. This intuitively seems effective as images naturally have this kind of hierarchical composition which can naturally be exploited by the network, and resembles many classical image processing techniques.

$$f\left(\sum_{i=1}^{n} x_i w_{i,j} + \theta_j\right)$$

The commonly known multi-layer perceptron consists of an array of inputs, an intermediate or hidden layer of neurons with non-linear activation functions f(·), and an output layer of neurons with linear or non-linear activation functions [19]. This architecture allows us to create non-linear classification boundaries. The output of a neuron is the result of an activation function applied to the linear combination of the weights of the connection between neurons and the input:where x_i denotes the i'th input value, ω_{ij} denotes the synaptic strength between the ith neuron and the jth neuron, θ_j is a bias and n is the number of neurons of the previous layer that are connected with the jth neuron.

Convolution is a mathematical term, defined as applying a function repeatedly across the output of another function. In the context of CNNs it means to apply a 'filter' over an image at all possible offsets. A filter consists of a layer of connection weights, with the input being the size of a small 2D image patch, and the output being a single unit. Since this filter is applying repeatedly, the resulting connectivity looks like a series of overlapping receptive fields, which map to a matrix of the filter outputs (or several such matrices in the common case of using a bank of several filters). Thus CNNs exploit local spatial structure and relationship that is prevalent in images which explains why it outperforms classical MLPs(which isn't designed to exploit like CNNs) in image recognition tasks.

A convolutional layer has a lot of parameters to learn and subsequent convolution can mean the parameter number explodes. To prevent this, pooling is used in the form of either taking maximum or averaging over multiple values from the convolutional layer. Thus pooling subdues overfitting to some extent.

3. Model Description

MLP Model

As demonstrated by the famous XOR example in Minsky and Papert's book, *Perceptrons* (1969), perceptrons alone can only approximate linearly separable functions and thus are not universal. To rectify this issue, more layers can be added, each separated by a nonlinear activation function such as a sigmoid. Without these nonlinear activation functions, the system becomes equivalent to a series of matrix multiplications of the weights and, due to associativity, collapses into a single matrix which would be equivalent to a single layered perceptron.

To tackle this problem, we used MLPs. The first classifier is a two hidden-layer MLP network. Each layer contained 800 hidden units separating them. The final output was selected using a SoftMax layer to clamp the output classes to their approximate probabilities. The architecture is as follows:

· Input Layer (28x28 nodes) with dropout=0.2

· Hidden layer: size=800 units with dropout of 0.5 probability, activation function is ReLU

· Hidden layer: size=800 units with dropout of 0.5 probability, activation function is ReLU

· Output: size=10

· Softmax Loss Function

Pseudo-code for this model

```
Read in the training images

Split the images into a training set and a
validation set

INITIALIZE the MLP model based on the
architecture

DEFINE the size of each training batch

DEFINE the number of epochs

FOR each epoch of iteration:

   FOR each batch in training set:

      Feed the batch forward through MLP to
      obtain classification scores

      Compute the mean square error for all
      classifications in the batch.

      Back-propagate the error through the MLP

      Update parameters of the layers based on
      backpropagation update rule


FOR each batch in testing set:

    Feed the batch forward through network to
    obtain classification scores.

    During this stage, according to dropout
    rule, the output of the neurons with
    dropout is the expected output based on
    the dropout probability. So, if output is
    1 and dropout was 0.5, the expected output
    is 0.5*1=0.5.

    Compute the mean square error for all
    classifications in the batch.
```

Result of the model:

Using this architecture, we achieved an accuracy of 98.82% after 300 epochs of training. The result showed that about 1.2% of testing images were misclassified.

Discussion on the model:

This model resembles some the models develoepd by Simard et al. in 2003, which can be viewed on the MNIST website. We achieve a misclassification of only 1.18% using mean square error, roughly in the middle of what they achieved but with much less of the work.

The choice of the ReLU function makes sure that the output of any neuron is always positive. So, this acts as a kind of switch in the sense that it can only be "on" or "off" and this models the way human brains work more closely as opposed to the classical sigmoid function. In practice ReLU has been found to perform much better than sigmoid or other activation functions.

For a mutually exclusive multi-class classification problem like digit classification, the final layer is usually applied to a softmax so that we get a concrete value of the probability of each class rather than some real number output which necessarily doesn't give us probabilities if we instead had used sigmoid or any other function as output.

To improve results and reduce overfitting, an input dropout is employed. Dropping neurons randomly makes it so they are unable to expect which other neurons will fire simultaneously, and so discourages them from offsetting each others' output. This encourages specialization of each node. The choice of a sigmoid (in our case SoftMax) allows multiple specialized neurons to fire simultaneously safely, since excessively high output values are clamped to a higher output. When the network is not training, all neurons fire their expected output(due to dropout probability), forcing all specialized nodes to fire when they are expected to.

Tweaking the model:

In order to see some of the effects different parameters had, we added one more hidden layer with 800 units and a dropout of 0.5. The resulting accuracy was 98.72%. Due to the large number of paramaters, it is likely that this is a case of overfitting. We can prevent this overfitting by increasing the number of data points to train on or decreasing the number of weights to train on.

We then used our original 2 hidden layered MLP but changed dropout from 0.5 to 0.65 on each hidden layer. The resulting accuracy was 98.45%. It seemed that 50% dropout gives a better accuracy. This is probably a case of simply losing too much data between layers and not giving the model a chance to learn what was needed.

CNN Models

A Simple CNN Model

We used the MATLAB library, DeepLearnToolbox, to test a simple convolutional neural network. The architecture is as follows:

- Input (28x28 image)
- Convolution: layer_size=3, filter_size=5x5
- Mean pooling: factor of 2
- Convolution: layer_size=6, filter_size=5x5
- Mean pooling: factor of 2
- FC layer: layer_size=150
- Output: size=10

Pseudo-code for this model

```
Read in the training images
INITIALIZE CNN model based on the architecture
DEFINE the size of each training batch
DEFINE the learning rate
DEFINE the number of epochs
FOR each epoch of iteration:
   FOR each batch in training set:
      Feed the batch forward through the cnn
      network to obtain classification scores
      Compute the mean square error for all
      classifications in the batch.
      Back-propagate the error through the
      network
      Update parameters of the layers based on
      the weight update rule of backpropagation

FOR each batch in testing set:
    Feed the batch forward through network to
    obtain classification scores.
    Compute the mean square error for all
    classifications in the batch.
Find the number of mis-classified images
```

Result of the model:

Using this architecture, we achieved an accuracy of 97.11%. The result showed that about 3% of testing images were misclassified.

Tweaking the model:

We also tried increasing the size of the first convolution layer from 3 to 6, but the accuracy did not improve significantly. Unfortunately, the DeepLearnToolbox has some performance issues, as well as some missing functionality such as a lack of a built-in ReLU function. Instead, the test was repeated using Lua's Torch library to take advantage the accelerated graphics processing in CUDA that is available. Because of this, we were able to increase the number of epochs from 10 to 100, and the accuracy peaked closer to 98.53%, or a 1.47% misclassification rate.

Discussion on the model:

This model performed surprisingly well. Even with such a tiny network, the performance was comparable to the best MLP we tested.

In order to improve this, it may help to include a ReLU activation function in future trials. Furthermore, the network was very tiny and shows the power that convolutional neural networks have over simple MLPs.

A Better CNN Model

With a larger model, it is reasonable to expect a better result. A different and slightly more complicated architecture was developed in the Python library, Lasagne. The architecture is as follows:

- Input (28x28 image)
- Convolution: layer_size=32, filter_size=5*5
- Max pooling: factor of 2
- Convolution: layer_size=32, filter_size=5*5
- Max pooling: factor of 2
- FC layer: layer_size=256, with dropout of 0.5
- Output: size=10, with dropout of 0.5.

Pseudo-code for this model:

The training and testing phase is the same as the one from MLP model.

Result of the model:

The best result we achieved using this model, was achieved with an accuracy of 99.54%.

The result showed that only less than 0.5% of testing images were misclassified. The architecture we used this time achieved a much better result than the previous one.

Tweaking the model:

When tweaking this model, by setting the dropout of the hidden-layer from 0.5 to 0.4, we obtained the accuracy of 99.51%. It is not clear whether setting the dropout to 0.4 produced any noticeable diminished performance of the architecture. With a difference of only 0.03%, this could be the result of random variation due to different weight initialization or stochastic gradient descent, or could be the result of the changed parameter. Regardless, it opens up the range of potentially viable dropout ranges.

Discussion on the model:

We used ReLU as the activation function to threshold at zero. Training with ReLU is faster than with other activation functions mainly because the output of each neuron can go up to positive infinity which counters the vanishing gradient or exploding gradient problem caused by many sigmoids.

The dropout method on the fully-connect layer and output layer appears to have prevented our model from overfitting. To improve these results further, we might need to try pre-processing the input by whitening the data with PCA or other preprocessing and normalization techniques..

Another Good CNN Model

We have also tried a different model on Torch. Our architecture consists of 2 convolution layers, followed by a two, fully-connected hidden layers. Specifically, we trained a CNN with the following architecture:

· Input (28x28 image)

· Convolution (32 filters, 5x5)

· ReLU

· Max Pooling (factor of 2)

· Convolution (128 filters, 5x5)

· ReLU

· Max Pooling (factor of 2)

· Fully Connected Layer (256 units, 50% dropout)

· Tanh Layer

· Fully Connected Layer (256 units, 50% dropout)

· Tanh Layer

· Output Layer (SoftMax loss, 10 units, 0.5 dropout)

Pseudo-code for this model:

The training and testing phase is the same as the one from MLP model.

Result of the model:

We trained and tested the model using the same techniques as the MLP models. After 100 epochs, the final accuracy was 99.43%±0.02% with 95% confidence over the last 10 epochs. The result showed that only a bit more than 0.5% of testing images were misclassified.

4. Summary of Results

In the training phase of the MLP model, we used 500 as the batch size, and iterate over 300 epochs.

After testing this architecture with 10000 test images, we achieved an accuracy of 98.82%.

In the training phase of the first CNN model, we used learning rate of 0.01, with batches of 10 to speed up training. We iterated the training loop for 10 epochs.

After testing this architecture with 10000 test images, we achieved an accuracy of 97.11%.

In the training phase of the second CNN model, we used 500 as the batch size, and iterate over 300 epochs.

After testing this architecture with 10000 test images, we achieved an accuracy of 99.54%.

Using this model we tried to examine how changing dropout can affect a CNN's performance. First we changed the dropout of the hidden layer to 0.4 and got accuracy of 99.51%. Then we retrained the model with dropout of 0.3 and got 99.54% and finally for the last time we retrained that model with dropout of 0.65 and got accuracy of 99.47%. This seems to have reduced the performance.

Since it has a much larger architecture, the second CNN model runs slower in training and testing than the first CNN model.

5. Discussion

One of our goals of this project was to examine how changing dropout probability values impact the accuracy of the classifier. In the case of MLPs an increase of dropout from 0.5 to 0.65 resulted in a decrease in accuracy of 0.37% which is a significant change. This was expected as we are actually increasing the number of nodes firing which ultimately results in an increase in overfitting and decreases the extent of specialization of neurons. We

noticed the same with our CNN models too. But in our 2nd CNN model, when we decreased the dropout to 0.4 from 0.5 only a change of 0.03% was noticed in accuracy. We suspect that this might not be significant and may have arisen from noise from online stochastic gradient descent training and random initialization. To test if that is true, we further decreased the dropout to 0.3, but surprisingly, we got the same accuracy as in the case of dropout rate of 0.5. The reason for this may be that with less neurons firing, there were enough neurons firing at the same time to specialize and learn the underlying required functions necessary for classification. Since we used 2 consecutive convolution-pooling pairs combined with a 250 node hidden layer, there were a large number of weights to train and so lead to great deal of function approximation capability.

As for the CNN vs MLP performance difference, we highlighted previously the main strength of CNN is that it is able to take advantage of local spatial relationship information. With the same training and testing methods, CNNs perform better and offer more flexibility with their architecture overall. The main explanation for this is that MLPs ignore the spatial locality of the data and has to relearn what the convolutional variant already excels at. CNNs are thus great at solving spatially sensitive problems such as image classification.

6. Conclusion

We have shown that convolutional neural networks perform better and with less training effort than multilayer perceptrons. With the same training and testing methods, CNNs perform better and offer more flexibility with their architecture overall.

Further, we discussed the equivalence of CNNs to MLPs and explained why an MLP is in fact a special case of a CNN rather than an extension. Simply put, a linear layer in an MLP is equivalent to a convolution with a maximized kernel. Because of this, CNNs could potentially generalize to non-spatial problems, but MLPs will likely be more effective in that case since they aren't so sensitive to nearby interacting terms.

Increasing the number of layers in an MLP increases the amount of training that is required. While it might achieve a better result eventually, with a restricted training time, it's hard to say whether the comparison is valid. However, our tests indicated that either the training was incomplete, or adding layers did not significantly improve the optimization that could be achieved.

We experimented with different dropout rates and examined what range produced the smallest validation error by the final epoch. Decreasing it below this range eliminated too many inputs and the model couldn't effectively train; increasing it above this range included too many inputs and the model likely became dependent on other neighboring nodes. We achieved the highest accuracy on all tests when the dropout was set to 0.5.

The ease of setting up the architecture depends on the toolbox used. We chose to setup three different toolboxes for our experiments: DeepLearnToolbox, Torch, and Theano/Lasagne. These each have their own strengths and weaknesses, and ultimately produced roughly the same results when validated against each other. Thus, the authors suggest one uses the toolbox they are most comfortable with.

Given the amount of data we used and the architecture, training wasn't difficult specially with the CNNs and thus we didnt have to use deep learning techniques such as greedy supervised training or unsupervised layerwise training. As for the MLP, we noticed overfitting when trained with 3 hidden layers. As we did not have enough time in hand, we didnt try to use deep learning techniques and see if we get better results with it. We do expect one may get better results if they were to do this.

References

[1] Rosenblatt, Frank. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, 1961

[2] Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation, 1986

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998

[4] Yann LeCun, Corinna Cortes. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/

[5] Geoffrey Hinton, Ruslan Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks, 2006

[6] Touqir Sajed. Unsupervised learning and Deep learning,2014.https://drive.google.com/file/d/0B1KKO4MV EtqDSTFTMTkzRk5HY1k/view

[7] Yoshua Bengio. Learning Deep Architectures for AI, 2009. http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf

[8] Wan Li, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus. Regularization of Neural Network using DropConnect, 2013

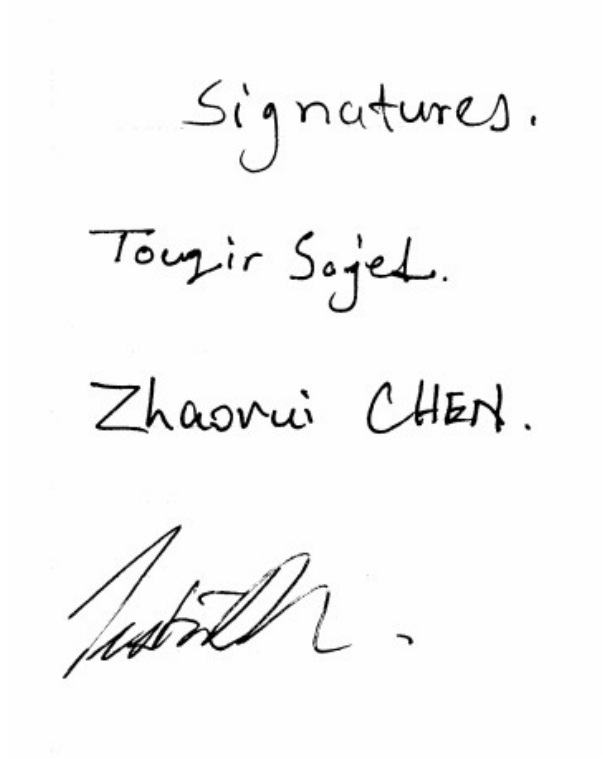
[9] Matthew D. Zeiler, Rob Fergus. Visualizing and Understanding Convolution Networks, 2013

Signature and Contribution

Contribution:

Touqir: Worked on the MLPs and the second CNN model(2 consecutive conv-pool layers). Contributed to almost all of the parts of the paper. Time spent: max 30 hours

Tristan: Worked on the 3rd CNN. Contributed to almost all of the parts of the paper. Time spent: max 30 hours



Zhaorui: Worked on the 1st CNN. Mainly responsible for pseudocode layout and also on methods of choice. Time spent: max 30 hours