Project Report

Alexander Golis (12/13/2016)

Introduction:

In this project, I worked on the MNIST dataset of handwritten digits and applied various machine learning algorithms to classify the images to one of the digits 0-9. This is a very well documented problem and many works [1] have been done on this topic. For me, the most interesting part of it was the fact that I could apply the same statistical techniques that we studied in the class on a completely different problem. Especially, applying my own implementation of MaxEnt model from PA3. During the course, we mostly spoke about natural language processing and concentrated on textual datasets in our assignments. In this project, my dataset includes images, rather than text, but still, the same concept of feature vectors could be applied here too.

Project overview:

It has been studied in various researches that image recognition problems had much success using deep learning techniques [2]. One of the objectives for this project was to see how an additional hidden layer with nonlinear activation function improved the accuracy of the classification. For that reason, I started testing with a simple MaxEnt classifier, then a simple neural network with only one hidden layer, and finally multiple hidden layers. Moreover, since convolutional neural networks are known to better generalize in image recognition tasks, because of the aspects like scaling and position/alignment of an object inside the image, I also experimented with convolutional neural network. Currently, the best performance of a single convolutional neural network trained in 74 epochs is 0.23% error rate [3].

Toolkit and environment:

I planned to work with google Tenserflow deep learning toolkit first, but found that it is not yet supported on Windows OS, which I use. After a research, I decided to use **Theano**, which is a numerical computation library for Python. I also used **Lasagne**, which is a high-level neural network library that rans on top of Theano. One of the benefits of Theano is that it allows efficient computations using GPU. Unfortunately, my machine doesn't have GPU so I used g++ compiler compatible with Theano to make computations be much faster than a regular Python code. The python code is translated in the background to a C++ efficient code and compiled with g++ compiler.

Dataset:

I used MNIST dataset of handwritten digits [1]. There are 60,000 sample images. I used 50,000 for training and 10,000 for dev set. Additional 10,000 images are used for the test purpose. This dataset is well normalized; every digit appears in the center of the image, the size of the digits is scaled to same proportions, all digits are vertically aligned. Each image is in white-gray colors and the size is 28 X 28 pixels.

Experiments:

- **MaxEnt model**

  Since my implementation of MaxEnt model was general enough, I could use it to make image classifications. I represented the image as a 1-dimensional array of 784 (28 times 28) numbers. The features in this case were binary indicators whether a color is set or no at a given pixel.

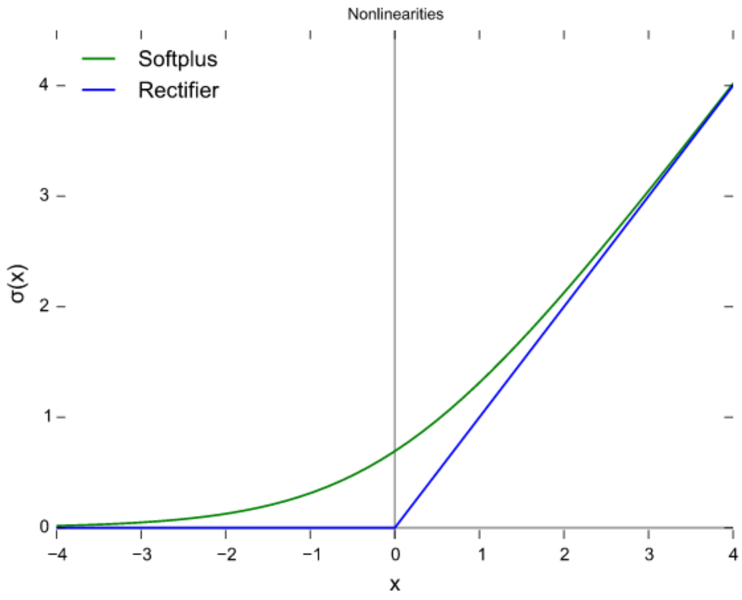
  Accuracy: 87.4%

  This may seem as a good result, but it is not. We got a 12.6% test error rate. As mentioned on the official MNIST database site [1], this result was achieved in 1998 using a linear classifier (1 layer NN). This is a baseline and we can improve it.

- **Neural network with 1 hidden layer**

  I chose to use 800 units in the hidden layer, and because of its high popularity, I chose a rectifier activation function for the neurons [4]. In addition, I applied 20% random dropout to the input data and 50% random dropout to the hidden layers. The output layer was a Softmax layer.

  I used a 2-dimensional representation of the images. In this case, each image was represented as a 28X28 matrix, where each cell represented a normalized color in a gray scale.



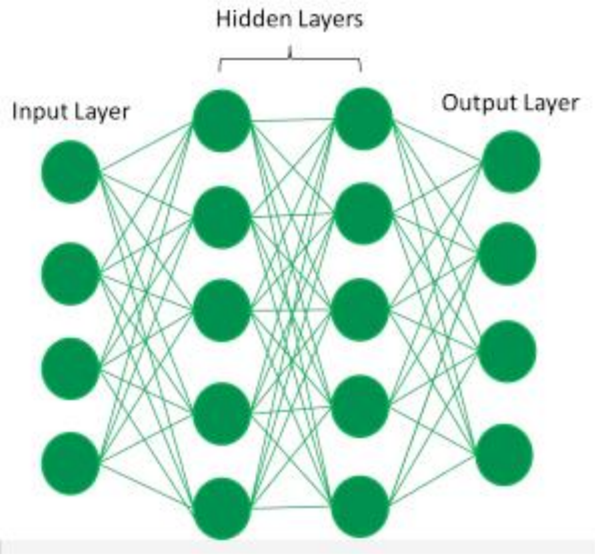
  Accuracy: 98.28%

  As we studied in the class, I used an early stopping method to identify the convergence of the model. The model converged after 140 epochs gaining 98.3 % accuracy on the dev set. Each epoch took about 8 seconds. As expected, with neural nets we got much better accuracy then with a log linear MaxEnt classifier. We are having only 1.72% of test error rate. From the MNIST page [1], we can see that this performance is equal to K-nearest-neighbors performance with deskewing, noise removal and blurring. It is also very close to the performance of Convolutional net with subsampling to 16x16 pixels. But still, most deep networks and convolutional nets gain much better overall accuracy.

- **Neural network with 2 hidden layers**

  I used the same properties here:

  - 800 units in each layer
  - Rectifier as an activation function
  - 20% input dropout
  - 50% hidden layer dropout
  - Softmax output layer.
  - Each layer was fully connected to the subsequent layer.



  Accuracy: 98.45%

  Using the early stopping technique, this model converged after 120 epochs. Each epoch took around 15 seconds.

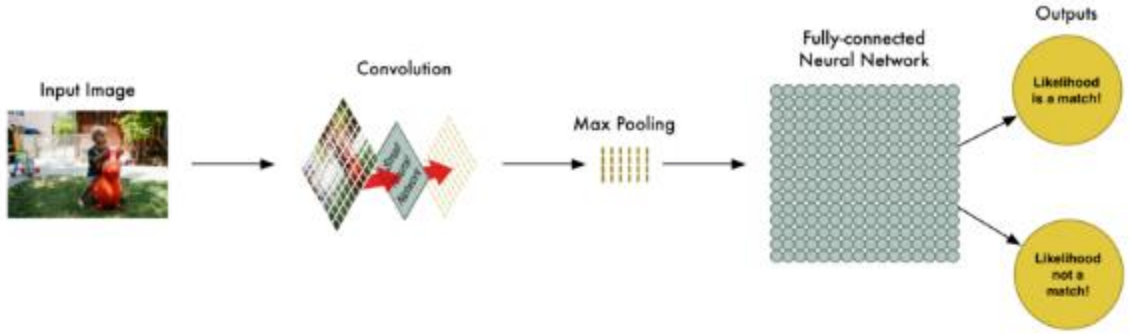
  As expected, we got even better performance. Although it is as small improvement (0.17%), it is still very valuable. As we reach "human-like" classification performance, it is getting harder to gain better accuracy.

  According to MNIST webpage [1], having error rate of 1.55% is between the performance of 2-layer NN, 800 HU, Cross-Entropy Loss, and 3-layer NN, 500+300 HU, Softmax, Cross-Entropy, weight decay. Therefore, we can see that our model reproduced the same expected accuracy.

- **Convolutional Neural Network**

  I studied about Convolutional Neural Networks from this [5] article, and decided to use the following properties for our task:

  - 2 convolution stages with 32 filters of size 5x5
  - 2 max pooling stages of factor 2 in both dimensions
  - Rectifier as an activation function
  - Fully connected hidden layer of 256 units with dropout of 50%
  - Softmax output layer with dropout of 50%



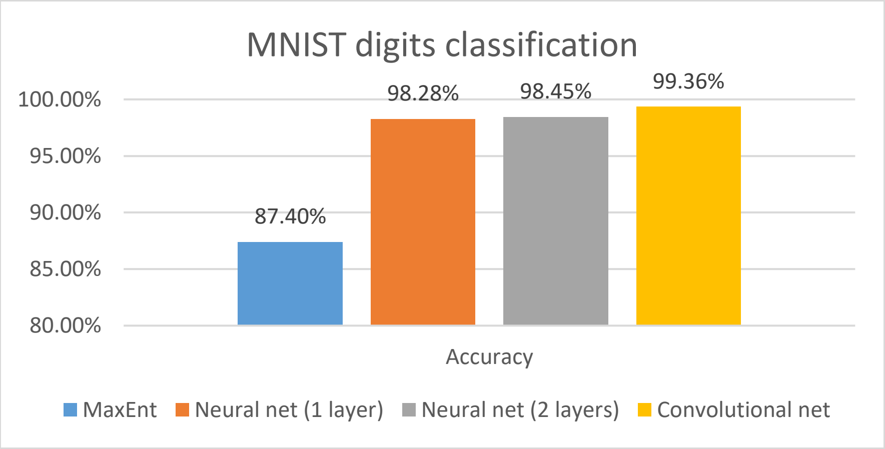
- **Convolutional Neural Network**
  Accuracy: 99.36%
  This is the best result we got so far! It is amazing to get such a high accuracy. The training was computationally very expensive. It took 119 epochs to converge and each epoch took about 320 seconds. Almost 11 hours on my computer. Gaining performance of 0.64% test error rate is better than most of the approaches documented on the MNIST webpage. It is close to Virtual SVM, deg-9 poly model and several configurations of Convolutional neural networks.

Conclusion:

Each additional layer improved the accuracy of the classification. We can conclude that deep neural networks improve our classification accuracy and are beneficial especially for image recognition tasks.

Moreover, the complexity of the neural network, like with convolutional network, also improves the accuracy. According to the MNIST page, some sophisticated convolutional networks achieved error rate as low as 0.23%. For example, the committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions] from Ciresan work in 2012.

It was interesting to see, how each model I experimented in this project reproduced the expected performances from the works that have been done in the past and are documented on the MNIST webpage.



References

[1] - http://yann.lecun.com/exdb/mnist/

[2] - https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721#.wdfz2u0wq

[3] - https://en.wikipedia.org/wiki/MNIST_database

[4] - https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[5] - https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721#.wdfz2u0wq