Project: Derivative in Neural Networks and Eigenvalues and Eigenvectors in PCA
IB3702 Mathematics for Machine Learning

Finn Alberts Peter Pelgrims

July 3th, 2024

1 Introduction

In this project, we dive into the role of derivatives in neural networks (NNs) and the inner workings of principal component analysis (PCA). Both are important topics in the world of artificial intelligence and machine learning. The goal is to gain a deeper understanding in both topics, and to get a good grasp on their mathematical inner workings.

We will start by introducing neural networks. One of the most important properties of a neural network is its ability to improve itself, by looking at a lot of (labelled) data. When explained on a high level, during model training, a neural network makes a prediction and improves itself based on how right or wrong that particular prediction was using so-called gradient descent. In this project we dive deeper into how this is done mathematically. We will not explore back propagation (which uses the chain rule on gradient descent), and will limit ourselves to gradient descent for just one layer of the NN. Furthermore, we will explore the ADAGRAD algorithm as a way to optimize training. We will, however, not be exploring other optimization techniques.

Additionally to exploring how neural network models improve themselves, we dive deeper into the role of principal component analysis (PCA). PCA is a method to reduce dimensionality of a dataset. This naturally leads to a loss of accuracy, because we have less data to work with, but can greatly improve training time when working with models like neural networks. We will thorougly explore the role of eigenvalues and eigenvectors in PCA. Additionally, we touch upon singular value decomposition (SVD) and its role in PCA. We will, however, explore SVD for PCA more high-level. Also, we will touch upon how exactly PCA improves training time for neural networks and how it can help to visualize data.

By combining these two topics, we can explore how these techniques mathematically work and can thoroughly understand their inner workings.

2 Preliminaries

2.1 Notation and techniques

In this section, we will list all notations and techniques that will be used in the report and will be considered a preliminary. Please note that we expand upon the notation of the Werkboek Premaster AI: Wiskunde voor machine learning [1] and the No bullshit guide to linear algebra [2] and will not reiterate that notation here. We will, however, list techniques required from these books. For the used notation, see table 1.

Table 1: Used notation

Notation	**Explanation**
$\overline{x}$	The average of x
ϕ	An angle, with $\tan\phi = \frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}$
∇f	The gradient of f

The techniques used can be found in table 2.

Table 2: Used techniques

Technique	**Source**
Calculating eigenvalues and eigenvectors	No bullshit guide to linear algebra
Singular value decomposition (SVD)	No bullshit guide to linear algebra
Calculating the derivative of a function	Werkboek Premaster AI: wiskunde voor machine learning
Sequences	Werkboek Premaster AI: wiskunde voor machine learning

2.2 Example problems

2.2.1 Finding eigenvalues and eigenvectors

As an example, we will find the eigenvalues and eigenvectors for the following matrix:

$$A = \begin{bmatrix} 3 & 1 & 0 \\ 0 & 5 & 0 \\ 2 & -1 & 1 \end{bmatrix}$$

To find the eigenvalues, we have to solve $\det(A - \lambda I) = 0$, let us find the eigenvalues for which this is true:

$$\det\left(\begin{bmatrix} 3-\lambda & 1 & 0 \\ 0 & 5-\lambda & 0 \\ 2 & -1 & 1-\lambda \end{bmatrix}\right) = 0$$

$$(3-\lambda)(5-\lambda)(1-\lambda) + 0 + 0 - 0 - 0 - 0 = 0$$

$$(3-\lambda)(5-\lambda)(1-\lambda) = 0$$

$$\lambda = 3 \vee \lambda = 5 \vee \lambda = 1$$

So the eigenvalues are, in decreasing order: 5, 3 and 1. To find the eigenvectors, we must solve for each eigenvalue the equation $A\vec{e}_\lambda = \lambda\vec{e}_\lambda$. Let us start with $\lambda = 5$

$$\begin{gathered} A\vec{e}_\lambda = 5\vec{e}_\lambda \\ (A-5I)\vec{e}_\lambda = 0 \\ \begin{bmatrix} 3-5 & 1 & 0 \\ 0 & 5-5 & 0 \\ 2 & -1 & 1-5 \end{bmatrix} \begin{bmatrix} e_{\lambda,x} \\ e_{\lambda,y} \\ e_{\lambda,z} \end{bmatrix} = 0 \\ \begin{bmatrix} -2 & 1 & 0 \\ 0 & 0 & 0 \\ 2 & -1 & -4 \end{bmatrix} \begin{bmatrix} e_{\lambda,x} \\ e_{\lambda,y} \\ e_{\lambda,z} \end{bmatrix} = 0 \end{gathered}$$

From here we can already see that we have one free variable. Let us take $-2e_{\lambda,x}+e_{\lambda,y} = 0$. Let us choose $e_{\lambda,x} = 1$, resulting in $-2 + e_{\lambda,y} = 0 \Rightarrow e_{\lambda,y} = 2$. This gives us (for the bottom row): $2e_{\lambda,x} - 1e_{\lambda,y} - 4e_{\lambda,z} \Rightarrow 2-2-4e_{\lambda,z} = 0 \Rightarrow e_{\lambda,z} = 0$, making the eigenvector for $e_{\lambda=5} = [1, 2, 0]$.

We can repeat these steps for $\lambda = 3$ and $\lambda = 1$, which would give $e_{\lambda=3} = [1, 0, 1]$ and $e_{\lambda=1} = [0, 0, 1]$.

2.2.2 Finding a derivative

Suppose we have the function $f(x) = 4xcos(2x^2) - 3x$. We can find the derivative of f by first finding the derivative of $cos(2x^2)$. Using the chain rule we find this to be $-4xsin(2x^2)$. Next, we can combine this with the factor $4x$ in front of it using the product rule, to find the derivative of $4xcos(2x^2)$. This results in $4cos(2x^2) - 16x^2sin(2x^2)$. Finally using the sum rule we subtract the derivative of $3x$, which is 3, to find the derivative of f, being: $f'(x) = 4cos(2x^2) - 16x^2sin(2x^2) - 3$.

3 Example of a Neural Network (NN)

On 3blue1brown [3], we can find an example of a neural network that tries to read handwritten letters (see figure 1).

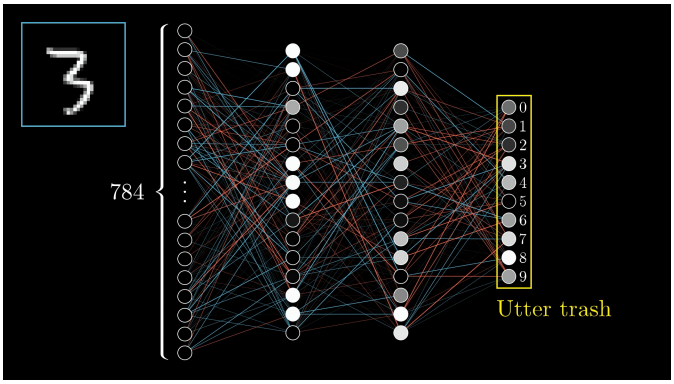


Figure 1: A neural network for recognising handwritten symbols i.c. the number "3"

Without going into depth, we will give the purpose of the neural network: on the left side, the input (consisting of multiple features) is the handwritten number "3". On the right side, only output "3" should be activated. As you can see, other outputs were also activated. This network still needs to **"learn"** to only produce the correct output "3". Obviously, it also needs to produce the correct outputs for other handwritten letters as input.

To learn how a neural network does this, we will need to explore some of the math behind neural networks in the next sections.

4 Gradient Descent and Weight Updates

4.1 What is a Gradient?

Let's start with exploring its definition at Wikipedia [4]:

"The gradient of a scalar-valued differentiable function f of several variables is the vector field ∇f whose value at a point p gives the direction and the rate of fastest increase. The gradient transforms like a vector under change of basis of the space of variables of f. If the gradient of a function is non-zero at a point p, the direction of the gradient is the direction in which the function increases most quickly from p and the magnitude of the gradient is the rate of increase in that direction, the greatest absolute directional derivative. Further, a point where the gradient is the zero vector is known as a stationary point. The gradient thus plays a fundamental role in optimization theory, where it is **used to minimize a function by gradient descent**. In coordinate-free terms, the gradient of a function f may be defined by:

$$df = \nabla f \cdot dr$$

When a coordinate system is used in which the basis vectors are not functions of position, the gradient is given by the vector whose components are the partial derivatives of f

at p. That is for $f : \mathbb{R}^n \leftarrow \mathbb{R}$, Its gradient is defined by $\nabla f : \mathbb{R}^n \leftarrow \mathbb{R}^n$ at point $p = (x_1, x_2, x_3, \ldots, x_n)$ as:

$$\nabla f(p) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(p) \\ \frac{\partial f}{\partial x_2}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \\ \end{pmatrix}$$

Note that the above definition for gradient is only defined for the function f, if it is **differentiable at p.** ... The gradient can also be used to measure how a scalar field changes in other directions, rather than just the direction of greatest change, by taking a dot product." This definition will be further explored in following sections.

4.2 How is a Gradient related to a Derivative?

For a function $f(x)$ with one variable x, the derivative is defined [5] as:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

You can rewrite this for "small"[1] Δx:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$
$$f(x + \Delta x) \approx f(x) + f'(x) \cdot \Delta x$$

Let us explore a function with more variables. We will continue our investigation by looking at the next logical step: a function $f(x, y)$ with two variables x and y.

$$f(x + \Delta x, y + \Delta y) \approx f(x, y) + \frac{\partial f}{\partial x} \cdot \Delta x + \frac{\partial f}{\partial y} \cdot \Delta y \tag{1}$$

We can rewrite this as a vector dot product:

$$f(x + \Delta x, y + \Delta y) \approx f(x, y) + (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

We recognize the gradient vector: $\begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$, also known as $grad(f)$.

Numerical example: For $f(x, y) = x^2 + y^2$, the gradient vector in (x, y) is $\begin{pmatrix} 2x \\ 2y \end{pmatrix}$.

If we have a function with n variables $f(x_1, x_2, \ldots, x_n)$, we similarly can write at point $p = (x_1, x_2, \ldots, x_n)$:

$$f(p + \Delta p) \approx f(p) + \nabla f(p) \cdot (\Delta p) \tag{2}$$

[1] $(\Delta x)^2$ must be negligible compared to Δx.

As mentioned before; the dot denotes the dot product on $\mathbb{R}^n$. Formula (2) is the best "linear" approximation to a function and now it is expressed in terms of the gradient, rather than the derivative.

4.3 What is the interpretation of a Gradient? [6]

Let us look again at (1), with $\Delta x = r \cdot cos\phi$ and $\Delta y = r \cdot sin\phi$, this becomes:

$$f(x+\Delta x, y+\Delta y) \approx f(x,y)+\frac{\partial f}{\partial x}\cdot r\cdot cos\phi+\frac{\partial f}{\partial y}\cdot r\cdot sin\phi = f(x,y)+r\cdot(\frac{\partial f}{\partial x}\cdot cos\phi+\frac{\partial f}{\partial y}\cdot sin\phi)$$

We can consider the function $h(\phi) = r \cdot (\frac{\partial f}{\partial x} \cdot cos\phi + \frac{\partial f}{\partial y} \cdot sin\phi)$. This is maximum when: $h'(\phi) = r \cdot (\frac{\partial f}{\partial x} \cdot (-sin\phi) + \frac{\partial f}{\partial y} \cdot cos\phi) = 0$ or $\frac{\partial f}{\partial x} \cdot sin\phi = \frac{\partial f}{\partial y} \cdot cos\phi$ or $tan\phi = \frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}$. This means the angle ϕ is exactly the direction of the gradient vector: $\begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$.

> The gradient vector is the direction where a function increases the most.

Let us go back to our numerical example: $f(x,y) = x^2 + y^2$ with gradient vector: $(2x, 2y)^T$. For any point (x, y), the direction of the gradient vector is the same as the direction of (x, y) and its magnitude is the norm of the vector: $2\sqrt{x^2 + y^2}$. Let's imagine for a given value of $z = f(x, y)$, these are all circles $z = r^2$. All points (x, y) on that circle have gradient vector $(2x, 2y)^T$ and the same magnitude: $2r$. So, the direction of the gradient vector is the same as (x, y). This means, at each point (x, y), the gradient vectors are perpendicular to the circle. If you move perpendicular from the circle along the gradient vector, you will move in the direction of the greatest change for $f(x, y)$. If you move along the circle, obviously there is no change, as we keep $z = r^2$ constant. As such, the change in $f(x, y)$ in "any" direction from (x, y) can be calculated by making the dot product with the gradient vector.

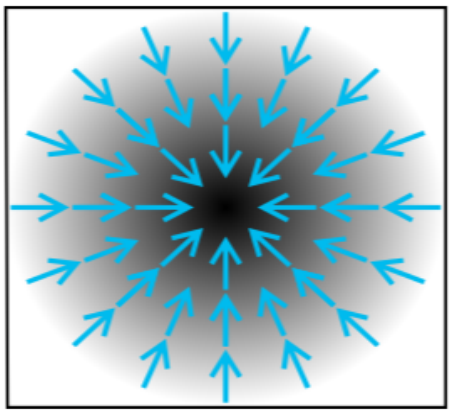
4.4 What is Gradient Descent?

Earlier, we found (2) for approximating a function in the "neighborhood" of a point. As noted: $f(p)$ increases fastest in the direction of the gradient vector and consequently, decreases fastest if one goes in the opposite direction i.e. in the direction of the negative gradient vector $-\nabla f(p)$ also known as "Gradient descent". By its nature, **Gradient descent is a method of finding a local "minimum".**

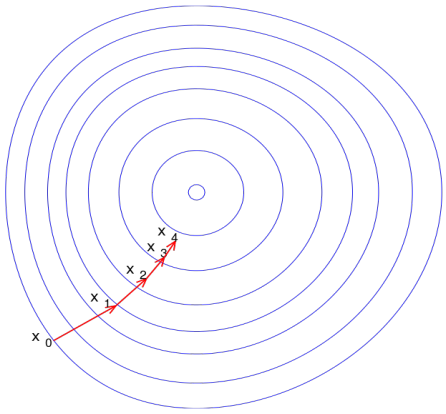
Back to our example: $f(x, y) = x^2+y^2$, the negative gradient vector is: $(-2x, -2y)^T$

At *Wikipedia*, we can find a corresponding visualisation (2a) [4].

From this visualisation, it is clear that in our example this leads to one minimum. Indeed, the minimum for $f(x, y) = x^2 + y^2$ can be found at $(0, 0)$. With gradient descent, we can



(a) Gradient Vectors



(b) Gradient Descent in 2D

Figure 2: Gradient vectors and gradient descent

take repetitive "small' steps in the direction of minimum. At *Wikipedia* (2b), there is another visualisation for "Gradient descent" for a function with two variables starting from x_0, x_1, to x_2, et cetera, we will try to find the minimum step by step.

The series of steps can be seen as a mathematical "sequence". If we use our earlier notation p for points in the n-dimensional vector space. Starting from point p_0, the sequence for subsequent steps can be calculated with (adaptation from [7]):

$$\vec{p}_{n+1} = \vec{p}_n - \gamma \nabla f(\vec{p}_n) \tag{3}$$

with $\gamma > 0 \in \mathbb{R}$ is the step size or learning rate.

Notice that (3) is the same as using (2) repetitively, when displacements can vary at each step and we need to move in the opposite direction of the gradient vector toward the local minimum. There is also a additional learning factor γ for having variable step sizes. The purpose is that the series $\vec{p}_0$, $\vec{p}_1$, $\vec{p}_2$, ..., $\vec{p}_n$ converges to a minimum.

4.5 How is Gradient descent being used in Neural Networks?

"In a neural network we have no control on the activations, we can only influence the weights" [3]

Let's go back to our earlier network [3], where the output was not correct (see figure 3).

What is the loss function (also known as the cost function)? The cost is explained on 3blue1brown:

"To say that a little more mathematically, add up the squares of the differences between each of these trash output activations and the values you want them to have. We'll call

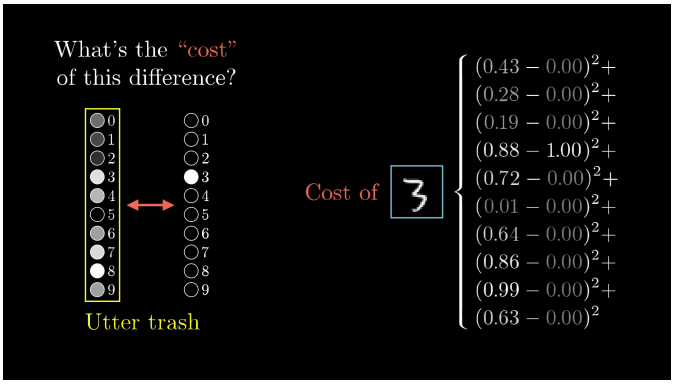


Figure 3: The cost or loss in a neural network

this the "cost" of a single training example."[3]

More in general: [8]: "A loss function is a function that compares the target and predicted output values and measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target".

So, the loss function is chosen and there are many types of loss functions. The one in our example (3), is one of the most popular loss functions: Mean Squared Error (MSE), MSE finds the average of the squared differences between the target and the predicted outputs.

A lecture at Cornell University explains more on the cost function and its constraints: "How does the loss function look like? The problem is: we don't know.[2] but it needs to be continuous, differentiable and convex. Basically, the loss function measures how many points are wrong" [9] We will use the the same source [9] for further exploration and adapt it to fit this into our story and make it consistent within this report.

In machine learning, the cost or loss function is called $l(\vec{w})$ with $w \in \mathbb{R}^n$ and for small displacements $\vec{s}$, can be linearly approximated with:

$$l(\vec{w} + \vec{s}) \approx l(\vec{w}) + g(\vec{w}) \cdot \vec{s} \tag{4}$$

With g being the gradient of function l in $\vec{w}$, this is the same equation as (2).

As we now need to travel through a unknown landscape where the points are now the weights, the goal is to find the weights where the cost function $l(\vec{w})$ is minimal, We can use gradient descent, and take steps down with step: $\vec{s} = -\alpha g(\vec{w})$ when $\alpha > 0$:

$$l(\vec{w} - \alpha g(\vec{w})) \approx l(\vec{w}) - \alpha g(\vec{w})^T \cdot g(\vec{w}) \tag{5}$$

This goes down as $g(\vec{w}_n)^T \cdot g(\vec{w}_n) = \|g(\vec{w}_n)\|$ is positive and $\alpha > 0$. "Why will it eventually reach its minimum? Because the function is convex." [9]

[2]We do of course know the loss function, as we choose it ourselves. What we do not know is how the landscape of the loss function looks like.

Consequently, to find the minimum, we can follow the same steps from (3). With points being the weights, we can update the weights in the same manner:

$$\vec{w}_{n+1} = \vec{w}_n - \alpha \cdot g(\vec{w}_n) \tag{6}$$

With each step, **the weights are updated** and the neural networks **"learns".** For the best learning result, it is important, and most challenging, to find a 'global' minimum and not get stuck in a 'local' minimum. This is probably why Cornell University gives the additional constraint for having a 'convex' function for the loss function [9].

Gradient descent is useful in machine learning for minimizing the cost or loss function.

4.6 Optimization Techniques

How can we optimise our weight update formula (6) to reach the minimum as fast as possible? From a lecture at Cornell University "It comes down to choosing α. It is clear that if we take very small steps (very small α), this will take a long time to reach a minimum. Also, we can't take too large steps, as we are bound by our linear approximation (4) still needs to hold! One safe approach is to choose: $\alpha = \frac{C}{t}$, with C being a constant and t depending on the number of updates you have already taken. Eventually this will be small enough to converge." [9]

Nowadays, there are more sophisticated techniques where the gradient is adapted at each step like the ADAGRAD algorithm (what's in a name):

We start with $\vec{s} = \vec{0}$. Then at each step, we vary the step size:

$$\vec{s} \leftarrow \vec{s} + g(\vec{w}_n)^2$$

and we keep $g(\vec{w}_n)^2$ as a vector by just squaring the elements and not taking the dot product! And we modify the weight update formula into:

$$\vec{w}_{n+1} = \vec{w}_n - \alpha \cdot \frac{g(\vec{w}_n)}{\sqrt{s + \epsilon}}$$

Note: ϵ is just a small constant like 10^{-5} for numerical stability. This translates into: for very steep functions we take small steps and for very shallow functions we take larger steps. Obviously, in order not to continue endlessly, we also have to stop the series, we will stop when we are close enough. Mathematically speaking: $\|\vec{w}_{n+1} - \vec{w}_n\| \leq \delta$, with δ being a small number.

5 Principal Component Analysis (PCA)

5.1 What is Principal Component Analysis (PCA)?

As introduced in chapter 1, PCA is a technique for dimensionality reduction for datasets. Let us first look at a more complete definition. According to Wikipedia, its definition is

as follows:

"Principal component analysis (PCA) is a linear dimensionality reduction technique with applications in exploratory data analysis, visualization and data preprocessing.

The data is linearly transformed onto a new coordinate system such that the directions (principal components) capturing the largest variation in the data can be easily identified." [10]

This implies that we want to reduce the dimensions of a dataset, but at the same time still want to capture the largest variation in the dataset. We do this by projecting our dataset on the principal component space (being made up of the principles axes) and getting the principal components of it. By doing this, we can train models faster (because we have less variables to work with), but still keep them as accurate as possible.

5.2 Techniques for Principal Component Analysis (PCA)

When exploring the techniques behind PCA, we quickly find that there are two main methods which are most commonly used. The first technique uses eigendecomposition on the covariance matrix of the dataset. The second techniques uses singular value decomposition (SVD). It is important to note that both techniques have the same end result.

5.2.1 Eigendecomposition for Principal Component Analysis (PCA)

Let us start by exploring eigendecomposition as a method for PCA. This method consists of six main steps [11].

Step 1: Create a matrix based on the data

This step is arguably the easiest one. To perform PCA we need to create a n-dimensional matrix M to contain the values of our dataset. Each column j contains a specific feature and each row i represents a different row of the dataset as shown in matrix 7.

$$M = \begin{bmatrix} Feature_1, value_1 & \dots & Feature_n, value_1 \\ \dots & \dots & \dots \\ Feature_1, value_n & \dots & Feature_n, value_n \end{bmatrix} \tag{7}$$

As a numerical example, we will imagine a dataset, represented in matrix M. In the following steps we will reduce its dimensions from three to one.

$$M = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix}$$

Step 2: Find the mean of each dimension

Because each column in the matrix represents a feature of the dataset, we can see these columns as the dimensions of the dataset. For each of these dimensions we will calculate the mean, as shown in 8.

$$\overline{M} = \begin{bmatrix} \overline{Feature_1} & \dots & \overline{Feature_n} \end{bmatrix} \tag{8}$$

For our numerical example, we find $\overline{M}$ to be $\begin{bmatrix} 66 & 60 & 60 \end{bmatrix}$

In step 3, we will use the centered matrix C (meaning we will subtract the mean of M: $C = M - \overline{M}$) to calculate the covariance matrix.

Step 3: Find the covariance matrix of C

Next, we compute the covariance of the centered matrix C. The covariance is defined as the variance between two variables and is calculated using formula 9.

$$Cov(x, y) = \frac{1}{n} \sum_{i=1}^{n} (x - \overline{x})(y - \overline{y}) \tag{9}$$

The easiest way to calculate this is to take the centered matrix C and multiply it by its transpose: $B = \frac{1}{n} C^T C$ (n being the number of rows of C), resulting in matrix B, as shown in formula 10.

$$B = \frac{1}{n} C^T C = \begin{bmatrix} Cov(Feature_1, Feature_1) & \dots & Cov(Feature_1, Feature_n) \\ \dots & \dots & \dots \\ Cov(Feature_n, Feature_1) & \dots & Cov(Feature_n, Feature_n) \end{bmatrix} \tag{10}$$

Please note that $Cov(Feature_n, Feature_1) = Cov(Feature_1, Feature_n)$ and therefore matrix B is symmetrical.

When we plug in our example matrix C, the centered version of M, we find:

$$B = \frac{1}{5} \begin{bmatrix} 24 & 24 & -6 & -6 & -36 \\ 0 & 30 & 0 & 0 & -30 \\ 30 & -30 & 0 & 30 & -30 \end{bmatrix} \begin{bmatrix} 24 & 0 & 30 \\ 24 & 30 & -30 \\ -6 & 0 & 0 \\ -6 & 0 & 30 \\ -36 & -30 & -30 \end{bmatrix} = \begin{bmatrix} 504 & 360 & 180 \\ 360 & 360 & 0 \\ 180 & 0 & 720 \end{bmatrix}$$

Step 4: Compute eigenvalues and eigenvectors of covariance matrix B

Next, we need to find the eigenvalues and eigenvectors of B. Using the methods from the "No bullshit guide to linear algebra" [2] we can compute the eigenvalues by solving

$\det(B - \lambda I) = 0$. Next, we can use these eigenvalues to calculate the corresponding eigenvector $\vec{e}_\lambda$ by solving for each value of λ the equation $B\vec{e}_\lambda = \lambda\vec{e}_\lambda$.

For our example, we thus have to solve:

$$\begin{gathered}
\det\left(B - \lambda I\right) = 0 \\
\det\left(\begin{bmatrix} 504-\lambda & 360 & 180 \\ 360 & 360-\lambda & 0 \\ 180 & 0 & 720-\lambda \end{bmatrix}\right) = 0 \\
(504-\lambda)(360-\lambda)(720-\lambda) + (360 \cdot 0 \cdot 180) + (180 \cdot 360 \cdot 0) \\
-(180 \cdot (360-\lambda) \cdot 180) - (0 \cdot 0 \cdot (504-\lambda)) - ((720-\lambda) \cdot 360 \cdot 360) = 0 \\
-\lambda^3 + 1584\lambda^2 - 641520\lambda + 25660800 = 0
\end{gathered}$$

Using WolframAlpha[3] we find $\lambda_1 \approx 44.81966, \lambda_2 \approx 629.11039, \lambda_3 \approx 910.06995$. Because $\lambda_3 \approx 910.06995$ is the largest eigenvalue and we want to reduce to one dimension, we will only be calculating the eigenvector for this one and not for the other eigenvalues (more on this in step 5). Using WolfgramAlpha we find this vector to be $\vec{e}_\lambda = [1.05594, 0.69108, 1]$.

Step 5: Use the largest eigenvalues to create a matrix W

The largest eigenvalues correspond to the eigenvectors which contain the largest variance. Therefore, we use the eigenvectors corresponding to these eigenvalues to create a matrix W, which will transform our original data into a lower dimension. Depending on how many dimension you want your final dataset to have (after performing PCA), we pick the k largest eigenvalues and corresponding eigenvectors. This is shown in matrix 11. This matrix is thus made up out of the principle axes of the dataset.

$$W = \begin{bmatrix} \vec{e}_{\lambda_1} & \dots & \vec{e}_{\lambda_k} \end{bmatrix} \tag{11}$$

Plugging in our found vector in step 4, we get:

$$W^T = \begin{bmatrix} 1.05594 & 0.69108 & 1 \end{bmatrix}$$

Step 6: Calculate principal components

In this last step, we calculate the principal components in a matrix P by taking the transpose of W, W^T and multiplying it with our original dataset M transposed. See equation 12.

$$P = W^T A^T \tag{12}$$

Each column j of P corresponds to the principal components of row i of M.

[3]This equation is a third-degree equation and therefore out-of-scope to solve by hand.

For our example, our result will be:

$$P = W^T M^T$$
$$P = \begin{bmatrix}1.05594 & 0.69108 & 1\end{bmatrix} \begin{bmatrix}90 & 90 & 60 & 60 & 30\\ 60 & 90 & 60 & 60 & 30\\ 90 & 30 & 60 & 90 & 30\end{bmatrix}$$
$$P = \begin{bmatrix}226.495 & 187.225 & 164.816 & 194.816 & 82.4082\end{bmatrix}$$

The values in P correspond to the principal components of each row of M.

5.2.2 Singular Value Decomposition (SVD) for Principal Component Analysis (PCA)

Another way to calculate the principal components is to use singular value decomposition (SVD) [12]. This method is more often used in computers when calculating the principal components, because it is a more optimized method.

As we know, SVD allows us to decompose any $m \times n$ matrix A into a $m \times m$ orthonormal matrix U, a $m \times n$ matrix Σ and a $n \times n$ orthonormal matrix V^T: $A = U\Sigma V^T$. Consider the matrix A to contain our centered data, where each column is a feature and each row is a datapoint. In that case, $U\Sigma$ is actually the matrix containing our data in the principal components space and V is the matrix we use to convert our original data A into the principal component space. Is that not beautiful?[4]

5.3 Visualizing Principal Component Analysis (PCA)

PCA can help us in multiple ways. One way is in visualizing the data in a lower dimension, so we can more easily interpret it.

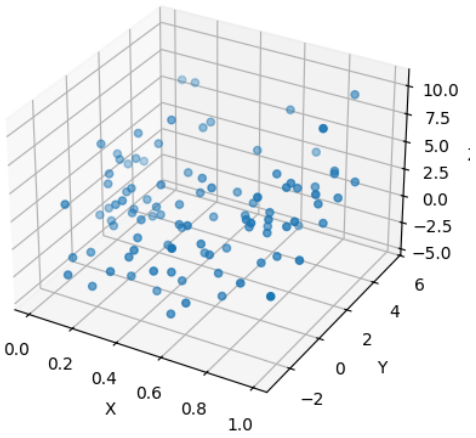
Say, for example we have a dataset with 10 features. If we wanted to plot these datapoints in a visual way, we would get a 10 dimensional plot. We, as humans, can only imagine space in 3 dimensions, making it impossible for us to visualize the data.

However, getting insight into how the different features might be correlated can be hugely beneficial, as it can help in understanding the data better. PCA can help us here. By preforming PCA, and plot the first two (or three) components of each datapoint, we can see if there's a relation and how strong that relation is.

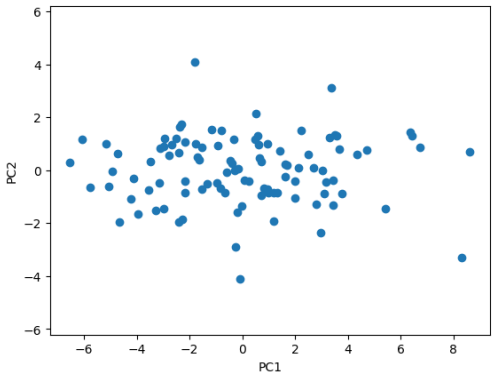
As a numerical example, suppose we have a dataset with three variables: x, y and z. x is a random number between 0 and 1, y has a linear relation with x, being $y = 2x$, and z is also linearly related with x and y: $z = x + y$. The relation is not perfect, as both y and z have a little noise added in. In figure 4a we can see the data plotted in 3D. As we can see, an pattern is not immediately clear. However, after performing PCA and

[4]To comply with the page limit, we will not be giving a numerical example here. For the steps to perform SVD, we refer to page 333 of the No bullshit guide to linear algebra.

visualizing the first two principal components in 2D (see figure 4b) the relation becomes a lot more visible.[5]



(a) Data plotted in three dimensions



(b) Principal components plotted in two dimensions

Figure 4: Comparison of two data plots

5.4 The impact of Principal Component Analysis (PCA) on training Neural Networks

Apart from PCA as a way to more easily visualize data, it is more commonly used to improve training times for neural networks. As explained before, every feature in a neural network corresponds to one neuron for the input layer. The more features you have, the more neurons with corresponding weights you have. This in turn results into more calculations to be made and thus longer training times. By using PCA, we can reduce the number of features we feed into the neural network, which will thus result in a shorter training time.

6 Collaboration

During the project we collaborated in a efficient way where we divided tasks among the two of us. To ensure we could both make progress, without being to dependent on each others schedules, we decided to split the project up in two "iterations" of one week each.

During the first iteration, Peter worked on laying the groundwork for the part about derivatives in neural networks, creating a big contribution there. At the same time,

[5]For the code used to create this dataset and these plots, see https://github.com/FinnAlberts/PCA_for_visualization

Finn established the foundation for the part about PCA and diving deeper into this part of the project.

After the first iteration, we critically assessed each others contributions, before having a meeting using Microsoft Teams. In the meeting we taught each other about the subject we worked on. Additionally we asked each other critical questions and suggested improvements. We both found this to work quite well.

After the meeting, we had our second iteration, in which we improved our subjects based on the feedback and questions asked during our meeting. This ensured we wrote a clear report. After this second iteration, we had two final meetings where we worked on dotting the i's and finishing up the project.

Aside from our scheduled meetings, we also had regular contact through WhatsApp whenever we found something which might be relevant or whenever we got stuck on something. This was a nice addition to ensure we would stay on track.

7 Reflection

7.1 Reflection by Finn Alberts

During this project I really learned a lot about the inner workings of neural networks. I really enjoyed learning about how parts of the course material relate to real AI applications. Especially the way eigenvalues and eigenvectors relate to PCA and how it helps in improving model training times was interesting to see. Additionally, the meetings with Peter where we explained things which researched were really useful and enjoyable. Discussing new mathematical topics with someone is definitely something I would continue in the future

Most challenging for me was balancing the amount of depth for each part of the report, especially with the limited amount of pages. However, when Peter and I decided to merge to methods and numerical example sections into one, not only the story became a lot more clear, but also we had some more space to work with.

7.2 Reflection by Peter Pelgrims

What I learned during this project is some mathematical background om neural networks. I also enjoyed working with Finn and recognized we had different perspectives which was quite an enrichment. The most challenging was to set the scope and not to drown in the overwhelming, sometimes incorrect and inconsistent material that is available on the internet. I did not learn any new mathematical techniques, but I learned how calculus and linear algebra can be better interpreted and combined.

In the future, I would use the same approach to study new mathematics: by following a formal course to set a proper foundation (and avoid drowning), and later extend it with online material from reliable sources.

References

[1] *Werkboek Premaster AI: wiskunde voor machine learning*, First edition ed. Heerlen: Open Universiteit, 2022.

[2] *No bullshit guide to linear algebra*, Second edition, v2.2. ed. Montréal: Minireference Co., 2020.

[3] "Chapter 2: Gradient descent, how neural networks learn," Jun. 2024. [Online]. Available: https://www.3blue1brown.com/lessons/gradient-descent

[4] "Gradient," May 2024. [Online]. Available: https://en.wikipedia.org/wiki/Gradient

[5] *Wiskunde voor Bachelor en Master: Deel 1 - Basiskennis en vaardigheden.* Utrecht: Syntax Media, Utrecht, 2015.

[6] *Wiskunde voor Bachelor en Master: Deel 2 - Differentiaalrekening en integraalrekening.* Utrecht: Syntax Media, Utrecht, 2016.

[7] "Gradient decent," Jun. 2024. [Online]. Available: https://en.wikipedia.org/wiki/Gradient_descent

[8] "Loss function," Aug. 2022. [Online]. Available: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9

[9] "Lecture 12: Gradient descent (and beyond)," 2019. [Online]. Available: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote07.html

[10] "Principal component analysis," Jun. 2024. [Online]. Available: https://en.wikipedia.org/wiki/Principal_component_analysis

[11] "The mathematics behind principal component analysis," Dec. 2018. [Online]. Available: https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643

[12] "Pca and svd — appearance matching," Jun. 2021. [Online]. Available: https://www.youtube.com/watch?v=ILu4-Lk-gZQ