AML – EX4 – Deep learning and Convolutional neural nets.

Part 1: Toy example

In order to better understand the results it is of course crucial to understand the main difference between the two models at hand, intuitively, a linear learning model can solve problems that are linearly separable while a convolutional neural net is a non-linear learning model implying it can solve more complex problems by applying a transformation on the data (technically neural nets can approximate any function). This intuitive approach is noticed quite clearly when observing our results.

The first function can be written as:

$$y(X_1, X_2, X_3, X_4) = \alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \alpha_4 X_4$$

While α_i are scalar coefficients, obviously, y is a linear function, which of course means it is linearly separable and therefore we would expect that the linear learning model will be able to learn this function, we would also expect that our CNN will also be able to approximate this function(given the right hyper-parameters).

As for the second function:

$$y(X_1, X_2, X_3, X_4) = \left(W_{1,1}X_1 + W_{2,2}X_2\right)\left(W_{3,3}X_3 + W_{4,4}X_4\right)$$

Clearly, this function isn't linear, therefore we would expect our linear learning model to fail and that our CNN would succeed in the approximation.

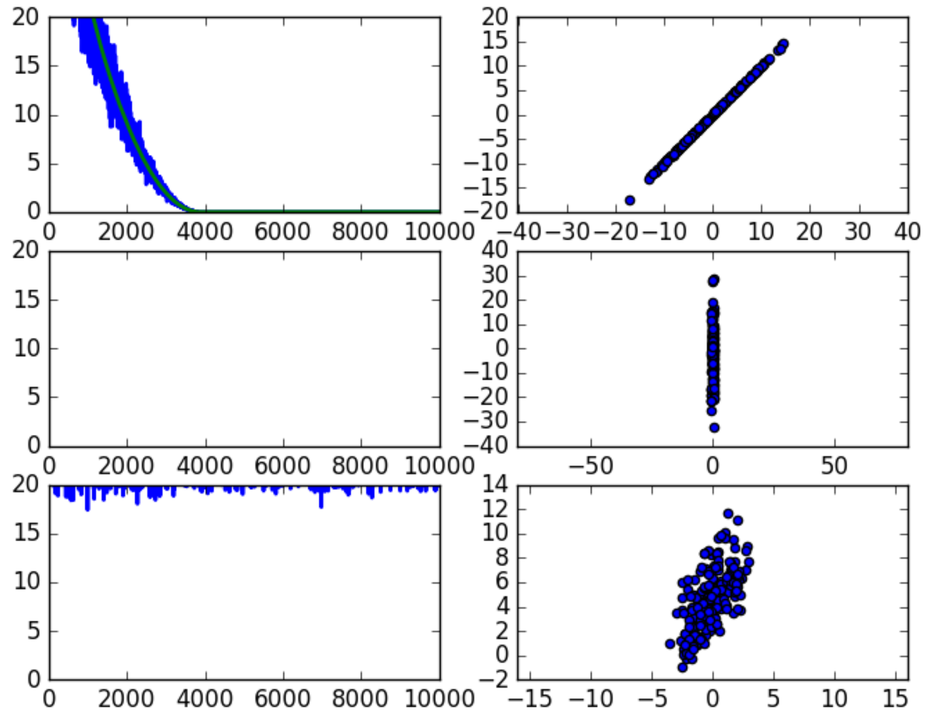
The third function can be written as:

$$y(X_1, X_2, X_3, X_4) = \log(e^{\alpha_1 X_1} + e^{\alpha_2 X_2} + e^{\alpha_3 X_3} + e^{\alpha_4 X_4})$$

Now, although this function is not a linear function, a linear model could learn a bad approximation of it since for some inputs the function's behavior is linear or relatively linear, and of course we would also expect for our CNN to be able to approximate it as well.

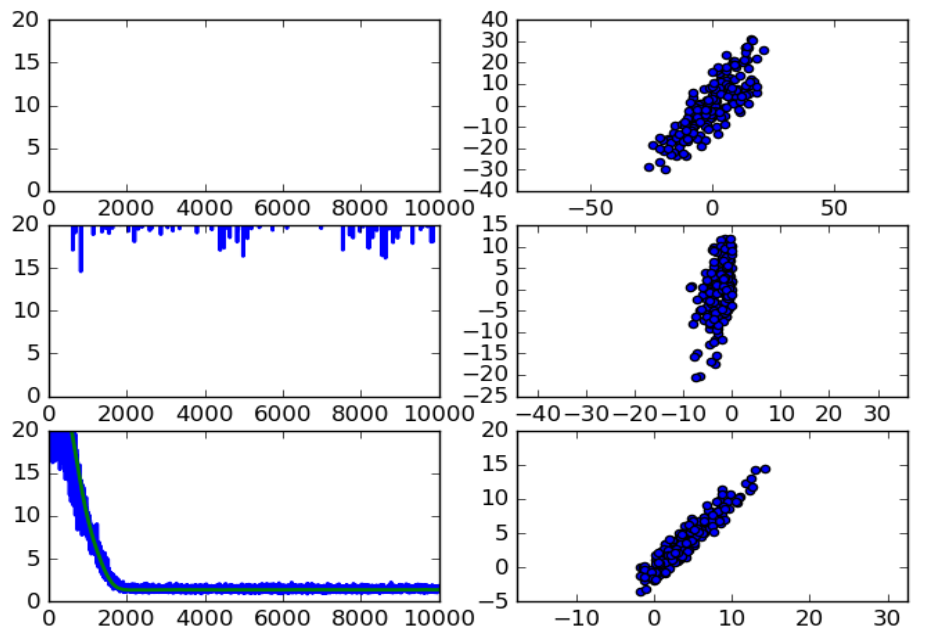
** In the following results, regarding the CNN model – we tried our best to find the ideal values for the hyper parameters but we ended up with mediocre results unfortunately.

These are the results for the linear model (for all three functions) while setting the epochs parameter to 1000:



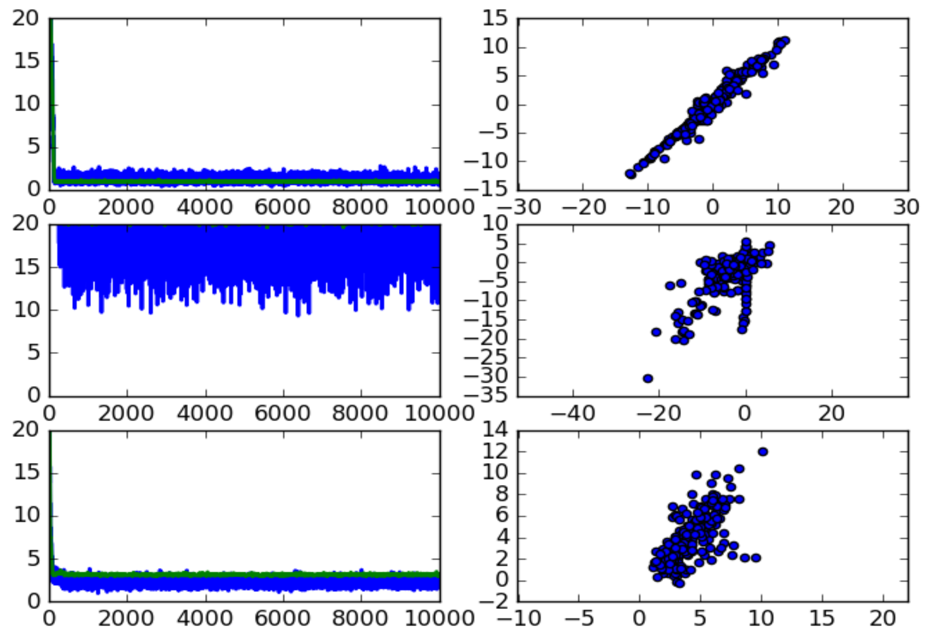
And these are the results for the CNN:

For epochs = 1000 and beta1 = 0.3 learning rate = .01:



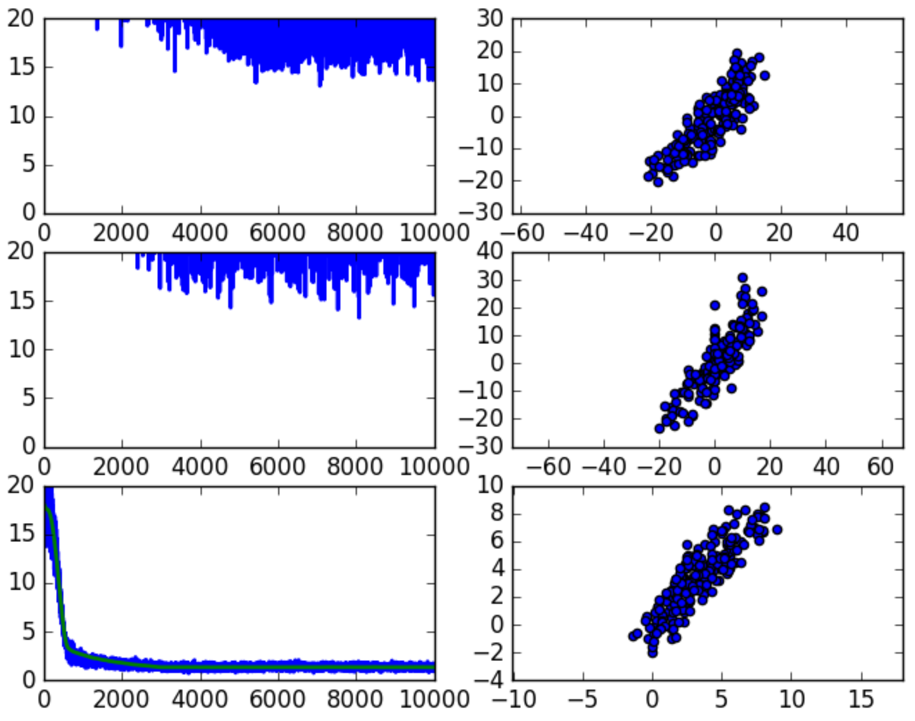
We get a reasonable (try of (:) approximation for the third function.

For epochs = 1000 epsilon = e^{-5} learning rate = .01:



We get a reasonable approximation of the first function.

And for epochs = 1000 and beta1 = .6:



We get an approximation for the second function (a quite bad one unfortunately) which was the toughest one to approximate.

** All the hyper-parameters that are not stated are equal to their default value.

Part 2: TensorFlow

The 3 sets of data we've used are:

- Block reduced data – meaning we took the original 28x28 images and we've reduced them to 14x14.
- White noise – we've added white noise to all the images.
- Block reduced data & White noise – we did both.

Examples of the images:

Original image:

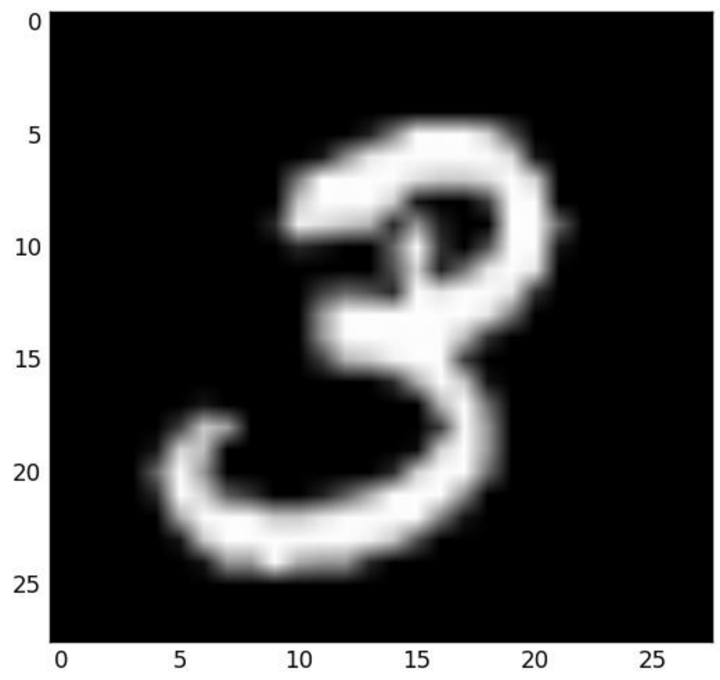
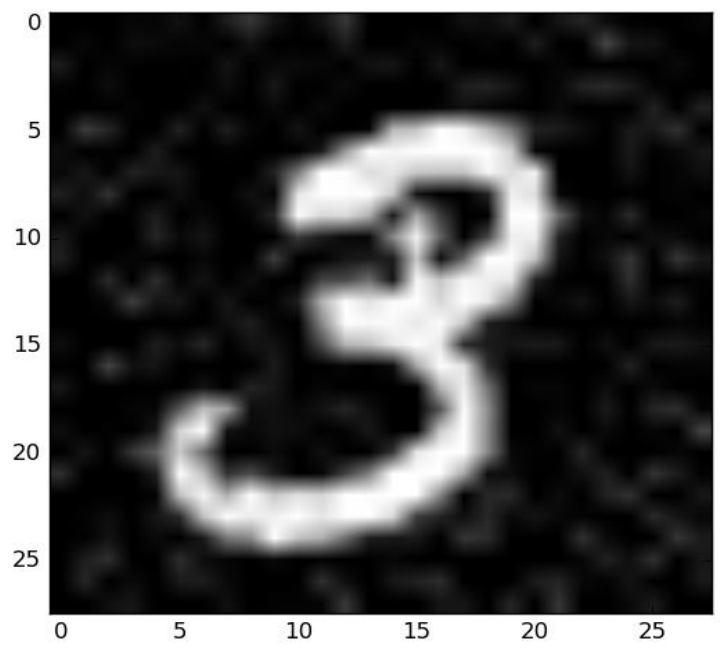
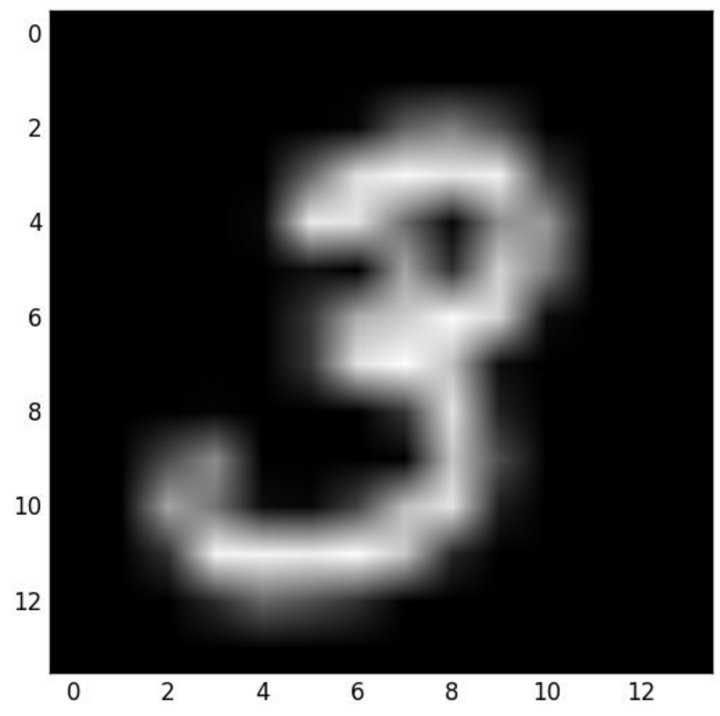


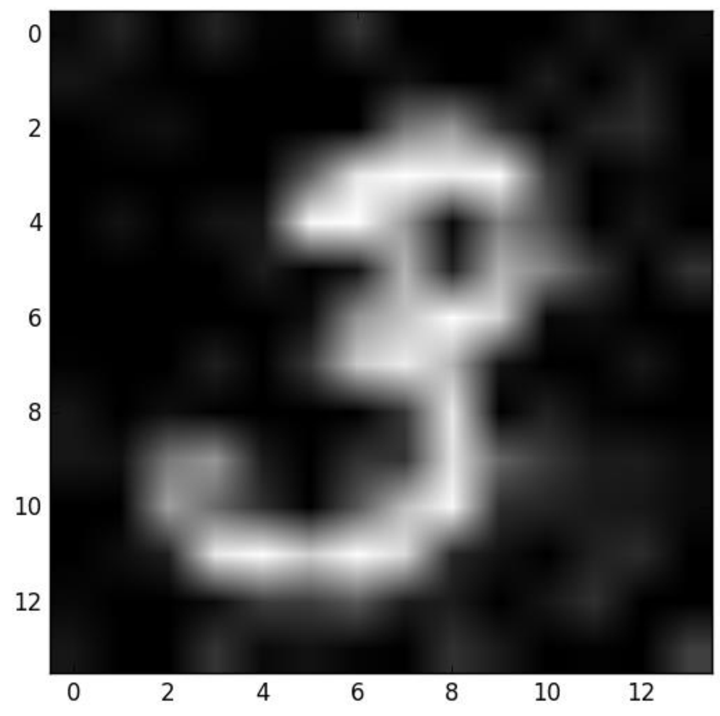
Image with white noise:



Block reduced image:



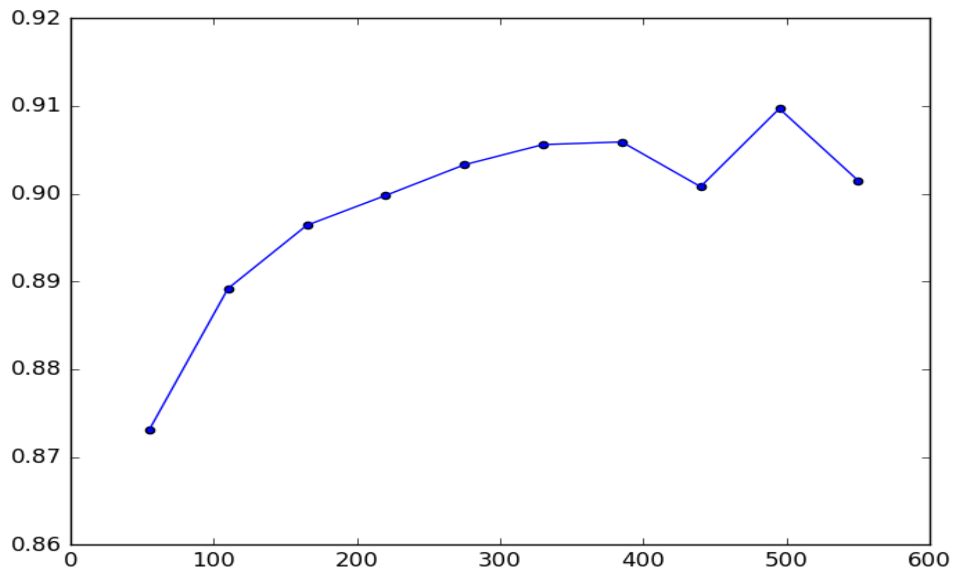
And both block reduction & white noise:



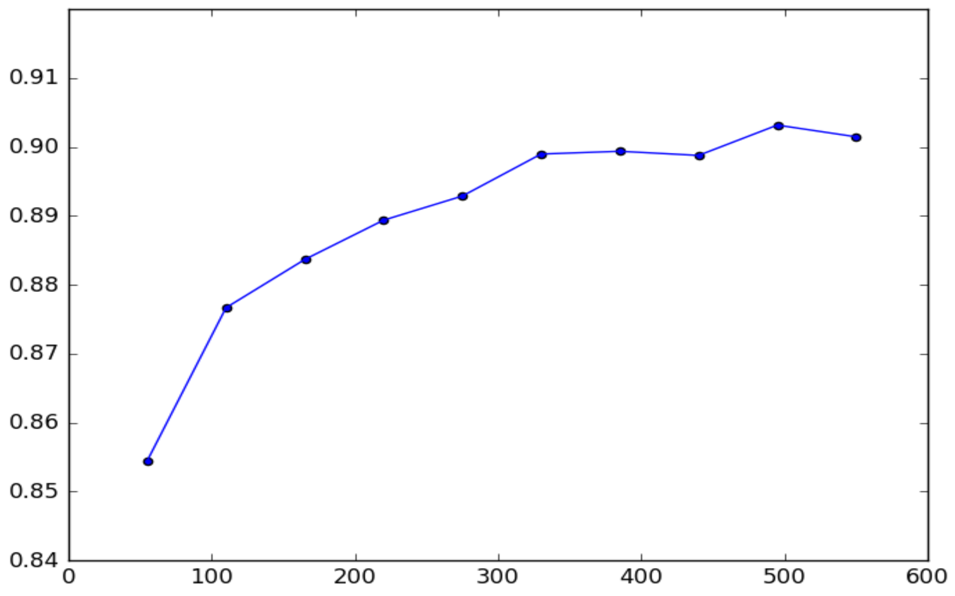
Linear model: In this model, there is no hidden layer in the neural network which means, the input nodes (784 or 196 depending on the input of course) are directly connected to the 10 output neurons, and this simple net is fully connected. As for the loss function – we've used the cross entropy function and for its optimization we've used the gradient decent algorithm. Furthermore, we've also used softmax so that the output would a vector of size ten stating the probabilities for each of the 10 possible answer options.

This model gave the worst results, its learning curve was drawn based on ten points such that the distance between two adjacent points represents 55 iterations while in each of those iterations, a batch of size 100 was used.

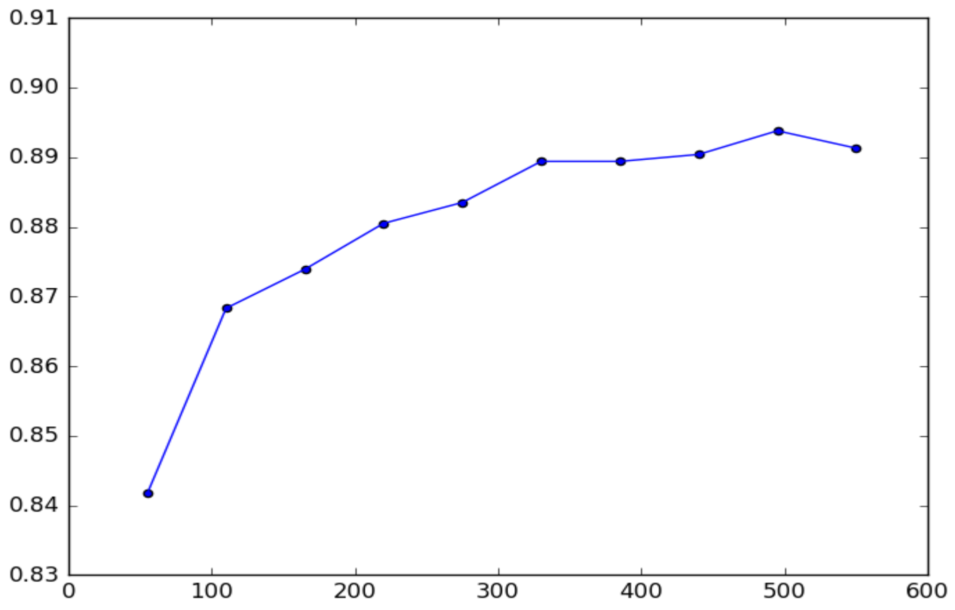
Learning curve for white noise only:



Learning curve for block reduction only:



And the learning curve for the mixture:



The final accuracy measured on the test data was:

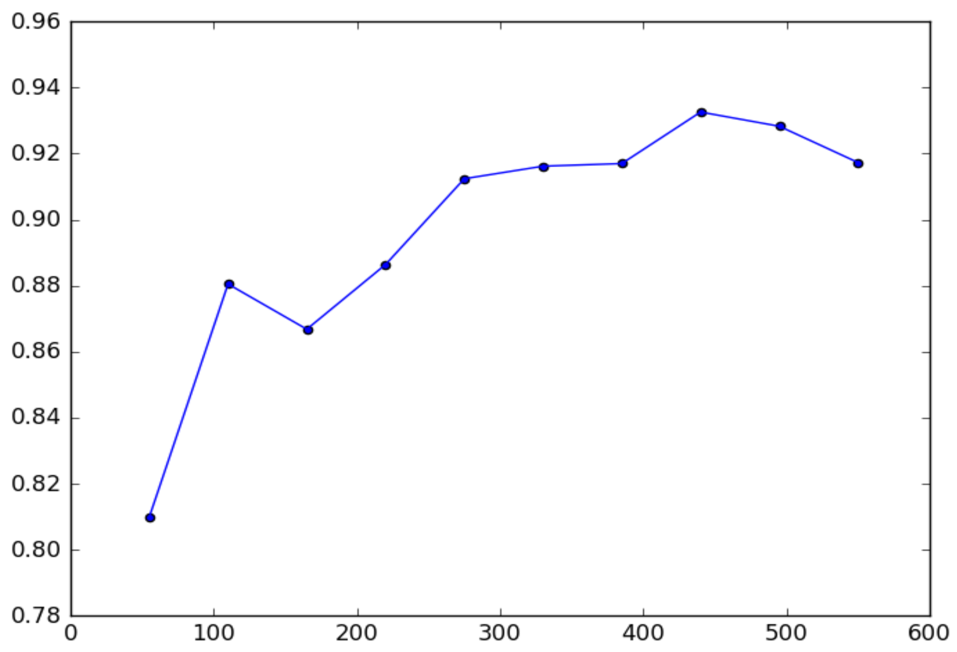
- White noise: approximately 0.903
- Block reduction: approximately 0.9015
- Block reduction& white noise: approximately 0.891

Quite predictable in fact.

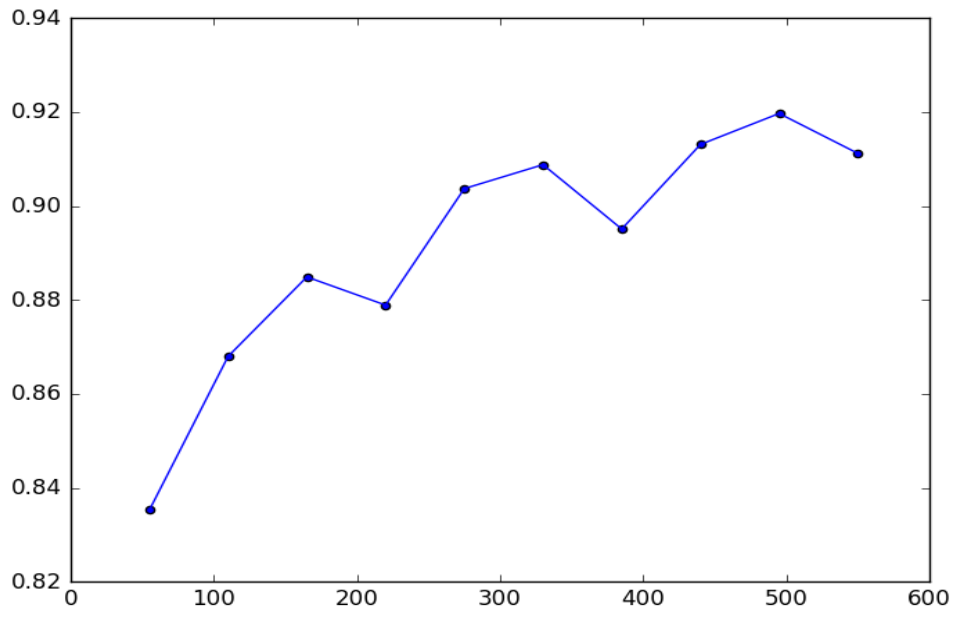
Deep model: In this model we've constructed a fully connected neural network consisting of two hidden layers – each containing the same number of neurons (as the number of input neurons in the net), each layer is fully connected to the next layer. Same as before, we've used the cross entropy as the loss function, the gradient descent for the optimization algorithm, the softmax function for a clear output and the relu function was used as an activation function.

The results were better than the linear model:

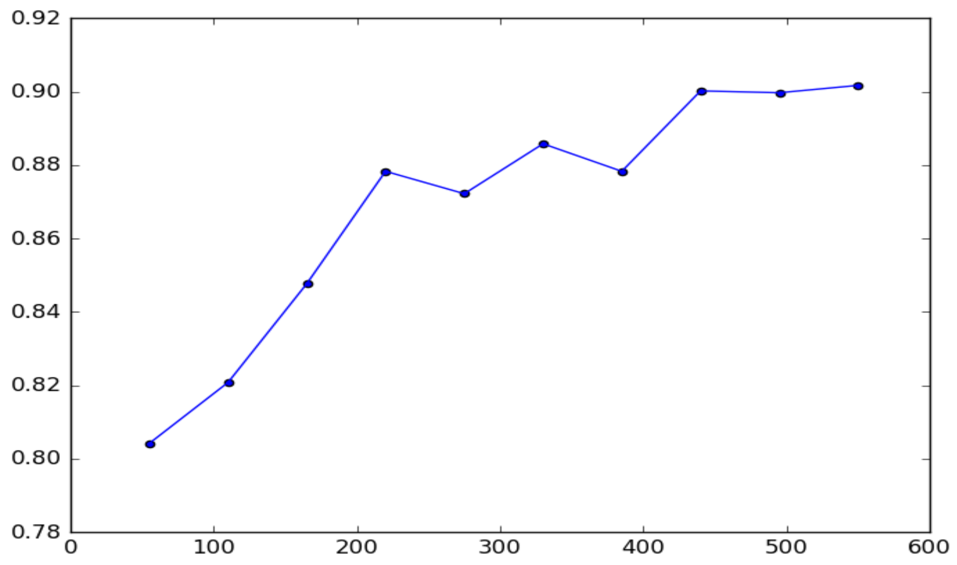
Learning curve for white noise:



Learning curve for block reduce:



And the learning curve for the block reduce and the white noise data:



The final accuracy we've measured for this model is:

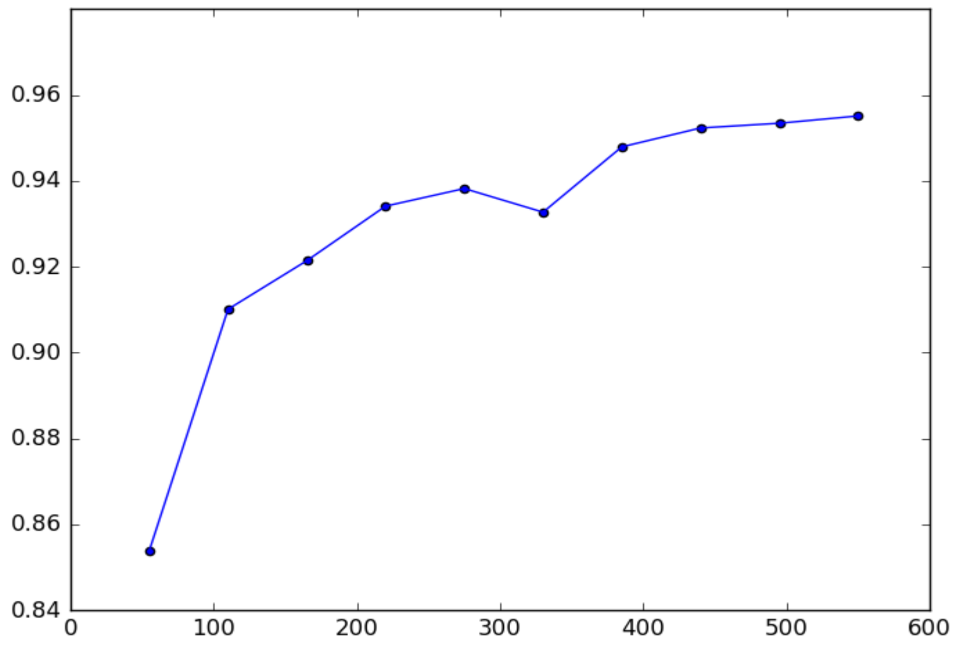
- White noise: approximately 0.912
- Block reduction: approximately 0.927
- Block reduction& white noise: approximately 0.902

CNN model: In this model we've constructed a Convolutional neural net consisting of the following layers:

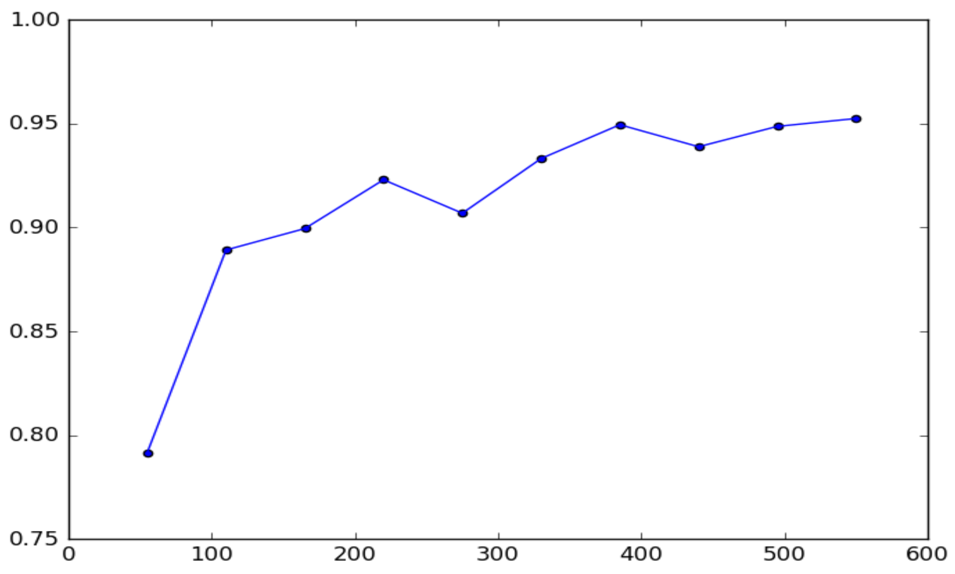
The first layer performs convolution between the input and 32 5x5 filters, the activation function here is the relu which basically means each cell in each activation map (of the 32 activation maps that are created) is fed as an input to a relu function, the size of each activation map is the same size of the input (since we use zero padding). The second layer is a 2x2 pooling layer. The next layer is a convolution layer consisting of 64 5x5x32 filters (again the activation function here is a relu function). The next layer is a layer consisting of 1024 neurons and this layer is fully connected to the previous layer. After that there is the output layer consisting of 10 neurons, all fully connected to the previous layer.

Again we've used the cross entropy as the loss function, we've used the 'adamoptimizer' as the optimization algorithm and the softmax function for the output. These are the results:

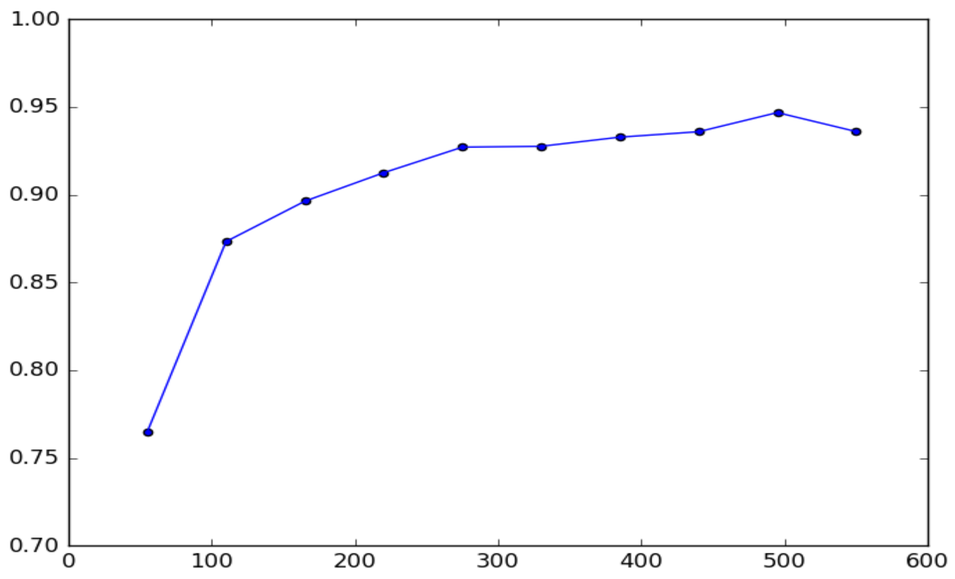
Learning curve for block reduce:



The learning curve for white noise:



The learning curve for both block reduction and white noise:



The final accuracy we've measured for this model is:

- White noise: approximately 0.9525
- Block reduction: approximately 0.9551
- Block reduction& white noise: approximately 0.9388

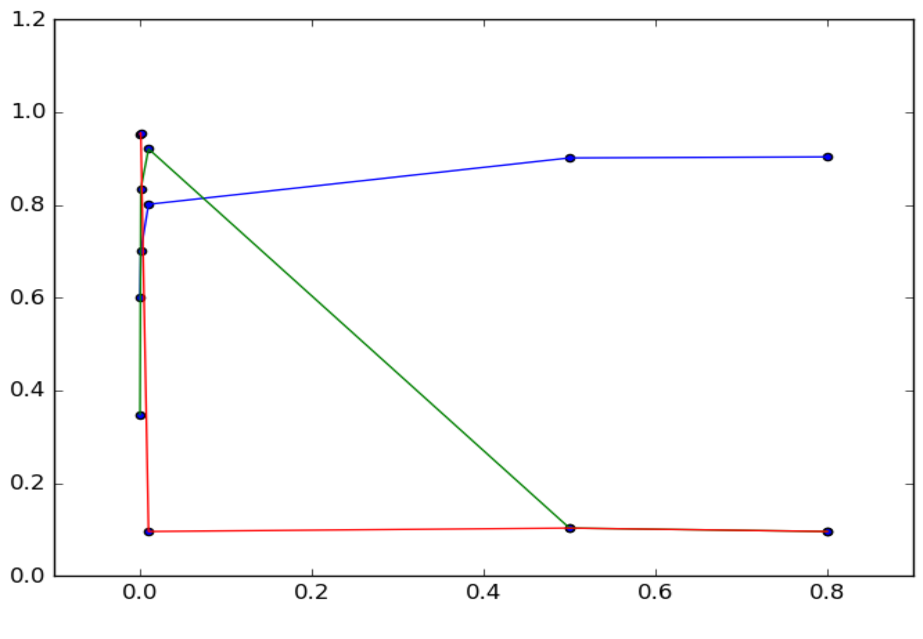
Now this results are obviously better than the previous models, but as we've read it still very bad regarding this problem. Obviously, changing the model's parameters could play a critical rule in the accuracy and finding the optimal parameters appears to be quite tricky.

Regarding our choice of the optimizing algorithm 'adamoptimizer' – we've chosen it because it gave better results (for several learning rates). As for the cross entropy, we've chosen it since the ln () function in it takes into account the closeness of a prediction – it suits perfectly if one chooses to use softmax (more generally – more data is deduced from calculating the loss function), and of course, it gave better results than other loss function (naturally).

As for the sudden drops of accuracy in the graphs – that could be the outcome of several reasons such as – inaccurate data batch, overfitting and **learning rate** (in most of the cases).

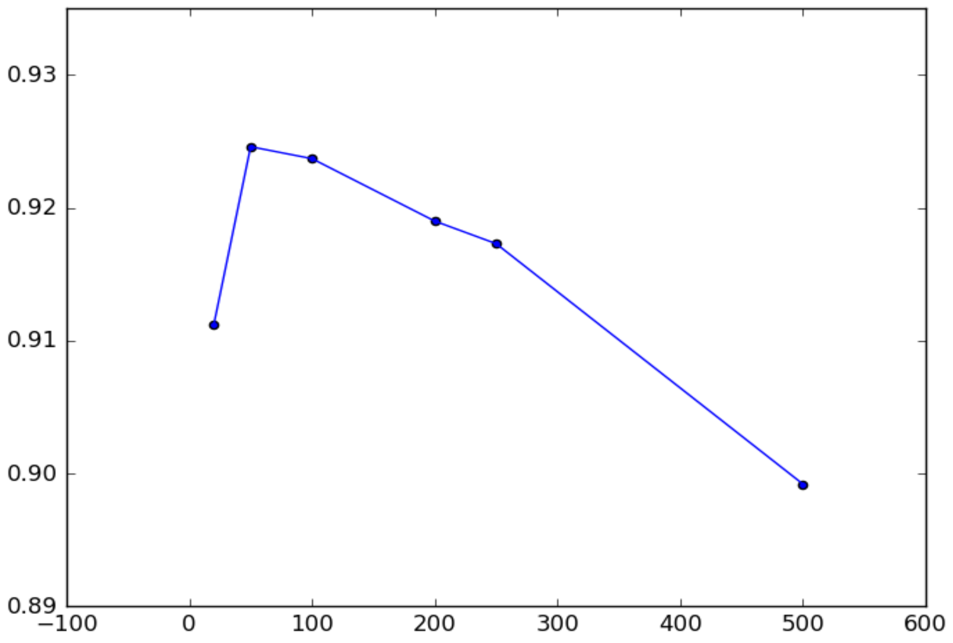
We've produced a graph of the accuracy of the 3 different models (for the block reduce data) as a function of the learning rate (since it appeared to us as a very significant parameter), we tried the following learning rates:

[0.0001, 0.001, 0.01, 0.5, 0.8] and here are the results:



Blue : Linear model, Green: Deep model, Red: CNN model

And we also graphed the accuracy of the Deep model as a function of batch size:


0.93
0.92
0.91
0.90
0.89
−100
0
100
200
300
400
500
600