KU Leuven

Data Mining and Neural Networks

# Assignment 4

Zachary Jones
Master of Statistics

January 6, 2020

# Contents

# 1 Digit Classification with Stacked Autoencoders and CNN

A difficult facet of deep neural networks is that the gradient, an essential element in the backpropagation algorithm, becomes too small to affect the weights. To avoid this, we use a greedy layerwise approach that allows us to optomize each layer individually before performing back propagation to fine tune the results. However, while the effect of finetuning is small for shallower nearual networks, the effect on deeper networks is much more noticable as errors accrue when layers are stacked on top of one another. If we had not used a greedy training algorithm, the problem of the vanishing gradient would begin to limit the effectiveness of the backpropagation algorithm as we made deeper and deeper networks. As can be seen in figure 1, a stacked autoencoder with three layers, (100, 100, 50), having a classification accuracy of 99.94, effectively beats a regular perceptron with a single hidden layer of 100 neurons, which has an accuracy of around 98%.

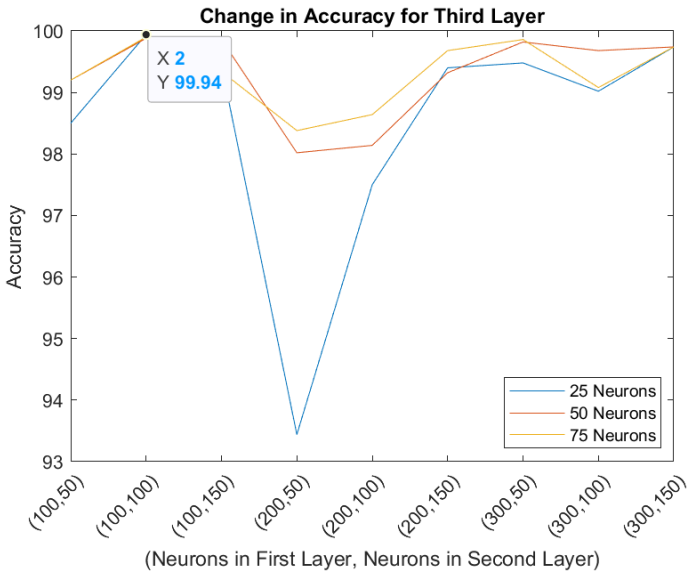


Figure 1: Graph Illustrating maximum classification accuracy for three layer stacked autoencoder, there the 2 refers to a third layer with 50 neurons

| Run | Change | % Accuracy |
|---|---|---|
| 1 | none | 82 |
| 2 | Add batchNormRegularization between convolutions and RELU units | 71 |
| 3 | Set RNG('default') | 69 |
| 4 | Double Layer Size of first convolution | 67 |
| 5 | Change first Convolution Layer to double neurons as well | 68 |
| 6 | Change Convolution sizes/neurons to (5,12) and (10, 48) | 82 |
| 7 | Change First Layer Size to (10,24) | 90 |
| 8 | Switch Convolution Layers | 91 |
| 9 | change maxPool to size 3 stride 3 | 94 |
| 10 | maxPool to 4 stride 4 | 96 |
| 11 | maxPool to 8 stride 8 | 97 |
| 12 | maxPool to 16 stride 16 | 83 |
| 13 | maxPool to 10 stried 10 | 95 |
| 14 | maxPool to 7 stride 7 | 96 |
| 15 | add a second fully connected layer | 95 |
| 16 | add another conv — batch — relu | 98 |
| 17 | add maxpool 2 stride 2 | 80 |
| 18 | add another conv — batch —relu | 99 |

When it comes to convolutional neural networks, the architecture changes slightly. Instead of having stacked layers of perceptrons performing feature reduction, convolutional arithmatic is used to try and "extract" features from an image and then those convolutional results are pooled together, in this case by finding the maximum value inside a set matrix, in order to perform dimensionality reduction. Again, network architecture has to be sussed out via trial and error.

The guiding principle behind a CNN is that they are able to determine "general" shapes, lines and blobs, that can then be recombined, and in the case of digit recognition we want to perform edge detection, as the digits are essentially collections of edges. Hence we would like to answer, is it better to have a large convolution $N \times N$ where $N$ is large? or a small one? When pooling data, when do the pools get too big and obfusicate the results? How many convolutional layers should we include? How many fully connected layers are appropriate?

With regards to the figure above, what was found was that the convolutions should be large, a wide matrix is better able to detect edges than a narrow viewport and larger convolution matrices lead to better results. Additionally, with regards to steps 4-6, the number of neurons is best when overparameterizing the convolution layer.

With regards to pooling, it was found that pooling the data was benificial until the pool size grew larger than about $8 \times 8$. It is proposed that pooling the data homogenizes it to a certain degree and can ultimately cause the features to dissapear. A second pooling procedure was tried between the convolutions, but it was detrimental to the final percentage accuracy.

It appears as though the more convolutional layers the better, and in this case, the more large convolutional layers the better. There was only an increase in accuracy when the number of convolutional layers was increased.

Finally, the fully connected layers did not seem to be particularly beneficial. It may be that their purpose is to combine smaller features and larger features into one cohesive whole, but the featureset for numerals is fairly simple and broad, so the fully connected layer does not have much need for refinement.

# 2

## 2.1

Examining the differences between VAE˙tensorflow and stacked autoencoder example in MATLAB. In both instances, the neural network structure is built through a series of weights, which are then optimized with respect to some loss function. In the case of the VAE, the loss function is the ELBO function, a combination of the Kullback-Liebler divergence and the expected reconstruction error. A key difference between the two is the architechture of the model and what exactly they are learning. The stacked autoencoder uses a set of layers with increasing sparsity to reduce a large dataset down to fewer dimensions and then uses backpropagation to reduce the error over the whole ensemble. It optimizes a nested function of the form:

$$W_3 tanh(W_2 tanh(W_1 X + \beta_1) + \beta_2) + \beta_3 ...$$

Where the W's represent the matrices of weights and are increasingly sparse. the whole ensemble is then fed into a softmax classifier and optimized using backpropagation. and has a smaller architecture:

$$QW_{2\mu} relu(QW_1 x + b_1) + Qb_{2\mu}$$

$$QW_{2\sigma} relu(QW_1 x + b_1) + Q_{b2\sigma}$$

While this can be vectorized it is kept in a seperate form to illustrate that the neural net is optimizing a *distribution* and learning the $\mu$ (mean) and $\sigma$ (variance).

## 2.2

The stacked autoencoder uses an algorithm called conjugate gradient descent, in which the solution is presented as a linear combination of conjugate vectors, an orthogonal basis found by minimizing a quadratic function of the inputs and weights. The algorithm then uses backpropagation to calculate the gradient of a performance function to change the weights themselves. It has the advantage of theoretically producing exact solutions, and rapidly converging to an acceptable solution. However, it is extremely sensitive to initial conditions and random error as most directions are not conjugate vectors.
The Variational auto encoder uses the Adam algorithm. It is in a similar family as gradient descent, however the learning rate for each network weight is maintained and adapted seperately. It is also known to be quickly converging, uses little memory, not require much tuning of hyperparameters, and to be robust against noise; making it good for large problem sizes.

# 3

## 3.1

The size of the first convolution layer is [11 11 3 96]. At first it may seem strange that a two dimensional image is associated with a four dimensional vector, however when one takes into account the number of colors (3) and the number of convolutions (96), it becomes apparent that this is an $11 \times 11$ matrix convolving across the image and extracting general features - lines and blobs that can later be combined to make moare advanced shapes.

## 3.2

There should be 96 inputs into layer 6, one for each convolution performed during layer 2.

## 3.3

The convolutional neural network started with $154587$ inputs ($227 \times 227 \times 3$ and ended with $1000$, one for each classification. A fraction of the total.