Department of Mathematics and Computer Science
Center for Analysis, Scientific Computing and Applications (CASA)
Postbus 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Author
Menno Hofsté (0996144)

Supervisor
Michiel Hochstenbach

Date
July 15, 2019

Digit recognition using Linear Algebra and Deep Learning

2WH40 Bachelor End Project

Menno Hofsté (0996144)
m.l.hofste@student.tue.nl

Abstract

Applications of digit and character recognition are everywhere. This paper explores three different ways to classify handwritten digits, which are k-nearest neighbour algorithm, singular value decomposition, and neural network. First, we illustrate the techniques by numerical examples. Fundamentals of these techniques are explained, and their computational complexities and accuracies are discussed. Finally, we conclude that the different properties of the methods makes them useful for different applications.

Contents

1 Introduction

Applications of automatic digit and character recognition are everywhere. Examples are recognising street signs, digitising old documents, or searching for words that appear in a photograph. The potential in this field is enormous, and as such there is also a lot of research done in this area [1]–[4]. The automatic classification of handwritten digits is often considered as a standard problem in pattern recognition, because many challenges of pattern recognition are present [2]. The research done in this area led to a number of ways to solve this problem. This thesis will give an overview of some of those popular techniques and compare them.

To compare the various techniques, we used labelled images from a subset of the US Postal Service Database which was packaged with a book from Hastie et al [5]. This data set is split up into 2 sets. There is a training set, which consists of a digit and an image representing that digit. This is the only set which the algorithms and models have complete access to. The other set is called a test set. This set also consists of a digit and an image representing that digit; however in this case the algorithms and models only have access to the images. This set is given as input after which a classification attempt is made by using the technique under consideration on the test set. Then the corresponding digit in the test set can be used to decide whether or not the classification is correct.

Each image in the data set has a dimension of 16×16 and thus consists of 256 pixels. The image is a black-and-white monochrome image, where each pixel has a certain gray-scale value between -1 and 1 with a step size of $5 \cdot 10^{-4}$. In this paper the lowest pixel value in an image is chosen to be completely white and the highest pixel value black. This is merely a convention such that the black digits are displayed on a white background.

The following structure is present in the paper. First, in Section 2, test images will be compared with the training images with the k-nearest neighbour algorithm. Secondly, in Section 3, test images are classified on their distance to subspaces spanned by the training images for a particular number. Finally, in Section 4, images are classified through the use of a neural network model.

Section 2 starts with the remark that each image can be represented by a point in a 256-dimensional space. In this space each element of the vector corresponds to the brightness of a pixel, hence this space will be called the pixel space. In this section, the similarity between two images is made more concrete. Then the idea is to classify a test digit as the digit corresponding to the most similar digit.

Section 3 exploits the fact that all training images for a particular number form a subspace. Using the singular value decomposition this space is represented in a more convenient way. The idea is to classify a test digit as the digit corresponding to the closest subspace.

In Section 4 we train a model on the training images in the hope that patterns pertaining to different images are extracted. Test images are then given as input to the model, where the model can use the different patterns to recognise a digit. Fundamental parts of a neural network will be explained with the use of a small example, after which it will be applied to the digit recognition problem.

Finally, we draw some conclusions in Section 5. In the appendix the reader can find parts of the code created in MATLAB 2019a.

2 *k*-Nearest Neighbour (kNN)

As mentioned in Section 1 the test set is a subset of the data set from which both the label and the image may be used. If an unlabelled image needs to be classified, we compare this image to each image in the training set and use the label of the most similar image. The nearest neighbour algorithm is based on this idea [6].

The nearest neighbour approach is very susceptible to outliers. Imagine an image of a number zero that is written in such a way that it closely resembles a six, and that there are two other images of a six written in almost the same way as the zero. Suppose you want to classify a six that most closely resembles the zero. In that case all information of the other two images of a six are completely ignored, hence returning an incorrect result. This particular problem can be avoided by looking at the three most similar images and choosing the most occurring one. Looking at the three nearest neighbours is known as the k-nearest neighbour algorithm, where $k = 3$. For simplicity in the remainder of this section assumes that $k = 1$ unless stated otherwise.

It is important to note that, disregarding the space and time constraints, one of the most difficult task for this particular technique is measuring similarity. Section 2.1 explores ways to relate different measures of similarity. Accuracy is important, however for many applications the speed of the algorithm is also important and as a result the space and time complexity are the focus in Section 2.2. In Section 2.3 we try to optimise the technique further at the cost of accuracy.

2.1 Similarity

Before this technique can be used to solve the handwritten digit recognition problem, a measure of similarity needs to be established. In mathematical terms, the pixel space needs to become a metric space.

Definition [7]. *A metric space is a pair (X, d), where X is a set and d is a metric on X (or distance function on X), that is, a function defined on X×X such that for all $x, y, z \in X$ we have:*

- *d is real-valued, finite and non-negative.*
- *$d(x, y) = 0$ if and only if $x = y$.*
- *$d(x, y) = d(y, x)$.*
- *$d(x, y) \leq d(x, z) + d(z, y)$.*

For the pixel space to become a metric space a suitable distance function needs to be defined. A lot of research has been done in this area resulting in many different distances. There are more generic distances such as the 1-norm and Euclidean distance, and there are more specialised distances. Several distances useful for digit recognition are the Fuzzy Image Metric (FIM) [8], Generalized Hausdorff distance [9], Tangent distance [10], and IMage Euclidean Distance (IMED) [11].

The FIM is primarily used in image quality assessment for image compression. The Generalized Hausdorff distance is very useful in comparing shapes in binary images. The Tangent distance is developed particularly for digit recognition and takes various transformations into account. IMED is similar to the more general Euclidean distance, however small perturbations result in a smaller difference [11].

First, we discuss the Euclidean distance and the 1-norm applied to the classification problem and their performance is measured. After this, the Tangent distance will be inspected further as, according to Wang et al [11], it results in the highest accuracy among the distances relevant to digit recognition.

2.1.1 Euclidean distance / 1-norm

Both the Euclidean distance and the 1-norm are used to compare points in the pixel space. Flattening an image consisting of 16 by 16 pixels results in a point in the pixel space. A visualisation of flattening a 3 by 3 image can be found in Figure 1.

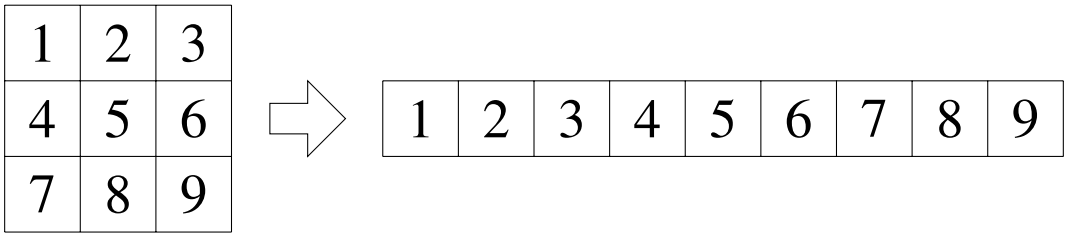


Figure 1: Flattening, or reshaping, of a 3 by 3 image

.

In this section, the pixel space is constructed as a subspace of the 256-dimensional Euclidean space, $\mathbb{R}^{256}$, which is a Hilbert space [7]. This implies that the pixel space is a complete metric space and hence has a valid metric. Given two vectors, $\mathbf{x}$ and $\mathbf{y}$, the Euclidean distance [7] in the pixel space is defined as

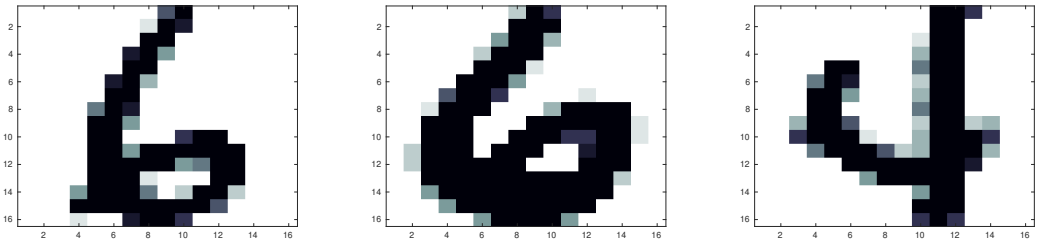
$$d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{\sum_{i=1}^{256}(x_i-y_i)^2} = \sqrt{(\mathbf{x}-\mathbf{y})\cdot(\mathbf{x}-\mathbf{y})} = \sqrt{\left\|(\mathbf{x}-\mathbf{y})^{\circ 2}\right\|_1}.$$

Here, $\circ$ denotes the Hadamard power or entrywise power. The 1-norm, otherwise known as the Taxicab or Manhattan distance, is also a valid metric [12]. Given the vectors $\mathbf{x}$ and $\mathbf{y}$ in the pixel space, the 1-norm distance is defined as

$$d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_1 = \sum_{i=1}^{256}|x_i-y_i| = |\mathbf{x}-\mathbf{y}|\,.$$

We define $\mathbf{x}$ as the image that needs to be classified. Using these metrics, the distances between $\mathbf{x}$ and all images of the training set can be calculated. Then, $\mathbf{x}$ is classified as the digit corresponding to the image with the smallest distance to $\mathbf{x}$.

The explanation has been abstract thus far, so to make these ideas more concrete a very small example is presented. For this example, the digit that needs to be classified can be seen in Figure 2a. The training set only contains a 4 and a 6, which can be found in Figures 2b and 2c respectively. For convenience, the images in Figures 2a, 2b, and 2c will be referred to by the vectors $\mathbf{x}, \mathbf{y}$, and $\mathbf{z}$.



(a) Test digit $\mathbf{x}$. (b) Training data $\mathbf{y}$. (c) Training data $\mathbf{z}$.

Figure 2: Three different images from the US Postal data set.

The first step is to find the distance between $\mathbf{x}$ and $\mathbf{y}$ and the distance between $\mathbf{x}$ and $\mathbf{z}$. In Figures 3 and 4 the results of the operations

$$\mathbf{u} = (\mathbf{x} - \mathbf{y})^{\circ 2} \text{ and } \mathbf{v} = (\mathbf{x} - \mathbf{z})^{\circ 2}$$

can be found. These operations are the equivalent of taking the pixel wise difference between the two images and then squaring each individual pixel value. Notice that these operations are present in the definition of the Euclidean distance. To calculate the distance from this, for both $\mathbf{u}$ and $\mathbf{v}$ the sum of all the pixel values is taken. Notice how more overlap between the test and training images lead to "difference"-vectors with less black values.

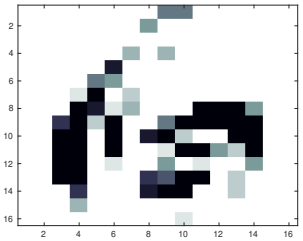


Figure 3: The vector $\mathbf{u}$.

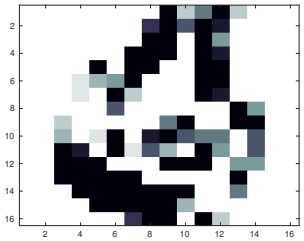


Figure 4: The vector $\mathbf{v}$.

The sum of all pixel values of $\mathbf{u}$ and $\mathbf{v}$ are 12.55 and 15.83. This means the following:

$$\begin{aligned}
12.55 &< 15.83 \\
\implies \sum_{i=1}^{256} u_i &< \sum_{i=1}^{256} v_i \\
\implies \sum_{i=1}^{256} (x_i - y_i)^2 &< \sum_{i=1}^{256} (x_i - z_i)^2 \\
\implies \sqrt{\sum_{i=1}^{256} (x_i - y_i)^2} &< \sqrt{\sum_{i=1}^{256} (x_i - z_i)^2} \\
\implies \|\mathbf{x} - \mathbf{y}\|_2 &< \|\mathbf{x} - \mathbf{z}\|_2 \\
\implies d(\mathbf{x}, \mathbf{y}) &< d(\mathbf{x}, \mathbf{z})
\end{aligned}$$

In other words, the result that $12.55 < 15.83$ implies that $\mathbf{x}$ more closely resembles $\mathbf{y}$ than $\mathbf{z}$. Since there are only two training samples this also means that the test digit will now be classified as the digit of the most similar image, which is a six. This is how a digit is classified using the nearest neighbour algorithm in conjunction with the Euclidean distance.

One may note that even though the example was set up to do a correct classification, the difference between $\mathbf{u}$ and $\mathbf{v}$ is pretty small. This is related to the fact that the 6 in Figure 2b is "wider" and "fatter". Furthermore, the Euclidean distance is also susceptible to rotations, scaling and transformations. The latter of which is illustrated in Figures 5 and 6.

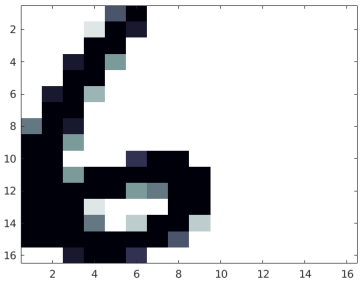


Figure 5: Transposition of $\mathbf{x}$.

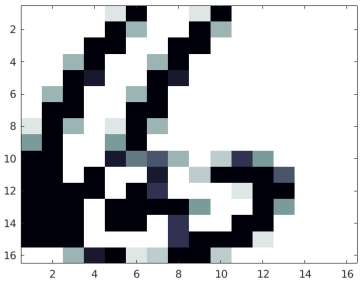


Figure 6: Difference $\mathbf{x}$ and its transposition.

The difference between the sum of the elements of Figure 2a and the same image shifted left by four pixels is 17.52. This distance is greater than both the other digits. To deal with these weaknesses the Tangent distance was developed.

2.1.2 Tangent distance

As mentioned before, the 1-norm and Euclidean metric are very susceptible to translations and rotations. The Tangent distance is developed to be invariant with respect to translation, rotation, scaling, shearing, line thickness, and two hyperbolic transforms [11]. An example of shearing is the mapping from (x, y) to $(x + 2y, y)$.

As we have already established, an image is a point in the 256-dimensional space $\mathbb{R}^{256}$. Take a point from this pixel space. Now, define a transformation. As example, we take a rotation, such that the parameter corresponds to the angle of the rotation. Transforming the point according to this rotation yields another point in the pixel space for each angle of rotation. All points obtained from rotating one point together form a 1-dimensional manifold [10], [13]. It helps to think of this as a line through the pixel space with all slightly rotated versions of the point. If we take P to be an image representing a three, then these rotations are the true rotations of P as can be seen in Figure 7.

In case multiple transformations are of interest, the set of transformations can be parametrised by n parameters. This in turn yields an n-dimensional manifold.

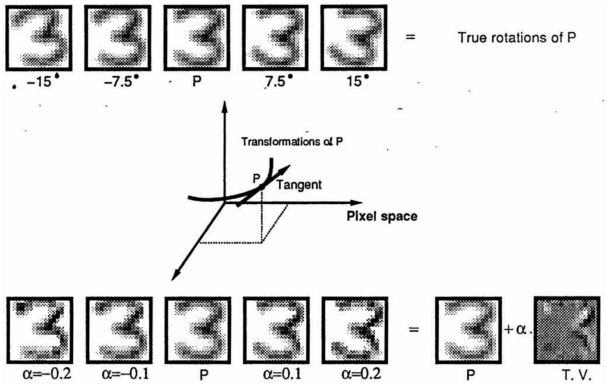


Figure 7: Visual representation of a manifold. Image from Simard et al [10].

The distance between two points can now be defined as the minimum distance between their respective manifolds. A big problem with this is that the manifold will not be linear, which means that the computation of this distance is a hard non-linear optimization problem [14]. Alternatively, the distance to the linear approximation of the manifold can be calculated as the approximation is very good for reasonably small angles [13]. The difference can be seen in Figure 7, where the resulting rotations of the linear approximation along with tangent vector are shown. Figure 8 also shows the difference between the true rotations and the ones created by the linear approximation along with a sketch of the approximation in the pixel space. Figure 9 visually shows the different distances and how they relate to each other.

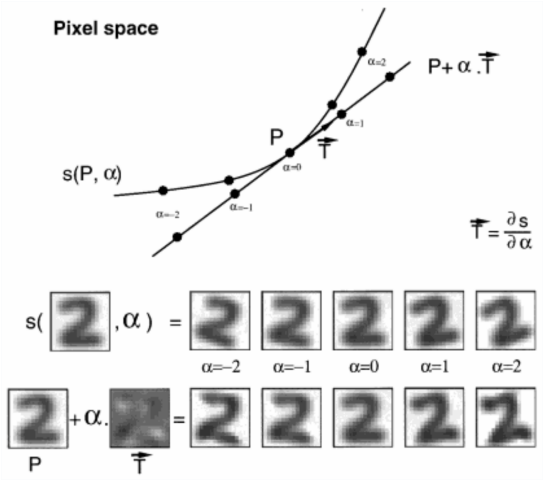


Figure 8: Tangent of manifold [13].

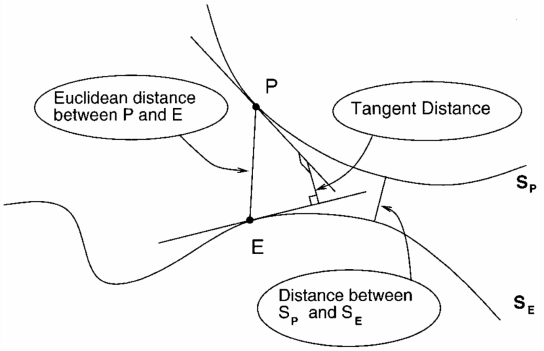


Figure 9: Distances where $\mathbf{P}$ and $\mathbf{E}$ are points in the pixel space [13].

The main advantage of this approach using the Tangent distance is accuracy as seven chosen transformations mentioned earlier, which should not influence distance, have less impact on the distance compared to Euclidean distance [11]. Naturally, computing the linear approximation results in a lot of computational overhead, increasing the time complexity. The results which confirm these statements can be found in Table 1.

Table 1: Results of nearest neighbour with $k = 1$.

Distance	1-norm	Euclidean	Tangent
Accuracy	0.91	0.92	0.94
Time (s)	7.20	6.73	59.92

The code to calculate the tangent distance between two images is taken from Keysers et al [14] and was slightly modified to fit the images with a different dimension.

2.2 Complexity

Certain applications of digit recognition require a faster classification time than others, such as self-driving cars as compared to writing input on a tablet. The time complexity describes the amount of time an algorithm takes to run. All previous subsections discuss the classification of a single digit. For this single image a distance needs be calculated to every single training image. This cannot be avoided, however when a second image needs to be classified the distance to each training image should not have to be calculated all over again as the point distribution stays the same. Certain data structures, such as the k-d tree, could reduce the amount of distances that need to be calculated.

Besides the computational cost, k-nearest neighbour as implemented until now is very sensitive to the composition of the training data. For example, having a lot of sixes compared to zeros will mean that the number six is much more likely to be chosen for a classification. To solve this, a lot of variations of kNN such as the weighted k-nearest neighbour, and k-d tree nearest neighbour have been developed [15].

The computational complexity will be derived where useful.

2.2.1 Linear Search

The way the classification problem is solved up until this point is with the use of linear search, sometimes called exhaustive or naive search. It is the "normal" implementation of kNN and does not rely on any structuring of the data. For each image $\mathbf{y}$, the distance between $\mathbf{y}$ and every other image in the training set is calculated. The training images are then sorted on their distance to the image $\mathbf{y}$. The most occurring digit in the top k is picked as the result of the classification.

A tie happens when the two images closest images have different labels. The case $k = 2$ results in 293 ties in the used data set, which is 14.6% of all cases. The case $k = 3$ results in no ties happening, which has to do with the fact that a three-way tie is only possible when the three nearest neighbour all represent a different number. In case of a tie there are several tie-breaking actions that could be chosen:

- Choose a different arbitrary k until there is no longer a tie [16].
- Randomly choose between tied values. This is the simplest way, however since there is no reason to make a particular a choice. This typically leads to the lowest accuracy [16].
- Start with $k = 2$ and increase k until the tie is broken [16].
- Adding weights to the k values. There are several different functions for assigning weights [17]. A simple example is $\frac{1}{d(\mathbf{x},\mathbf{y})}$ as the vector $\mathbf{y}$ has the smallest distance to $\mathbf{x}$ of the k-nearest and there-

fore has the most influence [18]. This solution corresponds to the weighted k-nearest neighbour mentioned earlier [15].

Now, we reason about the time complexity for linear search. Let t_r be the number of training images and t_e the number of test images. Since both test and training images have a fixed dimension of 16×16, the time complexity of calculating a distance between two images is $\mathcal{O}(1)$, which is constant. To classify one image, t_r number of distances need to be calculated, so the time complexity is $\mathcal{O}(t_r)$. To classify t_e images, the classification of one image is repeated t_e times, so the total time complexity is $\mathcal{O}(t_r \cdot t_e)$.

2.2.2 *k*-d tree

Calculating the distance between all training images and test images requires a lot of computation, which in turn takes time. Recall that training images can also be represented by a vector in the Euclidean space and Euclidean space is a vector space in which all points can be ordered with respect to a single dimension. The idea is to use these properties to create a data structure, such that a lot of candidates can be pruned. One such data structure is the k-dimensional tree, or k-d tree [19].

The recursive creation of such a k-d tree with dimensionality 2 will be explained. All entries in the training data set form a set S_0. Sort all the points with respect to their first dimension, here x coordinates. Now, take the median point as the root node, in the case of Figure 10 this is $A(50, 50)$. Divide all points into two groups, S_1 and S_2, which contain the points with smaller x coordinates and larger x coordinates respectively. Now, do this recursively on S_1 and S_2 while using the next dimension, here y coordinates. Repeat this while cycling through the dimensions until all points are in the k-d tree. A k-d tree can be built in $\mathcal{O}(kn \log n)$ time [20]. According to [19], the asymptotic running time for nearest neighbour queries is empirically observed to be $\mathcal{O}(\log n)$.

An example of a query is as follows. Suppose we want to find the nearest point to the point $\mathbf{Z}(60, 85)$. The first coordinate of **Z** is bigger than **A**, so in the tree we traverse to the right side. The second coordinate of **Z** is bigger than **C**, so in the tree we traverse to the right side. The first coordinate of **Z** is smaller than **F**, so in the tree we reach a leaf node and **F** is thus the closest node. Now, we draw a circle with radius $d(\mathbf{Z}, \mathbf{F}) = 10$. This circle crosses no splitting boundaries, so there cannot be a point closer than **F**. In case the circle does cross a splitting boundary, there can be a closer point, so you have to traverse up the tree a check each element in the branch corresponding to the plane the circle crosses.

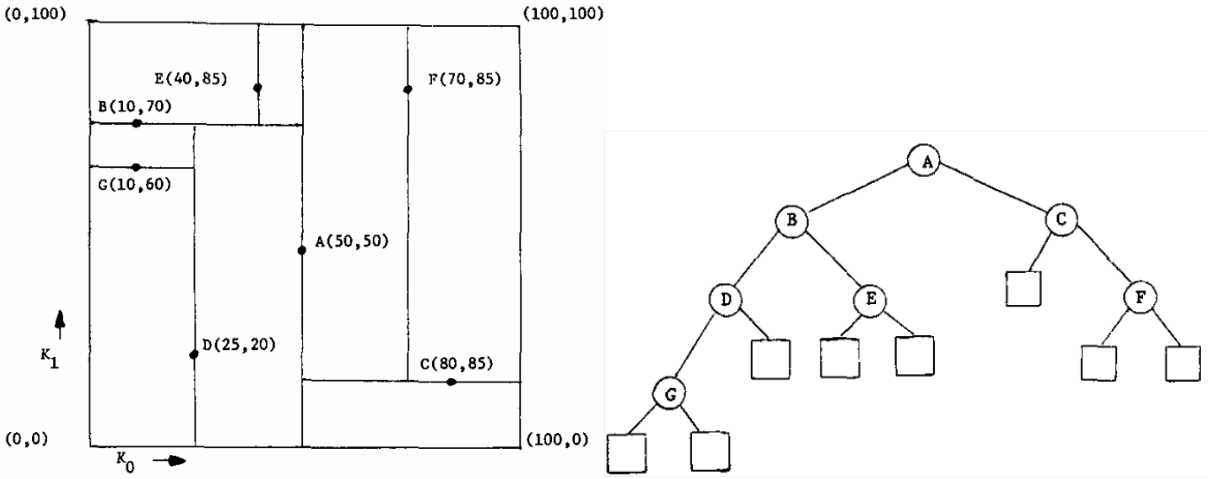


Figure 10: Points in 2-dimensional space (left) stored as nodes in a 2-d tree (right).

2.2.3 Performance

The asymptotic running time for nearest neighbour queries of $\mathcal{O}(\log n)$ only holds for relatively small k, where the logarithmic behaviour is observed on a large number of training samples. Due to the sparsity of the data, for large k this effect becomes even more pronounced. For example, for $k = 16$ a search of a k-dimensional tree of 76,000 records examines almost every record, which means that (almost) no pruning is happening and that the tree has lost its purpose [21]. To put this into perspective, the data set in this paper has a dimensionality of $k = 256$ with only 1,707 training images. As a result, using the k-d tree still requires that the distance to all training images needs to be calculated to find the most similar one, while incurring the overhead of precomputing the data structure. This can be seen in the results in Table 2. Since the k-d tree nearest neighbours queries take as much time as the linear search, the estimated data structure overhead is as much as 25%.

Table 2: Results of kNN on training data set. The time includes preprocessing and the querying of all test images.

k	1	2	3	4	5
Accuracy	0.92	0.90	0.91	0.90	0.90
Time Linear (s)	0.34	0.30	0.31	0.29	0.30
Time k-d tree (s)	0.59	0.36	0.34	0.36	0.37

Furthermore, unexpectedly the accuracy only decreases as k increases. This can be explained as a combination of sparse data and no labelling errors in the training set. A consequence of the sparsity of the data is that images with a bigger distance to the unknown image are included and exercise influence on the classification of that image. Paraphrased, this means that more and more irrelevant images are used to classify an image, hence leading to less accurate results.

Note that the times are not comparable with Table 1. This is related to the fact that the algorithm was run on an implementation which is specifically crafted to only be influenced by the time the distance calculation takes without any other optimizations such as to have a level playing field.

2.3 Approximate methods

Storing the data in k-d trees did not prove successful in bringing down the running time for applications with a time constraint. For these kinds of applications, derivatives of the k-nearest neighbour method have been developed under the name approximate nearest neighbour.

This class of nearest neighbour techniques exploits redundancy to reduce the computational cost. The training data set in this paper has 1,707 entries, which contains 252 images representing the number one. There is guaranteed to be a lot of overlap between these images. The name contains "approximate", because reducing overlap almost always comes in some form of data simplification where some information will be lost.

The term approximate nearest neighbours is a hypernym, meaning that there are many algorithms falling under the name such as locality sensitive hashing, clustering, etcetera [15], [22]. Ultimately, approximate nearest neighbour only yields improvement in time complexity and memory use, while sacrificing some accuracy.

2.3.1 Nearest Mean

One way to reduce redundancy and overlap is by grouping together duplicate images. All metrics have the properties and results for vectors such that $\mathbf{x} = \mathbf{y}$:

$$\begin{aligned} \mathbf{x} = \mathbf{y} &\implies d(\mathbf{x}, \mathbf{y}) = 0 && \text{See [7], [10]} \\ d(\mathbf{x}, \mathbf{z}) &\leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) = d(\mathbf{y}, \mathbf{z}) \\ d(\mathbf{y}, \mathbf{z}) &\leq d(\mathbf{y}, \mathbf{x}) + d(\mathbf{x}, \mathbf{z}) = d(\mathbf{x}, \mathbf{z}) \\ \implies d(\mathbf{x}, \mathbf{z}) &= d(\mathbf{y}, \mathbf{z}) \end{aligned}$$

This proves that the distance from an arbitrary image to two identical training images will be the same. This means that duplicates in the training data set can be removed without affecting the accuracy of the algorithm.

The sparsity of the data implies that the amount of duplicates in the training set will be very limited, hence this will barely improve the speed. If more performance is still desired, images with very small differences can then be merged. The threshold for clustering two images has to be experimented with to find the optimal value. However, this is a trade-off, since a lower threshold will yield better accuracy, yet a lower performance and vice versa. Setting the threshold to 0 will give the same results as the "standard" nearest neighbours, while setting no bound for the threshold will result in merging everything and yield the nearest mean.

With no bound, all images of the same class are merged into a vector. This vector will simply be the average image for each class. On the left side in Figure 11 a normal 3 can be seen beside the "average" 3. Here, the loss of information is also clearly illustrated.

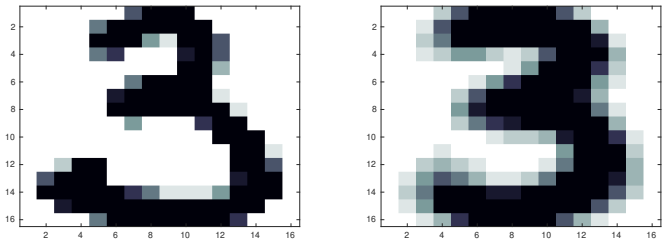


Figure 11: A 3 occurring in the training set and the average 3 vector.

As stated before, the computation of all of the mean images takes $\mathcal{O}(t_r)$ time, with t_r the number of training images. The main advantage of this approach is speed, since each test image has to be checked against only 10 other images. As a result, a single classification has a time complexity of $\mathcal{O}(1)$ for an image of a fixed size. For t_e number of classifications the total asymptotic computational complexity is $\mathcal{O}(t_e + t_r)$.

The results, as seen in Table 3, portray a significant speed advantage over the results seen in Table 1. An important thing to note is that each class now only has one representative. As a result, the linear approximations of the manifolds cannot overlap with images already present in the training set. This in turn leads to the Tangent distance yielding a much bigger improvement over the Euclidean distance, while having relatively the same performance penalty.

Table 3: Results of Nearest Mean.

Distance	1-norm	Euclidean	Tangent
Accuracy	0.71	0.81	0.88
Time (s)	0.04	0.03	0.34

Note that the Tangent distance in these tests is implemented in C compiled with Mex, so the time values of the Tangent distance can only be meaningfully compared with the same implementations in this paper.

In this section the distance from a new image to (a subset of) all images in the training set was used to identify images. The images associated with the same number span a subspace of $\mathbb{R}^{256}$. In the next section the distance to this subspace will be used to classify digits instead.

3 Singular Value Decomposition (SVD)

The approach using SVD is similar to, and could even be viewed as an implementation of, the approximate nearest neighbour approach. The application of kNN to the task of classifying images is known to suffer from various phenomena that arise when analysing high-dimensional spaces. Problems that we have seen until now that result from this are the severely reduced effectiveness of the k-d tree and the fact that increasing k reduces accuracy. These various phenomena are known as the curse of dimensionality [23].

Dimensionality reduction is the process of reducing the number of random variables under consideration [24]. To retain as much information as possible, the data is transformed from the high-dimensional space to a space of fewer dimensions, in a process known as feature extraction. In this paper, the singular value decomposition (SVD) is used. This technique is also known as the principal component analysis (PCA) in the field of statistics.

SVD is regularly researched in the context classifying images of faces and handwritten digits [2], [4]. In this section the SVD and particularly the left-singular vectors are used to classify handwritten digits.

3.1 Training

In this subsection, several concepts of subspaces and the SVD are explained together with their role in the context of image classification. As mentioned before, it is easier to work with images in the form of a vector as compared to a matrix or grid and therefore, images are flattened first. Next, a matrix $\mathbf{A}_n$ is created for each of the digits, such that the columns consist of all column vectors representing the number n. Suppose a_i with $i = 1, 2, 3, \ldots, k$ are the training vectors representing the number n. The matrix has the following structure:

$$\mathbf{A}_n = \begin{pmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_k \\ | & | & & | \end{pmatrix}.$$

For example, the columns of the matrix $\mathbf{A}_3$ consists of all vectors which are labelled as the number three.

The column space of a matrix $\mathbf{A}$ is a space containing all linear combinations of its column vectors. The column space of $\mathbf{A}$ is also called the image or range, $\mathcal{R}(\mathbf{A})$, of $\mathbf{A}$. The set of all possible linear combinations is also called the (linear) span or linear hull.

The column space of $\mathbf{A}_n$ is now a space containing all linear combinations of the images that represent the number n. Each number has a subspace of the pixel space associated with it. This can already be used for classification. All that needs to be done is to calculate this distance between the test image and all linear subspaces. The number associated with the subspace that is closest to the test image is chosen as the label of the test image.

Each linear subspace is spanned by a lot of vectors. This results in a very expensive distance computation and a lot of overlap of the different subspaces. An image that is in both subspaces will have a distance of zero to both, so overlapping subspaces are guaranteed to result in worse accuracy.

This way of classifying is thus both very slow and inaccurate. To improve this, the SVD will be used to find vectors that span smaller subspaces.

Theorem. (SVD) *Any matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, with $m \geq n$ can be factorised*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ orthogonal, and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is rectangular diagonal,

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n),$$

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

The values on the diagonal of the matrix $\mathbf{\Sigma}$ are known as the singular values, while the columns of $\mathbf{U}$ and $\mathbf{V}$ are called the left-singular and right-singular vectors respectively [2], [25].

The derivation below shows that the left-singular vectors are the set of eigenvectors of $\mathbf{A}\mathbf{A}^T$. These are thus obtained by solving the eigenvalue problem $\mathbf{A}\mathbf{A}^T\mathbf{x} = \sigma\mathbf{x}$.

$$\begin{aligned}
\mathbf{A} &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\
\mathbf{A}^T &= \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\
\Rightarrow \mathbf{A}\mathbf{A}^T &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\
\Rightarrow \mathbf{A}\mathbf{A}^T &= \mathbf{U}\mathbf{\Sigma}(\mathbf{V}^T\mathbf{V})\mathbf{\Sigma}^T\mathbf{U}^T \\
\Rightarrow \mathbf{A}\mathbf{A}^T &= \mathbf{U}(\mathbf{\Sigma}\mathbf{\Sigma}^T)\mathbf{U}^T
\end{aligned}$$

$\mathbf{\Sigma}\mathbf{\Sigma}^T$ is a real symmetric matrix, so the eigenvectors can be chosen to be orthogonal. The remainder of this section assumes that the eigenvectors are orthonormal, which is obtained by dividing the eigenvectors by their norm.

We define p as the rank of $\mathbf{A}$, then we have the following:

$$\begin{aligned}
\mathcal{R}(\mathbf{A}) &= \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\} \\
&= \{\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\} \\
&= \{\mathbf{U}\mathbf{\Sigma}\mathbf{y} : \mathbf{y} \in \mathbb{R}^n\} \\
&= \{\sum_{j=1}^{n} \mathbf{u}_j c_j : c_j \in \mathbb{R}\} \\
&= \mathcal{R}(\begin{pmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_p \\ | & | & & | \end{pmatrix}).
\end{aligned}$$

This means that the first n columns of $\mathbf{U}$ form an orthonormal basis for $\mathcal{R}(\mathbf{A})$, that is the first k columns of $\mathbf{U}$ are orthogonal and they are all unit vectors [4]. In Section 3.2.2 the importance of orthonormality will be explained. To clarify, the first k columns of $\mathbf{U}_n$ form the orthonormal basis, so from this point onwards $\mathbf{U}_n$ refers to the first p columns of $\mathbf{U}_n$.

The fact that the singular values are in decreasing order means that the corresponding left-singular vectors are in decreasing order of importance. Intuitively, the first k columns of $\mathbf{U}$ span a space using the k most important orthogonal directions. The first three columns of $\mathbf{U}_3$ are visible in Figures 12, 13 and 14.

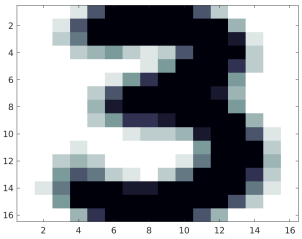


Figure 12: Column 1 of $\mathbf{U}_3$.

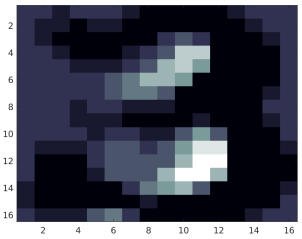


Figure 13: Column 2 of $\mathbf{U}_3$.

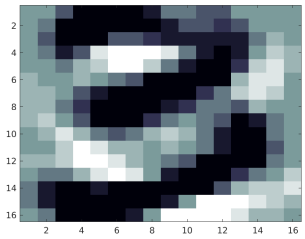


Figure 14: Column 3 of $\mathbf{U}_3$.

In this section several concepts were explained and how they are useful in the context of image classification. In the next subsection these concepts, and the SVD in particular, together with some mathematical operations will be used to identify the images from the test set.

3.2 Classification

In this section the classification procedure using these ideas is explained in more detail and more information is given with regard to finding the distance to a linear subspace.

3.2.1 Subspace spanned by training vectors

The distance between an image and a subspace cannot be computed directly. To do this, the distance between an image and its orthogonal projection on the subspace needs to be calculated instead.

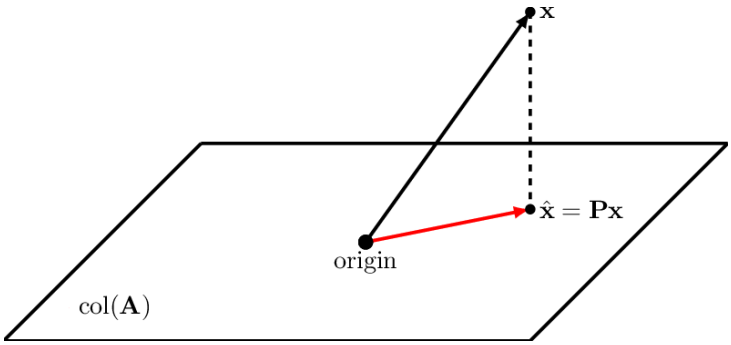


Figure 15: Orthogonal projection on the column space of $\mathbf{A}$, or col$(\mathbf{A})$ [26].

Suppose that $\mathbf{x}$ is the vector that needs to be classified and that $\hat{\mathbf{x}} = \mathbf{P}_n\mathbf{x}$ is the orthogonal projection of $\mathbf{x}$ on the subspace representing the number n. The distance between $\mathbf{x}$ and $\hat{\mathbf{x}}$ is (see Figure 15)

$$\|\mathbf{x} - P\mathbf{x}\|_2 = \left\|\mathbf{x} - \mathbf{A}_n(\mathbf{A}_n^T\mathbf{A}_n)^{-1}\mathbf{A}_n^T\mathbf{x}\right\|_2.$$

For each new vector the distance to each subspace is calculated. Suppose that the closest subspace is $\mathcal{R}(\mathbf{A}_n)$, which would mean that $\mathbf{x}$ is classified as the number n. Notice that the column space of $\mathbf{A}_n$ must be spanned by linearly independent vectors for the inverse of $\mathbf{A}_n^T\mathbf{A}_n$ to exist.

Intuitively, the linear projection of a vector is in fact the best approximation of the vector using a linear combination of the basis vectors. An image representing the number three (Figure 16) and number nine (Figure 18) can be found below. To see how this projection affects the image, the orthogonal projections on the column space of $\mathbf{A}_3$ of those numbers can be found besides them in Figures 17 and 19 respectively. Note that both Figure 16 and 18 are not from the training set, since a projection of a training image representing the number n on the range of $\mathbf{A}_n$ is just the vector itself.

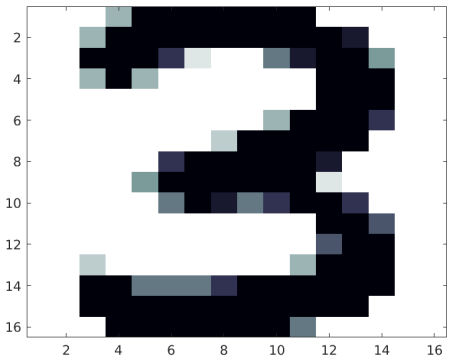


Figure 16: Image representing a 3 from the test set.

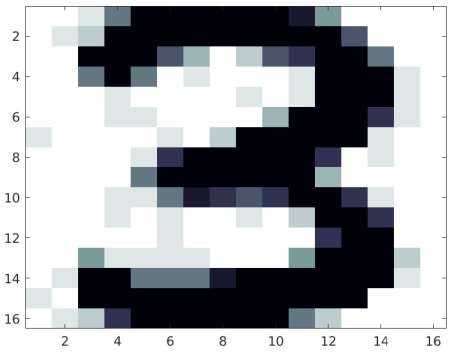


Figure 17: Orthogonal projection of Figure 16 on the range of $\mathbf{A}_3$.

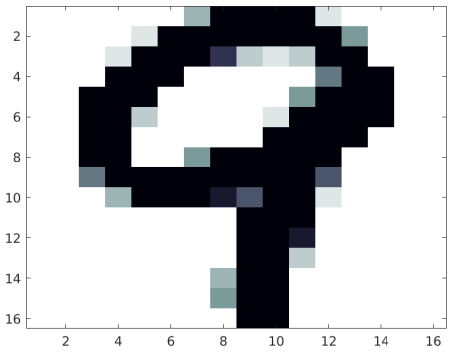


Figure 18: Image representing a 9 from the test set.

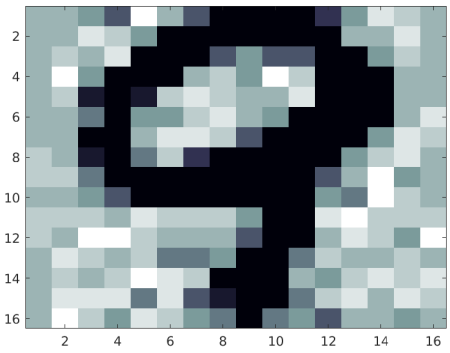


Figure 19: Orthogonal projection of Figure 18 on the range of $\mathbf{A}_3$.

Recall that the column space of $\mathbf{A}_3$ contains all linear combinations of the training images representing a three. We expect there to be a linear combination closely matching the number three, while no such combination for the number nine exists. Figures 17 and 19 confirm this expectation as the projection of an image representing the number three on $\mathcal{R}(\mathbf{A}_3)$ yields a very similar image to the original, while projecting a nine on this space introduces a lot of noise.

3.2.2 Using the SVD

The first step to classify digits using the SVD is to create the same matrices $\mathbf{A}_n$ for each digit n as created before. We define k_n as $\text{rank}(\mathbf{A}_n)$, avoiding having to write $\text{rank}(\mathbf{A}_n)$ everywhere. As mentioned in Section 3.1, the singular value decomposition of each $\mathbf{A}_n$ results in the matrix $\mathbf{U}_n$, which is a $256 \times k_n$ matrix whose columns span $\mathcal{R}(\mathbf{A}_n)$. Hence, the columns of $\mathbf{U}_n$ are $u_1, \ldots, u_{k_n}$. So $\mathbf{U}_n^T$ and $\mathbf{U}_n$ are:

$$\mathbf{U}_n^T = \begin{pmatrix} — & u_1^T & — \\ — & u_2^T & — \\ & \vdots & \\ — & u_{k_n}^T & — \end{pmatrix}, \qquad \mathbf{U}_n = \begin{pmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_{k_n} \\ | & | & & | \end{pmatrix}.$$

As mentioned before, the column vectors of $\mathbf{U}_n$ are all orthogonal and unit vectors. This means that $u_i \cdot u_i = 1$ and $u_i \cdot u_j = 0$ for all $i \neq j$, where $\cdot$ denotes the dot product. The matrix multiplication of $\mathbf{U}_n^T$ and $\mathbf{U}_n$ then yields:

$$\begin{aligned} \mathbf{U}_n^T \mathbf{U}_n &= \begin{pmatrix} u_1 \cdot u_1 & u_1 \cdot u_2 & \cdots & u_1 \cdot u_{k_n} \\ u_2 \cdot u_1 & u_2 \cdot u_2 & \cdots & u_2 \cdot u_{k_n} \\ \vdots & & \ddots & \vdots \\ u_{k_n} \cdot u_1 & u_{k_n} \cdot u_2 & \cdots & u_{k_n} \cdot u_{k_n} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \\ &= I \end{aligned}$$

Substituting this in the formula for the distance between a vector and its orthogonal projection yields:

$$\begin{aligned} \|\mathbf{x} - P\mathbf{x}\|_2 &= \left\|\mathbf{x} - \mathbf{U}_n (\mathbf{U}_n^T \mathbf{U}_n)^{-1} \mathbf{U}_n^T \mathbf{x}\right\| \\ &= \left\|\mathbf{x} - \mathbf{U}_n I^{-1} \mathbf{U}_n^T \mathbf{x}\right\|_2 \\ &= \left\|\mathbf{x} - \mathbf{U}_n \mathbf{U}_n^T \mathbf{x}\right\|_2 \end{aligned}$$

Therefore, the distance between a vector and a subspace spanned by the columns of $\mathbf{U}_n$ is $\left\|\mathbf{x} - \mathbf{U}_n \mathbf{U}_n^T \mathbf{x}\right\|_2$.

In Table 4, the subspace corresponding to each number is stated with the amount of basis vectors and the amount of column vectors of $\mathbf{A}_n$. Note that the training set for some numbers has an equal amount of basis vectors compared to orthonormal vectors. This means that every vector in $\mathbf{A}_n$ was already linearly independent.

Table 4: Number of basis vectors used to span the different subspaces.

n	0	1	2	3	4	5	6	7	8	9
k_n	245	100	202	131	122	88	151	166	144	132
number of columns $\mathbf{A}_n$	319	252	202	131	122	88	151	166	144	132

3.3 Performance

The difference in speed between using linearly independent columns of $\mathbf{A}_n$ and the orthonormal columns of $\mathbf{U}_n$ consists of two main parts. First, the basis columns of $\mathbf{A}_n$ must be chosen. To do this, each column vector needs to be checked whether or not they are a linear combination of the previous columns. This is an expensive computation when compared to computing the SVD. The second part responsible for the difference is the distance function. In Section 3.2.2 we saw that the orthonormality condition resulted in a simplified calculation. In Table 5 the difference between the running time of both distances can be seen.

At this point, all the columns of $\mathbf{U}_n$ are used to span the space on which the vectors are projected. As mentioned before, the singular vectors are ordered from important to less important. The first r vectors of $\mathbf{U}_n$ span a subspace of $\mathcal{R}(\mathbf{A})$ that contains the most important features for the number n. A feature then is a certain combination of pixel values. The amount of orthogonal vectors r to be taken as our basis is yet to be determined.

The choice for r must be independent from the test set, so different amounts of vectors are tested on the training set and their results are plotted in Figure 20. A projection onto a subspace spanned by more elements naturally has a higher computational cost. Figure 21 shows a clear upwards trend with respect to the time the algorithm takes. The slight peak in Figure 21 can be explained by caching optimisations between subsequent runs and would disappear when repeating this test multiple times.

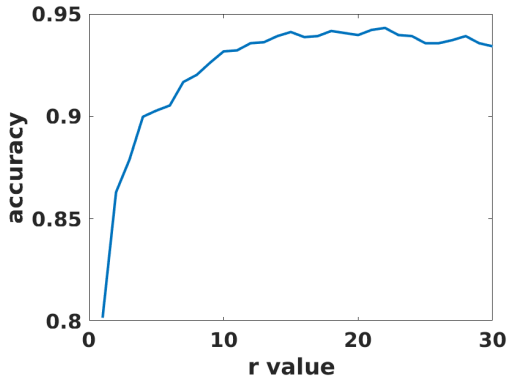


Figure 20: Accuracy plotted against r values.

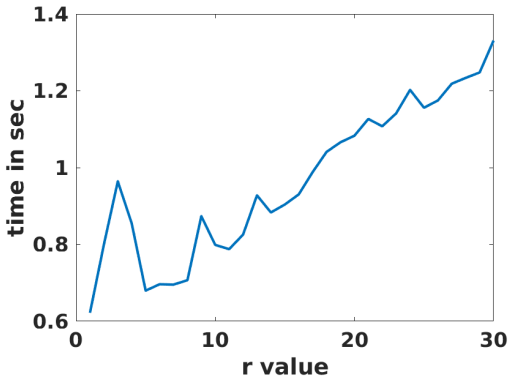


Figure 21: Time plotted against r values.

For a better overview, some data values at different intervals can be found in Table 5. The time difference between the simplified distance based on the orthonormality condition and the standard linear independence can also be seen here. The graph in Figure 20 indicates that $r = 15$ strikes a good balance between speed and accuracy.

Table 5: Results of SVD classifier.

r **value**	1	2	5	10	15	20	25
Accuracy	0.80	0.86	0.90	0.93	0.94	0.94	0.94
Time orthonormal (s)	0.62	0.80	0.68	0.80	0.90	1.08	1.16
Time linear independent (s)	1.17	1.57	1.86	2.19	2.58	3.23	3.95

The last two sections, kNN and SVD, tried to classify digits by computing the distance between a new image and other objects to determine the label. Another entirely different way of doing this is by constructing a model based on the training images. In the next section a neural network will be used to label the image.

4 Neural Network (NN)

Another popular way of solving the problem is through the use of Artificial Intelligence (AI) [3]. AI is, at the time of writing, a hot topic with Artificial Neural Networks (ANN) receiving special attention. In this section, a simple feedforward neural net will be created to attempt the classification of the digit from the data set. The neural net created by Higham will be taken as a basis [27]. The XOR problem based on the logical exclusive-or operator is a test problem that will be solved to showcase the different elements of a neural network [28].

In Section 4.1 the structure of a neural network is explained. Section 4.2 contains information about forward propagation, followed by the loss function in Section 4.3. The way a neural network is trained can be found in Section 4.4. The choices for the example problem are motivated in Section 4.5. Section 4.6 analyses the results of the neural network that solves the XOR problem. Finally, the discussed concepts are applied to the classification problem in Section 4.7.

4.1 Structure

There are many different complex structures such as recurrent or convolutional neural networks, however for simplicity we will focus on a feedforward neural network. A feedforward neural network has two main components: nodes, often called neurons, and edges. A node can have a value which is dependent on the values of nodes connected to it, the weights of the edges, and optionally the bias. Typically, these artificial neurons are aggregated into layers (see Figure 22).

The first layer is called the input layer. This layer contains the input neurons, which are the neurons with a direct correspondence to the input. The last layer is called the output layer. The calculated values in the output neurons are the final result and thus output of the neural network. All other layers are called hidden layers as their functionality is not exposed to the user. In the example, illustrated by Figure 22, there is only one hidden layer.

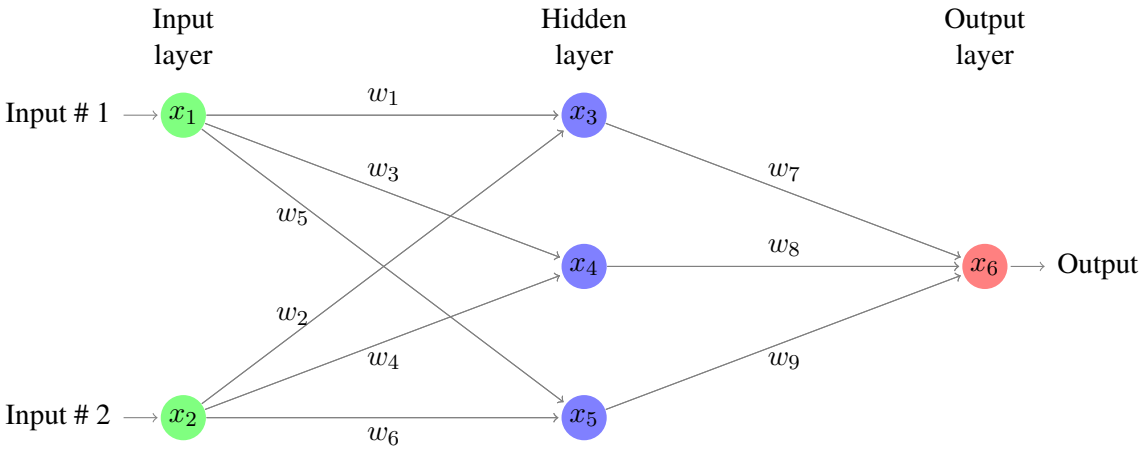


Figure 22: Neural network.

The sizes of the input and output layers are often largely dictated by the problem that you are trying to solve. Generally, deciding on the number of hidden layers and the number of nodes within each

layer is not an exact science. Rules of thumb have been suggested, however there is no widely accepted technique [27].

One property of a feedforward neural network is that edges can only be connected between two consecutive layers. Consequently, the value of all nodes in the first layer need to be calculated, then the values in the nodes of the second layer and so on. The process of calculating the nodes is aptly named forward propagation.

4.2 Forward propagation

First, forward propagation is illustrated on the XOR example. Then, a more general formula is derived and some notation is introduced.

The input of a neural network consists of values. The forward propagation of the example, seen in Figure 22, goes as follows:

$$\begin{aligned} x_1 &= \text{given by input}, \quad x_3 = f(w_1 \cdot x_1 + w_2 \cdot x_2 + b_3), \quad x_6 = f(w_7 \cdot x_3 + w_8 \cdot x_4 + w_9 \cdot x_5 + b_6), \\ x_2 &= \text{given by input}, \quad x_4 = f(w_3 \cdot x_1 + w_4 \cdot x_2 + b_4), \\ & \qquad\qquad\quad x_5 = f(w_5 \cdot x_1 + w_6 \cdot x_2 + b_5), \end{aligned}$$

Notice that these equations can easily be written in matrix form. These matrices will be referred to as the weight matrices. This yields the following:

$$\begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = f\left(\begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \\ w_5 & w_6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_3 \\ b_4 \\ b_5 \end{pmatrix} \right), \qquad x_6 = f\left(\begin{pmatrix} w_7 & w_8 & w_9 \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} + b_6 \right).$$

A simple substitution yields the following expression for computing the output in a single step:

$$x_6 = f\left(\begin{pmatrix} w_7 & w_8 & w_9 \end{pmatrix} f\left(\begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \\ w_5 & w_6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_3 \\ b_4 \\ b_5 \end{pmatrix} \right) + b_6 \right).$$

The function f is called the activation function. Popular choices for this function are the linear function $f(x) = x$, the Rectified linear unit (ReLU) function $f(x) = x \cdot \mathbb{1}\{x > 0\}$, and the sigmoid function $f(x) = \sigma(x) = 1/(1 + e^{-x})$ [27]. Furthermore, the XOR problem has outputs that can be divided into two classes, true and false, making it a classification problem. We will take the sigmoid function as the activation function as it is typically chosen in case of classification problems.

For use further in this paper, a general formula is useful. A general formula is thus derived and some notation is introduced. Let ℓ be a number corresponding to a layer, let $\mathbf{W}^{[\ell]}$ be the weight matrix of layer from layer $\ell - 1$ to ℓ, and let $\mathbf{b}^{[\ell]}$ be the biases of layer ℓ. We define the weighted input of layer $2 \leq \ell \leq L$ before and after the activation function to be

$$\begin{aligned} \mathbf{z}^{[\ell]} &= \mathbf{W}^{[\ell]} \mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]}, \\ \mathbf{a}^{[\ell]} &= f\left(\mathbf{z}^{[\ell]} \right), \end{aligned}$$

where L is the number of layers of the neural net and where $a^{[1]}$ corresponds to the input.

The last, and arguably the most important, thing that is needed is a way to improve the weights and biases such that the network as a whole behaves as desired. Changing the weights and biases for the purpose of improving performance is called training a neural network. Before a network can be trained, a loss function needs to be computed.

4.3 Loss function

The loss function, also known as the cost function, is necessary to gauge the performance of the neural network. A loss function takes the predicted output, $x_p \in \{0, 1, \ldots, 9\}$, and the actual output, x_6 as inputs and gives a measure of difference between the two. In this paper the Mean Squared Error (MSE) is used, where n is the number of training samples. The MSE is differentiable and has some other convenient properties, which made it very popular for feedforward neural networks such as this one. The formula is displayed below.

$$C = \frac{1}{n} \sum_{i=1}^{n} (x_6 - x_p)^2.$$

4.4 Training the network

Training the network is the act of improving the weights and biases in such a way that the neural network behaves as desired. The two ways discussed in this paper of improving these are through neuroevolution and back propagation. Before the weights can be improved, initial weights need to be chosen. There are several ways to pick these, however in this paper for the sake of simplicity they are initialised to a random number. This can be any random number, however since very big or small values make the process slower we take random numbers from a normal distribution.

4.4.1 Neuroevolution

Neuroevolution (NE) is the artificial evolution of neural networks using Genetic Algorithms (GA). GA are inspired by the process of natural selection. Neuroevolution starts with creating several neural networks initialised with random weights and biases. Then the models are tested on the training set and given a fitness score based on how well they performed. The fittest neural networks have a chance to create so-called offspring, which is a new neural network with a combination of the parents weights and biases. For each of the fittest there is also a chance to mutate, which randomly changes the weights by some small amount [29].

There are also more advanced NE algorithms such as the Neuroevolution with Augmenting Topology (NEAT). Mutations with NEAT also include the possibility to change the topology or structure of the network [30].

4.4.2 Gradient descent

A popular way of improving an NN is through gradient descent with back propagation. In Section 4.3, the loss function has been defined, which consists only out of the weights and inputs x_1, x_2, and x_3. That makes this an optimisation problem, where the loss function needs to be minimised. The calculation of the gradient requires the loss function to be differentiable and hence also continuous. Even though there

are solutions to solve this, however in this case this would add unnecessary complexity. Differentiable activation functions are thus easier to work with and hence are generally preferred.

To make use of back propagation, first define the error in the jth neuron to be

$$\delta_j^{[\ell]} = \frac{\partial C}{\partial z_j^{[\ell]}}, \qquad \text{for } 2 \leq j \leq n \text{ and } 1 \leq j \leq n_\ell$$

where n_ℓ is the number of neurons in layer ℓ. We point out that the usage of the term error is somewhat ambiguous. The idea of referring to $\delta_j^{[\ell]}$ as an error seems to have arisen because the cost function can only be minimum if all partial derivatives are zero, so $\delta_j^{[\ell]} = 0$ is a useful goal.

Substituting the various formulas together with the chain rule, which can be found in the paper of Higham et al [27], yields the following:

Lemma [27]. *We have*

$$\delta^{[L]} = \sigma'(\mathbf{z}^{[L]}) \circ (\mathbf{a}^L - \mathbf{x}_1), \tag{1}$$

$$\delta^{[\ell]} = \sigma'(\mathbf{z}^{[\ell]}) \circ (\mathbf{W}^{[\ell+1]})^T \delta^{[\ell+1]}, \qquad \text{for } 2 \leq \ell \leq L-1, \tag{2}$$

$$\frac{\partial C}{\partial w_{jk}^{[\ell]}} = \delta_j^{[\ell]} a_k^{[\ell-1]}, \qquad \text{for } 2 \leq \ell \leq L, \tag{3}$$

$$\frac{\partial C}{\partial b_j^{[\ell]}} = \delta_j^{[\ell]}, \qquad \text{for } 2 \leq \ell \leq L. \tag{4}$$

Here, $\circ$ also denotes the Hadamard or entrywise product. Recall that in Section 4.2 the input is propagated forward. This results in an output $\mathbf{a}^L$, which corresponds to x_6 in the XOR example. The error of the output $\boldsymbol{\delta}^{[L]}$ is computed. Now, in a process called back propagation, the errors $\boldsymbol{\delta}^{[p]}$ of the layer p with $p = L-1, \ldots, 2$ are calculated. The error for layer 1 is not calculated as the input is given and cannot be changed with adjusting weights.

At this point a vector $\boldsymbol{\delta}^{[\ell]}$ exists for each layer. The last step to perform is to update the weights and biases using this vector and a single gradient descent step.

$$\begin{aligned}
\mathbf{W}_{new}^{[\ell]} &= \mathbf{W}^{[\ell]} - \eta \frac{\partial C}{\partial \mathbf{W}}^{[\ell]} \\
&= \mathbf{W}^{[\ell]} - \eta \boldsymbol{\delta}^{[\ell]} \mathbf{a}^{[\ell-1]}, && \text{see (3),} \\
\mathbf{b}_{new}^{[\ell]} &= \mathbf{b}^{[\ell]} - \eta \frac{\partial C}{\partial \mathbf{W}}^{[\ell]} \\
&= \mathbf{b}^{[\ell]} - \eta \boldsymbol{\delta}^{[\ell]}, && \text{see (4).}
\end{aligned}$$

The symbol η is the learning rate or step size. This learning rate influences how quickly the weights of the neural network are changed. Generally, a lower learning rate results in slower training, however a larger learning rate will result in a change of weight values that is too large which means that the local minimum may never be obtained. This is how gradient descent with the use of back propagation is applied to a neural network.

4.5 XOR problem

XOR stands for exclusive or, which is a logical operation that evaluates to true only when one of the inputs differ from each other [31]. The XOR is a logical operation, however the neural network is numerical, so an encoding must be chosen. In this paper, true will be encoded with the number 1 and false with the number 0. This yields the following inputs and corresponding outputs:

$$\text{xor}\left(\begin{pmatrix}0\\0\end{pmatrix}\right) = 0, \qquad \text{xor}\left(\begin{pmatrix}1\\0\end{pmatrix}\right) = 1, \qquad \text{xor}\left(\begin{pmatrix}0\\1\end{pmatrix}\right) = 1, \qquad \text{xor}\left(\begin{pmatrix}1\\1\end{pmatrix}\right) = 0,$$

This example is chosen since the solution space is not linearly separable. This means that there does not exist a single line that can divide the different classification outputs. As a result, the problem cannot be solved by methods such as the single layer perceptron, hence making for a simple but non-trivial example [28]. The training set will then be the inputs with their corresponding labels. Normally, a proper subset of the input space is trained on, however in this case there are already such few solutions that all of the different input combinations are trained on.

In Figure 23 the reader can find the inputs with their corresponding labels. A blue cross means true, whereas a red circle means false. For example, the exclusive-or operation on false and false yields false, hence a circle on $(0, 0)$

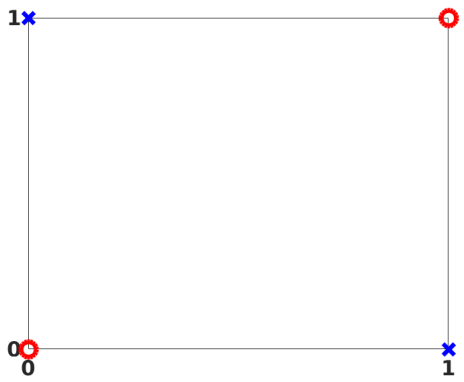


Figure 23: Output space of the encoded XOR operation.

4.6 Performance

At this point all the necessary parts of a neural network are described. The model is now trained, however we do not know when to stop training. To do this the performance needs to be measured. Based on those measurements it can be decided whether or not do another forward propagation followed by a back propagation step.

The training is done by simply cycling through the four possible inputs and training them. This guarantees that all elements from the input space are equally important to the final neural network. After adjusting the weights for a training input, the neural network is tested on a similar testing data set which it has not seen before. The values of the loss function can then be plotted to see how the neural network is improving. In Figure 24 this plot can be seen.

The test set used to calculate the loss in the plot is the same as the training set, so in this case the graph cannot be used to infer any useful information. Figure 25 visually represents the solution space, where a point in the grey areas is classified as a one, otherwise zero. Since the points from the same class fall in the same coloured area, the conclusion can be made that the neural network behaves as desired.

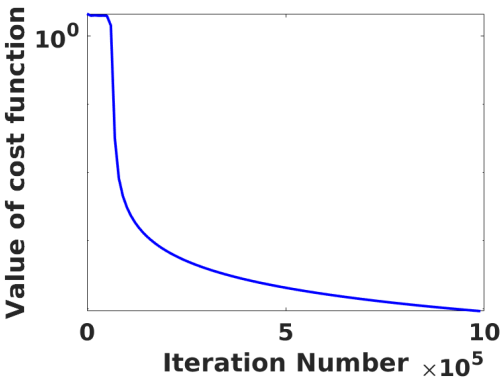


Figure 24: Cost, or loss, function of the neural network over the number of iterations.

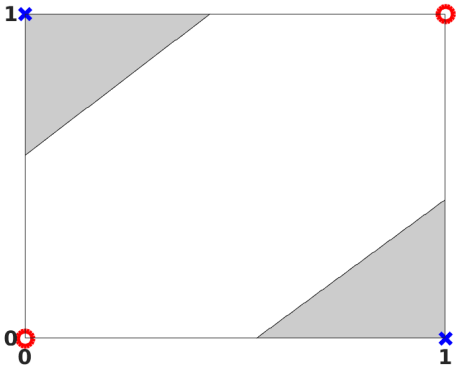


Figure 25: Solution space of the neural net. All inputs in the grey areas will be classified as true, otherwise false.

4.6.1 Overfitting

One notorious problem neural networks can have is overfitting. Overfitting produces a model that corresponds too closely to the training data set, and therefore fails to capture the desired behaviour [32]. In case of the example, through Figure 25, the model is not overfitted. This visualisation can only be made because the input dimensionality is two, so for higher dimensional inputs such as the image classification other methods need to be employed.

To deal with this when classifying digits, both the loss of the training set as well as the loss of the testing set is graphed. Overfitting then can be recognised if the loss of the training set continues to decrease while the loss of the test set does not [27].

4.7 Digit recognition

The basic concepts of a neural network are explained through the use of the XOR example. To apply this to the problem of digit recognition, all important differences are highlighted and the necessary changes described.

Structure: The example has two nodes as inputs, while an image is a vector $\mathbf{x} \in \mathbb{R}^{256}$, so the input layer now has 256 nodes. The output is now one of 10 classes from 0 to 9. Hence, the output is a vector $\mathbf{y} \in \mathbb{R}^{10}$, where the first element corresponds to the confidence 0 is the correct answer, the second corresponds to 1 and so on [27]. As for the hidden layers, a hidden layer with 64 then a hidden layer with 32 neurons is chosen. This structure is an estimated guess as there are no clear rules for the structure.

Performance: As mentioned in Section 4.6, the performance in the form of time and accuracy will now be measured on a separate test set, while plotting the error of the training set to make sure the model is

not overfitted.

At this point the model is trained with the MSE of the data plotted in Figure 26 and the accuracy plotted in Figure 27. As can be seen in Figure 27, the MSE of the training data is nearly zero without overfitting. This difference suggests that more training data would increase the performance of the neural network as the training data has an accuracy near one, while there is no overfitting present.

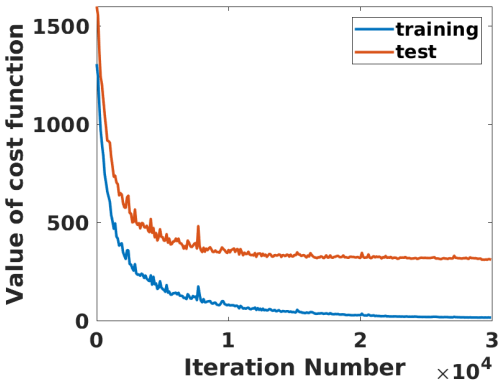


Figure 26: MSE of test and training data.

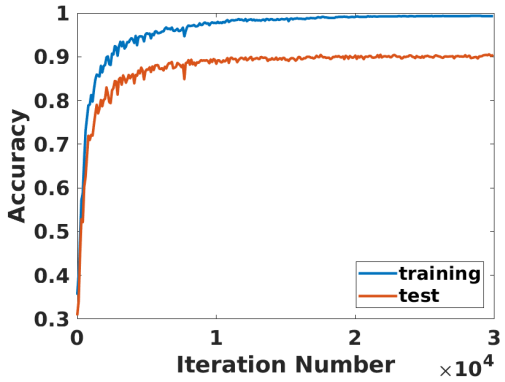


Figure 27: Accuracy of test and training data.

In Figure 27, the number of iterations is directly proportional to the amount of time used. In Table 6, a few points are sampled with the respective duration for that run. Concluding, the point where the variation is bigger than the improvement is around $2 \cdot 10^4$ number of iterations.

Table 6: Results of the neural network on the data set.

Accuracy	0.893	0.905	0.900
Number of iterations	$1 \cdot 10^4$	$3 \cdot 10^4$	$9 \cdot 10^4$
Time (s)	0.52	1.10	4.31

It is important to note that the neural network, just like the SVD, has a fixed classification time. As mentioned earlier, the model could be improved by adding a lot more training data as the current training data gets a classification accuracy of nearly one. These facts combined make for a model that after the initial training period can get very accurate while having a short classification time.

We have now seen all the different techniques and in the next section some conclusions will be drawn.

5 Conclusion

Two factors that are important are the accuracy and the speed with which the digits are recognised. Three distinct methods to recognise digits were tested and timed. These methods are the k-nearest neighbour algorithm, calculating distances to subspaces using the singular value decomposition, and neural networks.

The first technique, k-nearest neighbour, relies on the similarity between any image that needs to be classified and all images of the training set. Different measures of similarity, or metrics, such as the Euclidean distance and Tangent distance were explored. The Tangent distance is more accurate than the other distances, however this came at a significant cost. kNN is inherently computationally expensive, because it requires the computation of the distance between the image that needs to be classified and all the training images. The k-d tree was explored in an attempt to lower the computational cost by providing the possibility to skip certain distance calculations. This was unsuccessful due to the sparsity of the data. Approximate nearest neighbour in the form of the nearest mean did provide the increase in speed that was sought after, however this came with a relatively low accuracy.

For the second technique, SVD, the distance to the space consisting of the linear combinations of the training images per digit was computed instead of the distance to each point in the training set. This was even slower and less accurate than the standard nearest neighbour. With the use of the singular value decomposition the same subspace could now be spanned by orthonormal vectors. This gave similar accuracy, however this method was significantly faster as a result of some nice properties provided by the orthonormality of the vectors. Since the same space is now spanned by orthonormal vectors, the most important vectors could be used to span a subspace with the most important features of a particular number. This lead to a big improvement in both speed and accuracy.

The third and last technique, NN, makes use of neural networks. Neural networks is a typical example of the eager learning method. A neural network takes a long time to train, however once it is trained a single classification is extremely fast when compared to the other methods. As for the accuracy, the neural network performed comparable to kNN with the 1-norm. It was noted, however, that the neural network did not over-fit at any point and that more training images could improve this method without any extra computational complexity with regards to the classification.

We have thus seen three techniques: kNN, SVD, and NN. We argued about their time complexities where possible, and we have seen the strengths and weaknesses of each technique.

From our testing the SVD was the clear winner in our tests with the highest accuracy and fastest running time. In real use case scenarios, the neural network is more likely to be chosen as this application scales better with more training samples and more sophisticated structures.

References

[1] M. Anthonissen, Numerical Linear Algebra, Week 7: Recognizing digits written by hand.

[2] B. Savas and L. Eldén, Handwritten digit classification using higher order Singular Value Decomposition, *Pattern recognition*, vol. 40, no. 3, pp. 993–1003, 2007. DOI: 10.1016/j.patcog.2006.08.004.

[3] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, Deep, big, simple Neural Nets for handwritten digit recognition, *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010. DOI: 10.1162/NECO_a_00052.

[4] L. Cao, Singular Value Decomposition applied to digital image processing, *Division of computing studies, Arizona State University*, pp. 1–15, 2006.

[5] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009. DOI: 10.1007/978-0-387-84858-7. arXiv: arXiv:1011.1669v3.

[6] T. Cover and P. Hart, Nearest Neighbor pattern classification, *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967. DOI: 10.1109/TIT.1967.1053964.

[7] M. C. Bramwell and E. Kreyszig, Introductory Functional Analysis with Applications. John Wiley & Sons, 1978. DOI: 10.2307/3616033.

[8] J. Li, G. Chen, and Z. Chi, A fuzzy image metric with application to fractal coding, *IEEE transactions on image processing*, vol. 11, no. 6, pp. 636–643, 2002. DOI: 10.1109/tip.2002.1014995.

[9] D. Huttenlocher, W. Rucklidge, and G. Klanderman, Comparing images using the Hausdorff distance under translation, in *IEEE computer society conference on computer vision and pattern recognition*, 1992, pp. 654–656. DOI: 10.1109/cvpr.1992.223209.

[10] P. Simard, Y. LeCun, and J. Denker, Efficient pattern recognition using a new transformation distance, *Advances in neural information processing systems*, pp. 50–58, 1993. DOI: 10.1006/scdb.1998.0249.

[11] L. Wang, Y. Zhang, and J. Feng, On the Euclidean distance of images, *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1334–1339, 2005. DOI: 10.1109/tpami.2005.165.

[12] P. E. Black, Manhattan distance, 2019. Available: https://www.nist.gov/dads/HTML/manhattanDistance.html (visited on 06/05/2019).

[13] P. Y. Simard, Y. A. Le Cun, J. S. Denker, and B. Victorri, Transformation invariance in pattern recognition: Tangent distance and propagation, *International journal of imaging systems and technology*, vol. 11, no. 3, pp. 181–197, 2000. DOI: 10.1002/1098-1098(2000)11:3<181::AID-IMA1003>3.0.CO;2-E.

[14] D. Keysers, J. Dahmen, T. Theiner, and H. Ney, Experiments with an extended tangent distance, in *IEEE proceedings 15th international conference on pattern recognition*, 2000, pp. 38–42. DOI: 10.1109/icpr.2000.906014.

[15] N. Bhatia and Vandana, Survey of Nearest Neighbor techniques, *International journal of computer science and information security*, 2010. DOI: 10.1377/hlthaff.2014.1186. arXiv: 1007.0085.

[16] N. Pylypiw, Breaking ties in kNN classification, 2017. Available: https://www.linkedin.com/pulse/breaking-ties-k-nn-classification-nicholas-pylypiw (visited on 06/03/2019).

[17] J. E. S. Macleod, A. Luk, and D. M. Titterington, A re-examination of the distance-weighted k-Nearest Neighbor classification rule, *IEEE transactions on systems, man, and cybernetics*, vol. 17, no. 4, pp. 689–696, 1987. DOI: 10.1109/TSMC.1987.289362.

[18] S. Majewski, Dealing with ties, weights and voting in kNN, 2012. Available: https://stats.stackexchange.com/questions/45580/dealing-with-ties-weights-and-voting-in-knn (visited on 06/03/2019).

[19] J. L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975. DOI: 10.1145/361002.361007. arXiv: arXiv:1011.1669v3.

[20] R. A. Brown, Building a balanced k-d tree in O(kn log n) Time, *Journal of computer graphics techniques*, vol. 4, no. 1, pp. 50–68, 2015. arXiv: 1410.5420.

[21] R. F. Sproull, Refinements to Nearest-Neighbor searching in k-dimensional trees, *Algorithmica*, vol. 6, no. 1, pp. 579–589, 1991. DOI: 10.1007/BF01759061.

[22] A. Rajaraman and J. D. Ullman, Mining of Massive Datasets. Cambridge University Press, 2011. DOI: 10.1017/CBO9781139058452. arXiv: arXiv:1011.1669v3.

[23] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, When is "Nearest Neighbor" meaningful?, in *International conference on database theory*, Springer, Berlin, Heidelberg, 1999, pp. 217–235. DOI: 10.1007/3-540-49257-7_15.

[24] S. T. Roweis and L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000. DOI: 10.1126/science.290.5500.2323.

[25] L. Eldén, Matrix Methods in Data Mining and Pattern Recognition. SIAM, 2007. DOI: 10.1137/1.9780898718867.

[26] N. B. Erichson, S. Voronin, S. L. Brunton, and J. N. Kutz, Randomized matrix decompositions using R, 2016. arXiv: 1608.02148.

[27] C. F. Higham and D. J. Higham, Deep Learning: An introduction for applied mathematicians, Tech. Rep., 2018. DOI: DOI:10.1016/S0022-5096(98)00067-2. arXiv: 1801.05894.

[28] D. Shiffman, Nature of Code, 2019. Available: https://natureofcode.com/ (visited on 06/06/2019).

[29] D. Whitley, S. Dominic, R. Das, and C. W. Anderson, Genetic reinforcement learning for neurocontrol problems, *Machine learning*, vol. 13, no. 2, pp. 259–284, 1993. DOI: 10.1023/A:1022674030396.

[30] K. O. Stanley and R. Miikkulainen, Evolving Neural Networks through augmenting topologies, *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002. DOI: 10.1162/106365602320169811.

[31] R. E. Simpson, Introductory Electronics for Scientists and Engineers. Allyn & Bacon, 1974, pp. 400–402.

[32] D. M. Hawkins, The problem of overfitting, *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004. DOI: 10.1021/ci0342472.

6 Appendix

```
% Load the initial data.
% For this to work you need to have a file data.mat in which
% there are structs train and test containing images and labels
load data;

% get result from the chosen classifier
% svd_comp can be replaced by one of the below defined functions
% only the name has to be changed
tic
result = svd_comp(train, test, 15, false);
toc

% Finally, check the result against the test labels and display the
% accuracy
check(result, test)

function result = svd_comp(train, test, r, orthonormal)
    train.image = (train.image * 2) - 1;
    test.image = (test.image * 2) - 1;

    result = zeros(length(test.label), 1);

    A = cell(1, 10);
    U = cell(1, 10);
    S = cell(1, 10);
    V = cell(1, 10);

    % create a matrix for each digit
    for i = 1 : length(train.label)
        A{train.label(i) + 1} = [A{train.label(i) + 1}, ...
            train.image(:, i)];
    end

    % decompose the matrix for each digit
    for i = 1:length(A)
        [U{i}, S{i}, V{i}] = svd(A{i});
    end

    % find the minimum relative residual
    min_rel_res = zeros(1, length(A));
    for j = 1 : length(test.label)
        for i = 1:length(A)
            if (orthonormal)
                orth_proj = U{i}(:, 1:r) * ...
                    transpose(U{i}(:, 1:r)) * test.image(:, j);
            else
                orth_proj = U{i}(:, 1:r) * ...
                    inv(U{i}(:, 1:r)' * U{i}(:, 1:r)) * ...
                    U{i}(:, 1:r)' * test.image(:, j);
            end
```

```
            min_rel_res(i) = norm(test.image(:, j)- orth_proj);
        end

        % store index of the smallest norm
        [~, i] = min(min_rel_res);
        result(j) = i - 1;
    end
end

function result = neural_net_simple(train, test)
    result = zeros(length(test.label), 1);

    % One hot vector
    y = train.label == (0:9)';

    % Initialize weights and biases
    rng(5000);
    W2 = 0.5 * randn(length(train.image(:, 1)), 64)';
    b2 = 0.5 * randn(64, 1);
    W3 = 0.5 * randn(64, 32)';
    b3 = 0.5 * randn(32, 1);
    W4 = 0.5 * randn(32, 10)';
    b4 = 0.5 * randn(10, 1);

    % Forward and Back propagate
    % learning rate
    eta = 0.3;
    % number of SG iterations
    N = 3e4;

    for counter = 1:N
        % choose a training point at random
        k = randi(length(train.label));
        x_train = train.image(:, k);
        % Forward pass
        a2 = activate(x_train, W2, b2);
        a3 = activate(a2, W3, b3);
        a4 = activate(a3, W4, b4);
        % Backward pass
        delta4 = a4 .* (1 - a4) .* (a4 - y(:, k));
        delta3 = a3 .* (1 - a3) .* (W4' * delta4);
        delta2 = a2 .* (1 - a2) .* (W3' * delta3);
        % Gradient step
        W2 = W2 - eta * delta2 * x_train';
        W3 = W3 - eta * delta3 * a2';
        W4 = W4 - eta * delta4 * a3';
        b2 = b2 - eta * delta2;
        b3 = b3 - eta * delta3;
        b4 = b4 - eta * delta4;
    end

    for i = 1:length(test.label)
```

```
        x_test = test.image(:, i);

        % Forward pass
        a2 = activate(x_test, W2, b2);
        a3 = activate(a2, W3, b3);
        a4 = activate(a3, W4, b4);

        % Check whether the neuron with the highest value is correct
        [~, j] = max(a4);
        result(i) = j - 1;
    end
end

function result = nearest_neighbour(train, test)
    k = 1;
    model = fitcknn(transpose(train.image), transpose(train.label));
    model.NumNeighbors = k;
    result = predict(model, transpose(test.image));
end

function result = nearest_neighbour_tangent(train, test, k)
    result = zeros(length(test.label), 1);
    % For each testcase
    for i=1:length(test.label)
        best_norm = Inf;
        best_labels = -ones(k, 1);

        % Determine the best norm
        for j=1:length(train.label)
            current_norm = norm(train.image(:,j)- test.image(:,i));
            if (current_norm <= best_norm)
                best_norm = current_norm;

                for p = 2 : k
                    best_labels(p) = best_labels(p - 1);
                end
                best_labels(1) = train.label(j);
            end
        end
        result(i) = best_labels(1);
    end
end

function result = nearest_neighbour_mean(train, test)
    % average vector 1 contains the average vector of the digit 0
    % create an average vector for each digit
    average_vectors = zeros(256, 10);
    number_of_vectors = zeros(1, 10);
    result = zeros(length(test.label), 1);

    % compute the total vector for each digit
    for i = 1 : length(train.label)
```

```
        digit = train.label(i) + 1;
        average_vectors(:, digit) = average_vectors(:, digit) ...
            + train.image(:, i);
        number_of_vectors(digit) = number_of_vectors(digit) + 1;
    end

    % normalise the vectors
    for i = 1 : 10
        average_vectors(:, i) = average_vectors(:, i) ...
            / number_of_vectors(i);
    end

    % check which average vector is closest with respect to the dist
    for i = 1 : length(test.label)
        best_norm = Inf;
        best_guess = -1;

        % Determine the best norm
        for j = 1 : 10
            current_norm = sum(abs((average_vectors(:, j) ...
                - test.image(:, i))));
            if (current_norm < best_norm)
                best_norm = current_norm;
                best_guess = j - 1;
            end
        end
        result(i) = best_guess;
    end
end

function successrate = check(result, test)
    success = zeros(10, 1);

    for i = 1 : length(test.label)
        if test.label(i) == result(i)
            success(test.label(i) + 1) ...
                = success(test.label(i) + 1) + 1;
        end
    end

    % Calculate success rate
    successrate = sum(success) / i;
end
```