2.1 PCA

a)

$$X = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}$$

X with mean 0:

$$X = \begin{bmatrix} -\frac{3}{2} & -\frac{1}{4} \\ -\frac{1}{2} & \frac{3}{4} \\ \frac{1}{2} & -\frac{1}{4} \\ \frac{3}{2} & -\frac{1}{4} \end{bmatrix}$$

We continue with the matrix with mean zero.

$$X^T * X = \begin{bmatrix} -\frac{3}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{3}{2} \\ -\frac{1}{4} & \frac{3}{4} & -\frac{1}{4} & -\frac{1}{4} \end{bmatrix} * \begin{bmatrix} -\frac{3}{2} & -\frac{1}{4} \\ -\frac{1}{2} & \frac{3}{4} \\ \frac{1}{2} & -\frac{1}{4} \\ \frac{3}{2} & -\frac{1}{4} \end{bmatrix} = \begin{bmatrix} 5 & -0.5 \\ -0.5 & 3 \end{bmatrix}$$

Now we can compute the eigenvalues:

$$(5 - \lambda) * (3 - \lambda) - 0.25 = 0$$

$$15 - 8\lambda + \lambda^2 - 0.25 = 0$$

$$\Rightarrow \lambda_1 = 2.88197, \lambda_2 = 5.11803$$

$$\begin{bmatrix} 5 - 5.11803 & -0.5 \\ -0.5 & 3 - 5.11803 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow -\frac{1}{2} * x_1 - 2.11803 * x_2 = 0$$

$$\Rightarrow \text{eigenvector: } \begin{bmatrix} 4.23606 \\ 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 4.23606 \\ 1 \end{bmatrix}$$

Encoder: $f(x) = D^T * x$
Decoder: $g(x) = D * D^T * x$
Encoding:

- $f(x^{(1)}) = \begin{bmatrix}4.23606, & 1\end{bmatrix} * \begin{bmatrix}1\\1\end{bmatrix} = 5.23606$
- $f(x^{(2)}) = \begin{bmatrix}4.23606, & 1\end{bmatrix} * \begin{bmatrix}2\\2\end{bmatrix} = 10.47212$
- $f(x^{(3)}) = \begin{bmatrix}4.23606, & 1\end{bmatrix} * \begin{bmatrix}3\\1\end{bmatrix} = 13.70818$
- $f(x^{(4)}) = \begin{bmatrix}4.23606, & 1\end{bmatrix} * \begin{bmatrix}4\\1\end{bmatrix} = 17.94424$

b)

$$X = \begin{bmatrix}-1 & 1\\-2 & 2\\-1 & 3\\-1 & 4\end{bmatrix}$$

X with mean 0:

$$X = \begin{bmatrix}\frac{1}{4} & -\frac{3}{2}\\-\frac{3}{4} & -\frac{1}{2}\\\frac{1}{4} & \frac{1}{2}\\\frac{1}{4} & \frac{3}{2}\end{bmatrix}$$

We continue with the matrix with mean zero.

$$X^T * X = \begin{bmatrix}\frac{1}{4} & -\frac{3}{4} & \frac{1}{4} & \frac{1}{4}\\-\frac{3}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{3}{2}\end{bmatrix} * \begin{bmatrix}\frac{1}{4} & -\frac{3}{2}\\-\frac{3}{4} & -\frac{1}{2}\\\frac{1}{4} & \frac{1}{2}\\\frac{1}{4} & \frac{3}{2}\end{bmatrix} = \begin{bmatrix}0.75 & 0.5\\0.5 & 5\end{bmatrix}$$

Now we can compute the eigenvalues:

$$(0.75 - \lambda) * (5 - \lambda) - 0.25 = 0$$

$$3.75 - 5.75\lambda + \lambda^2 - 0.25 = 0$$

$$\Rightarrow \lambda_1 = 0.691969, \lambda_2 = 5.05803$$

$$\begin{bmatrix} 0.75 - 5.05803 & 0.5 \\ 0.5 & 5 - 5.05803 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow \frac{1}{2} * x_1 - 0.05803 * x_2 = 0$$

$$\Rightarrow \text{eigenvector: } \begin{bmatrix} 1 \\ 8.61623 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 \\ 8.61623 \end{bmatrix}$$

Encoder: $f(x) = D^T * x$
Decoder: $g(x) = D * D^T * x$
Encoding:

- $f(x^{(1)}) = \begin{bmatrix} 1 & 8.61623 \end{bmatrix} * \begin{bmatrix} -1 \\ 1 \end{bmatrix} = 7.61623$
- $f(x^{(2)}) = \begin{bmatrix} 1 & 8.61623 \end{bmatrix} * \begin{bmatrix} -2 \\ 2 \end{bmatrix} = 15.23246$
- $f(x^{(3)}) = \begin{bmatrix} 1 & 8.61623 \end{bmatrix} * \begin{bmatrix} -1 \\ 3 \end{bmatrix} = 24,84869$
- $f(x^{(4)}) = \begin{bmatrix} 1 & 8.61623 \end{bmatrix} * \begin{bmatrix} -1 \\ 4 \end{bmatrix} = 33.46492$

Decoding:$D * D^T * x$ ($D^T * x$ is already computed)

- $\begin{bmatrix} 1 \\ 8.61623 \end{bmatrix} * 7.61623 = \begin{bmatrix} 7.61623 \\ 65.6231 \end{bmatrix}$
- $\begin{bmatrix} 1 \\ 8.61623 \end{bmatrix} * 15.23246 = \begin{bmatrix} 15.23246 \\ 131.2463788 \end{bmatrix}$
- $\begin{bmatrix} 1 \\ 8.61623 \end{bmatrix} * 24,84869 = \begin{bmatrix} 24,84869 \\ 214.1020282 \end{bmatrix}$
- $\begin{bmatrix} 1 \\ 8.61623 \end{bmatrix} * 33.46492 = \begin{bmatrix} 33.46492 \\ 288,3414477 \end{bmatrix}$

c) TODO

2.2 Numerical Computation 1

a) As explained in the deeplearning book:
Both of these difficulties can be resolved by instead evaluating softmax(z) where $z = x - \max_i x_i$. Simple algebra shows that the value of the softmax function is not changed analytically by adding or subtracting a scalar from the input vector. Subtracting $\max_i x_i$ results in the largest argument to exp being 0, which rules out the possibility of overflow. Likewise, at least one term in the denominator has a value of 1, which rules out the possibility of underflow in the denominator leading to a division by zero.

b) As explained in the deeplearning book:
Underflow in the numerator can still cause the expression as a whole to evaluate to zero. This means that if we implement log softmax(x) by first running the softmax subroutine then passing the result to the log function, we could erroneously obtain $-\infty$.

2.3 Numerical Computation 2

a) see File

b) as you can see in Figure 1 and 2 our Function has the look of an sloap.
If we now take a look at the steps of the gradient descent, displayed in Figure 3 and 4 we can see that we are starting at point $(-2|4|84)$. the next descent goes down to $(-0.4|3.9|7)$ as shown in 5.

The descent continues down the slope with decreasing X values fairly quickly towards zero. The same happens for the Y Coordinates but less quickly.

This is caused by the way the function is being setup. The derivative in X direction is growing much quicker (distancing from 0) compared to the derivative in Y direction. Thats why we are approaching $X = 0$ much much quicker than we are $Y = 0$. The value for z changes accordingly and also tends towards 0 just as expected.

c) as you can see in Figure 6 down below this is a very interesting special case because with this ϵ the gradient descent alternates between the "sides" (in respect to X) of our surface. This is caused by the derivative which is alternating between the same value positive and negative for the descent in direction X. So we are esentially endlessly looping in X-direction without any progress while only "progressing" in Y-direction. For every starting point there is exactly a single value of ϵ which results in this behaviour. This result is caused by the symetry of our surface!

2.4 Numerical Computation 3

a) see File

b) you can see the steps of Newtons Method displayed in Figure 7.
TODO EXPLAIN!!

2.5 Numerical Computation 4

stop descent if one of these conditions hold:

- the derivative of f in this point is smaller than a chosen constant 0.0001 for 3 iterations in a row.
- we also stop in a special case just like in 2.3c by checking if we start alternating between 2 points. We could do this for example by storing the last 4 values for the derivative and checking if

$$dx_0 - dx_2 \leq 0.0001 \quad \wedge \quad dx_1 - dx_3 \leq 0.0001$$

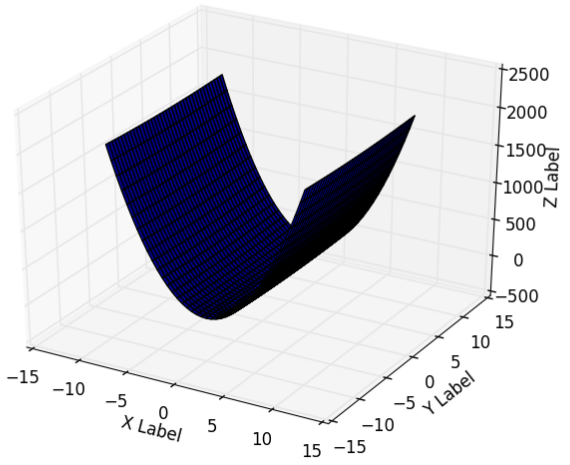

Figure 1: function surface perspektive 1

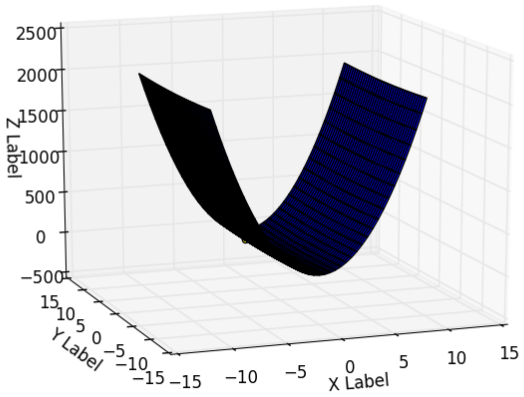

Figure 2: function surface perspektive 2

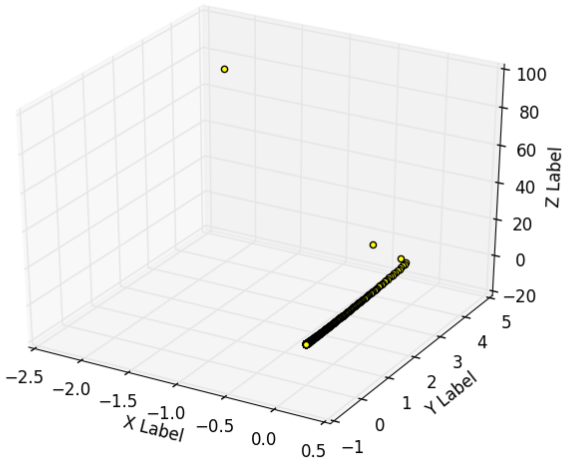

Figure 3: gradient descent steps, perspektive 1

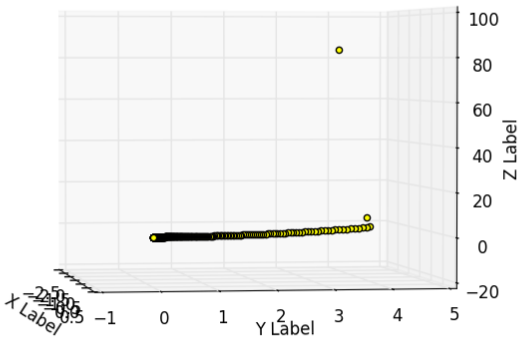

Figure 4: gradient descent steps, perspektive 2

Daniel Schäfer (2549458)
Christian Bohnenberger (2548364)
Dominik Weber (2548553)

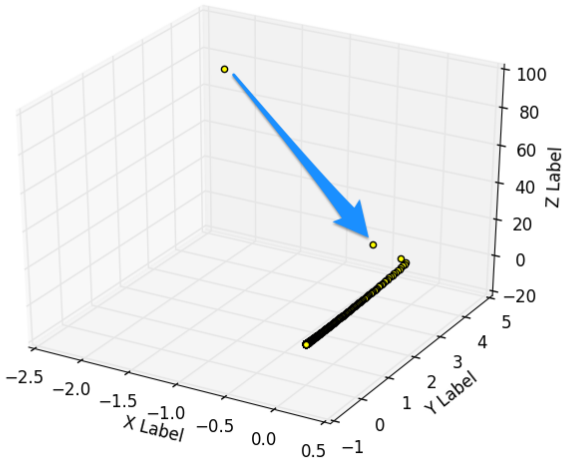

Figure 5: gradient descent direction

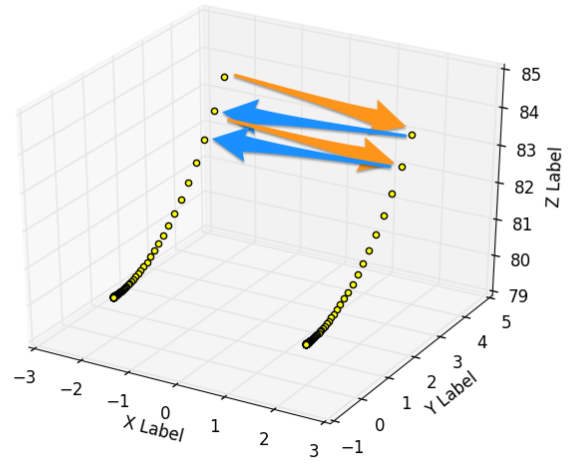

Figure 6: gradient Descent with $\epsilon = 0.1$

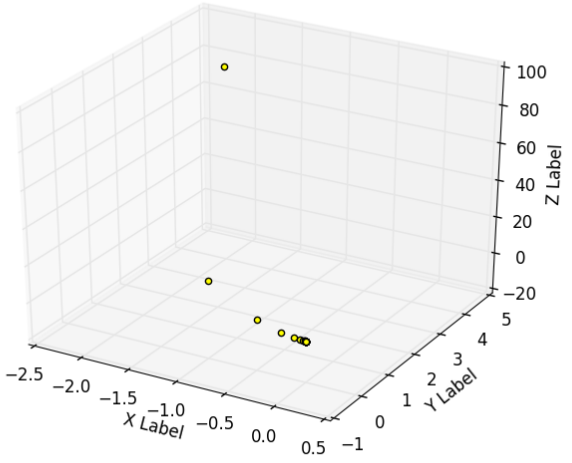

Figure 7: Newtons method steps