Ex2

Eviatar Ben Eliyahu

April 23, 2023

1 Practical part

1.1 Auto-Encoding

Explore the reconstruction error over the test set when using lower and higher latent space dimension d:
First and foremost the model indeed reconstruct the images adequately. As can be inferred by the following sanity check:

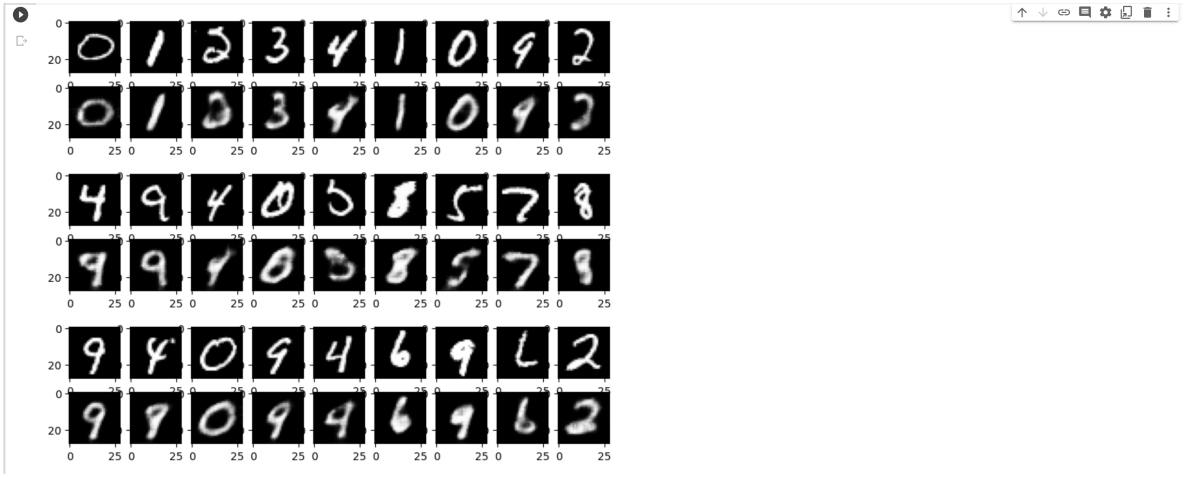

Figure 1: First line at any collection are the randomly chosen original images. Second line of images are the corresponded reconstructed images (after decoding and decoding).

Using lower and higher latent space dimension d, as expected, the model results in better performances as long as the latent dimension, d, increasing. As can one see, the higher value of d, the better test loss and train loss, the model achieves. Furthermore, another important observation is that's model with higher latent dimension converge quicker then models with lower latent dimension. In the next figure, the models' performances plotted, where all the models have the same architecture and vary only by the latent dimension, $d \in [5, 14]$.

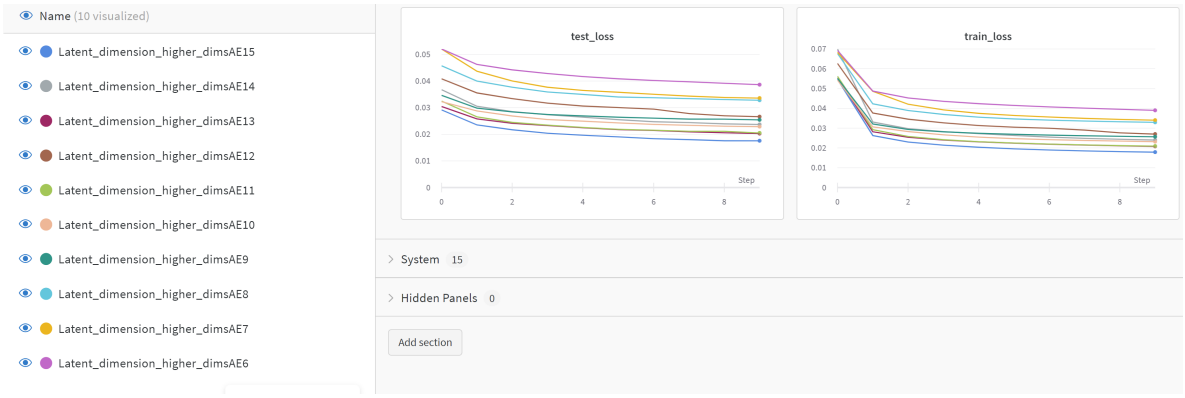

Figure 2: Models with different latent dimension, $d \in [5, 14]$.

Those results are expected since the encoder "projects" the given images and then reconstruct them. The projection results in loss of "information", the higher the latent dimension is, the less information is getting lost, and the reconstruction supposed to be more accurate.
As part of the exploration, to reassure the conclusion, same model examined in higher latent dimension where $d \in [50, 55]$. This, domain $[50, 55]$ chosen because its significantly higher then the previous

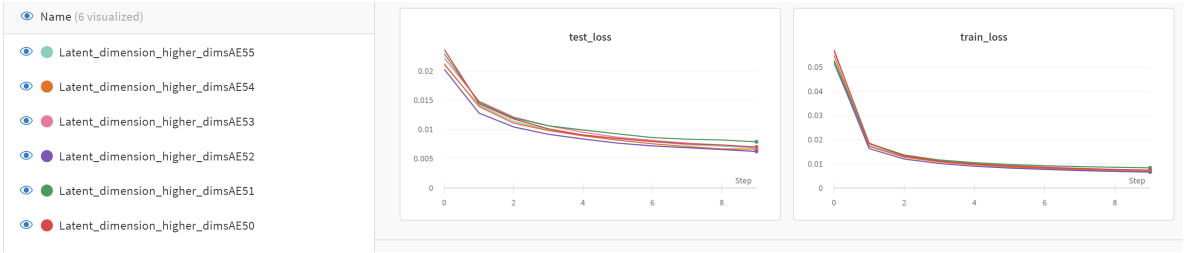

Figure 3: Models with different latent dimension, $d \in [100, 104]$.

domain , but not close enough to the image domain, which are $[28 \times 28]$ so a "projection" is still occurs. In addition to that, one can see that in aspects of "variance" the higher the latent dimension is, the smaller variance there is while the model is learning.
This part of the exploration indeed strengthens the hypothesis, even in higher dimension, this performance improvement is occur. Those higher dimension, as expected, even results in better performance, but worth-mentioning, we can see, **empirically only**, a moderate improvement as long as the latent dimension is increasing over the higher dimension.

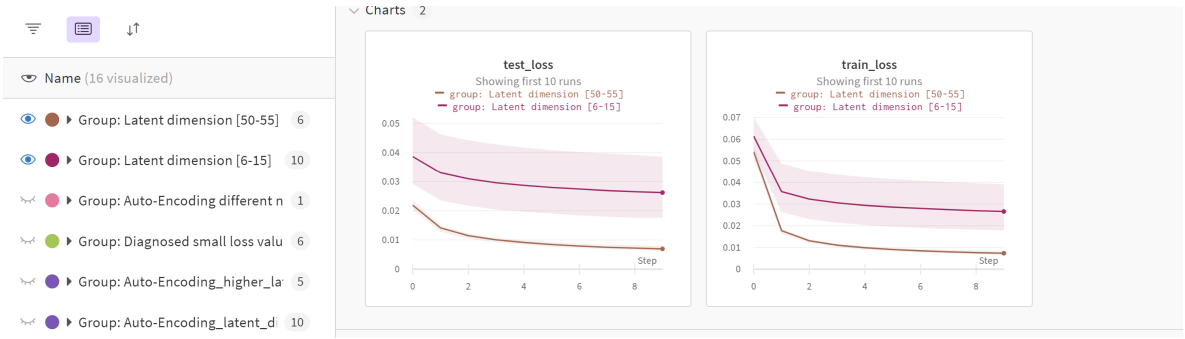

Figure 4: Models with different latent dimension, d, where $d \in [6, 15]$ in comparison to Models with latent dimension ,d, where $d \in [6, 15]$.

Explore the reconstruction error over the test set when using using a fixed latent dimension d but with encoder/decoder architecture with more or fewer layers/weights: As expected, increasing the networks' number of layers results in higher performances. Those results are expected as well- since increasing the networks' layer number allows the model to become more complex, extends the model's expressivity, and strengthen it's ability to learn complex patterns in the data.

Figure 5: Same BaseLine Model with different number of layers

1.2 Interpolation

Interpolations, with latent space, d = 10:

Figure 6: Interploation with latent dimension 10 between digits 9 to 6

Figure 7: Interpolation with latent dimension 10 between digits 5 to 4

Figure 8: Interpolation with latent dimension 10 between digits 9 to 2

Interpolations, with latent space, $d \in [10, 15, 40, 90]$ between digits 2 to 3:

Figure 9: d= 5

Figure 10: d= 10

Figure 11: d= 15

Figure 12: d= 40

Figure 13: d= 90

Analyzing the performance through visual inspection of reconstructed images we can see how higher latent dimension results in sharper images and quicker shift along the interpolation due to the fact that as the dimensionality of the projected space increases, the model will typically be able to learn and capture patterns that hidden in the data, That, in contrast to lower dimension, where the projection results in higher data loss and less expresivity and complexity of the learned model -where the interpoolated images seems to be more blurry, and to shifts to the "digit targrt" slowly. All that said, until a range where the latent dimension is so high that the model become to complex and the reconstructed images are basically - noise.

Figure 14: Interpolation between two images, where d = 100

Figure 15: Interpolation between two images digits are supposed to be 7 and 9, where d = 3

1.3 De-correlation

Looking "element-wise" on the correlation value- as long as d increase the correlation is decreasing. While calculating the *norm* of the covariance matrix, as long as the d increase the norm increasing. A possible explnation for that is that dealing with small latent space a model tends to not save a lot of nuances and Data's characters and so the correlation is small, while in compare to that dealing with higher latent dimension the projection is still keeping more characters and patterns of the data which yields to an high correlation. An intuitive way to explain this trend- looking on the "extreme case" where d = 1000, there are much more dimensions than pixels, so obviously that yields an higher corrrelation.

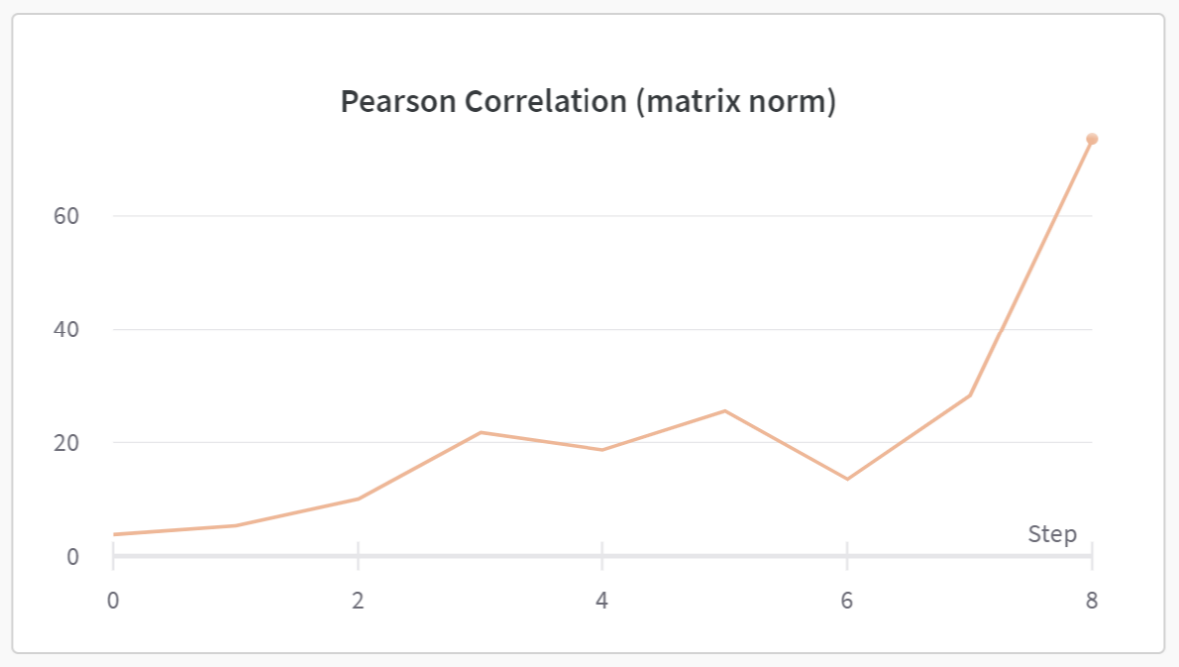

Figure 16: Pearson correlations metric = norm

After reconsideration, and applying other metrics then norm - such as matrices' determinant or matrices' mean (not take into account the main diagonal, and after abs() function - due to the significance of de-correlation) we find that, as said above, as long as d increase the correlation is decreasing, and our possible explanation was not necessarily right (an explanation to the above increasing can be explained as "curse of dimensionality" or in more specific words- the norm is a metric that increasing as the dimension is increasing, hence the increasing might be attached to that and not to the correlation increasing):

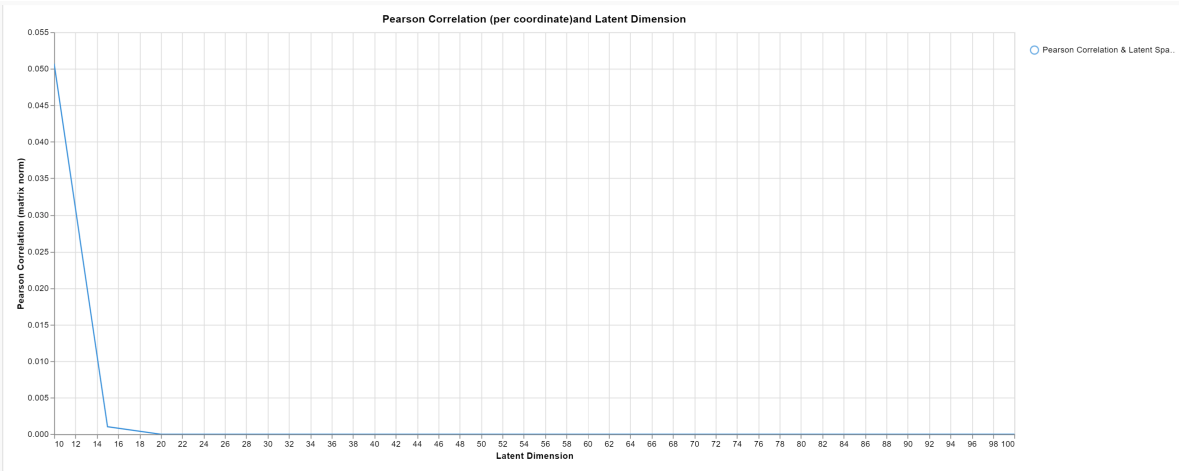

Figure 17: Pearson correlations metric = determinant

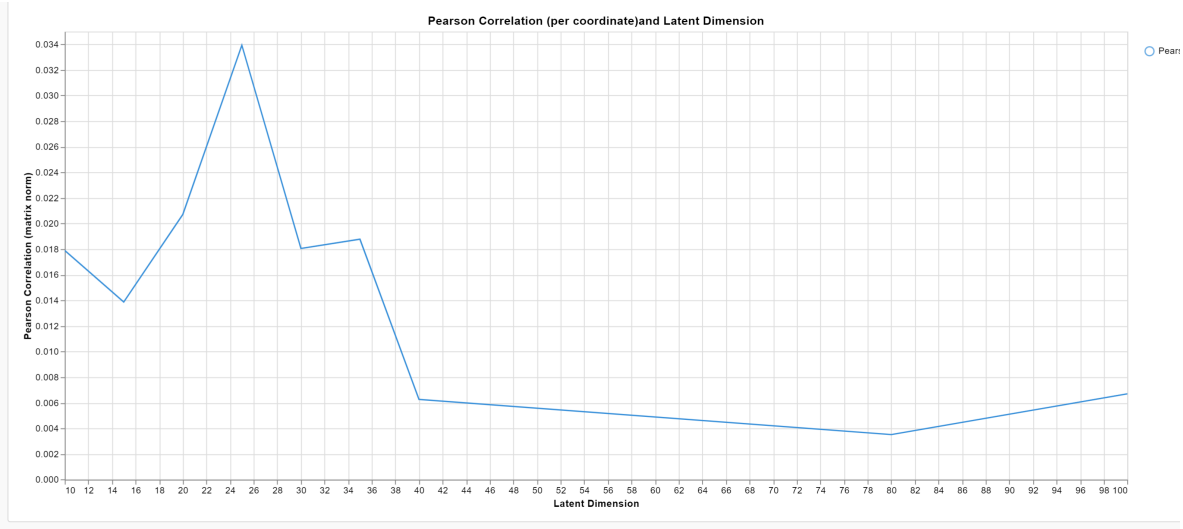

Figure 18: Pearson correlations metric = mean(abs())

Bottom line, as the dimensionality of the projected space (i.e., the latent dimension) increases, the model will typically be able to preserve more of the correlation structure present in the original data. This is because a higher-dimensional space provides more degrees of freedom to represent complex correlations between features. However, this does not mean that increasing the dimensionality of the projected space will always be beneficial. If the data has a low intrinsic dimensionality (i.e., the data is intrinsically low-dimensional and can be well-represented by a lower-dimensional space), then increasing the dimensionality of the projected space beyond this intrinsic dimensionality may not provide any additional benefits, and may even introduce noise and overfitting. Thus, we can understand that since our data is relatively "simple" and do not requires a lot of resources and high latent dimensions, indeed the correlation might increase as d increase - until a "sweet spot"(probably low- as said the CIFAR-10 data is relatively simple), where at this point the correlation might decreased as d increase.

1.4 Transfer Learning

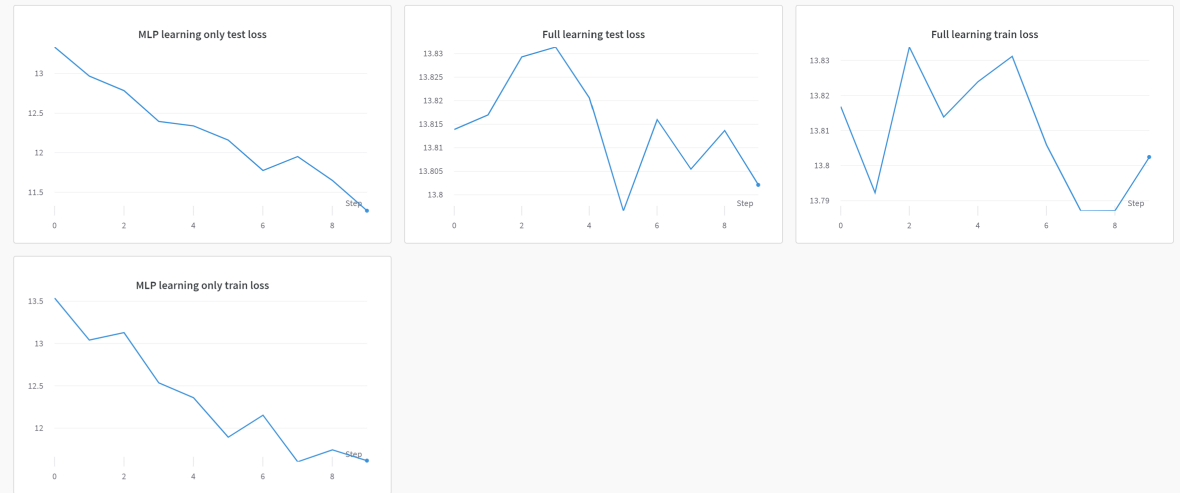

Figure 19: Transfer learning- full learning versus pre-trained model "freeze"

As can one deduced from the following plots and looses, "freezing" the learning in this specific architecture result in better (and more stable) performances. This behaviour might explained by the fact that the classifier is relatively poor at the beginning, and hence the loss which it's deliver to the

encoder is harmful (if the model has been trained over more examples, than it might have been more beneficial to learn all the the models' parameters. Furthermore, since the pre-learned encoder is not compatible to classify it's make sanse to learn its parameters. Those several images fits the model's parameters, and since the networks size is relatively small, the gradient is not vanishing, meanning that the representation is better and it is easier to the MLP to solve the task. Latent space dimension d = 15 (since in the previous task this dimension prooved as a dimension with good result and simultanisly small). Arch Encoder: 3 conv' layers (, with Relu), 2 fully connected layers, with Sigmoid. MLP: 3 linear layers (with Relu).

2 Theoretical part

2.1

Let f_1 and f_2 be a linear functions, $\vec{v}$, $\vec{u} \in F^d$, where $d \leq \infty$, and scalar $\alpha \in F$.

- $f_1(f_2(u+v)) = f_1(f_2(v) + f_2(u)) = f_1(f_2(v)) + f_1(f_2(u))$.
- $f_1(f_2(\alpha \cdot u)) = f_1(\alpha \cdot (f_2(u))) = \alpha \cdot f_1(f_2(u))$

Where in both subsection the first equality derives from linearity of f_2 and the second from linearity of f_1.

Let f_1 and f_2 be an affines functions with corresponding matrices and vectors $A, B, \vec{a}, \vec{b}$, and $\vec{x} \in F^d$.
Where $f_1(\vec{x}) = A\vec{x} + \vec{a}$ and $f_2(\vec{x}) = B\vec{x} + \vec{b}$
Then $(f_2 \circ f_1)(\vec{x}) = f_2(f_1(\vec{x})) = B(A\vec{x} + \vec{a}) + \vec{b} = (BA)\vec{x} + (B\vec{a} + \vec{b})$.
(And (if and) since A and B are invertible matrices, BA is invertible as well.)

2.2

The calculus behind the Gradient Descent method:

2.2.1

Ideally, the stopping condition of the form $\Theta^{n+1} = -\alpha \nabla f_{\theta^n}(x)$ **is** $\Theta^{n+1} = \Theta^n$. i.e., where gradient is basically $\vec{0}$.

Practically the stopping condition for the iterative Gradient Descent scheme is typically to continue the iterations until a certain convergence criterion is met. There are several convergence criteria that can be used, and the choice of criterion depends on the specific problem and the desired level of accuracy.
One common convergence criterion is to check the magnitude of the gradient of the objective function at each iteration. The iteration is stopped when the magnitude of the gradient falls below a specified tolerance level, indicating that the algorithm has converged to a minimum of the function.
Another convergence criterion is to monitor the change in the objective function at each iteration. The iteration is stopped when the change in the objective function falls below a specified tolerance level, indicating that the algorithm has converged to a minimum of the function.
Finally, the iteration can be stopped after a fixed number of iterations, which is typically set based on the computational resources available and the desired level of accuracy.
In summary, the stopping condition for the Gradient Descent method is typically based on a convergence criterion that is related to the magnitude of the gradient or the change in the objective function, or a fixed number of iterations.

2.2.2

Following the previous section:
$f(x+dx) = f(x) \iff f(x) + dx \cdot \nabla f(x) + dx^T \cdot H(x) \cdot dx + O(||dx||^3) \iff dx^T \cdot H(x) \cdot dx = 0$
Same goes for $<$ or $>$
That is, theoretically, if for every $dx \in R$ it holds that $dx^T \cdot H(x) \cdot dx > 0$ s is a minima(, and with

analogy to the another cases).
From another direction, to use this theorem to classify a stationary point (i.e., a point where the gradient vector is zero) as a local maximum or minimum, we need to examine the sign of the eigenvalues of the Hessian matrix.
If all eigenvalues of H are positive, then the quadratic form $dx^T \cdot (x) \cdot dx$ is positive definite, meaning that it is always positive except at the point $x + dx$. This implies that f(x) is locally convex near $x + dx$, and thus $x + dx$ is a local minimum.
If all eigenvalues of H are negative, then the quadratic form $dx^T \cdot (x) \cdot dx$ is negative definite, meaning that it is always negative except at the point x=$x + dx$. This implies that f(x) is locally concave near $x + dx$, and thus $x + dx$ is a local maximum.
If H has both positive and negative eigenvalues, then the quadratic form $dx^T \cdot (x) \cdot dx$ can take both positive and negative values, depending on the direction of d. In this case, $x + dx$ is a saddle point.
Note: if H has at least one zero eigenvalue, then the second-order approximation is not sufficient to determine the nature of the stationary point, and higher-order terms need to be considered.

2.2.3

In such case- Circular data, where the data has no true zero rather vector data has a periodicity - we can apply circular statistics (instead of linear). In particular, we can use the concept of circular variance.
That is, for example, for calculation the Circular mean: take every single data set ,α , and cut it up into its units or its components. By the following formula (assuming the data already converted to radians): $\hat{\alpha} = arctan(\frac{1}{N} \cdot \frac{\sum \sin(\alpha)}{\sum cos(\alpha)})$

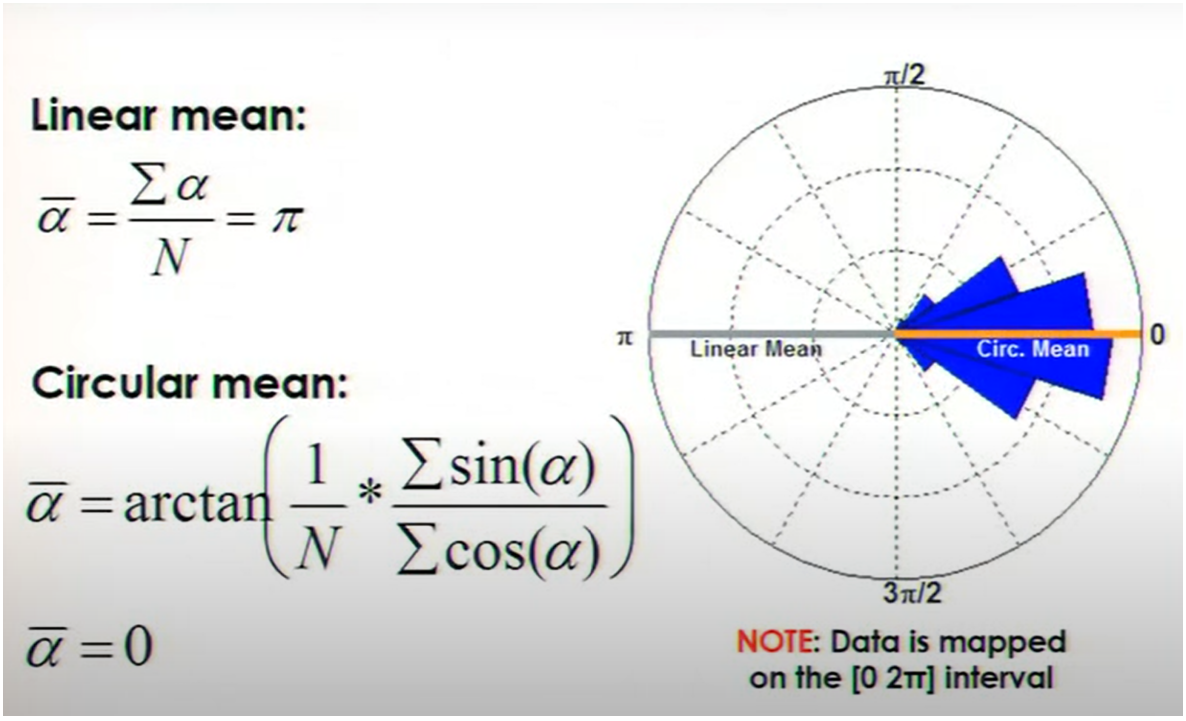

Figure 20: Directional statistics: (linear and circle mean), "Circular Statistics - Nour Malek and Frederic Simard

In the same logic the circular variance is $1 - \bar{R}$, where $\bar{R}$ is the mean resultant vector i.e. $|\bar{R} = \frac{1}{N} \cdot \sum \exp^{(} i \cdot \alpha)|$.
The resultant vector $\bar{R}$ represents the length and direction of the vector sum of all the angles in the sample. The circular variance measures the spread of the angles around the circle, with a value of 0 indicating that all the angles are located at the same point on the circle, and a value of 1 indicating that the angles are uniformly distributed around the circle.

```
def circular_variance(x, y):
    # Note: this is not my code, this code adapted from scipy opensource "scipy.stats.circvar"
    import numpy as np
    nsum = np.asarray(np.sum(~mask, axis=axis).astype(float))
    nsum[nsum == 0] = np.nan
    sin_mean = sin_samp.sum(axis=axis) / nsum
    cos_mean = cos_samp.sum(axis=axis) / nsum
    # hypot can go slightly above 1 due to rounding errors

    with np.errstate(invalid='ignore'):
        R = np.minimum(1, hypot(sin_mean, cos_mean))

    res = 1. - R
    return res
```

Figure 21: Circular variance loss

In addition to that, a loss function can formed by using sum of squared differences using "subtraction modulo 2π" (the following loss function with either modulo 180 or modulo 360.

```
def loss(x, y):
    return abs((x - y) +180) % 180 -180
```

Figure 22: Circular data loss function

Another option is to substract between points, unless they located in different semicircle, where in that case the loss will be calculated by the sum of both points from 0, or 360. e.g., loss(0, 358) = abs((0-0) - (358-360)) = 2

2.2.4

1 $\frac{\partial}{\partial x} f(x+y, 2x, z) = (\frac{\partial f}{\partial x+y} \cdot \frac{\partial x+y}{\partial x}, \frac{\partial f}{\partial 2x} \cdot \frac{\partial 2x}{\partial x}, \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial x}) = (\frac{\partial f}{\partial x+y}, 2 \cdot \frac{\partial f}{\partial 2x}, 0)$

2 $\frac{\partial f}{\partial x} f_1(f_2(f_3(...(f_n(x))) = \frac{\partial f_1}{\partial f_2}(f_2(f_3(...(f_n(x)))) \cdot \frac{\partial f_2}{\partial f_3}(f_3(f_4(...(f_n(x))))) \cdot ... \cdot \frac{\partial f_n}{\partial x}(x)$

3 $\frac{\partial f}{\partial x}(x, (f_1(...(f_n(x)))) = (\frac{\partial f_1}{\partial x}(x) \frac{\partial f_1}{\partial f_2}(x, (f_2(f_3(...(f_n)))))) \cdot \frac{\partial f_2}{\partial x}(x, (f_2(f_3(...(f_n(x))))))$

i.e. For any $0 < i < n - 1$
$\frac{\partial f_i}{\partial x}(x, (f_i(f_{i+1}(...f_n(x))))) = (\frac{\partial f_i(x)}{\partial x}(x), \frac{\partial f_i}{\partial f_{i+1}}(x(f_{i+1}(...(f_n(x)))))) \cdot \frac{\partial f_{i+1}}{\partial x}(x, (f_{i+1}(...(f_n(x))))))$

4 Denote $t(x) = x + h(x)$ and $q(x) = x + g(l(x))$
$\frac{\partial f}{\partial x}(q(x)) = \frac{\partial f}{\partial q}(q(x)) \cdot \frac{\partial q}{\partial x}(x) = \frac{\partial f}{\partial q}(q(x)) \cdot (1 + \frac{\partial g}{\partial x}(t(x))) = \frac{\partial f}{\partial q}(q(x)) \cdot (1 + \frac{\partial g}{\partial t}(t(x)) \cdot \frac{\partial t}{\partial x}(x)) = \frac{\partial f}{\partial q}(q(x)) \cdot (1 + \frac{\partial g}{\partial t}(l(x)) \cdot (1 + \frac{\partial h}{\partial x}(x)))$.