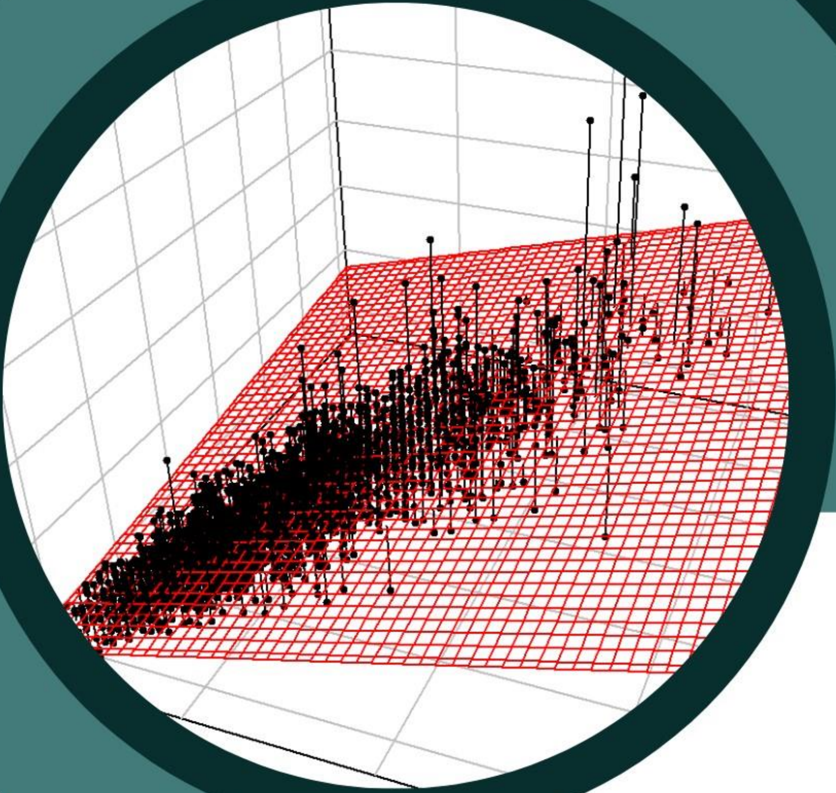
AUDREN ADRIEN
PHAN BENJAMIN
2023-2024

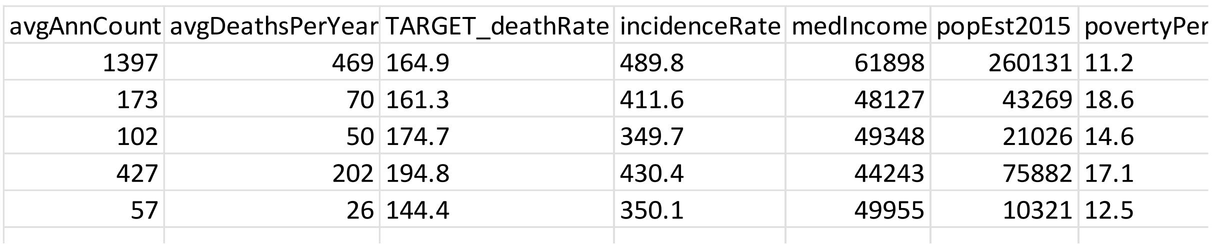


Deep Learning:
Homework 1

Cancer Death Rate Prediction

INTRODUCTION

The goal of this project is the prediction of future cancer death rate in United States based on available data which includes for instance poverty indicators, location where data has been collected, ethnicity indicators, marital status, age, employment statistics and many others with obviously each time a corresponding death rate value.



avgAnnCount	avgDeathsPerYear	TARGET_deathRate	incidenceRate	medIncome	popEst2015	povertyPer
1397	469	164.9	489.8	61898	260131	11.2
173	70	161.3	411.6	48127	43269	18.6
102	50	174.7	349.7	49348	21026	14.6
427	202	194.8	430.4	44243	75882	17.1
57	26	144.4	350.1	49955	10321	12.5

Table 0.1: First occurrences of raw data.

During this project we will compare different models of prediction: multi-linear regression, neural network regression with successively one more hidden layer to reach a total of five different models.

First, the raw data will be studied in order to extract a maximum of its potential and avoid losing information. After a step of normalization which will enable us to use statistics methods efficiently, we will be able to process the data for the training of our models. Eventually, after wisely calibrated our hyperparameters in order to obtain the best prediction, we will come back on what defines “the best”, we will compare the results to select the best model to perform the prediction of cancer death rate.

1) The dataset contains 3047 data samples.
2) The dataset is aggregating many indicators from various states and counties on cancer and patients. The dataset is trying to establish a relation between these indicators and a cancer death rate.
3) For the initial CSV file, the minimum value is 0 and the maximum is 10170292 (simple iteration through rows and columns with numerical values).
4) There are 34 features for each sample.
5) Cf. Imputation part.
6) The label “TARGET_deathRate”.
7) Cf. Split part.
8) Cf. Standardization & Normalization part.

I. DATA

A. Model Relevance

The raw data document includes a total of 3047 examples (rows) with 34 features (columns). The first interrogation we could have: Is a prediction model relevant? In other words, Is the correlation between data and what we want to predict sufficient to hope to predict future values with correctness?

The figure 1.1 represents how a feature behaves depending on the evolution of a second feature.

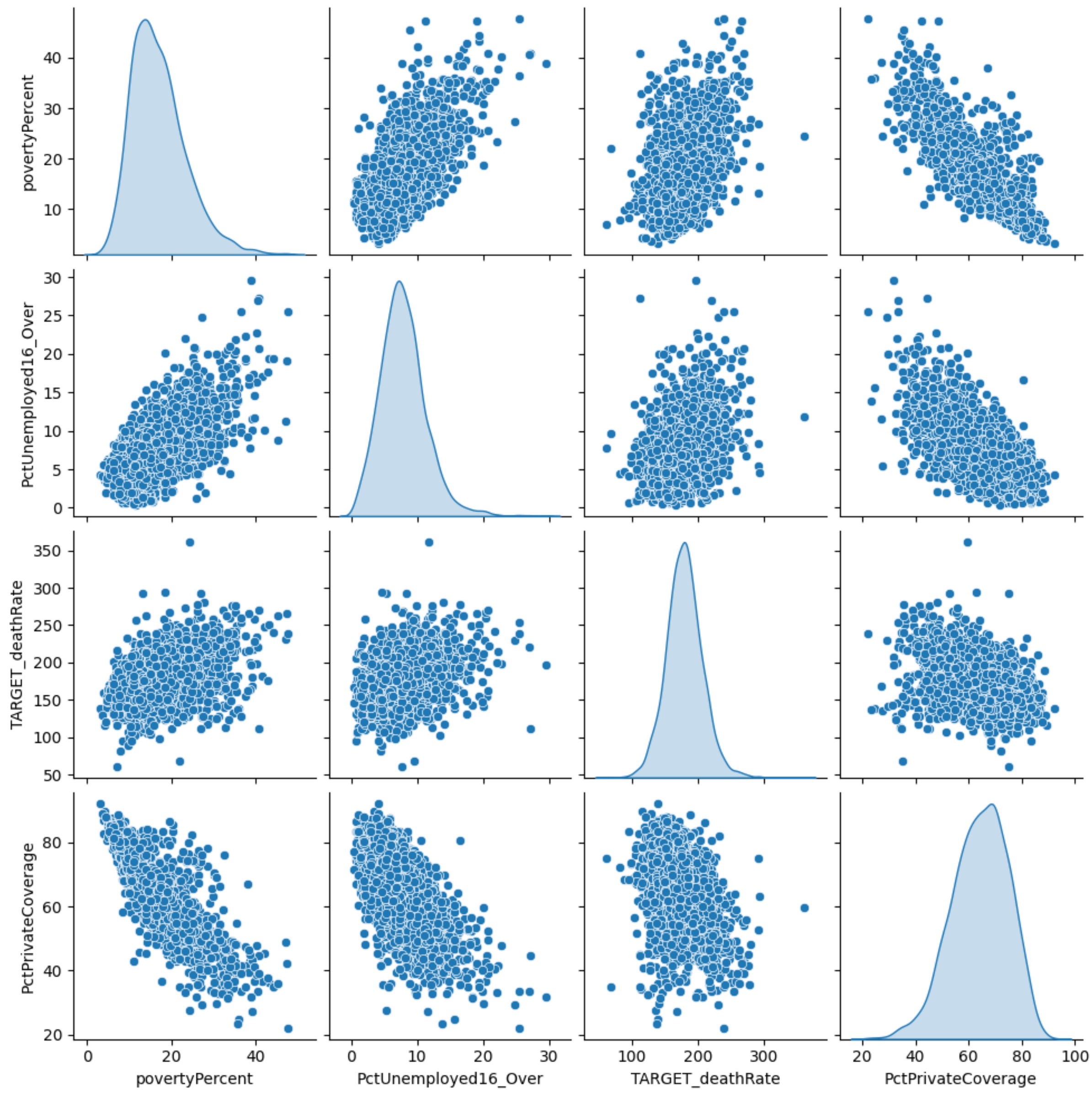


Figure 1.1: 2D visualization of features.

What we are looking for is a sort of trend when the death rate is on the y axis. The third row clearly shows a distribution of data correlated with some features. Of course, at this time, we cannot state the poverty percent is direct cause of mortality but based on their correlation we can expect to a same behavior of these two features in a broader sense.

B. Data Splitting

Now that we know a model of prediction is relevant, we need to separate our data set between a set of training that will be used to configure the five models, and a set of tests to evaluate the performance of the models.

We first need to verify that the data inferences/examples are basically the same. To take an example, we could imagine a set of physics measures to predict the weather. Some measures have been taken in summers and others in winter. Therefore, if we do not pay attention when splitting the original data set, we could have a major part of the summer data in the training set and a major part of the winter data in the testing set. When evaluating our model, the performances obtained would be bad. Indeed, we trained a model on summer data and evaluate it on winter data. To avoid this phenomenon, we need to make sure our original data set is not constituted of subsets. In this case we should divide proportionally each subset between the training set and the testing set.

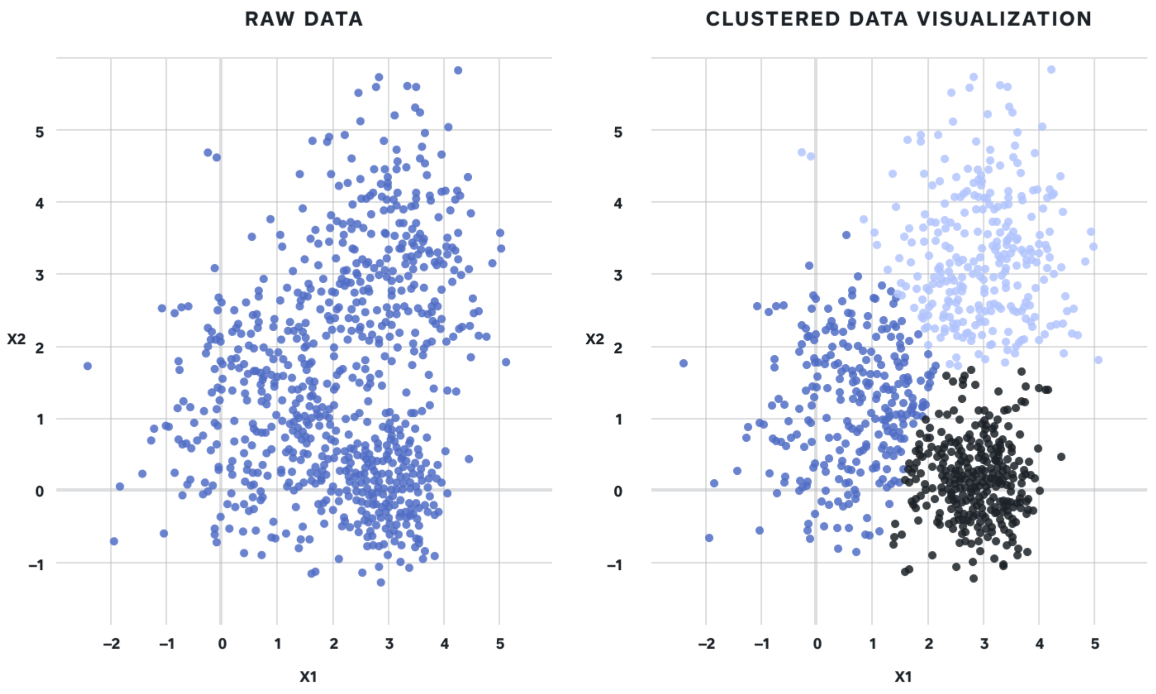


Figure 1.2: Example of subsets problem.

A K-means algorithm could be used to avoid the problem mentioned above. To make sure we are not facing to this type of problem with our data set we can represent our data in a three-dimensional space but as it includes thirty-four features, we will use a Principal Component Analysis (PCA) which will enables us to represent data on the three most relevant axis.

The PCA consist of finding the axes that are maximizing the data dispersion on these previous. This is equivalent to solve the following problem (which is simply obtained using Pythagorean theorem):

$$\begin{cases} \max\limits_{u} \phi = u^T X^T X u \\ u^T u = 1 \end{cases} \qquad \textit{with } X \textit{ the reduced centered data matrix}$$

Which is itself equivalent to find the eigen vectors u_i associated with the highest eigenvalues of the covariance matrix. The unit vector criteria ensure unicity of the solution.

We can draw the share of inertia of the data explained by each eigenvalue thanks to a singular value decomposition (SVD). Since we know the data matric can be reconstituted from this decomposition:

$$X \approx \hat{X}_q = \sum_{i=1}^{q} \sqrt{\lambda_i}\, v_i {u_i}^T$$

$$(order\ q\ approximation)$$

$$Let\ \ X \in \mathbb{R}^{n\times p} \qquad then:\ \hat{X}_q = V_1 S_1 U_1^T \qquad with\ V_1 \in \mathbb{R}^{n\times q},\ S_1 \in \mathbb{R}^{q\times q}, U_1^T \in \mathbb{R}^{q\times p}$$

To operate a dimension reduction on data matrix, we only need to multiply X with U_1, which project data on principal axes, and then keep the m first columns of $X_p = X.U$ depending on the number m of dimensions we want to keep.

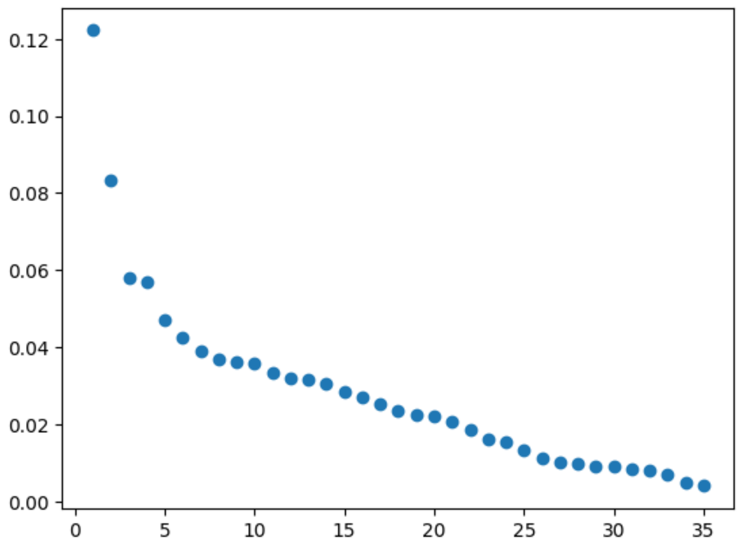


Figure 1.3: Share of data inertia explained by principal axes.

Note that the sum of all eigenvalues of covariance matrix $X^T X$ is equal to one.

With only the first three axes, we almost reach a third of the total inertia. Let represent the data on these axes.

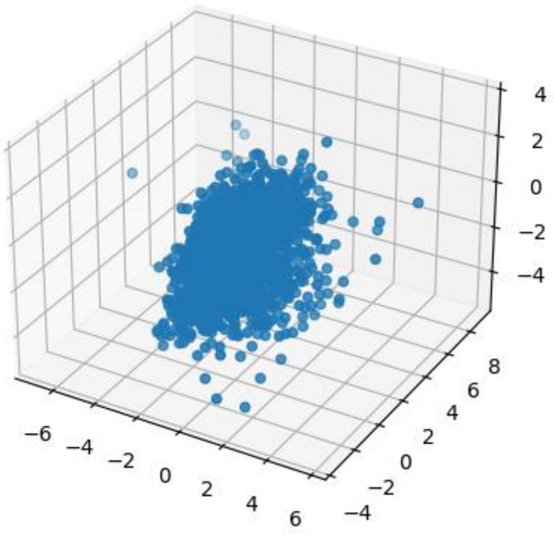


Figure 1.4: Data projection on the first three axes.

As the projection shows us, a clustering step before splitting data would be useless. Indeed, a major part of data seems to be aggregated to the same place which means each example is globally representative of all the data.

A commonly used fraction is 80/20, respectively for training and testing set. There will be no deviation from the rule for this project.

C. Data Normalization

Some columns of the data set are not completely numerical. Which is a problem because our model is based on statistic transformations that only accept numerical values. There are two columns concerned. The first one: binnedInc is composed of two values. We only need to separate it with two different columns. The second one is a little bit trickiest. It is important to not lose any information. Therefore, we will use cartesian coordinates of each place and we will use two columns. One for x axis and the other one for y axis.

D. Data Imputation

Some columns do not contain any value for certain rows.

Feature	**Number of "none" value**
PctPrivateCoverageAlone	609
PctSomeCol18_24	2285
PctEmployed16_Over	152

We decided to not use the column "PctEmployed16_Over" for the rest of the project as there are too many missing values, and an imputation model would be unable to predict accurately enough.

Regarding the two other columns, we decided to move forward with an imputation based on neural network regression (using the same models that those we are presenting below) to predict the missing values. We achieved to reach a correlation coefficient of 0.86 on test data for the label with 152 missing values.

E. Data Standardization

Nothing extraordinary, the data matrix must be reduced and centered. Center the data avoid getting a bias problem with the model and the reduction step avoids scale problems.

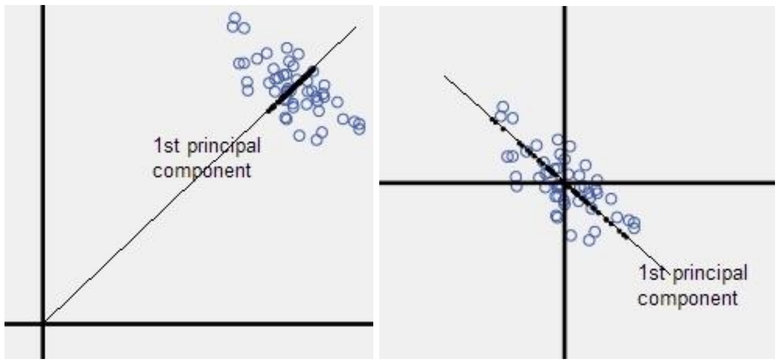


Figure 1.5: Centered and uncentered data comparison.

$$X_{norm} = \frac{X - E(X)}{Var(X)}$$

II. MODEL

Our goal is to predict as accurate as possible a cancer death rate from the initial data set. Our data has a CSV file shape, and we want an output with a numerical value. The type of model we must use to perform this task is a regression. There will be five different models. The first one will be a simple linear regression and the four other, neural network regressions with increasing number of hidden layers. We will dive in their architecture more deeply in the next parts.

A. Linear Regression

We are presenting here a proof for the multi-variables' linear regression formula:

$$Let\ X\ be\ the\ features\ matrix\ with\ an\ additional\ column\ with\ ones\ only\ at\ the\ end.$$

$$And\ Y\ the\ pediction\ vector\ (in\ this\ case\ the\ death\ rate\ column),$$

$$i.e\ X[1,p] = data\ and\ X[p+1] = \begin{pmatrix} 1 \\ \dots \\ 1 \end{pmatrix}$$

$$we\ are\ searching\ for\ a\ weights\ matrix\ \ w \in \mathbb{R}^{p+1}\ \ such\ as\ Xw = \hat{Y}$$

$$with\ \hat{Y}\ as\ close\ as\ possible\ to\ Y\ in\ a\ mean\ squared\ sense.$$

$$This\ is\ an\ unconstraint\ linear\ optimization\ problem:$$

$$arg\min_{w} \frac{1}{2}(Xw - Y)^2 <=> arg\min_{w} \frac{1}{2}[(Xw|Xw) - 2(Xw|Y) + (Y|Y)]$$

$$<=>\ arg\min_{w} \frac{1}{2}[(Xw|Xw) - 2(Xw|Y)]$$

$$we\ can\ introduce\ the\ Lagrangian\ of\ the\ probem\quad \mathcal{L} = \frac{1}{2}(Xw|Xw) - (Xw|Y)$$

$$When\ reaching\ the\ optimum\ of\ our\ problem,\ \ \frac{\partial\,\mathcal{L}}{\partial w} = 0$$

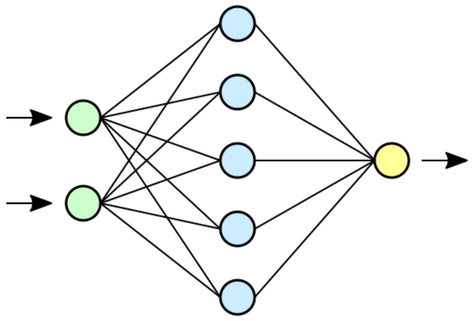
$$=> X^T X w^* - X^T Y = 0$$

$$Finally,\ \ w^* = (X^T X)^{-1} X^T Y$$

B. Neural Network Regression

Input Layer:

Our features without the one we want to predict.



Output Layer:

The value of the model prediction

For the four models, the number of hidden layer (blue) is ranging from 1 to 4. For each hidden layer there will be a pre-defined number of nodes. Every node of the model is fully connected. Let's find out how many parameters we should have for a model: ANN-fourL-32-16-8-4

When considering what was said in the previous parts, we have 34 features -1 feature to predict -1 unexploitable feature +2 divided column (BinnedInc and Geography)

34 times 32 weights + 32 bias

32 times 16 weights + 16 bias

16 times 8 weights + 8 bias

8 times 4 weights + 4 bias

4 weights + 1 bias (final layer for regression in yellow)

The result is 1825 trainable parameters. To optimize them we will use a stochastic gradient descent.

$$w' = w - \eta \nabla_w f(w) \quad \textit{if } w \textit{ represents the weights vector and } f \textit{ the loss function}$$

We will come back on the proof later.

This process will be used during what we call a backward propagation to optimize the weights of the model. For each epoch, we compute the output of every examples. We then spread the partial derivative of the loss function/criteria with respect to each weight and using the previous output of the model.

III. OBJECTIVE

There are many different objective functions (Likelihood/Log Loss, Hinge Loss, MSE, MAE, …)

The most simple and easy to understand/feel is the mean squared error function. This is simply defined by the sum (on the examples) of Euclidean distance between the model prediction and the real value.

$$MSE = \frac{1}{n} \sum_{n\ examples} \left(w_1x_1 + w_2x_2 + \cdots + w_px_p - y\right)^2 = \frac{1}{n}\|Xw - Y\|_2^2$$

IV. OPTIMIZATION

Our objective is to update the trainable parameters in a way that the loss function is decreasing the most. Indeed this means our model can predict better the expected value, he is learning somehow.

Let's look at the idea behind the gradient descent:

we are searching for the direction u (unit vector) such as a criteria f is decreasing the most

$$i.e\ arg \min_{\substack{u \\ u^Tu=1}} \frac{\partial}{\partial \alpha} f(x + \alpha u) <=> arg \min_{\substack{u \\ u^Tu=1}} u^T \nabla_x f(x) \quad \textit{using the chain rule}$$

$$then, using\ one\ of\ the\ dot\ product\ expression \quad arg \min_{\substack{u \\ u^T u=1}} \|u\|_2 \|\nabla_x f(x)\|_2 \cos(\theta)$$

$$And\ because\ \|u\|_2 = 1\ and\ \|\nabla_x f(x)\|_2\ do\ not\ depends\ on\ u,$$

$$arg \min_{\substack{u \\ u^T u=1}} \cos(\theta)\ \ with\ \theta\ tha\ angle\ between\ u\ and\ the\ gradient$$

$$This\ expression\ is\ minimized\ when \quad \theta \equiv \pi\ [2\pi]$$

$$which\ means\ u\ has\ the\ opposite\ direction\ of\ the\ gradient$$

$$We\ can\ now\ update\ our\ weights\ with\ a\ direction\ where\ the\ criteria\ is\ decreasing:$$

$$w' = w - \eta \nabla_w f(w)$$

The stochastic gradient descent (SGD) is only computing the gradient of the loss function from a random sample of the data set. This enables to estimate the real gradient and save computational resources when working with a large training set.

V. MODEL SELECTION

By comparing MSE scores for training and testing dataset, we can easily determine if a model is under/over fitting on the data. If, for instance, a model performs well on training but the MSE score is much higher on test set then the model is overfitting. If the performances are relatively bad on both set, then the model is underfitting.

During the training step, even if the accuracy on training data is increasing, the accuracy on validation data can decrease with the number of epochs. This is the direct consequence of overfitting. An early stop can be used to avoid this problem and keep the best parameters for the model.

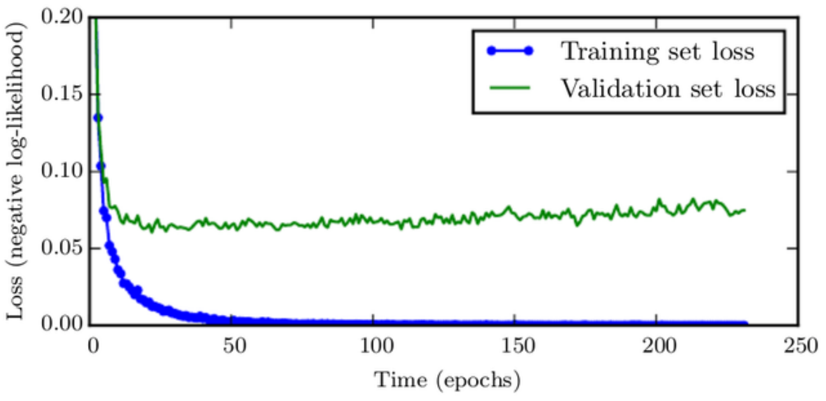


Figure 5.1: U-shaped curve, deep learning-Goodfellow-Bengio-Courville

To set up an early stop, which is equivalent to fix a maximum number of epochs, and optimize other hyperparameters such as the learning rate, we can use the Keras Tuner API.

Regarding the optimizer, a method with an adaptative learning rate would be better than a stochastic gradient descent. For instance, the learning rate of AdaGrad algorithm follows the gradient variations. Add of a momentum could increase the performance by avoiding local minimums (Cf. Adam).

VI. MODELS PERFORMANCES

Model	MSE
Linear Regression	0.46
ANN-oneL-16	0.44
ANN-twoL-32-8	0.45
ANN-threeL-32-16-8	0.46
ANN-fourL-32-16-8-4	0.46

Table 6.1: Models 'means squared error.

As shown in the table 6.1, the number of layers doesn't change the MSE on this data set. It means we don't need to increase the number of layers, and which reduces the time complexity.

The loss function evolution trough epochs does not clearly show any overfitting. Indeed, the loss function on test data reach a "plateau" even if the loss is decreasing for training. Since the loss on test data does not significantly increase which implies that the model does not become that much specific to the training data. The variance of the model remains relatively low. This is not surprising when considering the performance (R^2) on training data which is not incredible.

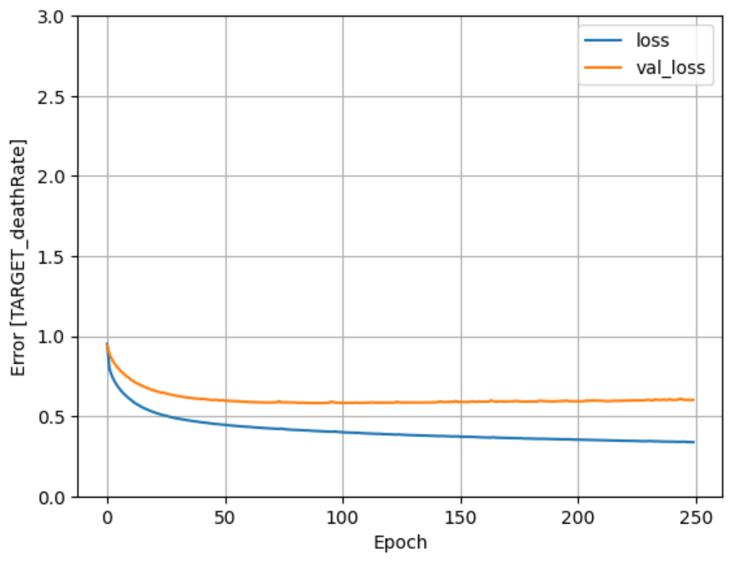


Figure 6.1: Loss function - 3 layers - LR = 0.001

Model	LR: 0.1	LR: 0.01	LR: 0.001	LR: 0.0001
ANN-oneL-16	0.48	0.53	0.56	0.51
ANN-twoL-32-8	0.48	0.55	0.50	0.49
ANN-threeL-32-16-8	0.53	0.43	0.52	0.51
ANN-fourL-32-16-8-4	0.48	0.51	0.51	0.54

Table 6.2: Models 'R^2 with different learning rates.

A high learning rate typically results in a shorter training time, requiring fewer epochs to converge to a satisfactory model performance. Nonetheless, the model may overshoot and oscillate around the minimum of the loss function.

A low learning rate typically results in a slower learning process, requiring more epochs to converge to an optimal solution because the model updates take smaller steps toward minimizing the loss function.

In the table 6.2, we only kept the results of the best correlation coefficient. It seems that LR = 0.01 may produce good results in some trials, but it might not perform consistently well across all trials, where LR = 0.001 was more consistent but needed more time.