

Deep Learning for MSc 2025/2026 – week 3

exercise tasks

Prof. Alexander Binder

Week 03: A bit on generalization

Vorwort:

- understand the variability of test loss measures over draws of test sets
 - You will compute average error measures on test sets, and measure their variability over different draws of test sets
- understand the variability of selected training mappings over different draws of training sets

Task 1 – Draw samples from 2 gaussians

- implementiere eine Funktion, welche Paare (x, y) erzeugt, derart dass n_1 paare das label $y = +1$ haben, n_2 paare das label $y = -1$ haben, und dass x aus einer zwei-dimensionalen Normalverteilung stammt

$$x \in \mathbb{R}^2, m1 \in \mathbb{R}^2, m2 \in \mathbb{R}^2$$
$$p(x|Y = +1) = \text{Normal}(m1, \sigma_1^2 * I_2)$$
$$p(x|Y = -1) = \text{Normal}(m2, \sigma_2^2 * I_2)$$
$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Wie zieht man aus einer 2-dimensionalen Normalverteilung?

Entweder ihr nehmt eine Funktion fuer das Ziehen aus einer multivariaten Normalverteilung ... oder:

- torch und numpy koennen aus einer 1-dimensionalen Normalverteilung ziehen ... ask google
- falls $x_0 \sim N(0, 1)$ und $x_1 \sim N(0, 1)$, dann folgt daraus dass $x = (x_0, x_1) \sim N((0, 0), I_2)$

- falls $x = (x_0, x_1) \sim N((0, 0), I_2)$ und $y = \sigma x + \mu$ mit $\sigma \in \mathbb{R}$, dann $y \sim N(\mu, \sigma^2 I_2)$
 (ein allgemeineres Ergebnis erzielt hier: falls $x \sim N(0_n, I_n)$ und $y = Ax + \mu$ wobei 0_n der Nullvektor in n Dimensionen ist, I_n die Identitätsmatrix in n Dimensionen, und, A eine (n, n) -Matrix ist, dann $y = Ax + \mu \sim N(\mu, A^\top A)$)
- d.h. zuerst zieht ihr 2 Punkte aus einer 1-dim Normalverteilung, dann verkettet ihr diese in einen 2-dim Vektor (x_0, x_1) , danach wendet ihr eine affine Abbildung an, um $y = \sigma x + \mu$ aus x zu berechnen, und das hat die gewünschte Verteilung

Die Funktion sollte dieses Interface aufweisen:

```
def datagen(n1,n2,m1,m2,sig1,sig2, rng):
```

wobei n_1 die Anzahl Punkte, m_1 der 2-dimensionale Mittelwert (ein Vektor), $sig1$ die Standardabweichung – für $Y = +1$ ist, und n_2 die Anzahl Punkte, m_2 der 2-dimensionale Mittelwert, $sig2$ die Standardabweichung – für $Y = -1$ ist. Der Rückgabewert sollte das Format $(n_1 + n_2, 2)$ haben.

- Mit dieser Funktion generiere synthetische Klassifikationsdaten (x, y) mit $n_1 = 50 = n_2$, $m_1 = [1, 1]$, $m_2 = [-1, -0, 5]$, $sig1 = sig2 = 2.0$.
- Plotte diese in matplotlib mit 2 Farben, welche vom Label in y abhängen.

EN:

- implement a function , which generates pairs of feature and label (x, y) such that n_1 pairs have the label $y = +1$ and, n_2 pairs have the label $y = -1$ and, such that x is drawn from a 2-dimensional normal distribution

$$x \in \mathbb{R}^2, m1 \in \mathbb{R}^2, m2 \in \mathbb{R}^2$$

$$p(x|Y = +1) = Normal(m1, \sigma_1^2 * I_2)$$

$$p(x|Y = -1) = Normal(m2, \sigma_2^2 * I_2)$$

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- How to draw samples x from a 2-dim normal distribution?

Either you use a function which can directly draw samples from a multivariate normal ... or:

- torch und numpy can be used to draw from a 1d normal ... ask google
- if $x_0 \sim N(0, 1)$ and $x_1 \sim N(0, 1)$, then we obtain $x = (x_0, x_1) \sim N((0, 0), I_2)$

- if $x = (x_0, x_1) \sim N((0, 0), I_2)$ and $y = \sigma x + \mu$ with $\sigma \in \mathbb{R}$, then we obtain $y \sim N(\mu, \sigma^2 I_2)$
(a more general result states that if $x \sim N(0_n, I_n)$ und $y = Ax + \mu$ such that 0_n is the zero vector in n dims, I_n the identity matrix in n dims , and, A is a (n, n) -matrix, then we obtain $y = Ax + \mu \sim N(\mu, A^\top A)$)
- in summary: at first you draw 2 samples from a 1-dim Normal distribution, then you concatenate them into a 2-dim vector (x_0, x_1) , then you apply an affine mapping, in order to compute $y = \sigma x + \mu$ from x . y has the desired distribution

The function should have this interface:

```
def datagen(n1,n2,m1,m2,sig1,sig2, rng):
```

where $n1$ is the number of samples, $m1$ the 2-dim mean vector, $sig1$ the standard deviation – for $Y = +1$, and $n2$ die Anzahl is the number of samples, $m2$ the 2-dim mean, $sig2$ the standard deviation – for $Y = -1$.

The return value should be an array of shape $(n1 + n2, 2)$.

- use the above function to draw samples with
 $n1 = 50 = n2$, $m1 = [1, 1]$, $m2 = [-1, -0, 5]$, $sig1 = sig2 = 2.0$.
- plot those using matplotlib with 2 colors which depend on the label

Task 2 – Test errors for one particular classifier

Generate samples (x, y) . Measure the (binary) classification accuracy (= binaere Klassifikationsgenauigkeit) for the following mapping

$$a = w \cdot x + b, \quad w = [2, 1.5], \quad b = -3.0/8 \\ y = sign(a)$$

the binary classification accuracy (= binaere Klassifikationsgenauigkeit) for a single point is:

$$1 - 1[f(x) \neq y] = 1[f(x) = y]$$

Repeat the experiment as follows:

- Compute the averaged (binary) classification accuracy for $n1 + n2$ many samples drawn using `def datagen(n1,n2,m1,m2,sig1,sig2, rng):`.
- Repeat the experiment 100 times. This results in an array with 100 numbers of averaged (binary) classification accuracies. Compute the mean and standard deviation over that array .

- Repeat the experiment 100 times for $n1 = n2 = 100$. Compute the mean and standard deviation over that array .
- Repeat the experiment 100 times for $n1 = n2 = 1000$. Compute the mean and standard deviation over that array .

something to observe

Compare the means and the standard deviations of the mean test data accuracies for the 3 choices of $n1 = n2 = 10, n1 = n2 = 100$ and $n1 = n2 = 1000$.

What can you observe ?

The classification mapping used here should result in the minimal expected loss (in case of $sig1 = sig2$)

Task 3

Finish task 2 from the last week (NNs old school (simple)).

Task 4 - Overfitting und Varianz des Trainingsergebnisses

In dieser Aufgabe implementieren Sie den 1-NN classifier (1-naechster Nachbar). Er macht eine Vorhersage wie folgt:

$$f(x) = \sum_i y_i 1[x_i = \operatorname{argmin}_i \|x_i - x\|]$$

Algorithmisch kann man ihn wie folgt implementieren:

- gegeben sei ein zu klassifizierender Vektor x und die Trainingsdaten $D = ((x_0, y_0), \dots, (x_{n-1}, y_{n-1}))$
- berechne fuer alle Trainingsdaten x_i den Abstand $\|x - x_i\|$ zum zu klassifizierenden Vektor x
- finde den index i derart, dass $\|x - x_i\|$ minimal ist ueber alle Trainingsdaten, d.h. den index des naechsten Nachbarn
- klassifizierte x durch das Label y_i , welches zum naechsten Nachbarn gehoert

Warum wollen wir mit dem arbeiten? Per definition hat 1-NN die Accuracy 1 und den Fehler 0 auf den trainingsdaten (jeder ist sich selbst der naechste!). Der overfittet aus Prinzip.

Ausserdem kann man diesen in 2D schoen mit voronoi plots visualisieren.

- erzeugen sie $n1 = n2 = 100$ Trainingsdaten fuer D , welche sie in dem 1-NN classifier einsetzen werden
 - erzeugen sie $n1 = n2 = 100$ Testdaten . Berechnen Sie die (binaere) Klassifikationsgenauigkeit auf diesen Testdaten fuer den 1-NN classifier, welcher mit den Trainingsdaten aus D "trainiert"/initialisiert worden ist
 - Wiederholen sie das experiment 5 oder 10-mal (jedes mal auch neue Trainingsdaten ziehen).
- Die gemittelte (binaere) Klassifikationsgenauigkeit sollte niedriger sein (+ der gemittelte 0-1-Verlust hoher) als bei der Abbildung aus Task 2.
- Bonus: nutzen sie scipy um einen Voronoi plot der Regionen des 1-NN Klassifikators zu erzeugen und vergleichen sie mit der Entscheidungsgrenze, welche durch $f(x) = 0$ bei der Abbildung aus Task 2 gegeben ist. Das ist optisch nett :)

EN: You will implement the 1-NN (nearest neighbour) classifier. It predicts as follows:

$$f(x) = \sum_i y_i [x_i = \operatorname{argmin}_i \|x_i - x\|]$$

It can be implemented as follows:

- let x be a vector to be classified and let $D = ((x_0, y_0), \dots, (x_{n-1}, y_{n-1}))$ be the training samples
- compute for all training samples x_i the distance $\|x - x_i\|$
- find the index i such that $\|x - x_i\|$ is minimal among all training samples, i.e. the index of the nearest neighbor for the test sample x within the training samples
- classify x by the label y_i , which belongs to the nearest neighbor

Why using this one ?

The 1-NN classifier has by definition the accuracy 1 and 0-1 error 0 on the training data.

It can be nicely visualized using voronoi plots.

- create $n1 = n2 = 100$ training samples for D
- create $n1 = n2 = 100$ test samples . Compute the 0-1 accuracy and the 0-1 error on the test samples for the 1-NN classifier which was "trained"/initialised using D as training samples
- Repeat the experiment 5 oder 10 times (not only draw new test samples, this is including a new draw of training samples every time).

Your classification accuracy should be lower (and the 0-1 error higher) than for the predictor in task 2.

- Bonus: use scipy to make a Voronoi plot of the regions of the 1-nn classifier and compare it to the boundary created by the mapping in Task 2.