Hashing as a preprocessing for learning with deep neural networks

SRIVATSAN RAMESH, NEERAJ GADKARI,

PATRICK ALRASSY, MARCELLO MORGANTINI

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

BIG DATA AND MACHINE LEARNING PROJECT

PROFESSOR:

KRZYSZTOF CHOROMANSKI

Department of
Industrial Engineering
and Operation Research
December 14, 2017

Abstract

Neural networks have been one of the most beautiful programming paradigms in the world that has simplified various tasks. The cutting edge technologies apply neural network models to achieve their desired targets. Due to a humungous amount of data available, it has been observed that the training and testing phase of neural networks take a longer time. Dimensionality reduction techniques help in compressing the data to reduce the computational efforts required to train the neural network without compromising on the performance of them. In this project, we deal with three types of hashing pipelines that helps in linear mapping of data to a lower dimensional space by giving appropriate weightage to the parts that are critical in defining the component and in performing extensive computation. These hashing pipelines are known to preserve the euclidean distance of vectors in a given space. The three hashing pipelines are Johnson LindenStrauss Transformation, Hadamard Transformations and Principal Component analysis. We are given the MNIST dataset to which the hashing pipelines are applied to generate a reduced matrix that preserves the details of the images which is applied to the neural networks to perform the classification and the stability of these algorithms is studied with respect to the classification accuracy of the neural network. The critical factors such as time complexity, computational efforts and accuracy are studied for these hashing pipelines to understand the performance of the hashing pipelines.

1. Introduction

With advancements in technology, it has become really easy for companies to understand the preferences and likings of a large population. Huge chunks of data are analyzed everyday to build prediction models and complex machine learning algorithms for effective decision making. Companies are constantly expanding their data storage capabilities to accumulate more data but there is always a cost associated with storage expansion. Also with more and more data, the time complexity associated with the algorithms increase exponentially and it takes more time to train and test these algorithms. Researchers have been working on various methodologies to reduce the dimensions of data by retaining certain critical features that define the dataset. *Dimensionality reduction* is the process of reducing the number of random variables under consideration by obtaining a set of principal values. It involves a linear mapping of data to a lower dimensional space such that the essential and critical features of the dataset are retained to perform extensive computation. There are various reduction techniques such as Johnson Lindenstrauss Transformation, Hadamard Transformations, Principal Component Analysis and so on. The main objective of this project is to understand the influence of data compression on the quality of classification with the use of deep neural networks. The MNIST database of handwritten digits is used as the dataset in this project. This dataset is preprocessed by the hashing pipelines elucidated above and fed as an input to the neural networks. The project also aims at analyzing the stability of various preprocessing pipelines in terms of time complexity, effectiveness and so on.

1.1. Why hashing?

In the era of modern computation where the feature space $\mathbf{X}$ has a dimension in the range of $40k$, it is tough to achieve accuracy with an optimal cost and computation time. Hashing helps in reducing the dimension of the feature space by preserving its integral properties in a lower dimensional space. This is also called as Randomized dimensionality reduction. The motivation behind dimensionality reduction is that it helps in improving the computational speed and reduces the time complexity of algorithms. It also helps in optimizing the available storage capacity and helps in storing the data within the constraints. Thirdly it helps in achieving a similar accuracy as that of a normal dataset when subject to processing by various algorithms. Therefore hashing can be helpful in reducing memory usage and time complexity.

1.2. Related work

Researchers have been working on hashing for many years and have been striving to make it better. In one of the early works Dasgupta [1] on random projections, it was mentioned that learning of high dimensional gaussian mixtures can be simplified when they are projected into a randomly chosen lower dimensional subspace. Nir Ailon [2] proposed a new low distortion embedding called fast Johnson-Lindenstrauss transform by preconditioning the sparse random projection with Fourier transform that is claimed to complete the operation in O(logN) time. From gaussians it extended to another hashing pipeline Hadamard (Tom St Denis), where the Fast pseudo Hadamard transformation was analyzed with respect to its speed and was proved that the transformation requires O(NlogN) time to complete. As shown in Reference [3] Xiancheng Zhou provided a new dimension towards Principal Component Hashing where it was highlighted that the PCH treats each principal component to have the same importance which leads to a loss of information in constructing hashed table thereby proposing alternatives such as Weighted PCH and Grid PCH which eliminates this uncertainty. Raja Giryes, Reference [4] proved that neural networks with random gaussian weights preserve the euclidean distance of the data with a special treatment for in-class and out-of- class data. Hashing pipelines followed by non-linear mappings have been proven effective and Anna Choromanska, in Reference [5], studied the action of random projection followed by non-linear mapping for various hashing pipelines on learning by deep neural networks and nearest neighbor classifier. Finally in Reference [6] Krzysztof Choromanski proposed a schema for recycling the Gaussian random vectors for approximating various kernel functions.

1.3. Expected Results

Research works conducted in the field of hashing have elucidated a fact that dimensionality reduction of data helps in improving the computational speed and memory utility. Past results have shown that the time complexity associated with hashing processes has become a function of logN where N is the dimension of the data. Hashing, in spite of its abilities to reduce computational efforts is found to produce less accurate results when compared with the results of non-hashed process. There exists a tradeoff between these 2 parameters, time complexity and accuracy for various hashing pipelines which will be presented in the further sections.

1.4. Scope

This project deals with hashing pipelines such as Johnson Lindenstrauss Transformations and Hadamard Transformations with and without non-linear mapping and Principal Component analysis. These pipelines are tested on the state of the art neural networks to compare the performance of these pipelines on various factors like classification errors with respect to hash size, time complexity, number of hidden layers in neural networks and so on.

2. Dimensionality Reduction Methods

2.1. Gaussian Circulant Matrix

The Gaussian circulant matrix is a variant of the random Gaussian matrix. The circulant matrix is not completely random, it has a certain structure to it. The matrix is of size $m \times n$, and reduces a vector from n dimensional space to a vector in m dimensions. The first row of the Gaussian circulant matrix is taken from a random Gaussian distribution. Each consecutive row is formed by shifting the first row to the right by one element. Hence, the matrix will look like:

$$\mathbf{C} = \begin{pmatrix} c_0 & c_{n-1} & \cdots & c_1 \\ c_1 & c_0 & \cdots & c_2 \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1} & c_{n-2} & \cdots & c_0 \end{pmatrix} \tag{1}$$

This matrix is constructed by randomly generating an array of n independent normally distributed numbers and shifting it to the right for generating each row. Since this matrix is used for dimensionality reduction, $m < n$ always holds. Hence the number of rows is always lower than the number of columns, ensuring that no row is repeated. Though the elements of two rows are different only due to the shift, the elements of each row are themselves identically and independently distributed. Each element comes from the standard normal distribution, with $mean = 0$. This ensures that the circulant Gaussian matrix is also, like the random Gaussian matrix, unbiased. The structure of the matrix, though, leads to a reduction in time complexity of the dimensionality reduction. This matrix was chosen as it is a partially structured unbiased matrix, it retains some randomness and is also unbiased ensuring the relation between input data and the label is not lost.

2.2. Hadamard Matrix

The HD matrix is the product of an $n \times n$ Hadamard matrix with an $n \times n$ diagonal matrix with diagonal elements being either +1 or -1. The resulting HD matrix will be an $n \times n$ matrix. This cannot be directly used for dimensionality reduction. m out of the n rows of the matrix are chosen resulting in a $m \times n$ matrix. This matrix can then be used for reducing dimensionality from n dimensions to m dimensions. The m rows are chosen randomly and without repetition from the $n \times n$ HD matrix. The Hadamard matrix, H, is a matrix such that:

1. All elements in the matrix are either 1 or -1.
2. The dot product of any two rows of the matrix should be 0.

There are two different ways to generate a Hadamard matrix. The easier method can only generate matrices of dimensions $2n \times 2n$. Since the number of features in each image from the MNIST is 784, the next size of a matrix produced by this method would be 1024×1024. The other method can only generate matrices of dimensions $(4n+4) \times (4n+4)$ when $4n+3$ is a prime. 787 is a prime of the form $4n+3$, where $n = 196$. Hence this method can be used to generate a 788×788 matrix. Since 788 is closer to 784, the second method is chosen to generate the Hadamard matrix. This will mean dropping fewer rows. Also, the input vector must be padded with zeros to match the number of columns of the HD matrix. Hence choosing 788, which is closer to 784, means retaining more properties of the original matrix. The $(4n+4) \times (4n+4)$ matrix is generated by first computing the quadratic residue of $4n+3$. A $(4n+3) \times (4n+3)$ matrix is then generated such that in the upper triangular matrix if the row number and column number belongs to the quadratic residue of $(4n+3)$ then the element in that position takes value $+1$, else it takes value -1. The lower triangular matrix is obtained by multiplying the corresponding element in the upper triangular matrix by -1. The diagonal elements are all set to $+1$. This results in a $(4n+3) \times (4n+3)$ matrix. To get a $(4n+4) \times (4n+4)$ matrix from this matrix, a row is added above the first row and a column added before the first column. All the elements in this row and column are set to -1. A $(4n+4) \times (4n+4)$ Hadamard matrix **H** is hence obtained. The **D** matrix is generated such that it is a diagonal matrix, with each diagonal element chosen from the set +1,-1 with uniform probability. The product of the Hadamard(**D**) and the diagonal(**D**) gives the **HD** matrix. 784 rows were chosen randomly and without replacement from the 788 rows of the **HD** matrix. The input were padded with 4 zero elements for the dimensions to match. The **HD** matrix was chosen as it also is an unbiased matrix and the time complexity of the dimensionality reduction can be greatly reduced making use of its properties. Also the variance resulting from using the **HD** matrix is lower than that from the Gaussian matrix, though it is slightly more complicated to construct.

2.3. Signed Hadamard Matrix

The signed **HD** transformation is nearly the same as the HD matrix, the only difference being that after the input vector has been multiplied by the **HD** matrix, a sgn function is applied on the resulting vector. The transformation can be represented as:

$$T(\mathbf{X}) = sgn(\mathbf{HDX})$$

Where, $T(\mathbf{X})$ is the transformation for the input **X** and **HD** is the **HD** matrix. This is a biased transformation but with a very small variance. This has been included in the hashing processes to test whether the neural network can adjust itself to nullify the bias of the transformation. If the neural network can train in this manner, then the lower variance of this method may result in a better output accuracy.

2.4. Principal Components Analysis

PCA, also known as Principal Component Analysis, is considered to be a robust method for dimensionality reduction. PCA projects the observations (possibly correlated) on orthogonal components in order to maximize the variance between them and convert them into a set of uncorrelated variables. It can be referred as a transformation that attempts to diagonalise the covariance matrix of observation data. After computing it, principal component vectors and their associated eigenvalues are available. The transformation is defined in such a manner that the first prinicpal component has the largest variance and each component after that in turn has the highest variance possible but under a constraint that it is orthogonal to the preceding components. While the former provide the directions in which observations are projected, the latter supply a kind of weight associated to them strictly related ti the singular values of the observation data set. The lowest singular values can be ignored as well as their respective principal components in order to reduce the dimensions of the dataset thereby preserving information about data variability. Thus when all the principal components and the eigenvalues are available one can always obtain the starting data set.

3. Numerical Test

Given an input-output time history ($\boldsymbol{u}$-$\boldsymbol{y}$), Neural Networks are used for fitting curves, pattern recognition and classification, clustering and dynamic time series. A classification problem is studied in this paper.The MNIST database of handwritten digits containing a set of 60,000 examples is used to perform the classification. The original black and white (binary) images from NIST have been previously elaborated by size normalization in order to fit in a 20x20 pixel box. Their aspect ratio has been preserved. Since an anti-aliasing technique has been used by the normalization algorithm, the resulting images are described by grey levels. The 60000 images (observation vectors) have been centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. As an example, four observations are plot in Fig.1 below. It can be noticed how all the numbers are perfectly centred allowing to the NN to achieve a good classification.

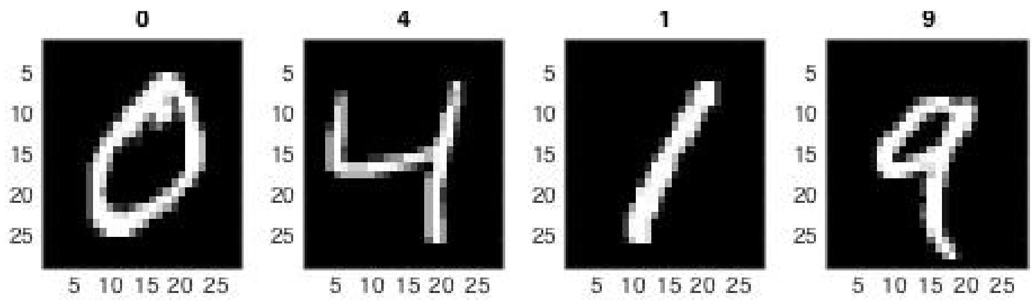


Figure 1. MNIST data set

3.1. Data Processing

Since each observation is described by a 28×28 pixels map, it is represented by a vector $\boldsymbol{u}$ of $28 * 28 = 784$ elements. Each observation is classified into one of the first ten digits 0-9. By adopting a feedforward Neural Network with output layers providing results in the range [0 ,1], 10 output layers have been adopted and classes have been turned into dummy variables. Fig.2 displays the Neural Network just described. Observations are vector of 784 elements each. One hidden layer has been implemented and it consists in 100 neurons which use sigmoid functions to compute weighted inputs and bias. Finally 10 output layers provide the result of the classification. The network is trained through scaled conjugate gradient backpropagation.

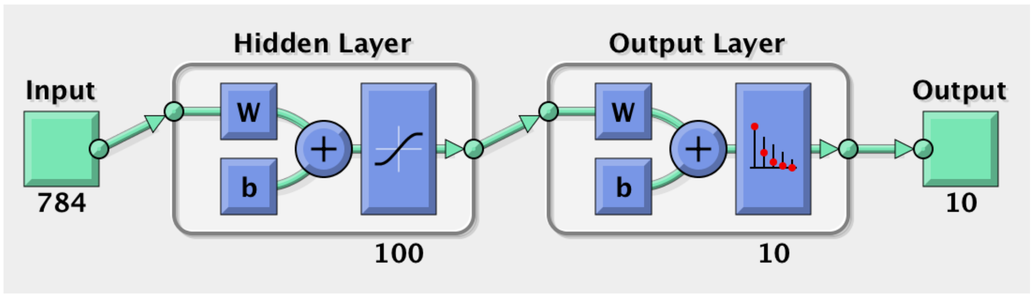
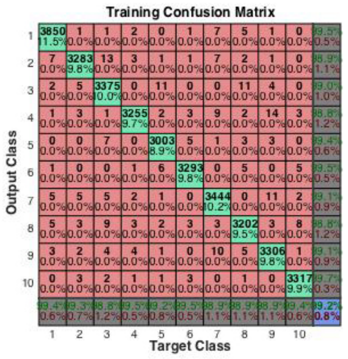


Figure 2. Neural Network

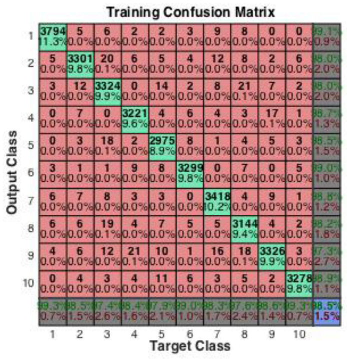
3.2. Network Settings

As mentioned, 60000 observations are available thanks to the MNIST data set. The first step consists of splitting the data into training set and testing set. The former and latter represent respectively the 80% and the 20% of the whole data set. Furthermore, in order to get a better classification *cross-validation* technique has been implemented. From the selected training set, the 70% of them has been used for training, the 30% for validation. 10-folds cross validation has been computed. Since the gradient descending theory relies on random starting condition, Monte Carlo simulation has been adopted to find reliable results. As several study cases have been investigated leading to a huge computational time, each neural network training has been performed 10 times only. Furthermore the quality of the neural network performance could be evaluated by the confusion matrix and the ROC curve. The former yields information about misclassified digits, the latter about true positive vs false positive rate



Training Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	3850 11.5%	1 0.0%	1 0.0%	2 0.0%	0 0.0%	1 0.0%	7 0.0%	5 0.0%	1 0.0%	0 0.0%	99.5% 0.5%
2	7 0.0%	3283 9.8%	13 0.0%	3 0.0%	1 0.0%	1 0.0%	7 0.0%	2 0.0%	1 0.0%	0 0.0%	98.9% 1.1%
3	2 0.0%	5 0.0%	3375 10.0%	0 0.0%	11 0.0%	0 0.0%	0 0.0%	11 0.0%	4 0.0%	0 0.0%	99.0% 1.0%
4	1 0.0%	3 0.0%	1 0.0%	3255 9.7%	2 0.0%	3 0.0%	9 0.0%	2 0.0%	14 0.0%	3 0.0%	98.8% 1.2%
5	0 0.0%	0 0.0%	7 0.0%	0 0.0%	3003 8.9%	5 0.0%	1 0.0%	3 0.0%	3 0.0%	0 0.0%	99.4% 0.6%
6	1 0.0%	0 0.0%	0 0.0%	1 0.0%	6 0.0%	3293 9.8%	0 0.0%	5 0.0%	0 0.0%	5 0.0%	99.5% 0.5%
7	5 0.0%	5 0.0%	5 0.0%	2 0.0%	1 0.0%	0 0.0%	3444 10.2%	0 0.0%	11 0.0%	2 0.0%	99.1% 0.9%
8	5 0.0%	3 0.0%	9 0.0%	3 0.0%	2 0.0%	3 0.0%	3 0.0%	3202 9.5%	3 0.0%	8 0.0%	98.8% 1.2%
9	3 0.0%	2 0.0%	4 0.0%	4 0.0%	1 0.0%	0 0.0%	10 0.0%	5 0.0%	3306 9.8%	1 0.0%	99.1% 0.9%
10	0 0.0%	3 0.0%	2 0.0%	1 0.0%	1 0.0%	3 0.0%	0 0.0%	1 0.0%	0 0.0%	3317 9.9%	99.7% 0.3%
	99.4% 0.6%	99.3% 0.7%	98.8% 1.2%	99.5% 0.5%	99.2% 0.8%	99.5% 0.5%	98.9% 1.1%	98.9% 1.1%	98.9% 1.1%	99.4% 0.6%	99.2% 0.8%

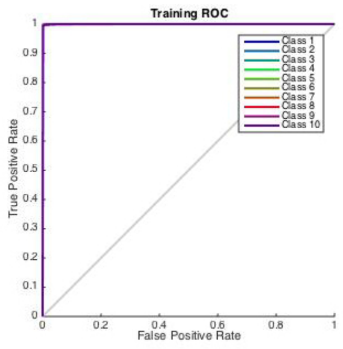


Training Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	3794 11.3%	5 0.0%	6 0.0%	2 0.0%	2 0.0%	3 0.0%	9 0.0%	8 0.0%	0 0.0%	0 0.0%	99.1% 0.9%
2	5 0.0%	3301 9.8%	20 0.1%	6 0.0%	5 0.0%	4 0.0%	12 0.0%	8 0.0%	2 0.0%	6 0.0%	98.0% 2.0%
3	3 0.0%	12 0.0%	3324 9.9%	0 0.0%	14 0.0%	2 0.0%	8 0.0%	21 0.1%	7 0.0%	2 0.0%	98.0% 2.0%
4	0 0.0%	7 0.0%	0 0.0%	3221 9.6%	4 0.0%	6 0.0%	4 0.0%	3 0.0%	17 0.1%	1 0.0%	98.7% 1.3%
5	0 0.0%	3 0.0%	18 0.1%	2 0.0%	2975 8.9%	8 0.0%	1 0.0%	4 0.0%	5 0.0%	3 0.0%	98.5% 1.5%
6	3 0.0%	1 0.0%	1 0.0%	9 0.0%	8 0.0%	3299 9.8%	0 0.0%	7 0.0%	0 0.0%	5 0.0%	99.0% 1.0%
7	6 0.0%	7 0.0%	8 0.0%	3 0.0%	3 0.0%	0 0.0%	3418 10.2%	4 0.0%	9 0.0%	1 0.0%	98.8% 1.2%
8	6 0.0%	6 0.0%	19 0.1%	4 0.0%	7 0.0%	5 0.0%	5 0.0%	3144 9.4%	4 0.0%	2 0.0%	98.2% 1.8%
9	4 0.0%	6 0.0%	12 0.0%	21 0.1%	10 0.0%	1 0.0%	16 0.0%	18 0.1%	3326 9.9%	3 0.0%	97.3% 2.7%
10	0 0.0%	4 0.0%	3 0.0%	4 0.0%	11 0.0%	6 0.0%	3 0.0%	5 0.0%	2 0.0%	3278 9.8%	98.9% 1.1%
	99.3% 0.7%	98.5% 1.5%	97.4% 2.6%	98.4% 1.6%	97.9% 2.1%	99.0% 1.0%	98.3% 1.7%	97.6% 2.4%	98.6% 1.4%	99.3% 0.7%	98.5% 1.5%

Figure 3. Confusion matrices

Fig.3 shows an example of confusion matrices in for the training error in both the case of uncompressed data (left) and data compressed through the Hadamard matrix. Generally, because of data compression and loss of information, the latter misclassifies more elements compared with the former. Also an example of the ROC curve is shown in Fig.4 below along the same line of the previous one. Each of the methods evidences a good performance.



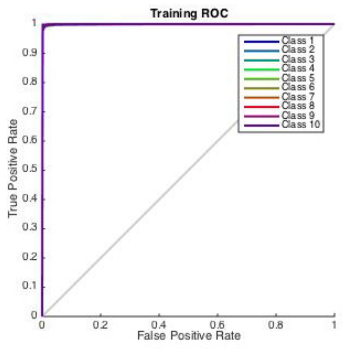


Figure 4. ROC

Even though those elements fully describe the classification quality, some other tools are necessary for the comparison at the core of this paper's purpose as shown in the next paragraphs.

3.3. Network Performance

The performance of the neural network is studied in this section.There are two main tools to assess the error entity: the mean squared errors and the cross-entropy. Whereas the former just computes a distance between the prediction and the actual output vectors, the latter heavily penalizes the output that are extremely inaccurate and by giving a little penalty for a fairly correct classification. They assign specific weights to the each element of the error vector. Furthermore there is no valid reason for why the assigned weights should be different from each other. Regularization and Normalization are possible in this scenario. Squared weights and biases do not require regularization so the regularization has not been introduced in this project work. Since the dummy variables have been employed ,therefore normalization has been valued to be unsuitable and so avoided.
In order to compute the accuracy of the Neural Network over the size of the hash, the output has been modified. The predicted output returned by the Neural Network lies in the range of [0,1] but the actual output has to be in binary digits. Therefore a threshhold of 0.5 has been set for better computation Given the NN predicted output $\boldsymbol{y}_{NN} = [y_{NN}^{(1)}, y_{NN}^{(2)}, ..., y_{NN}^{(10)}]$, its elements are modified as follows:

$$\begin{cases} y_{NN}^{(i)} < 0.5 \Rightarrow y_{NN}^{(i)} = 0 \\ y_{NN}^{(i)} \geq 0.5 \Rightarrow y_{NN}^{(i)} = 1 \end{cases} \tag{2}$$

Finally each predicted output and its coressponding actual output are compared. If all of their elements show the same value, the vectors match, otherwise if at least one of their elements is different then the prediction is not accurate. Following this policy the accuracy of the analyzed methods over the hash size has been reported in Fig.6.

3.4. Results

It is seen that without hashing test error is minimized, but computational efforts and time complexity is high. Qualitative results concerning the implemented hashing methods are plot in Fig.5 and reported in Table 1."Unreduced" shows the test error obtained when the observations are vectors of 784 elements. The Mean Squared Error and the Cross Entropy error show the same trends.
When the data is compressed to a small er dimensional space with hashing size is 64,it is seen that PCA is by far the best option for this case. However, it is very different from the other test errors as the error for PCA increases over the hashing size. A possible explanation for that unexpected behaviour may be provided by numerical issues. In fact, considering really low singular values after the SVD in the PCA may introduce numerical noise.
Given a hashing size of 512 , PCA provides a poor solution as well as the methods such as Gaussian circulant matrix. Hadamard hashing performs better than the other methods when the hashing size is over 150 and its test error converges to the one provided by unreduced data. Test errors developed using Gaussian circulant matrix and Hadamard *sgn* matrix are correlated to the one provided by the simple Hadamard hashing, but always higher. This behaviour is expected since the application of the *sgn* function to the Hadamard matrix compresses data, covering information, thus the accuracy is supposed to become lower as well as the memory required to memorize a binary matrix.

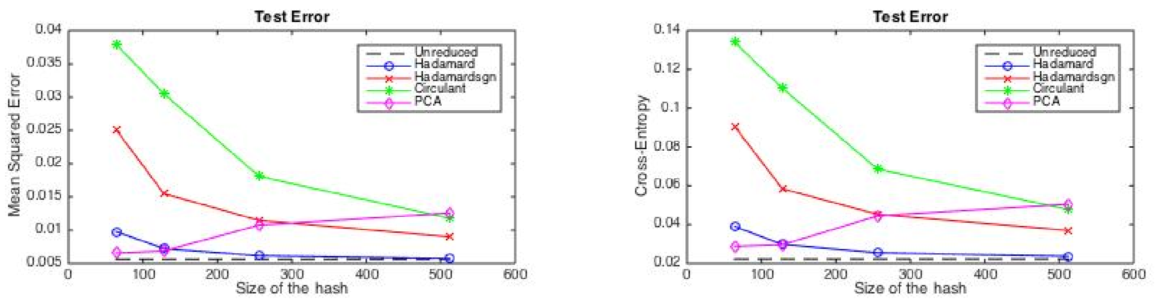


Figure 5. Mean Squared Error (left), Cross Entropy (right)

Size of the hashing	Hadamard	Hadamardsgn	Circulant	PCA
64	0.0098±0.0006	0.0251±0.0003	0.0379±0.0013	0.0065±0.0004
128	0.0072±0.0002	0.0155±0.0006	0.0305±0.0034	0.0068±0.0002
256	0.0061±0.0005	0.0115±0.0001	0.0181±0.0013	0.0107±0.0015
512	0.0057±0.0003	0.009±0.0002	0.0118±0.0003	0.0126±0.0002

Table 1: Mean Squared Error, Test Error [Mean, Std]

The accuracy of the hashing algorithms has been computed as defined in section 3.3.Results in Fig.6 and in Table 2 are consistent with the ones from Fig.5, as the error decreases over the hashing size, the accuracy increases whereas PCA and Gaussian circulant matrix hashing methods have the same accuracy for a hashing size of 512 and when it drops to 64 they differ completely.

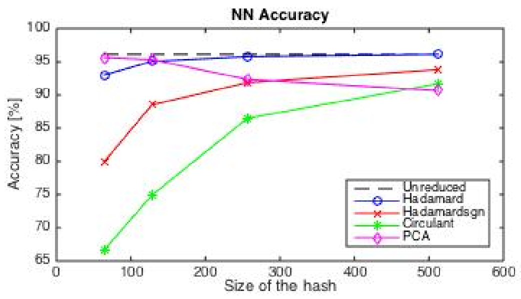


Figure 6. Neural Network Accuracy

Size of the hashing	Hadamard	Hadamardsgn	Circulant	PCA
64	93.00±0.55	79.94±0.31	66.64±1.41	95.63±0.17
128	95.06±0.16	88.52±0.54	74.89±3.60	95.32±0.18
256	95.77±0.32	91.86±0.10	86.48±1.14	92.35±1.35
512	96.11±0.19	93.82±0.15	91.69±0.34	90.65±0.33

Table 2: Hashing, Accuracy [%Mean, Std]

Test error and accuracy are not the only important factors to be considered. Other characteristics such as the net-training time and the dimensions of the compressed file may be a crucial aspect. For instance, data available from observations are described by a grey scale 0-255 defined through 8 bits, so each vector containing 784 elements needs $784 * 8 = 6272$ bits. Because of the sign function applied on the Hadamard matrix, each element will be 0 or 1, and 1 bit is enough to describe it. Thus the needed memory is remarkably reduced to $784 * 1 = 784$ bits. In addition to the fact elucidated above, the hashing numbe reduces it further.

3.5. Training time

Data compression represented as the size of the hash has a huge impact on the computational time employed to train the Neural Network. Fig.7 proves how hashing techniques reduce the training time drastically by using just 512 elements for each observation vector.
Computational time depends on the gradient descendent algorithm, based on initial random conditions. Hence computational time has been determined as the mean of 10 simulations.
As said, the observation vector consists of 784 elements. Training the Neural Network took 95.695 sec including all the aspects. Looking at Fig.7 the worst case scenario is represented by the Hadamard technique with a hashing size of 512. The training time is shown to be 40.982 sec. Even though this method takes a longer duration among the

prescribed methods, it is also the best performing as proven in Fig.6.
Hashing through a Gaussian circulant matrix allows to achieve optimal results in terms of time. By setting a hashing size of 256, 21.888 sec are required to perform the net training.

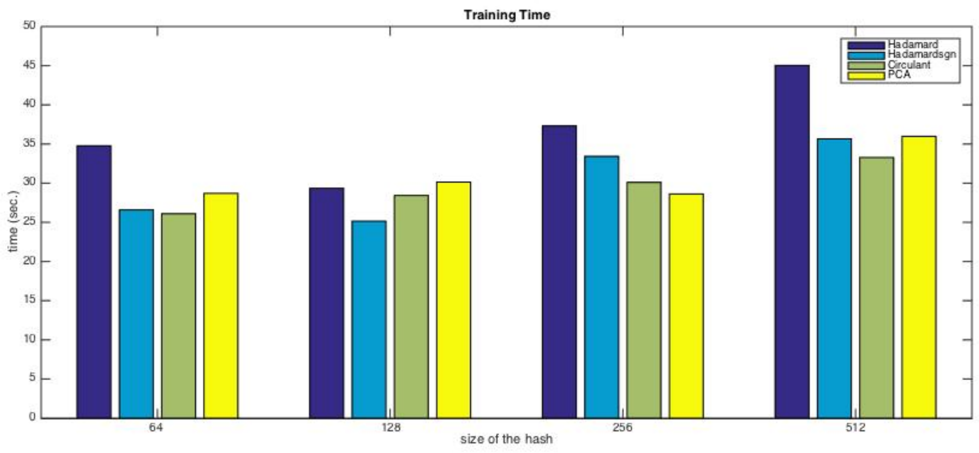


Figure 7. Neural Network Training Time

Although it does not offer the best time performance, the PCA evidences a competitive training time, especially for a huge data compression. When the hashing size is 64 the computational time is 29.517 sec, but its accuracy is the best by far as proven in the previous section. Table 3 also illustrates how PCA's computational time is competitive with the other methods both in terms of mean and standard deviation, especially when the compression due to hashing is maximum.

Size of the hashing	Hadamard	Hadamardsgn	Circulant	PCA
64	34.78±3.50	26.58±1.61	26.10±1.43	28.70±1.39
128	29.35±1.57	25.16±1.86	28.43±4.84	30.13±2.26
256	37.31±3.51	33.42±2.94	30.11±4.48	28.63±4.21
512	45.04±2.04	35.66±1.82	33.28±3.28	35.97±4.30

Table 3: Hashing, Time (sec.) [Mean, Std]

3.6. Possible Troubles

Some considerations about the neural network can be pointed out evaluating the trend for both the training error and the test error over the number of neurons employed in the hidden layer. If the training error increases (which is also very plausible), that means that adding neurons made the optimization harder, with the optimization methods and initialization in usage. Otherwise, if training error decreases and test error increases, overfitting phenomena occur in the net.

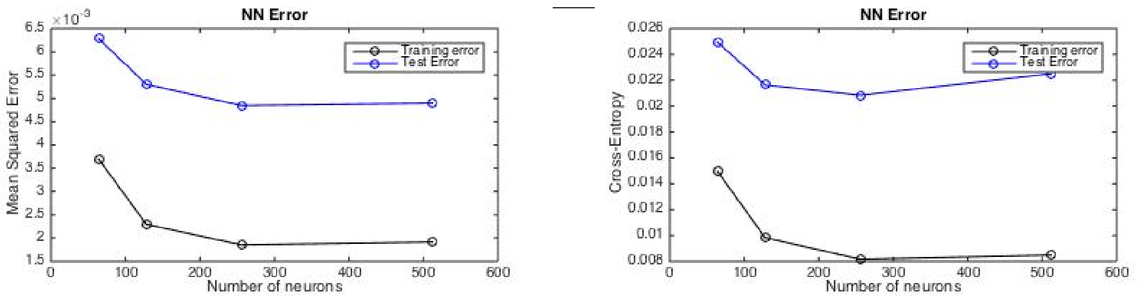


Figure 8. Mean Squared Error (left), Cross Entropy (right)

As shown in figure Fig.8 increasing the number of neurons both the test error decrease until 256 neurons are implemented. Over that value the training error keeps almost steady, but the test error increases underlining possible troubles due to overfitting giving penalties not only to the test error, but also to the training error fitting data not as well as hoped. Furthermore because of the huge amount of neurons it took 474 sec and 109 iterations to perform the net training. Mathematical issues may also be involved in this step.Table 4 displays mean and standard deviation for the test and training error over the number of neurons as a statistical support to what already shown previously in Fig.8.

Number of Neurons	Training Error	Test Error
64	0.0150±0.0016	0.025±0.0004
128	0.0098±0.0005	0.0216±0.0007
256	0.0082±0.0007	0.0209±0.0003
512	0.0085±0.001	0.0225±0.0006

Table 4: Uncompressed Data, Mean and Standard Deviation

3.7. Future works

The feedforward neural network developed by Hinton is a widely used algorithm to solve various machine learning problems. However there are certain limitations to the regular feedforward neural network. They are not known to scale larger images. We will need more neurons for the parameters to add quickly which increases the time complexity of the operation and a huge number of such parameters would quickly lead to overfitting. Also when compared to any other method, their training time is longer and therefore requires a cost-benefit tradeoff. Therefore Convolutional Neural Networks can be used in classifying images. They constrain the architecture in a better way. The layers of the convolutional neural networks have neurons arranged in 3 dimensions width, depth and height. Each layer of the convolutional neural network transforms the 3D input volume to 3D output volume of neuron activations thereby reducing the computational efforts and time complexity and it can also give better results for these hashing pipelines. Hence the study can be extended further to applying hashing pipelines to Convolutional neural networks. In this same hypothetical case where we implement a fully connected layer to retrieve the features, in Reference [7], Hijazi mentions in his paper that "the input image of size 32x32 and a hidden layer having 1000 features will require an order of 106 coefficients, a huge memory requirement". Secondly, it has been observed that Principal Component analysis has produced the best accuracy for a smaller hash size of 64. But it experiences a unique trend which is not quite common among other hashing pipelines. As per the comparison based on the accuracy, it is seen that the accuracy of principal component analysis reduces with increasing hashing size whereas the alternative would have been expected. This non-linearity in trend can be explained by the fact that the smaller singular variables after the Singular Value Decomposition can introduce some noise which can be a reason for the trend. But studies can further be extended to analyze the reasons behind the deviation.

3.8. Conclusions

The hashing mechanisms elucidated above can serve as an aid in intricate machine learning problems. It will help in getting a similar accuracy of a non-hashed projections with lesser computational efforts and time complexity. The pipelines were compared with each other for various hashing sizes based on accuracy, mean squared error, cross entropy and training time. It is seen that there exists a tradeoff between the critical parameters under study for various hashing pipelines such as accuracy, training time and size of the hash. Principal Component analysis had the best performance with hash size as 64 but it undergoes a competitive training time. Circulant gaussian matrix had a lesser training time but the accuracy is not that great. While choosing such dimensionality reduction techniques, the factors that are critical for a particular task need to be considered to select the best possible technique for effective performance. Although un-hashed projections offer a better and stable accuracy, it cannot perform complex operations in a competitive environment and thus hashing mechanisms are proved to be effective with our findings

References

[1] S. Dasgupta, "Experiments with random projection," *ATT Labs-Research*, 1999.

[2] B. C. Nir Ailon, "The fast johnson-lindenstrauss transform and approximate nearest neighbors," *Society for Industrial and Applied Mathematics*, 2009.

[3] X. Zhou, "Weighted grid principal component analysis hashing," *2014 International Conference on Machine Learning and Cybernetics*, 2014.

[4] R. Giryes, "Deep neural networks with random gaussian weights: A universal classification strategy?" *IEEE Transactions on Signal Processing*, 2015.

[5] A. Choromanska, "Binary embeddings with structured hashed projections," 2016.

[6] K. Choromanski, "Recycling randomness with structure for sublinear time kernel expansions," 2016.

[7] R. K. Samer Hijazi, "Using convolutiona neural networks for image recognition."