RIGA TECHNICAL UNIVERSITY

Faculty of Information Technology and Computer Science

Institute of Applied Computer Systems

Department of Artificial Intelligence and System Engineering

Iurii Lepesevich

Academic Bachelor Study Program “Computer Systems”

Student ID 191ADB051

Analytical comparison of normalization functions in regression tasks

Scientific adviser

Doctor of science, Researcher

Ēvalds, Urtāns

RIGA TECHNICAL UNIVERSITY

Faculty of Information Technology and Computer Science

Institute of Applied Computer Systems

Department of Artificial Intelligence and System Engineering

Work Performance and Assessment Sheet of the Bachelor Paper

The author of the graduation paper:

Student Iurii Lepesevich

\_\_\_\_\_\_\_\_\_\_\_\_\_ (signature, date)

The graduation paper has been approved for the defence:

Scientific adviser:

Doctor of Science, Researcher, Ēvalds, Urtāns

\_\_\_\_\_\_\_\_\_\_\_\_\_ (signature, date)

abstract in engish

Abstract in latvian

Table of contents

[1. Deep machine learning 8](#_Toc101323438)

[1.1. Basic architecture 8](#_Toc101323439)

[1.1.1. Linear layer 8](#_Toc101323440)

[1.1.2. Activation function 9](#_Toc101323441)

[1.2. Normalization functions 13](#_Toc101323442)

[1.3. Loss functions 15](#_Toc101323443)

[1.4. Optimization algorithm 17](#_Toc101323444)

[1.4.1. The gradient 18](#_Toc101323445)

[1.4.2. Stochastic gradient descent (SGD) 19](#_Toc101323446)

[1.5. Backpropagation algorithm 22](#_Toc101323447)

[1.6. R-Squared 23](#_Toc101323448)

[1.7. Regression tasks 24](#_Toc101323449)

[1.7.1. Inputs – Outputs 25](#_Toc101323450)

[1.7.2. Example of architectures 25](#_Toc101323451)

[1.7.3. Loss functions – MSE, MAE, Huber Loss 26](#_Toc101323452)

[1.8. Importance of normalization functions 27](#_Toc101323453)

[2. Methodology 27](#_Toc101323454)

[2.1. Datasets 27](#_Toc101323455)

[2.1.1. Dataset - Weather in Szeged 28](#_Toc101323456)

[2.1.2. Datasets – CalCOFI 36](#_Toc101323457)

[2.2. Normalization functions 39](#_Toc101323458)

[2.3. Architecture 39](#_Toc101323459)

[2.4. Metrics 39](#_Toc101323460)

[2.4.1. R-Squared 39](#_Toc101323461)

[3. References 40](#_Toc101323462)

introduction

Artificial Intelligence

Artificial Intelligence (AI) is a specific field in Computer Science that implies solving cognitive tasks that are created for human intelligence. Such tasks could be described as:

* Learning.
* Solving problems.
* Pattern recognition.

In modern world AI is widely used in different spheres. We can see development of this technology in such fields as healthcare, military, social field, etc.

For the last few years development in the discipline of statistical data has created such new domains as: Machine Learning (ML) and Deep Learning (DL). Those domains created background for further improvement of AI technology.

Deep Learning

Deep learning is a sub-field of AI. DL uses multilayer algorithms to deep analysis of inputted data. Such algorithms recognize patterns and find relationship between different features of data. If a huge dataset would be used, DL algorithm could recognize dependencies between individual variables.

As an example, image recognition could be given. DL is able to distinguish important parts of particular image and predict class of that image (such as is bird or fish is located on the image).

Deep learning models are widely used in such technologies as self-driving cars, cancer prediction, business production optimization, speech recognition, etc.

Aim

Aim of this paper is to perform analytical comparison of different normalization functions. It is required to create methodology so optimal normalization function would be applied in particular problem.

In this graduation paper – basic concepts of DL field will be described, several experiments will be performed and compared. Based on experiments results – conclusion will be made and required guidelines for normalization function will be created.

Section Content

Firstly, theoretical background will be described. There will be information about basic concepts of DL field:

* Linear layers.
* Loss functions.
* Backpropagation algorithm.
* Optimization algorithms.
* Metrics (NRMSE, R-2 Score).
* Regression tasks.
* Importance of normalization functions.
* Learnable methods.
* Basic architecture.
* Datasets.

After theoretical part – experimental part will be shown. Experimental results will be verified in verification part.

# Deep machine learning

## Basic architecture

### Linear layer

Linear layers are the basic structures of the DL model. It can take a particular or multiple argument as input and generate a particular or multiple argument as output. Some layers have no state, but in more frequent cases, the layers have states: the weight of the layers, one or several tensors learned in a stable gradient descent, and together contain the network knowledge (Charniak, 2018).

A common view of linear function is next:

where F(x) – output of the function that depends on argument ;

– coefficient by which independent variable is multiplied;

– coefficient which is added to independent variable .

Every linear function has view as Formula (1.1). For example, if there is such equation as: (you can see there are 2 arguments and 2 arguments) – it can be converted to such view: .

Below you can see graph of Formula (1.1):

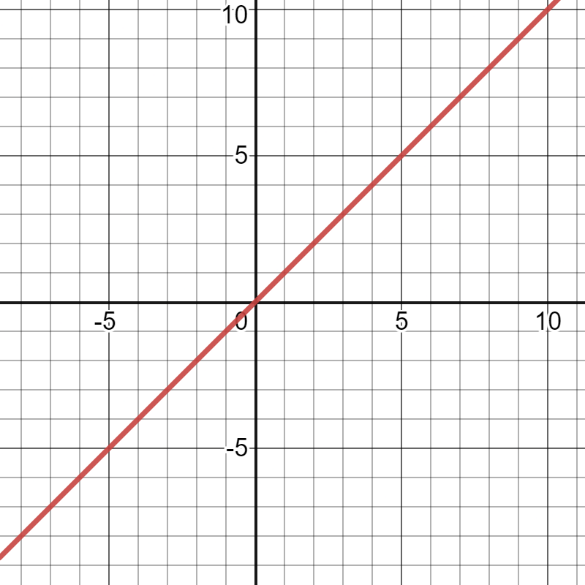


Fig. 1.1. Example of linear graph

As it is seen of Figure (1.1), every linear graph can be described as a straight slope. argument states the angle of the slope and argument states how far the line is from axes.

### Activation function

The activation function is a function that is applied to neurons at the top of the layer when predicting. An activation function is an overly simplified function, which can take a number and return another number. However, we have an infinite functions, not all of which are useful as activation functions. There are several restrictions to the functions that make them activation functions. As you can see, using functions outside these constraints is generally a bad idea (Vasilev *et al.*, 2019).

Speaking of constraints, some rules can be called several rules. The first is that the activation function in the domain must be continuous and infinite. You shouldn't put numbers in functions, and the function will not generate outputs for that value (Liang *et al.*, 2021).

As an example, let’s look at Figure (1.2) and Figure (1.3):

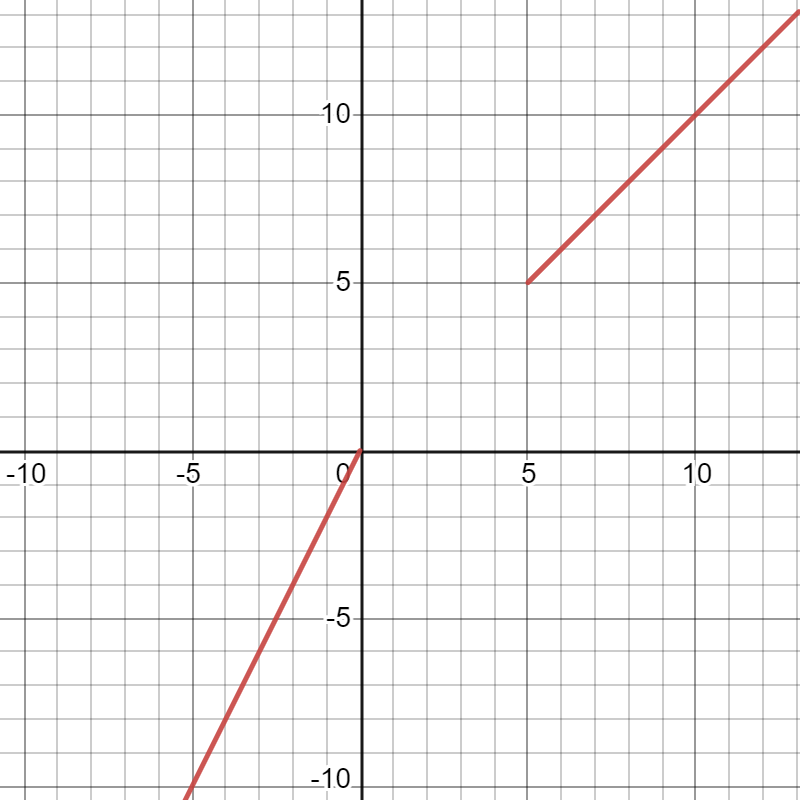


Fig. 1.2. Example of discontinuous function

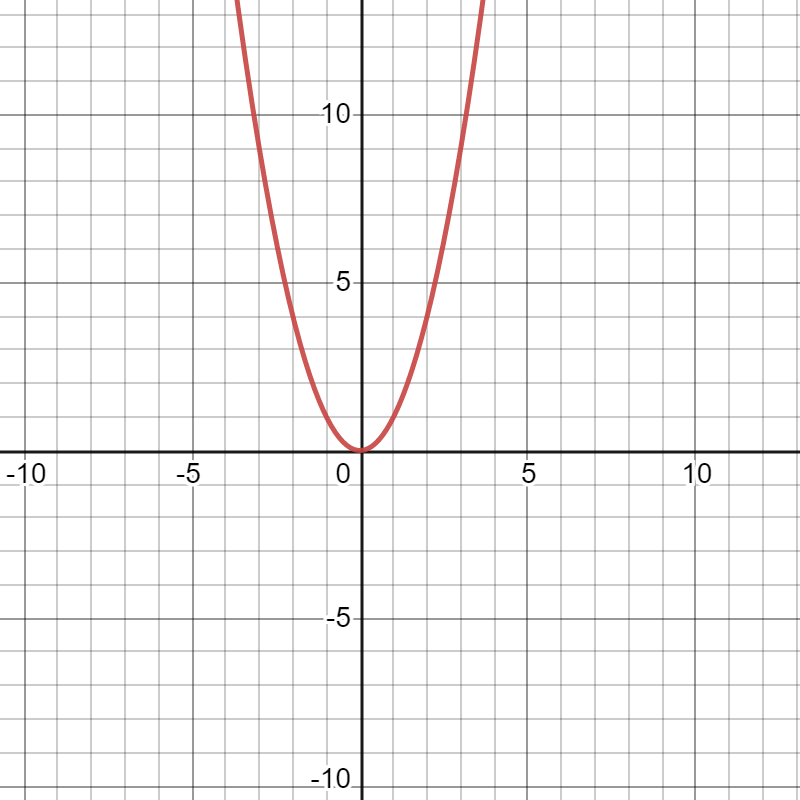


Fig. 1.3. Example of continuous function

As it is seen, on the first example, Figure (1.2) there are some values of that has no values of . This is a graph of an Formula (1.2):

where – the function of value;

– independent variable.

Discontinuous activation functions that have some ‘empty’ values of would be a horrible example for a DL model (Trask, 2019).

Talking about an alternative – on the Figure (1.3), you can see function that is continuous. This is a graph of an Formula (1.3):

Where – the function of value;

– independent variable.

This activation function seems very nice as no ‘empty’ values are on it’s domain and no problems would occur during learning algorithm.

The next important rule is that the activation function must be monotonous (it will never change direction) (Trask, 2019).

In Figures (1.3) and (1.4), examples of monotonic functions and non-monotons can be seen:

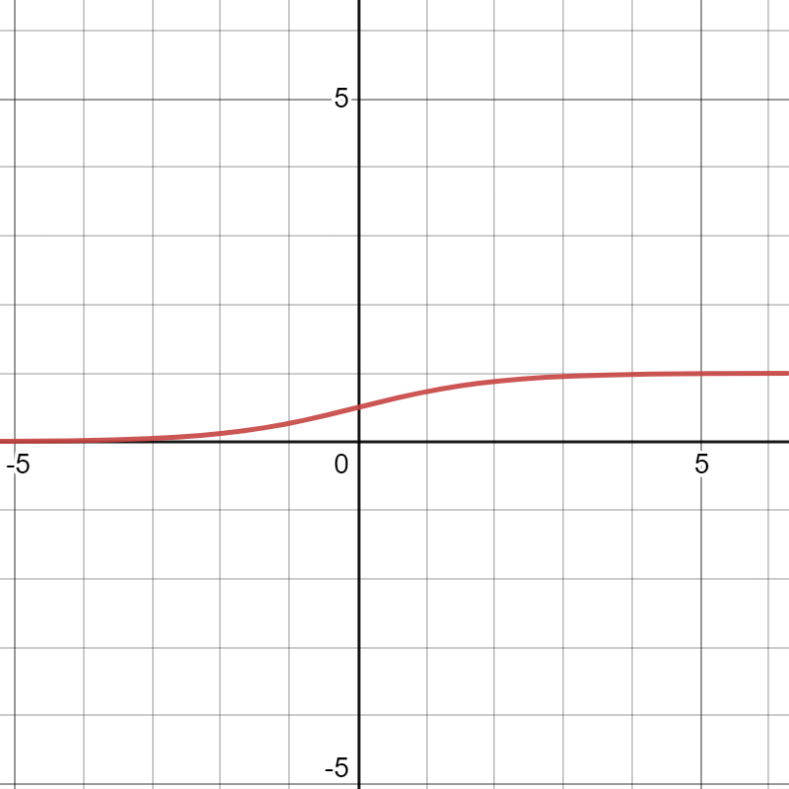


Fig. 1.4. Example of sigmoid function

Function on Figure (1.3) increases and decreases, in other words, it changes direction though it’s domain. On the other side, Figure (1.4) shows sigmoid functions that is monotonic. It increases from 0 to 1 though the whole slope.

Difference between those 2 types of functions could also be described as non-monotonic function has one or several places where for 2 or more values of there will be only 1 value of (in Formula (1.2) – for , ). On the other hand, monotonic equation assumes that for 1 value of independent variable there can only be 1 value of dependent variable (Trask, 2019).

As (Trask, 2019) states: “This particular constraint isn’t technically a requirement. Unlike functions that have missing values (noncontinuous), you can optimize functions that aren’t monotonic”.

Third rule is about activation function being non-linear. To achieve nonlinearity, neurons must allow selective correlation to input neurons. Therefore, a very negative signal from a particular input to a neuron can reduce the correlation of each input to each input. On the other hand, a function that looks like a straight line measures the weighted average that enters. Scaling increases the collective correlation represented by higher or softer values. However, activation does not allow a weight to affect the correlation between neurons and other weights. Given the activation function of a neuron, you want to increase or decrease the correlation between one received signal and all other received signals (Nielsen, 2015).

Examples of such functions can be seen on Figures (1.1) and (1.4). Linear function that is based on Formula (1.1) just scales the weighted average, however, function illustrated on Figure (1.4) allows to see the correlation of the inputted signal to all other signals.

The last but not least, as we are talking about DL as n algorithm that is performed by computer and that takes huge dataset as an input, activation functions are considered to be low on computational resources. (Trask, 2019) states: “Many recent activation functions have become popular because they’re so easy to compute at the expense of their expressiveness (relu is a great example of this)”.

Relu equation is based on Formula (1.4) and can be seen on Figure (1.5):

where – the function of value;

– independent variable.

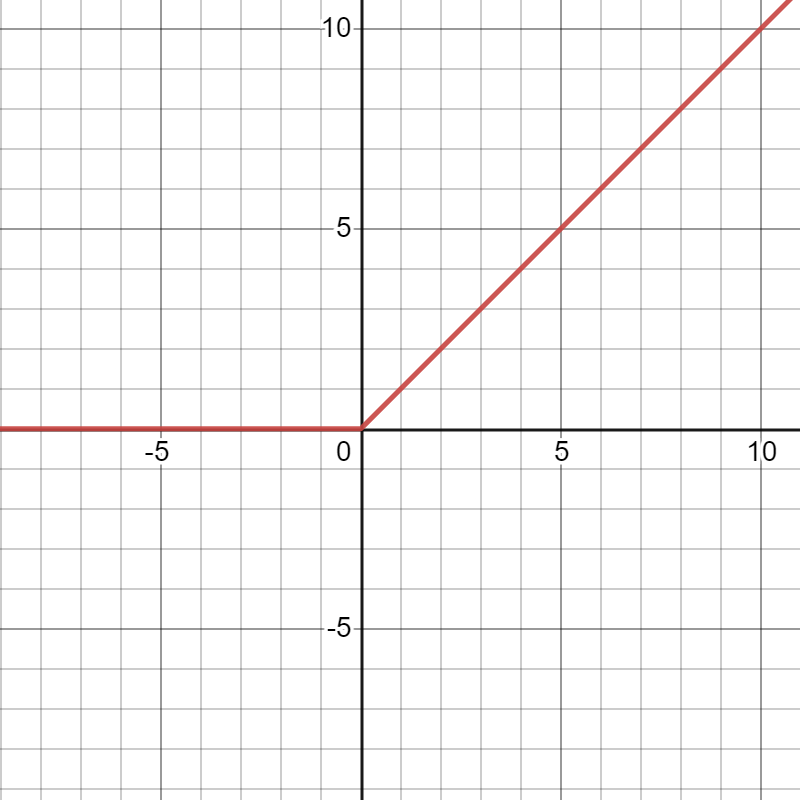


Fig. 1.5. Example of ReLU function

## Normalization functions

To understand, why we need normalization – let’s take an example of a model that predicts house prices. In such dataset – there will be a lot of features of a house, such as: number of floors, location, area, number of rooms, height of ceiling, etc. In such way values of those features would vary from ranges like (0 - 4) and (10000 - 1000000).

Usually, it is not he best practice to feed in big values in a neural network, such as if inputted value of a feature is a large integer compared to weight of that feature or data that is heterogeneous (if one of the features varies in a range of 0-1 and other in range 0-10000). Not normalized data could trigger “gradient explosion” and your network won’t converge (Buduma and Locascio, 2017).

Problem described above could be solved though applying normalization function on your dataset. Usually Formula (1.5) is used. It makes all the values of dataset (values of features) to vary in the range from 0 to 1:

where – new value of the feature (after recalculation);

– old value of the feature (before recalculation);

– minimal value among values of particular feature;

– maximal value among values of particular feature.

Another technic to which normalization is referred is “Batch Normalization”. Batch normalization is a new method for solving the problem of disappearance explosion of gradients, resulting in reduced or increased magnitude of activation gradients in consecutive layers. Another problem is that in training, parameters change, causing activation of hidden variables to change as well. If you change the input from the initial layer to the later layer, the training data of the later layer is unstable, so training data can be slower to adjust during training. Batch normalization allows us to get rig of that problem. In batch normalization, it is a function of creating some similar variances to create additional "normalization layers" that resist such behaviors by adding "normalization layers" between hidden layers (Aggarwal, 2018).

Referring to what was preciously said – our DL model consist of different layers of linear and activation functions. Batch normalization usually applied before each linear function except the first one. In such way, outputs from activation blocks are normalized and transferred to further linear layers.

Below you can see implementation of such model in Python:

class Model(torch.nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.layers = torch.nn.Sequential(

torch.nn.Linear(in\_features=10 + 2 \* len(dataset\_full.labels), out\_features=24),

torch.nn.ReLU(),

BatchNormLast(in\_features=24),

torch.nn.Linear(in\_features=24, out\_features=16),

torch.nn.ReLU(),

BatchNormLast(in\_features=16),

torch.nn.Linear(in\_features=16, out\_features=1)

)

On the example above, several layers are seen. Layers goes in such sequence: Linear – ReLU – Batch Normalization – Linear – ReLU – Batch Normalization – Linear. As it was already stated, each time values go through Linear and ReLU layer – before being transferred to next block of such layers, they are normalized with Batch Normalization layer.

The normalization of image input helps to strengthen the training process and increases its resistance to variations. Batch normalization is further a step towards standardization of inputs to all layers of our neural networks. In particular, we change the network architecture to include the operations of the network.:

1. Before entering the layer, grab the vector of the logit that enters before going through the nonlinearity.

2. Normalize all examples of minibatch, reduce average and divide by standard deviation, dividing the component of each logit vector.

3. Given a normalized input x, the affine transformation is used to restore the representation power of two vectors of trainable parameter: γx̂ + β (Buduma and Locascio, 2017).

## Loss functions

Choose of loss function is a crucial moment in DL model. Point of loss function is to determine how far predicted value is from the real value. More precise wording is stated in (Chollet, 2018): “Loss function (objective function)—The quantity that will be minimized during training. It represents a measure of success for the task at hand”.

Imagine, you have a goal to maximize average of well-being of all humans. With poor loss function your trained AI will choose to kill all humans, as in that case their average happiness will grow, however we understand how awful this solution is. We need to keep in mind that every AI model tries to lower loss functions with any cost (Chollet, 2018).

**Table 1.1** (Trask, 2019)

**Correspondence of loss function to particular problem**

|  |  |  |
| --- | --- | --- |
| **Problem type** | **Last-layer activation** | **Loss function** |
| Binary classification | sigmoid | binary\_crossentropy Formula (1.6) |
| Multiclass, single-label classification | softmax | categorical\_crossentropy Formula (1.7) |
| Multiclass, multilabel classification | sigmoid | binary\_crossentropy Formula (1.6) |
| Regression to arbitrary values | None | MSE Formula (1.8) |
| Regression to values between 0 and 1 | sigmoid | mse Formula (1.8) or binary\_crossentropy Formula (1.6) |

where – loss that should be calculated via formula;

– number of predictions;

– predicted probability;

– real probability of the class.

where – loss that should be calculated via formula;

– number of predictions;

– predicted probability;

– real probability of the class.

where – loss that should be calculated via formula;

– number of predictions;

– predicted probability;

– real probability of the class.

## Optimization algorithm

Previously, it was described how data is transformed through different (linear and activation) layers. Below in Formula (1.9) you can see data transformation via linear layer:

where – output of linear layer;

– weight of the input;

– bias;

– data that is inputted in the model.

Expression (1.9) shows you very simple prediction of based on learnable parameter . and are tensor or so called, learnable parameters of the layer.

Normally, the values of these parameters are randomly assigned at the beginning of the learning process, and these values are modified according to the algorithm during the learning process. The idea is to adjust those values through training loop (Zhang *et al.*, 2021).

(Chollet, 2018) describes training loop is such sequence:

* Calculate y\_pred values based on inputted x values (this is also called forward pass).
* Calculate distance between predicted values (y\_pred) and real values (y). This step is performed via predefined loss function.
* Based on the results of a loss function update values of weights so y\_pred would be closer to real values of y.

Iterating through this loop will make you loss dramatically decrease. Now let’s talk about each step. Step 1 is about forward passing inputted x values. In this step, the information (functions) you use go through the sequence of layers in your learning model. In step 2 we use loss functions (described in previous section) that allow us to see how far predictions from their real state are. It calculates the distance of the output of the model (predicted values) and real values. The most interesting part is step 3, on this stage we need to somehow decide in which direction and by how far to change the weights of features used. For example, we have one feature with its weight equals 0.5 – loss would be 10. After changing this coefficient value to 0.35 – loss changes to 15, however if we change it to 0.65 – the loss will be 5. From this iteration it seems that updating our coefficient with +0.15 is the right direction to minimize loss between real value and predicted one (Zhang *et al.*, 2021).

However, this solutions seems very insufficient as we would need to perform this operation to every feature of a dataset, which could be millions. A better way is to make use of all the differentiable operations in the network and calculate the loss gradient in relation to the network coefficient. After differentiation last step would be to go to the opposite direction from the derivative, making loss smaller and smaller (Chollet, 2018).

### The gradient

A gradient is a derivative that extends to multiple variables. It captures the local slope of the function and predicts the effects of small steps from any point. The gradient points towards the steepest upwards of the tangent hyperplane (Kochenderfer and Wheeler, 2019).

As an example let’s imagine we have weight W with value W0. In that case, derivative of that weight in the point W0 would be gradient:

where – gradient;

– function that is the dependable variable of a gradient;

– dependable variable of a function, value of weight that is under consideration.

where each coefficient describes the direction and size of the changes in our loss function that occur if we adjust the different values of the W0 coefficient.. Gradient described in Formula (1.10) is the gradient of the function:

where – function of gradient;

– dependable variable, value of weight;

– value of loss function.

in the point where weight = W0 (Chollet, 2018).

From math we know that derivative of a function with just a single coefficient can describe the slope of that function. In the same way, gradient in Formula (1.10) can describe curvature of function around point W0 (Chollet, 2018).

Summing up the knowledge above, we can understand how minimizing the loss in our model could be performed. Just as x can be moved in opposite directions to reduce the value of f(x), f(W) can also be reduced by moving W from the opposite direction of its gradient. As an example, such equation could be considered:

where – updated value of weight that is under consideration;

– value of weight that is under consideration before updating it;

– small scaling factor that represents how far we will go in the direction to update new weight;

– formula of the gradient described in Formula (1.10), derivative of the weight (Chollet, 2018).

### Stochastic gradient descent (SGD)

With a derivation function, you can find a point on the gradient where the derivative of the function is 0. Minimum of a function is a point where derivative of that function is equal to 0. In the scope of a neural network – our task is to find combination of weights using which in forward pass, lead us to the minimum loss (Zhang *et al.*, 2021). This could be done solving the equation:

where – gradient;

– function that is the dependable variable of a gradient;

– dependable variable of a function.

Such way of reducing the loss could be applied on tasks with small number of variables (somewhere around 2-4), however doing it in real tasks where datasets have thousands or millions of features – it would be insufficient. Instead, we could use a three-step algorithm that was previously described. With the help of differentiable function – calculation of a gradient could be used for optimization of step 3 in described Optimization algorithm (updating the weight in the opposite direction of its derivative will lower the loss every iteration):

* Calculate y\_pred values based on inputted x values (this is also called forward pass).
* Calculate distance between predicted values (y\_pred) and real values (y). This step is performed via predefined loss function.
* Perform backward pass. Calculate derivative of a distance (loss) function with respect to dataset features.
* Move the values of a features in the opposite direction of computed gradient.

Described algorithm above is an example of a SGD algorithm. Below you can see graphical representation of it.

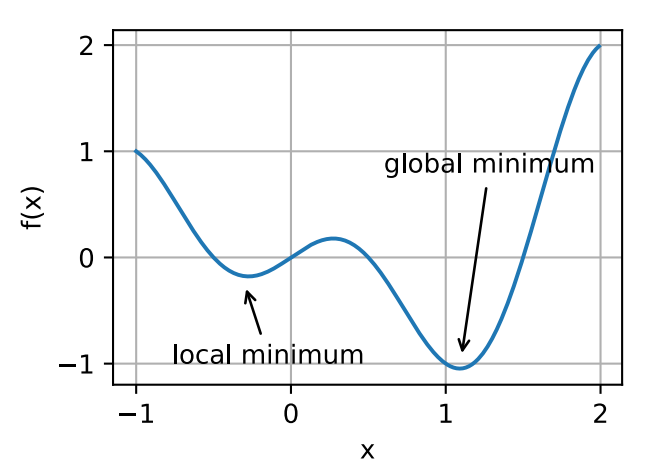


Fig. 1.6. Graphical representation of SGD algorithm (Zhang *et al.*, 2021).

It was already described, why we need step variable in Formula (1.12), however a note about this variable should be added. If too small value would be picked – model could be stuck in local minimum (Figure 1.7) or it would spend too much iterations and learning process will last for too long time. If large value would be chosen – updated weights could be found in completely random positions of the curve and end up not minimizing the loss, but even increasing (Chollet, 2018).

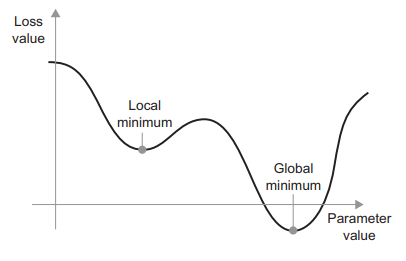


Fig. 1.7. Representation of local and global minimum (Chollet, 2018).

As it is seen on Figure (1.7), loss function has several points where its value could be described as minimum. Local minimum represents point of the curve where loss decreases, but not getting to the minimal value of the whole curve. With small step value, model could decide that this local minimum is the only minimum in our function and get stuck in it, but we need network to find minimal of the whole function.

This problem could be solved via applying technique that is based on physical phenomenon (momentum). With this technique, values of weights update based not only on the current state of a slope but also, on the value of predefined variable. Another words, we add new variable that control the speed of parameter modification (Chollet, 2018).

Here you can see a small algorithm in programming language that realizes that method (adopted from (Chollet, 2018)):

past\_velocity = 0.

momentum = 0.1

while loss > 0.01:

w, loss, gradient = get\_current\_parameters()

velocity = past\_velocity \* momentum + learning\_rate \* gradient

w = w + momentum \* velocity - learning\_rate \* gradient

past\_velocity = velocity

update\_parameter(w)

## Backpropagation algorithm

With the knowledge background of optimization algorithm, we can proceed further to backpropagation algorithm. This technic is based on widely known chain rule of derivation. The situation is that, in neural network, there are a lot of layers, and they are connected with each other in a certain sequence. To compute gradient of a whole model – chain rule is applied (Chollet, 2018; Vasilev *et al.*, 2019).

Tern of backpropagation is referred to the backward phase that is used in every multi-layer neural network. There are 2 phases (Aggarwal, 2018):

* Forward phase: we input values of the features of a dataset that are going to be used in computation of the output. After values are inputted – they go through defined layers (like linear or activation) and used for calculations with respect to their weight. We try to minimize loss between outputted value and the real value that we have. Next step would be to calculate derivative of the loss function with respect to previously used weight.
* Backward phase: idea of this phase is to calculate gradient of loss function with respect to weights of features to reduce this loss function. Calculation of those weights is performed via applying chain rule of differential calculus. Term back appears because, using chain rule we start calculation from the bottom layers of the model and sequentially listing up to the top layers.

Let’s consider an example (Chollet, 2018):

where – main function of the model that includes all the layers of the model (3 layers in our case);

– inner function of the model, that takes and as inputs (it is the top layer);

– inner function of the model, that takes and as inputs (it is an output for );

– inner function of the model, that takes inputs (it is an output for );

– weights of the features from dataset.

Using Function (1.14), we would need to apply chain rule in order to calculate its derivative. We would start from deriving , then and up with computing . A more formulaic form of this calculation is represented in Function (1.15) (Petersen and Pedersen, 2007):

where – main function that takes () as an input;

– inner function that takes as an input;

– argument of function.

## R-Squared

R-squared is a type of metric that is used in popular DL model to determine variation of dependent variables by the independent variables of the dataset. Book (Chicco, Warrens and Jurman, 2021) gives such explanation: “The coefficient of determination (Wright, 1921) can be interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variables”.

R-squared is based on Formula (1.16):

where – R-squared;

– number of samples;

– predicted value;

– dependent variable of a dataset (true value of the dataset);

– mean of real values.

As it was already described, by the output of the Formula (1.16), we can understand how teachable dataset is. Values of output could vary from to +1, where is the worst possible value, while 1 is the best (Chicco, Warrens and Jurman, 2021).

To understand how that metric could be used, let’s consider an example where we have 2 regression models with output that scales from 0 to 10 in one case and from 0 to 100 in another. With such metrics as MSE or MAE, it would be impossible to compare those 2 models, however, using R-squared we would get coefficient in the range (, which will show us the predictive performance of those datasets (Chicco, Warrens and Jurman, 2021).

## Regression tasks

Now, with theoretical background of deep learning models, we can dive into regression tasks. There are 2 main types of DL problems(Chollet, 2018; Russell and Norvig, 2021):

* Classification tasks: idea of classification task is that you model outputs probability for every class of your dataset. As an example, image classification can be given. Pixels are taken as feature and type of image as an output. You input set of images in required format and model predicts what kind of classes are shown on the image (model can predict either dog, cat or elephant is located on the image). In such tasks, output vary in the range 0 – 1, where 0 means that that exact class in not found or determined on the image and 1 means that probability of that class is 100%.
* Regression tasks: in such tasks, model also, teaches based on the features inputted and real values of the dataset, however, output values can vary in any range. As an example, prediction of house prices could be named. You input dataset to your model with features of houses, such as: number of floors, area of the house, distance between house and center of the city. Those features can, also, have not scalar values, but categorical, such as: location or type of house. In such case those categories will be transformed into scalar values so they can be used by the model (examples of such values will be given in the practical section). After all house knowledge is inputted, model starts to learn and after assigning faithful values to the weight of the features, it starts to can predict the price.

### Inputs – Outputs

First stage of building the model – implementation of dataset based on which learning would be performed. Usually, values of dataset can vary in a huge range and it can slow down the learning. Although networks can adapt automatically to such data, learning will certainly be more insufficient with regards to tome and space. A common solution for that problem is data normalization. We need to subtract from every dataset feature mean of features and divide this value by the standard deviation. In this case, the feature is concentrated around 0 and has a unit-standard deviation (Chollet, 2018).

This is an example, how normalization could be performed:

data\_mean = data.mean()

s\_deviation = data.std()

data = (data - data\_mean) / s\_deviation

### Example of architectures

Once the dataset is successfully implemented, we can build the model's logic unit. As it was previously mentioned, model consist of different layers of functions, such as linear or activation (sigmoid, ReLU, Tanh, etc…). Depending on size of dataset, complexity of the model and many other circumstances, different layers and their sequence will be built.

Normally, the last layer of the regression model is the linear layer. With that, model can output values in a various range, which is a crucial factor of a regression neural network. For example, if sigmoid function would be chosen, for the last layer, output could only vary in a range from 1 to 0 (which is acceptable for classification model, but not for regression) (Chollet, 2018).

Example of such architecture is given in scientific paper (Chollet, 2018). This is an example of Python code using opensource library for deep learning models “keras":

from keras import models

from keras import layers

def build\_model():

model = models.Sequential()

model.add(layers.Dense(64, activation='relu',

input\_shape=(train\_data.shape[1],)))

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(1))

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

return model

### Loss functions – MSE, MAE, Huber Loss

In the previous sub-section you can find background knowledge of loss functions. It was already said, that loss function allows you to determine distance between real values and predicted ones. Here briefly will be described 3 main types of such loss function that are used in regression models.

The mean average error (MAE) is the most common and simple function for calculating model loss. MAE is based on Formula (1.17). First step is to find distance between one particular output and real value connected with features of this output. Then all of those distances are summed up and divided by the number of outputs (Chai and Draxler, 2014):

where – value that represents loss that calculates via formula;

– number of observations;

– real value;

– output of the model;

– number of iteration.

Another common function that is used to determine loss, is the mean square error (MSE). First step is the same as in MAE, we calculate distance, however, after that, result of first calculation is squared. After that program must sum all of those squares and divide by the number of observations. Formula (1.18) represents that equation (Chai and Draxler, 2014):

where – value that represents loss that calculates via formula;

– number of observations;

– real value;

– output of the model;

– number of iteration.

Last example of a loss function in this section is Huber loss. This function is considered as a balance between MSE and MAE. Calculation of loss could be performed using Formula (1.19) (Meyer, 2021):

where – value that represents loss that calculates via formula;

– parameter defined by user;

– real value;

– output of the model.

## Importance of normalization functions

# Methodology

## Datasets

In the related work several datasets will be implemented to compare usage of normalization functions under different circumstances. In this section such dataset are described.

### Dataset - Weather in Szeged

The first dataset is “Weather in Szeged”. It consists of 96543 samples of data with 12 features. Each row has next features:

* Formatted Date: In this column date and time of weather observation is recorded. Data is stored in scalar type. Example: “2006-04-01 00:00:00.000 +0200”.
* Summary: Overall information about forecast. Data is stored in text format. Example: “Partly Cloudy”.
* Precip type: Type of precipitation during particular time. Data is stored in text format. Example: “rain”. Figure (2.1).
* Temperature: This column represents the temperature in Celsius. Data is stored in scalar type. Example: “9.47222222222222”. Figure (2.2).
* Apparent Temperature: This feature represents apparent temperature of particular time. Data is stored in scalar type. Example: “7.38888888888888”. Figure (2.3).
* Humidity: Here humidity of the air is observed. Data is stored in scalar type. Example: “0.89”. Figure (2.4).
* Wind Speed: Information about speed of the wind is stored (in kilometers per hour). Data is stored in scalar type. Example: “14.1197”. Figure (2.5).
* Wind Bearing: Wind bearing is observed in this column (in degrees). Data stored in scalar type. Example: “251”. Figure (2.6).
* Visibility: This feature represents visibility in kilometers. Data is stored in scalar type. Example: “15.8263”. Figure (2.7).

This dataset is based on weather observations during period from 2006 – 2016 in Szeged.

Below you can see histograms for most of the features to understand the statistics of this dataset:

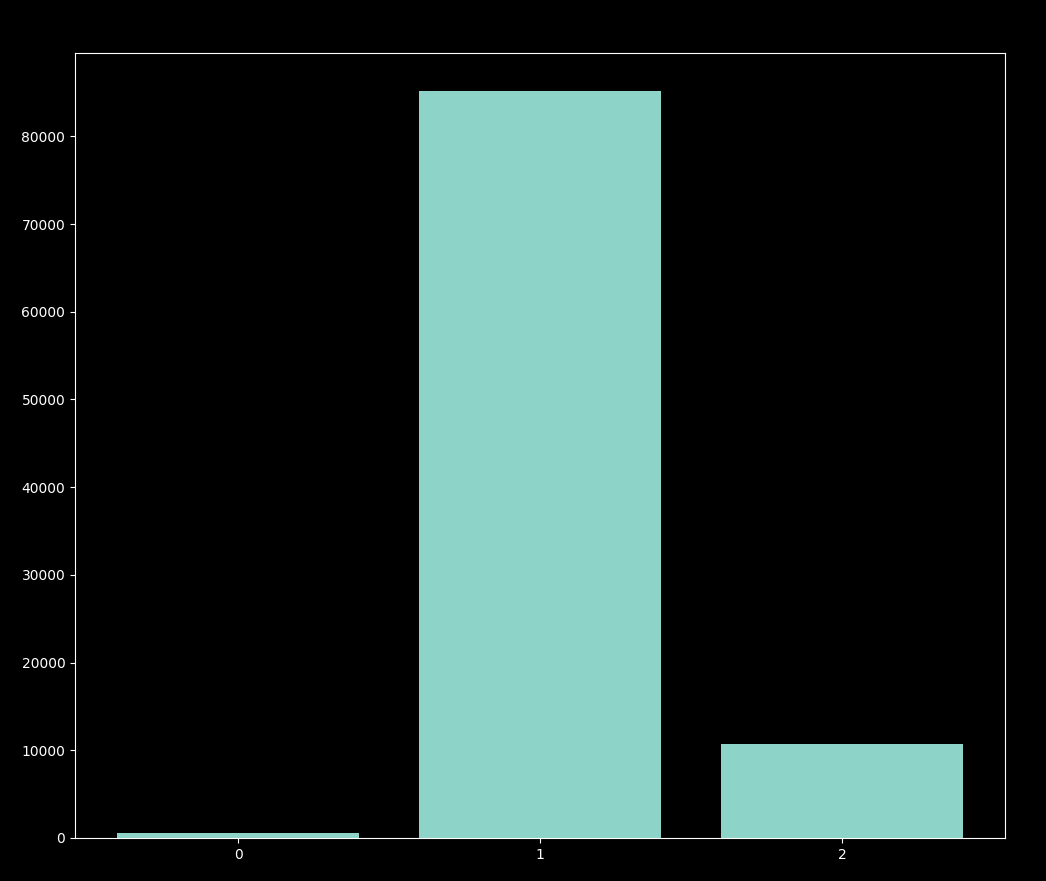


Fig. 2.1. Histogram for feature “Precip type”

On Figure (2.1), you can see histogram for the feature “Precip type” which shown how many number of different types of precipitation were stored in current dataset. Each number on x axes represents each type of precipitation:

* 0 – No precipitation was recorded,
* 1 – Rain,
* 2 – Snow.

It is clearly seen that rain is the dominant precipitation among dataset. To be exact, there are 85224 times rain was recorded, 10712 – snow and 517 times no precipitation was found (all together 96543 records).

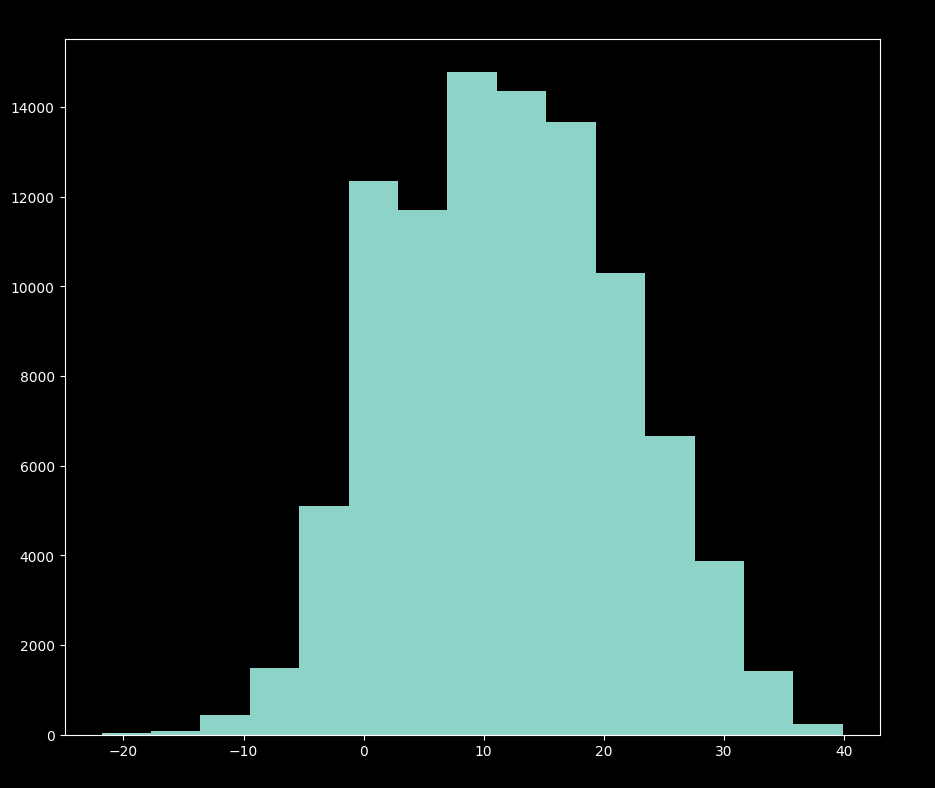


Fig. 2.2. Histogram for feature “Temperature”

Figure (2.2) allows us to notice normal or Gaussian distribution of the temperature samples. X axes represents temperature of a row. We can see that most of values of that feature are concentrated in the middle of a range. The lowest temperature recorded is -21.8 C, while maximum is 39.9 C.

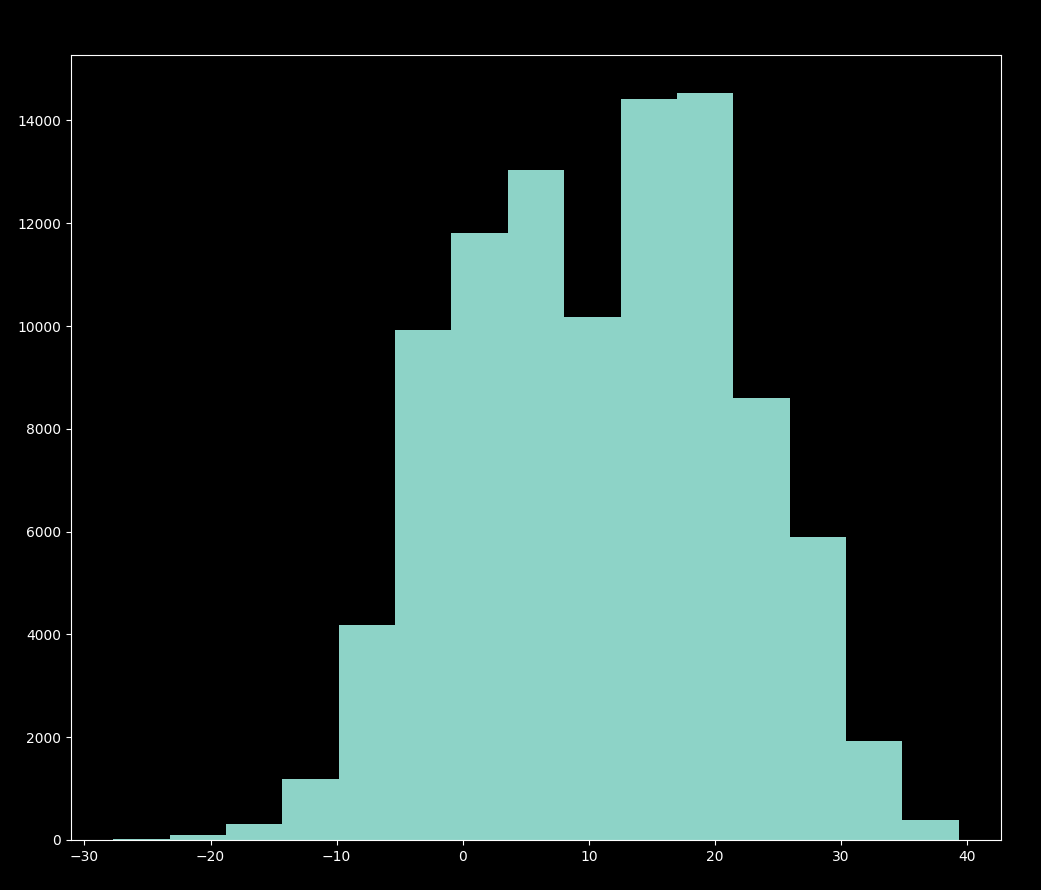


Fig. 2.3. Histogram for feature “Apparent temperature”

This histogram (Figure 2.3), as previous, shows us normal distribution of the values of current feature. Most of the values are located in the range from -5 C to 25 C. Minimal apparent temperature recoded is -27.7 C, while maximum is 39.3 C.

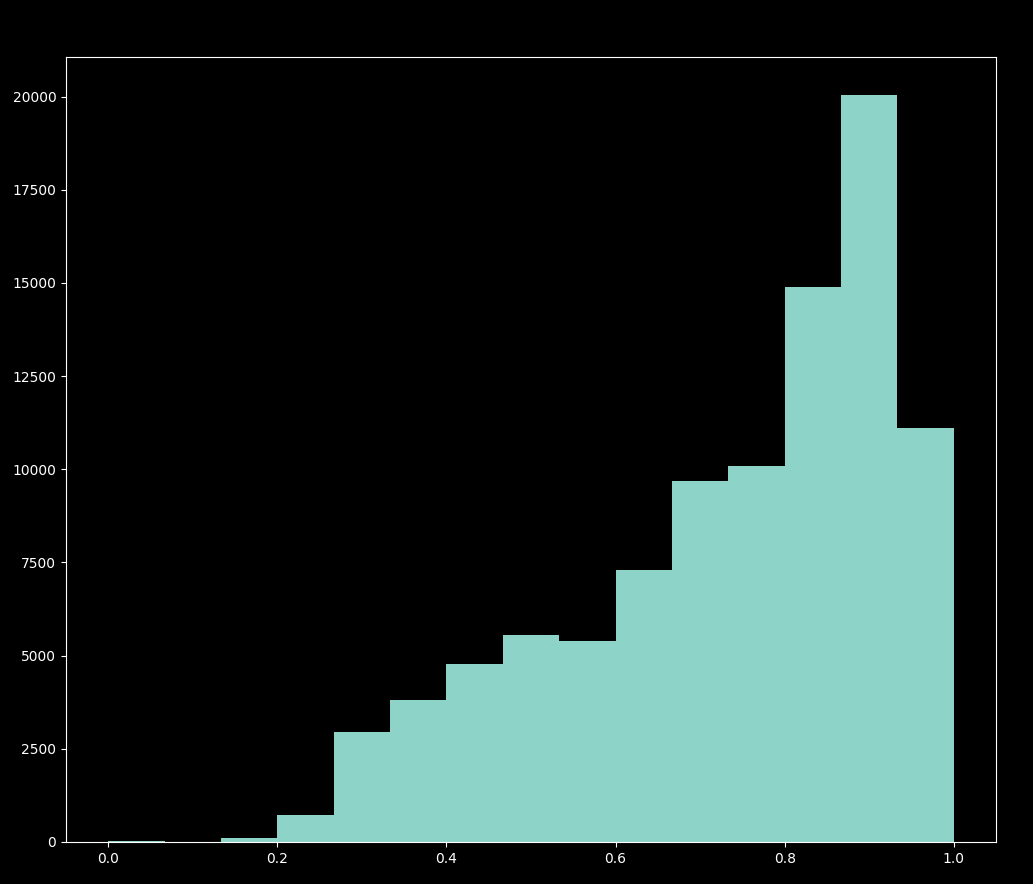


Fig. 2.4. Histogram for feature “Humidity”

On Figure (2,4) histogram for feature “Humidity” is located. Unlike Figure (2.2) and Figure (2.3) no Gaussian distribution is seen. As values grow their number of occurrences increases. Most of occurrences are seen at the highest numbers of humidity. In such sequence, most of the samples are located in the range from 0.7 to 1. Minimal value of humidity recorded is 0 and the maxim is 1.

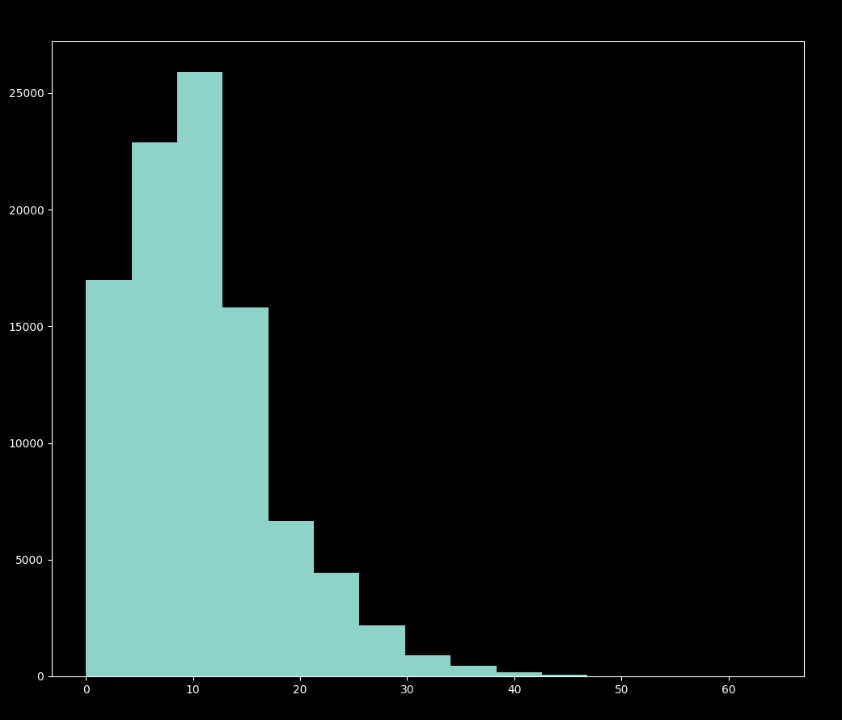


Fig. 2.5. Histogram for feature “Wind Speed”

Looking on Figure (2.5), it is clearly seen situation that is opposite to what is shown on Figure (2.4). As values of feature decreases – number of their occurrences increases. This kind of distribution is called exponential distribution. With that logic, most of the values of wind speed are distributed across the range from 0 km/h to 15km/h. Minimal wind speed that was stored in the dataset is 0 km/h, while maximum is 63.9 km/h.

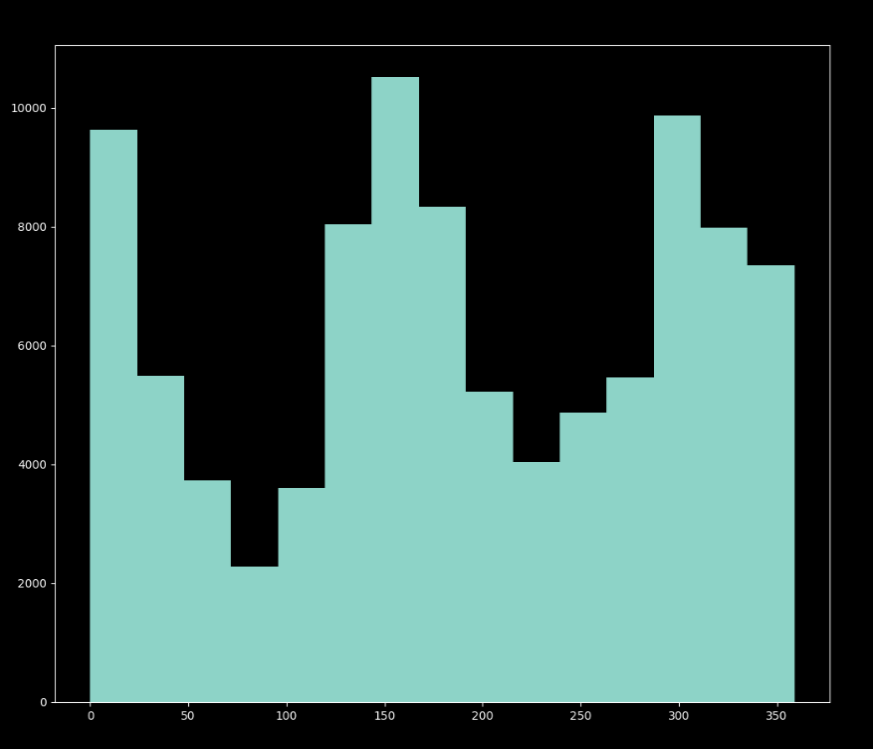


Fig. 2.6. Histogram for feature “Wind Bearing"

Looking at Figure (2.6) we can observe that most of the features are located on the extrema and at the middle part of the histogram. Biggest angle of wind bearing recorded is 0 degree, while maximum is 359 degree. This distribution is logical as we are talking about degrees, which are applied to a circle. It could, also, be described as: most of the samples are located in the places where sinus of their degree would be equal to 0 and vice versa, as sinus of their degree approaches 1 – number of occurrences decrease.

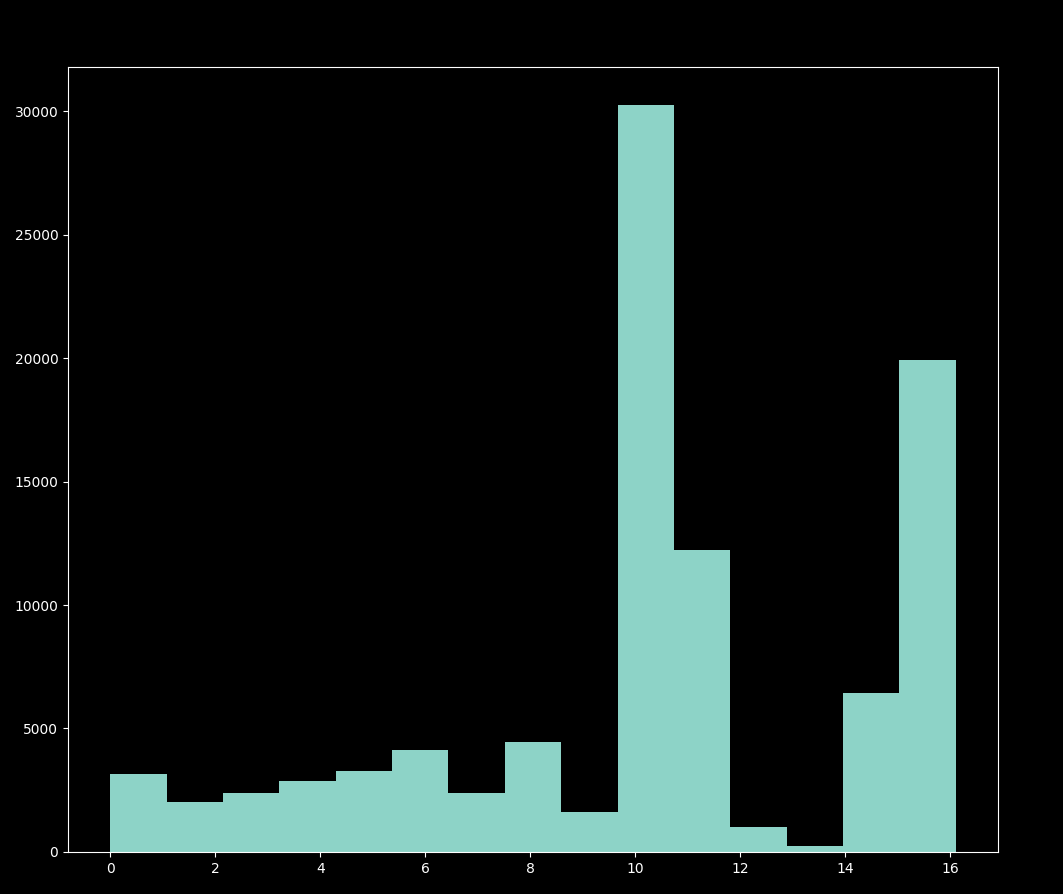


Fig. 2.7. Histogram for feature “Visibility"

Figure (2.7) allows us to understand normal visibility in the region when data was recorded. Most of the time, visibility is approximately 10 kilometers. Minimal value that was recorded is 0 km, while maximum is 16.1 km.

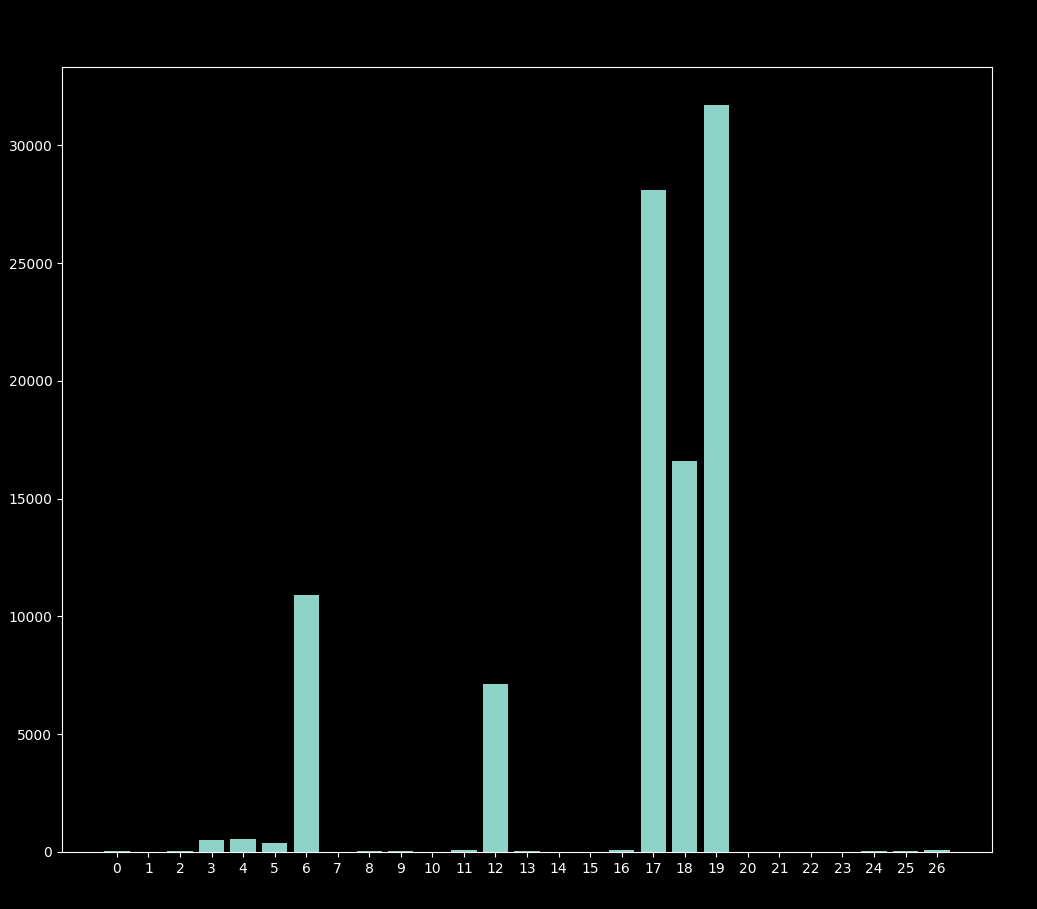


Fig. 2.8. Histogram for feature “Summary”

As on Figure (2.1), Figure (2.8) shows us distribution of text format data. That kind of data usage was described in previous section. To use text data in learning process – we need to represent it in a way of scalar values. For each value there is one concrete class:

* 0 – Breezy,
* 1 – Breezy and dry,
* 2 – Breezy and Foggy,
* 3 – Breezy and Mostly Cloudy,
* 4 – Breezy and Overcast,
* 5 – Breezy and Partly Cloudy,
* 6 – Clear,
* 7 – Dangerously Windy and Partly Cloudy,
* 8 – Drizzle,
* 9 – Dry,
* 10 – Dry and Mostly Cloudy,
* 11 – Dry and Partly Cloudy,
* 12 – Foggy,
* 13 – Humid and Mostly Cloudy,
* 14 – Humid and Overcast,
* 15 – Humid and Partly Cloudy,
* 16 – Light Rain,
* 17 – Mostly Cloudy,
* 18 – Overcast,
* 19 – Partly Cloudy,
* 20 – Rain,
* 21 – Windy,
* 22 – Windy and Dry,
* 23 – Windy and Foggy,
* 24 – Windy and Mostly Cloudy,
* 25 - Windy and Overcast,
* 26 - Windy and Partly Cloudy.

It is seen that mostly, dataset is filled with values of: “Clear”, “Foggy”, “Mostly Cloudy”, “Overcast”, “Partly Cloudy”.

Plotting of histograms was performed using python programming language with opensource library for plotting “matplotlib”. Below, you can see code which was written for plotting:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

X = np.array(pd.read\_csv("./weatherHistory.csv"))

for scalar\_feature in range(3, 6):

self.X = X[:, scalar\_feature:9]

plt.hist(self.X[:, 5], bins=15)

plt.show()

for text\_feature in range(1, 3):

self.Y = np.array((X[:, text\_feature])).astype(np.str\_)

unique, counts = np.unique(self.Y, return\_counts=True)

y\_hist = dict(zip(unique, counts))

names = list(range(0, len(y\_hist)))

values = list(y\_hist.values())

plt.bar(range(len(y\_hist)), values, tick\_label=names)

plt.show()

### Datasets – CalCOFI

CalCOFI is a dataset of a 60 years of oceanographic observation by California Cooperative Fisheries Investigations. It consists of 864864 samples with 74 rows of features each. There is data of:

* temperature,
* salinity,
* oxygen,
* phosphate,
* silicate,
* nitrate and nitrite,
* chlorophyll,
* transmissometer
* PAR,
* C14 primary productivity,
* phytoplankton biodiversity,
* zooplankton biomass,
* zooplankton biodiversity.

Even though, having such big data, not all of the rows are filled fully, however, this problem can be solved during the stage of implementation of the dataset.

Description of such big data would take too much space; only statistics of most usable and important features are listed below:

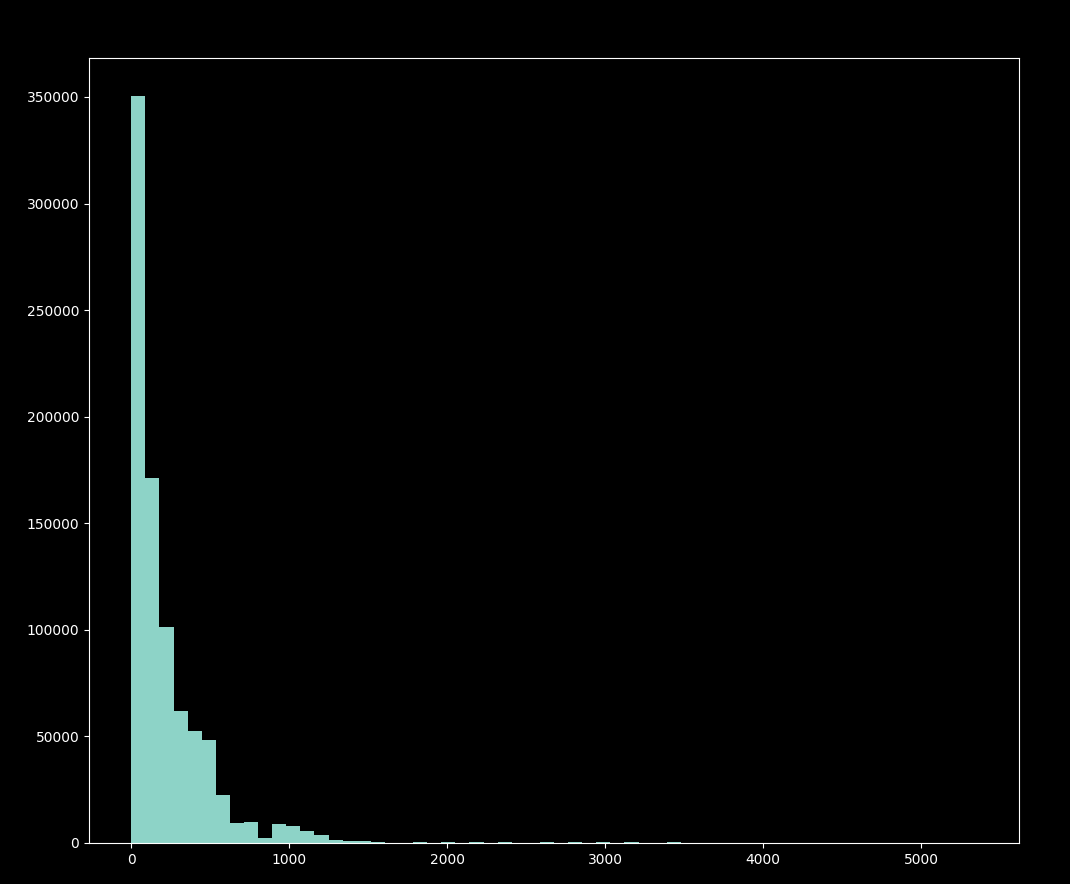


Fig. 2.9. Histogram for feature “Depthm”

Histogram located on Figure (2.9) shows us distribution of values of feature “Depthm”. This feature represents depth (in meters) that was recorded. All of those values are of scalar type. We can see that most of the occurrences are of a depth not more than 400 meters. As value of feature increases – number of samples – decreases.

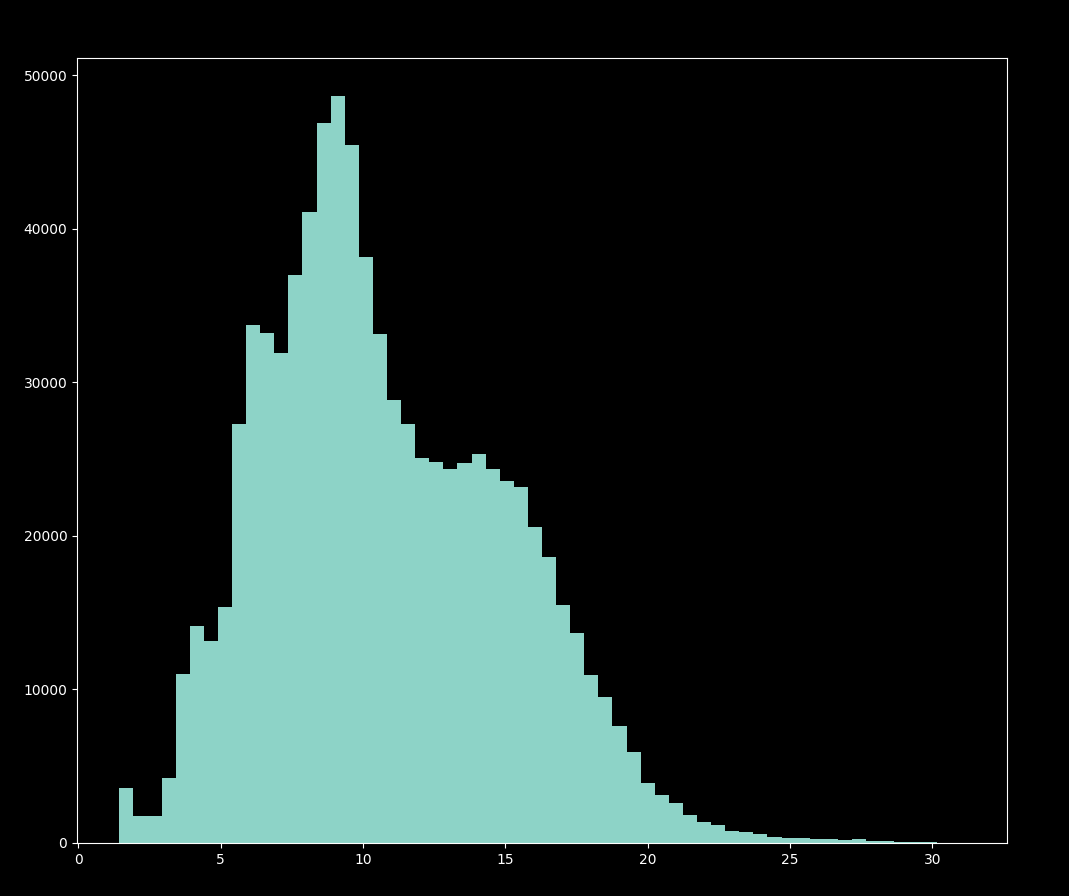


Fig. 2.10. Histogram for feature “T\_degC”

Figure (2.10) shown values of feature “T\_degC”. Most of the values are located in the range from 3 to 17.

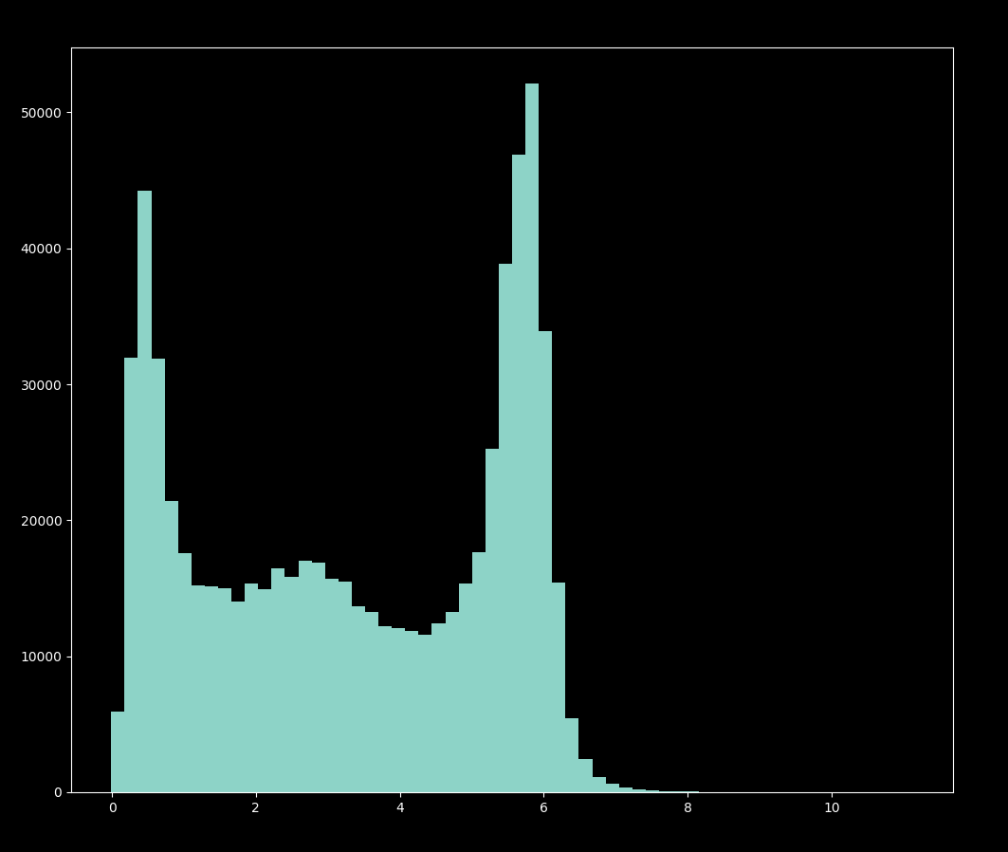


Fig. 2.11. Histogram for feature “STheta”

Figure (2.11) shown values of feature “STheta”. We can see 2 point of largest cluster of samples around values 0.4 and 5.5. Most of other values are located between those 2 clusters.

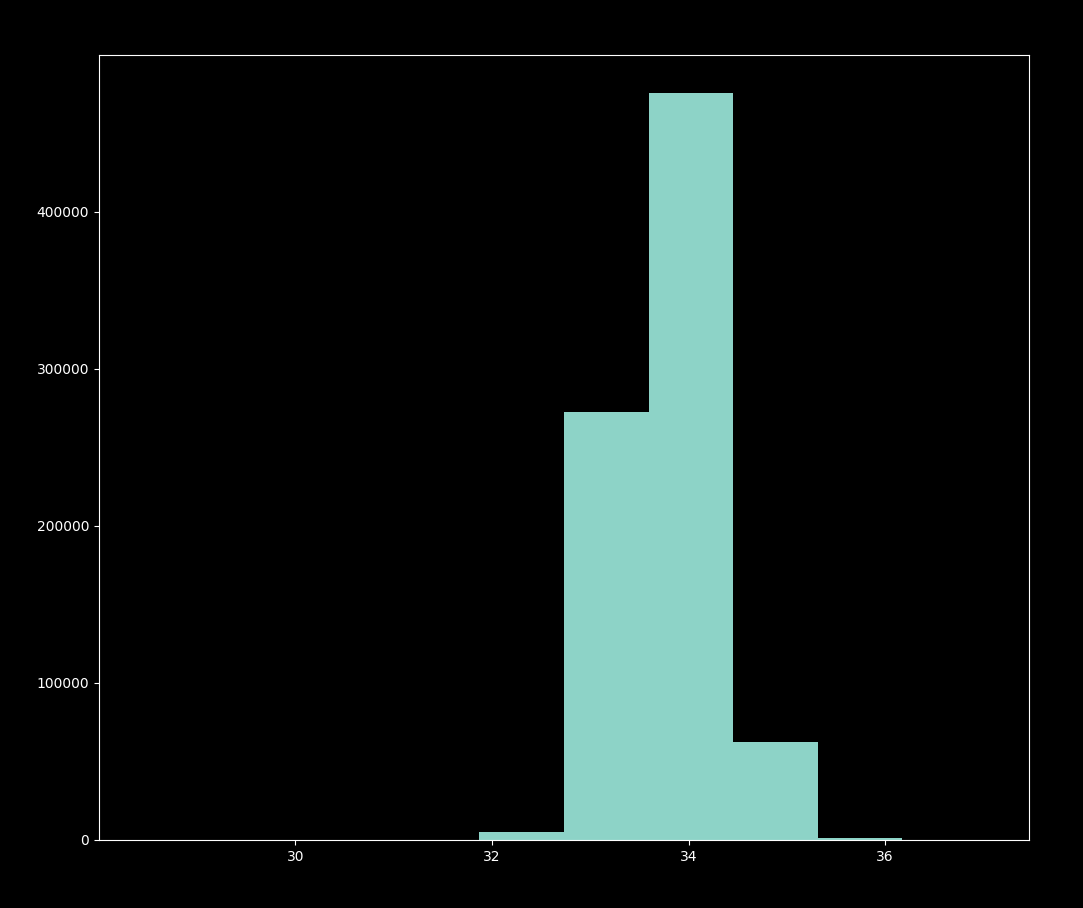


Fig. 2.12. Histogram for feature “Salnty”

On Figure (2.12) values of feature “Salnty” are shown. It is seen that most of values are equal to 34. Other don’t deviate from it very much. 99% of them are in the range from 33 to 35.

## Normalization functions

## Architecture

## Metrics

### R-Squared

R-squared is a type of metric that is used in popular DL model to determine variation of dependent variables by the independent variables of the dataset. Book (Chicco, Warrens and Jurman, 2021) gives such explanation: “The coefficient of determination (Wright, 1921) can be interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variables”.

R-squared is based on Formula (2.1):

where – R-squared;

– number of samples;

– independent variable of a dataset;

– dependent variable of a dataset (true value of the dataset);

– mean of real values.

As it was already described, by the output of the Formula (2.1), we can understand how teachable dataset is. Values of output could vary from to +1, where is the worst possible value, while 1 is the best (Chicco, Warrens and Jurman, 2021).

To understand how that metric could be used, let’s consider an example where we have 2 regression models with output that scales from 0 to 10 in one case and from 0 to 100 in another. With such metrics as MSE or MAE, it would be impossible to compare those 2 models, however, using R-squared we would get coefficient in the range (, which will show us the predictive performance of those datasets (Chicco, Warrens and Jurman, 2021).

# References

Aggarwal, C.C. (2018) *Neural networks and deep learning: a textbook*. Cham, Switzerland: Springer. doi:10.1007/978-3-319-94463-0.

Buduma, N. and Locascio, N. (2017) *Fundamentals of deep learning: designing next-generation machine intelligence algorithms*. First edition. Sebastopol, CA: O’Reilly Media.

Chai, T. and Draxler, R.R. (2014) ‘Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature’, *Geoscientific Model Development*, 7(3), pp. 1247–1250. doi:10.5194/gmd-7-1247-2014.

Chicco, D., Warrens, M.J. and Jurman, G. (2021) ‘The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation’, *PeerJ. Computer Science*, 7, p. e623. doi:10.7717/peerj-cs.623.

Chollet, F. (2018) *Deep learning with Python*. Shelter Island, New York: Manning Publications Co.

Kochenderfer, M.J. and Wheeler, T.A. (2019) *Algorithms for optimization*. Cambridge, Massachusetts: The MIT Press.

Liang, S. *et al.* (2021) ‘Reproducing Activation Function for Deep Learning’. arXiv. Available at: http://arxiv.org/abs/2101.04844 (Accessed: 16 May 2022).

Meyer, G.P. (2021) ‘An Alternative Probabilistic Interpretation of the Huber Loss’, in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA: IEEE, pp. 5257–5265. doi:10.1109/CVPR46437.2021.00522.

Nielsen, M.A. (2015) *Neural networks and deep learning*. Determination press San Francisco, CA, USA.

Petersen, K.B. and Pedersen, M.S. (2007) ‘The Matrix Cookbook’.

Russell, S.J. and Norvig, P. (2021) *Artificial intelligence: a modern approach*. Fourth edition. Hoboken: Pearson (Pearson series in artificial intelligence).

Trask, A.W. (2019) *Grokking deep learning*. Shelter Island: Manning.

Vasilev, I. *et al.* (2019) *Python deep learning: exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow*. Second edition. Birmingham Mumbai: Packt Publishing Limited.

Zhang, A. *et al.* (2021) ‘Dive into Deep Learning’, *arXiv preprint arXiv:2106.11342* [Preprint].