**RĪGAS TEHNISKĀ UNIVERSITĀTE**
Datorzinātnes un informācijas tehnoloģijas fakultāte
Lietišķo datorsistēmu institūts
Mākslīgā intelekta un sistēmu inženierijas katedra

**Oļegs Korsaks**
Bakalaura studiju programmas „Datorsistēmas”
students, stud. apl. nr. 051RDB146

# SALIDZINOŠĀ ANALĪZE DATU KOPU FORMĀTIEM PYTORCH ATTĒLU KLASIFIKĀCIJAS UZDEVUMIEM

**Atskaite par bakalaura darbu**

Zinātniskais vadītājs
Mg.sc.ing, Pētnieks
**ĒVALDS URTĀNS**

Rīga 2022

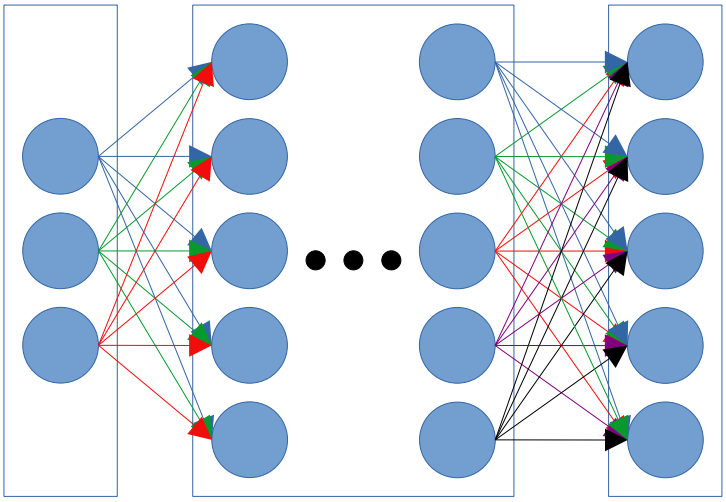
# Table of Contents

# 1. Ievads

## 1.1. Dziļā mašinmācīšanās

### Pamata arhitektūras

Makslīgais neironu tīkls sastāv no viena ieejas slāņa, viena vai dažiem slēptiem slāņiem, ka arī viena izejas slāņa. Katrs slānis satur vienu vai vairākus neironus. Tie ir saistīti ar blakusslāņu neironiem.



Eksistē tīkli, kuru neironi tiek pilnīgi saistīti ar visiem neironiem blakusslāņos, bet var būt saistīti tikai daļēji. Katrai saitei ir savs svars un nobīde. Tie ir apmācamie parametri. Apmācīšanas procesa mērķis ir atrast tādus svarus un nobīdes visām saitēm, ar kuriem tīkls veidotu pareizus rezultātus izejas slānī.

#### Linārie slāņi

Although there is no strong formal theory on how to select the neural network layers and configuration, and although the only way to tune **some** hyper-parameters is just by trial and error (meta-learning for instance), there are still some heuristics, guidelines, and theories that can still help us reduce the search space of suitable architectures considerably. In a previous blog post, we introduced the inner mechanics of neural networks. In this series of blog posts we will talk about the basic layers, their rationale, their complexity, and their computation capabilities.

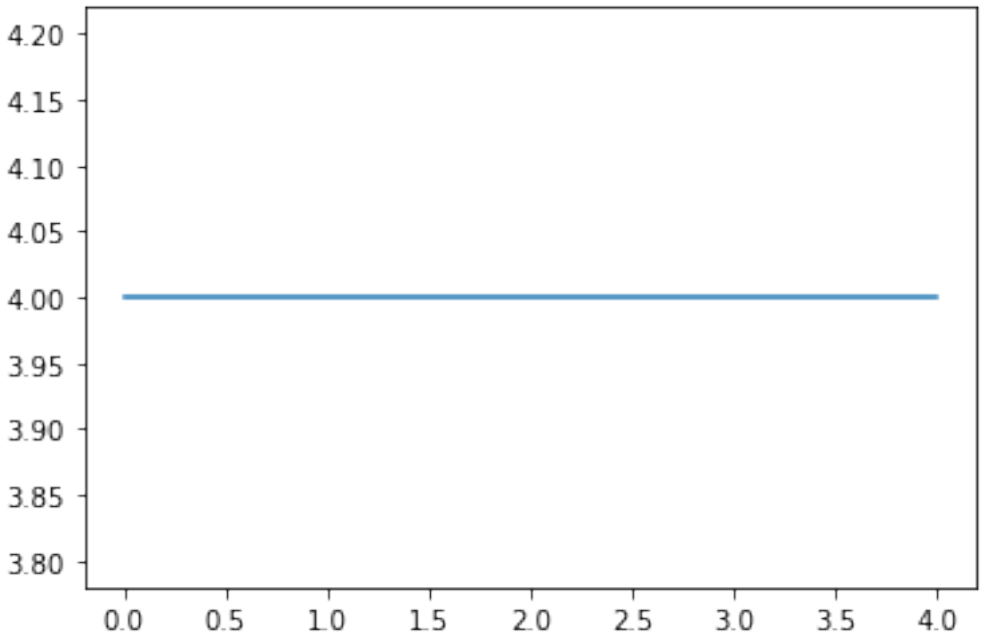
#### Lineārais nobīdes slānis

$$y = b$$

This layer is basically learning a constant. It’s capable of learning an offset, a bias, a threshold, or a mean. If we create a neural network only from this layer and train it over a dataset, the mean square

error (MSE) loss will force this layer to converge to the mean or average of the outputs.

For instance, if we have the following dataset {2,2,3,3,4,4}, and we're forcing the neural network to compress it to a unique value ***b***, the most logical convergence will be around the value **b=3** (which is the average of the dataset to reduce the losses to the maximum. We can see that learning a constant is kind of learning a DC value component of an electric circuit, or an offset, or a ground truth to compare to. Any value above this offset will be positive, any value below it will be negative. It's like redefining where the offset 0 should start from.

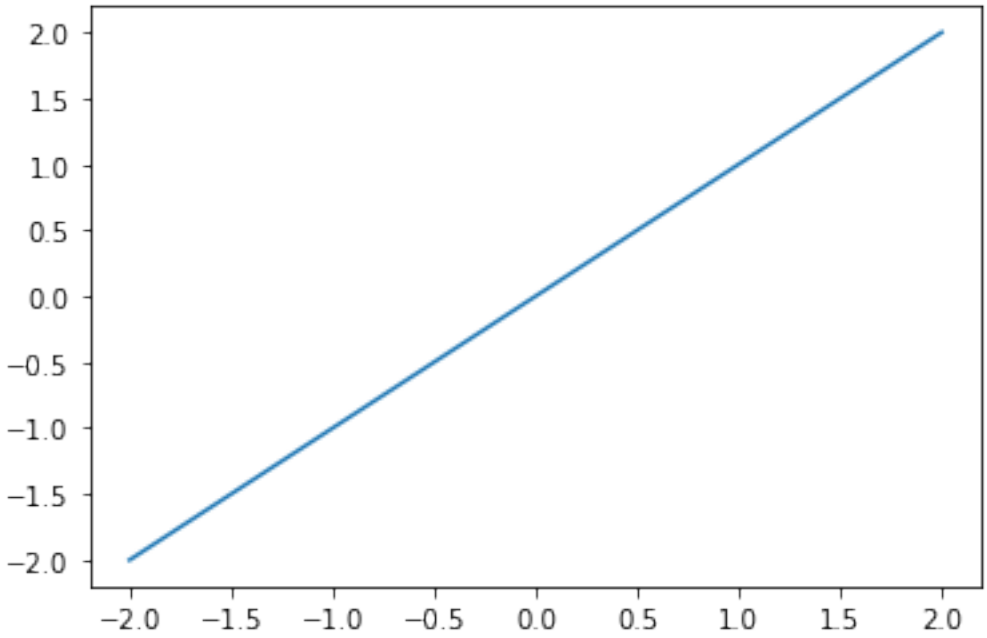



## Lineārais slānis

$$y = W \cdot x$$

A linear layer without a bias is capable of learning an average rate of correlation between the output and the input, for instance if x and y are positively correlated => w will be positive, if x and y are negatively correlated => w will be negative. If x and y are totally independent => w will be around 0.

Another way of perceiving this layer: Consider a new variable A=y/x. and use the "bias layer" from the previous section, as we said before, it will learn the average or the mean of A. (which is the average of output/input thus the average of the rate to which the output is changing relatively to the input).




## Lineārais padeves slānis

$y = W \cdot x + b$

A Feed-forward layer is a combination of a linear layer and a bias. It is capable of learning **an offset and a rate of correlation**. Mathematically speaking, *it represents an equation of a line*. In term of capabilities:

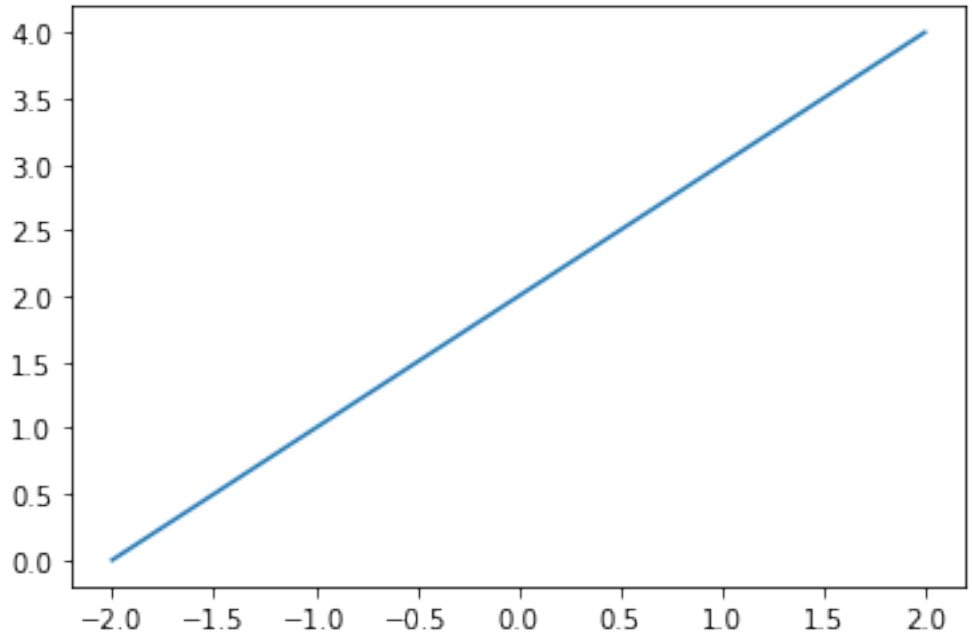
- This layer is able to replace both a linear layer and a bias layer.
- By learning that **w=0** => we can reduce this layer to a pure bias layer.
- By learning that **b=0** => we can reduce this layer to a pure linear layer.
- A linear layer with bias can represent **PCA** (for dimensionality reduction). Since PCA is actually just combining linearly the inputs together.
- A linear feed-forward layer **can learn scaling** automatically. Both a MinMaxScaler or a StandardScaler can be modeled through a linear layer.

By learning **w=1/(max-min)** and **b=-min/(max-min)** a linear feed-forward is capable of simulating a MinMaxScaler

$$y = \frac{x - min}{max - min} => y = \frac{1}{max - min}x - \frac{min}{max - min} => y = w.x + b$$

Similarly, by learning w=1/std and b=-avg/std, a linear feed-forward is capable of simulating a StandardScaler

$$y = \frac{x - avg}{std} => y = \frac{1}{std}x - \frac{avg}{std} => y = w.x + b$$




## Limitations of linear layers

- These three types of linear layer can only learn **linear** relations. They are totally incapable of learning any non-linearity (obviously).
- Stacking these layers immediately one after each other is totally pointless and a good waste of computational resources, here is why:

If we consider 2 consecutive linear feed-forward layers $\mathbf{y_1}$ and $\mathbf{y_2}$:

$$y_1 = w_1.x + b_1; y_2 = w_2.y_1 + b_2$$

We can re-write $\mathbf{y_2}$ in the following form:

$$y_2 = w_2.(w_1x + b_1) + b_2 => y_2 = (w_2.w_1)x + (w_2.b_1 + b_2) => y_2 = w.x + b$$

We can do similar reasoning for any number of consecutive linear layers. **A single linear layer is capable of representing any consecutive number of linear layers**.

## Aktivizācijas funkcijas

### Sigmoid

It squashes a vector in the range (0, 1). It is applied independently to each element of s si.

$$f(x)=\frac{1}{1+e^{-x}}$$

### ReLU

Traditionally, some prevalent non-linear activation functions, like sigmoid functions (or logistic) and hyperbolic tangent, are used in neural networks to get activation values corresponding to each neuron. Recently, the ReLu function has been used instead to calculate the activation values in traditional neural network or deep neural network paradigms. The reasons of replacing sigmoid and hyperbolic tangent with ReLu consist of:

1. Computation saving - the ReLu function is able to accelerate the training speed of deep neural networks compared to traditional activation functions since the derivative of ReLu is 1 for a positive input. Due to a constant, deep neural networks do not need to take additional time for computing error terms during training phase.

$$f(x)=\begin{cases} x, x>0 \\ 0, x\leq 0 \end{cases}$$

Solving the vanishing gradient problem
- the ReLu function does not trigger the vanishing gradient problem when the number of layers grows. This is because this function does not have an asymptotic upper and lower bound. Thus, the earliest layer (the first hidden layer) is able to receive the errors coming from the last layers to adjust all weights between layers. By contrast, a traditional activation function like sigmoid is restricted between 0 and 1, so the errors become small for the first hidden layer. This scenario will lead to a poorly trained neural network.

### Softmax (priekš klasifikācijas)

Softmax it's a function, not a loss. It squashes a vector in the range (0, 1) and all the resulting elements add up to 1. It is applied to the output scores s. As elements represent a class, they can be interpreted as class probabilities. The Softmax function cannot be applied independently to each si, since it depends on all elements of s. For a given class si, the Softmax function can be computed as:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Where sj are the scores inferred by the net for each class in C. Note that the Softmax activation for a class si depends on all the scores in s.

$$L(y, y') = -\frac{1}{N} \sum y \cdot \log(y')$$

$$\frac{\partial L(y, y')}{\partial y'} = -y \cdot \frac{1}{y'} = -\frac{y}{y'}$$

$$SoftMax(y = j|x) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

$$\begin{bmatrix} a_0(1-a_0) & -a_0 a_1 & -a_0 a_2 \\ -a_1 a_0 & a_1(1-a_1) & -a_1 a_2 \\ -a_2 a_0 & -a_2 a_1 & a_2(1-a_2) \end{bmatrix}$$

## Kļūdas funkcijas

Kļūdas funkcija palīdz noteikt cik tālu tekoša prognozēta vērtība ir no patiesas. Un ja to pielietot visiem datu eksemplāriem – ar to var noteikt, cik labi tekošais modelis var prognozēt rezultātus kopumā. Ideālā gadījumā kļūdai jābūt vienādai nullei, gan apmācības datu eksemplāriem, gan pārbaudes. Tātad apmācība cenšas kļūdu samazināt.

### MAE

Mean absolute error vai vidēja absolūta kļūda:

$$L_{MAE} = \frac{1}{N} \cdot \sum_{i=0}^{N} \left| (h_\theta(x_i) - y_i) \right|$$

Tā ir vidēja absolūta starpība starp pareizas un prognozētas vertībām. Kļūda pieaug lineāri un tiek pielietota regresijas uzdevumiem, kuru rezultāts ir viena vērtība.

### MSE

Mean squared error vai vidēja kvadrātiskā kļūda:

$$L_{MSE} = \frac{1}{N} \cdot \sum_{i=0}^{N} (h_\theta(x_i) - y_i)^2$$

Tā ir vidēja starpība starp pareizas un prognozētas vertībām, kas tiek pacelta kvadrātā. Kvadrāts palīdz izvairīties no negatīvām vērtībām. Kā arī kļūdas vērtība pieaug straujāk, salīdzinot ar MAE. Tiek pielietota regresijas uzdevumiem.

### BCE

Binary cross-entropy vai binārā krustentropija.

$$f(s_i) = \frac{1}{1 + e^{-s_i}}$$

Pielieto klasifikācijas uzdevumiem, kad modelim atļauts izvēlēties par pareizo vairāk nekā vienu klasi.

### CCE

Categorical cross-entropy vai kategoriskā krustentropija.

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Pielieto klasifikācijas uzdevumiem, kad vajag izvelēties tikai vienu klasi no dažādam.

## Atpakaļizplatīšanās algoritms

Šīs algoritms cenšas mainīt tīkla svarus un nobīdes tā, lai kļūda būtu 0.

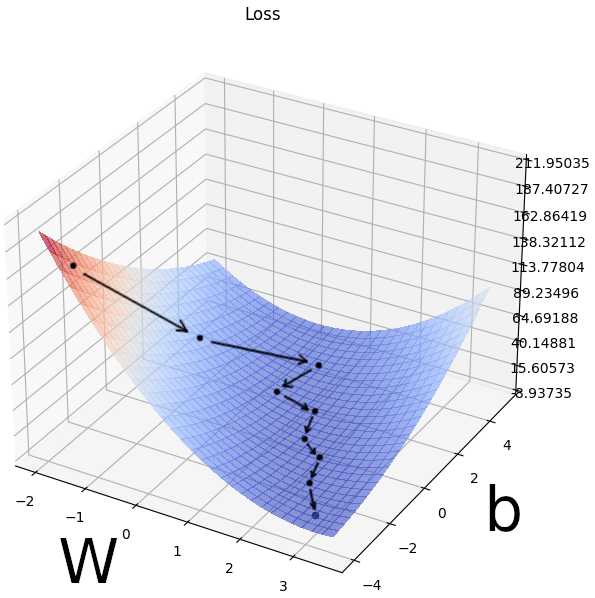
$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$b := b - \alpha \cdot \frac{\partial}{\partial b} J(b, W)$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$W := W - \alpha \cdot \frac{\partial}{\partial W} J(b, W)$$

Pieņemsim, ka modelis ir:

$$y' = M(x) = Linear(W_1, b_1, W_2, b_2, x) = Linear(W_2, b_2, ReLU(Linear(W_1, b_1, x)))$$
$$Linear(W_i, b_i, x_i) = W_i \cdot x_i + b_i$$
$$ReLU(x_i) = \begin{cases} x_i, x_i \geq 0 \\ 0, x_i < 0 \end{cases}$$

Kur, piemēram, $J(\theta_0, \theta_1) = L_{MAE} = \frac{1}{N} \cdot \sum_{i=0}^{N} (h_\theta(x_i) - y_i)$ ir MAE kļūdas funkcija un $\alpha$ ir apmācības koeficients, kurš noteic, cik strauji svars $W$ un nobīde $b$ tiek mainīti. Kļūdas funkcijas atvasinājums noteic vai parametru ir jāpalielina, vai jāsāmazina un uz kādu lielumu.




## Apmācāmo parametru optizācijas algoritmi

### Stohastiskā gradienta nolaišanās algoritms (SGD)

The goal of this is to update $\theta_0, b$ and $\theta_1, W$ to minimize cost function result.

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$b := b - \alpha \cdot \frac{\partial}{\partial b} J(b, W)$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$W := W - \alpha \cdot \frac{\partial}{\partial W} J(b, W)$$

but to simultaneously update both variables we need to assign them to temporary variables first, so b doesn't affect calculation of the W.

$\alpha$ - is a learning rate, which defines how fast we are going to change b and W.

By intuition what is going to happen – partial derivative of the cost function will be a positive slope (will return a positive number) if the mse will on the right side of the local minima, as a result we will subtract a slope value multiplied by learning rate from W (or b). Otherwise we will add it. In ideal situation once MSE is 0 – we don't move anymore.

So let's try to figure out partial derivatives:

For $\theta_0$ or $b$ :

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \cdot \frac{1}{N} \cdot \sum_{i=0}^{N} (h_\theta(x_i) - y_i)^2 = \frac{\partial}{\partial \theta_0} \cdot \frac{1}{N} \cdot \sum_{i=0}^{N} (\theta_0 + \theta_1 x_i - y_i)^2 = \frac{2}{N} \sum_{i=0}^{N} (\theta_0 + \theta_1 x_i - y_i) \cdot 1$$

equals to:

$$\frac{2}{N}\sum_{i=0}^{N}(\theta_0+\theta_1 x_i-y_i)\cdot 1=\frac{2}{N}\sum_{i=0}^{N}(b+W\cdot x_i-y_i)\cdot 1$$

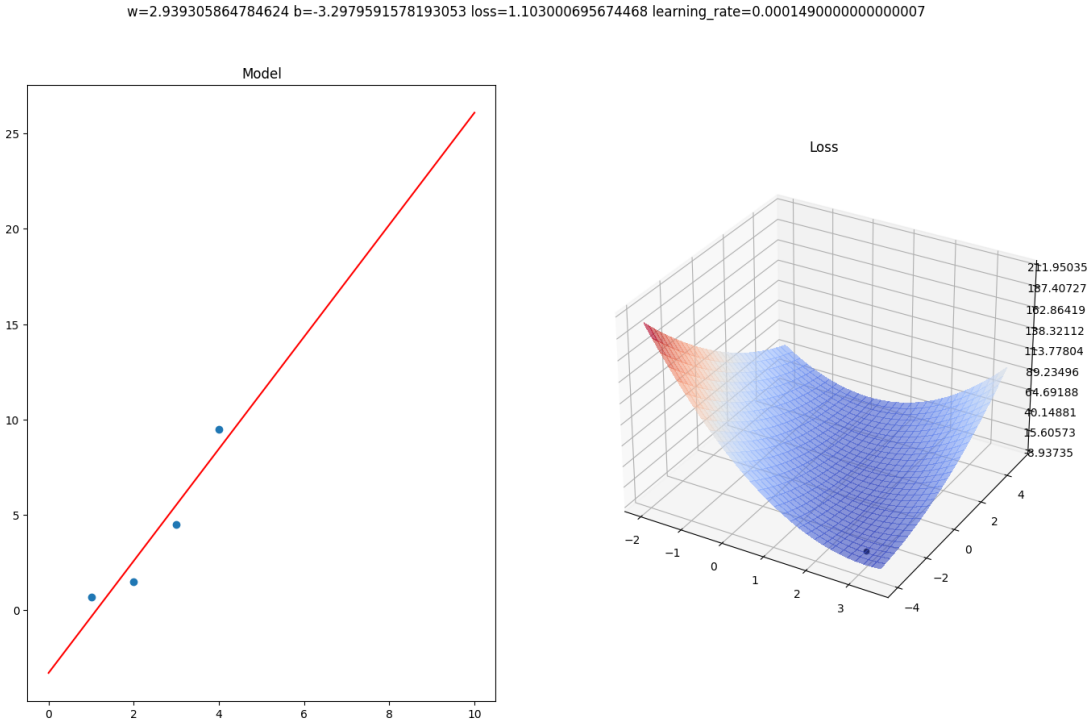
and for $\theta_1$ or $W$ :

$$\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)=\frac{\partial}{\partial\theta_1}\cdot\frac{1}{N}\cdot\sum_{i=0}^{N}(h_\theta(x_i)-y_i)^2=\frac{\partial}{\partial\theta_1}\cdot\frac{1}{N}\cdot\sum_{i=0}^{N}(\theta_0+\theta_1 x_i-y_i)^2=\frac{2}{N}\sum_{i=0}^{N}(\theta_0+\theta_1 x_i-y_i)\cdot x_i$$

equals to:

$$\frac{2}{N}\sum_{i=0}^{N}(\theta_0+\theta_1 x_i-y_i)\cdot x_i=\frac{2}{N}\sum_{i=0}^{N}(b+W\cdot x_i-y_i)\cdot x_i$$

For linear model:




For sigmoid model:

$$\frac{1}{1+e^{-x}}$$

We need to find a gradient descent:

$$\theta_0:=\theta_0-\alpha\cdot\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)$$

$$b:=b-\alpha\cdot\frac{\partial}{\partial b}J(b,W)$$

$$\theta_1:=\theta_1-\alpha\cdot\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)$$

$$W:=W-\alpha\cdot\frac{\partial}{\partial W}J(b,W)$$

$$\frac{\partial}{\partial \theta_0} J(\theta_0,\theta_1)=\frac{\partial}{\partial \theta_0}\cdot\frac{1}{N}\cdot\sum_{i=0}^{N}(h_\theta(x_i)-y_i)^2$$

where $h_\theta(x)=\frac{1}{1+e^{-(\theta_0+\theta_1\cdot x)}}=\frac{1}{1+e^{-(b+W\cdot x)}}$

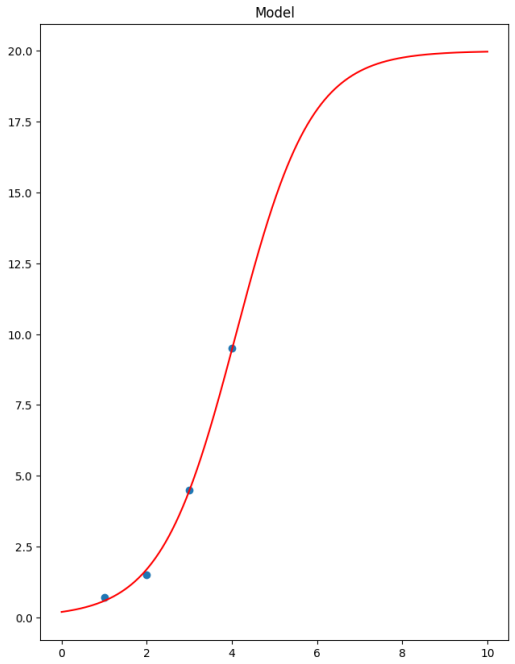
let's substitute $a=b+W\cdot x$ so $h_\theta(x)=\frac{1}{1+e^{-a}}$

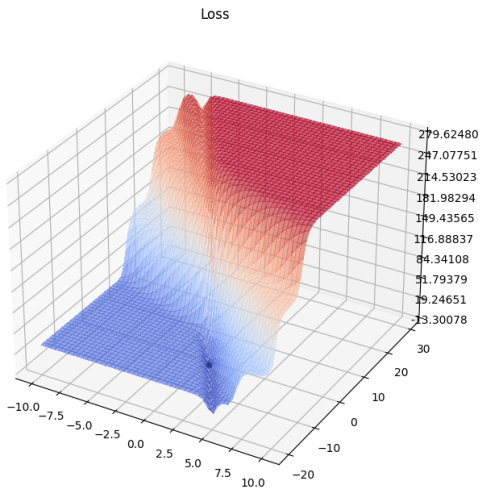
$$\frac{\partial}{\partial a}\frac{1}{1+e^{-a}}=\frac{\partial}{\partial a}(1+e^{-a})^{-1}=-(1+e^{-a})^{-2}\cdot\frac{\partial}{\partial a}(1+e^{-a})=-(1+e^{-a})^{-2}\cdot(\frac{\partial}{\partial a}1+\frac{\partial}{\partial a}e^{-a})=$$
$$=-(1+e^{-a})^{-2}\cdot(0+e^{-a}\cdot\frac{\partial}{\partial a}[-a])=-(1+e^{-a})^{-2}\cdot(e^{-a}\cdot-1)=\frac{e^{-a}}{(1+e^{-a})^2}=$$
$$=\frac{e^{-a}}{(1+e^{-a})\cdot(1+e^{-a})}=\frac{1\cdot e^{-a}}{(1+e^{-a})\cdot(1+e^{-a})}=\frac{1}{1+e^{-a}}\cdot\frac{e^{-a}}{1+e^{-a}}=$$
$$=\frac{1}{1+e^{-a}}\cdot\frac{e^{-a}+1-1}{1+e^{-a}}=\frac{1}{1+e^{-a}}\cdot(\frac{1+e^{-a}}{1+e^{-a}}-\frac{1}{1+e^{-a}})=\frac{1}{1+e^{-a}}\cdot(1-\frac{1}{1+e^{-a}})$$

$$\frac{\partial}{\partial \theta_0}\frac{1}{1+e^{-a}}=\frac{\partial}{\partial b}\frac{1}{1+e^{-a}}=\frac{\partial}{\partial a}\cdot\frac{\partial}{\partial b}=\frac{e^{-a}}{(1+e^{-a})^2}\cdot 1=\frac{1}{1+e^{-a}}\cdot(1-\frac{1}{1+e^{-a}})\cdot 1$$

$$\frac{\partial}{\partial \theta_1}\frac{1}{1+e^{-a}}=\frac{\partial}{\partial W}\frac{1}{1+e^{-a}}=\frac{\partial}{\partial a}\cdot\frac{\partial}{\partial W}=\frac{e^{-a}}{(1+e^{-a})^2}\cdot x=\frac{1}{1+e^{-a}}\cdot(1-\frac{1}{1+e^{-a}})\cdot x$$

w=1.1346469402126882 b=-4.6482628586296135 loss=0.0140181278184008896 learning_rate=0.0001490000000000007

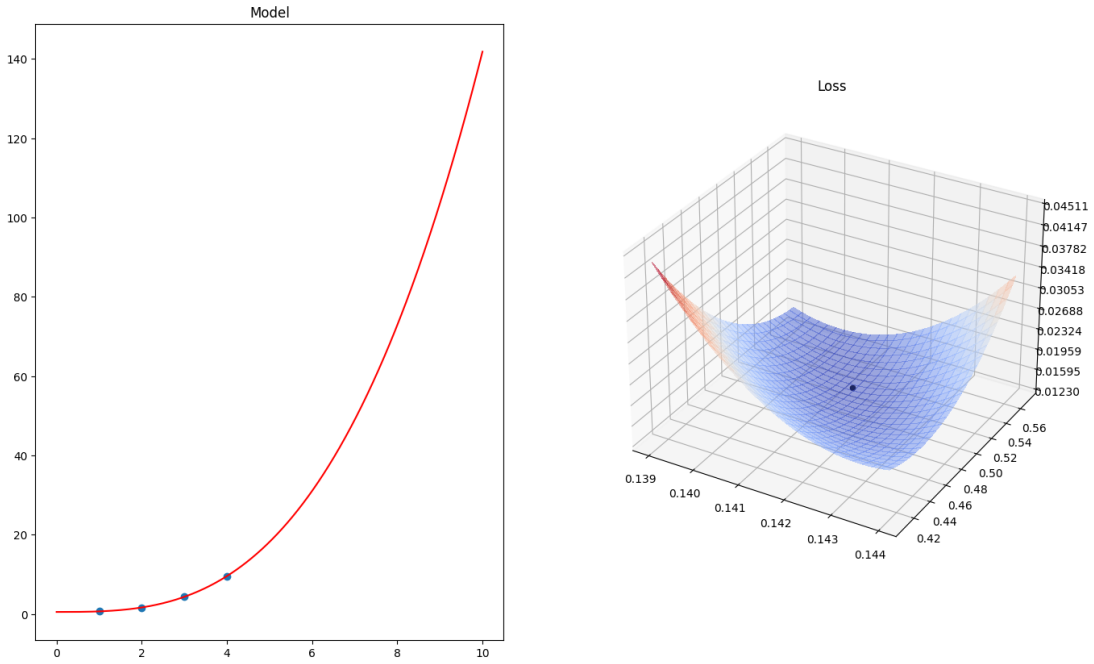



For cubic function $b+W\cdot x^3$, where $b$ would shift the line vertically and $W$ would regulate its width. That was my intuition on how to fit the line. The nice part here is that loss function derivative $dW, db$ is the same as for linear. And $dx$ is:

$$dx\,cubic: \frac{\partial}{\partial x}[W\cdot x^3+b]=W\cdot 3\cdot x^2$$

w=0.1413461538461152 b=0.5163461538462171 loss=0.013832747781064567 learning_rate=0.0014859999999999995




## RMSprop

Stohastiskā gradienta nolaišanas algoritmam ir potenciāla problēma – apmācības koeficients ir konstants un visiem parametriem ir viens in tas pats. Tas derētu viegliem uzdevumiem, kuriem kļūdas virsma ir glūda. Taču gadījumos, kad neironu tīkls ir sarežģīts un kļūdas virsma arī ir sarežģīta, gradients var vai nu pazust, vai nu "eksplodēt".

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

Situāciju var uzlabot adaptīvs apmācības koeficients, kas nozīmē, ka viņš mainās ar laiku un vairs neskaitās par hiperparametru. Šīs algoritms samazina soli lielam gradientam lai izvairītos no "eksplozijas" un palielina soli mazam gradientam lai izvairītos no pazušanas.

## Metrikas

Lai varētu saprast, cik labi modelis funkcionē, eksistē dažāda veida metrikas.

| | | Minējums | |
|---|---|---|---|
| | | Pozitīvs | Negatīvs |
| Patiesība | Pozitīvs | TP | FN |
| | Negatīvs | FP | TN |

Divu variantu gadījumā var sastādīt tabulu, kur **T*** ir situāciju skaits, kad minējums sakrīt ar patieso atbildi. Un **F*** ir situāciju skaits, kad minējums nesakrīt.

### Akurātība

Tā ir pareizo minējumu skaita pret kopēju minējumu skaitu attiecība:

$$akurātība = \frac{pareizo\ minējumu\ skaits}{kopējais\ minējumu\ skaits} = \frac{TP + TN}{TP + TN + FP + FN}$$

Šo metriku var izmantot gadījumos, kad dati ir sabalansēti. Piemēram, ja modelis neko neprognozē, bet tikai atgriež vienu un to pašu atbildi, kuras biežums ir 95%, tad sanāks, ka modelim akurātība ir 0.95. Kas nav taisnība un visos citos gadījumos modelis nekad neatgriezīs pareizo atbildi.

### Precizitāte

Precision should ideally be 1 (high) for a good classifier. Precision becomes 1 only when the numerator and denominator are equal i.e TP = TP +FP, this also means FP is zero.

$$precīzitāte = \frac{TP}{TP + FP}$$

As FP increases the value of denominator becomes greater than the numerator and precision value decreases (which we don’t want).

### Recall

Recall should ideally be 1 (high) for a good classifier. Recall becomes 1 only when the numerator and denominator are equal i.e TP = TP +FN, this also means FN is zero.

$$recall = \frac{TP}{TP + FN}$$

As FN increases the value of denominator becomes greater than the numerator and recall value decreases (which we don’t want).

## F1 reitings

The F1 score is needed when accuracy and how many of your ads are shown are important to you. We've established that Accuracy means the percentage of positives and negatives identified correctly. Now, we'll investigate "F1 Score," which is a way to measure how much accuracy is present in your dataset.

$$F1 reitings = 2 \cdot \frac{precīzitāte \cdot recall}{precīzitāte + recall}$$

We tend not to focus on many different things when making decisions, whereas false positives and false negatives usually have both tangible and intangible costs for the business. The F1 score may be a better measure to use in those cases, as it balances precision and recall.

## Hiperparametri

Hyperparameters are defined as the parameters that are explicitly defined by the user to control the learning process.

These parameters express "High Level" properties of the model such as its complexity or how fast it should learn. Hyperparameters are usually fixed before the actual training process begins.

Hyperparameters can be divided into two categories:-

1.Optimizer Hyperparameters

These are the variables or parameters related more to the optimization and training process then to model itself. These parameters help you to tune or optimize your model before the actual training process starts so that you start at the right place in the training process.

- Learning Rate: It is a hyperparameter that controls how much we are adjusting the weights of our neural network with respect to the gradient.
- Mini-Batch Size: It is a hyperparameter that has an effect on the resource requirements of the training and also impacts training speed and number of iterations.
- Number of Training Iteration or Epochs: It is one complete pass through the training data.

2.Model Hyperparameters

These are the variables which are more involved in the architecture or structure of the model. It helps you to define your model complexity on the basis of :

- Number of Layers: Layer is a general term that applies to a collection of 'nodes' operating together at a specific depth within a neural network. Different layers like Input layer, hidden layer and output layer.
- Hidden Units: Number of hidden layers in a neural network, the more complex model(means more hidden layers), the more learning capacity the model will need.

- Model Specific parameters for the architectures like RNN: Choosing a cell type like LSTM Cells, Vanilla RNN cells or GRU Cells) and how deep the model is.

## 1.2. Attēlu klasifikācija

Konvolūcijas tikls (ConvNet)

ResNet / DenseNet

InceptionNet

## 1.3. PyTorch vide

PyTorch ir optimizēta mašīnmācīšanas bibliotēka, kura atvieglo darbu gan ar procesoru (CPU), gan ar videokārti (GPU). Programēšanas valoda ir Python.

Modeļu implementāciju <> viendājumi

### Datu ielādes process

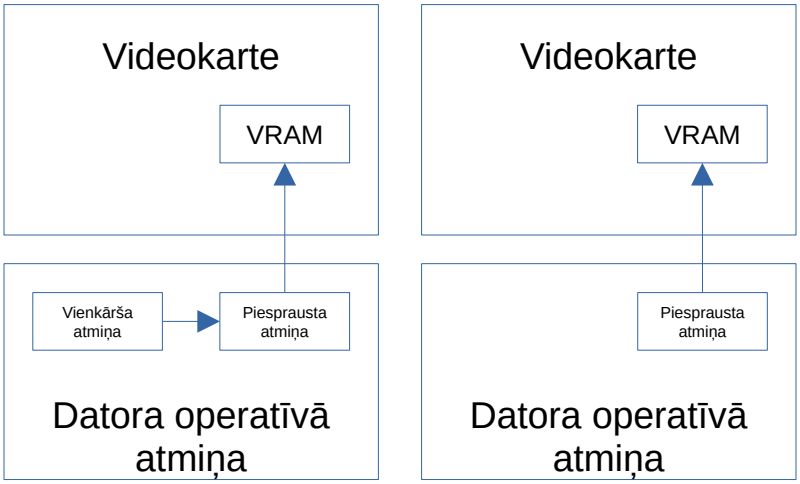
Datu ielādes process sastāv no datu kopas (Dataset) un datu ielādēja (DataLoader) klasēm. Dataset klase ir atbildīga par datu kopas izmēra noteikšanu caur **__len__()** metodi un pēc indeksa izvēlēta ieraksta atgriešanu caur **__getitem__(index)** metodi. Pašus datus var lejupielādēt no tīmekļa, vai ielādēt no diska inicializācijas solī.

```
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self):
        super().__init__()
        self.x = [1, 2, 3, 4, 5, 6]
        self.y = [0, 1, 0, 1, 0, 1]

    def __getitem__(self, index):
        return self.x[index], self.y[index]

    def __len__(self):
        return len(self.x)
```

Pēc datu kopas definīcijas un inicializācijas to sadala uz apmācīšanas un pārbaudes kopām. Tālāk katru no tām padod datu ielādejiem. Tālāk darbs notiek ar datu ielādejiem.

```
data_loader_train = torch.utils.data.DataLoader(
    dataset=dataset_train,
    batch_size=BATCH_SIZE,
    shuffle=True,
    pin_memory=USE_CUDA,
    num_workers=WORKER_COUNT
)
```

Jo lielāka datu kopa ir, jo lielāki vajadzīgi: operatīva atmiņa (RAM), operatīva atmiņa videokartē (VRAM), kā arī vieta uz cieta diska. Un jo lielāki dati, jo vairāk tie pārvietosies no diska uz RAM un pēc tam uz VRAM.
Gadījumos, kad RAM/VRAM nav pietiekoši daudz, var izmantot **batch_size** lai vienlaikus ielādētu tikai noteikto datu eksemplāru skaitu. Lai paātrinātu datu ieladi, var arī izmantot **pin_memory=True**, lai datus ielādētu "piespraustā" CUDA atmiņā bez vienkāršas atmiņas starpniecības.




Gadījumā, ja datu partijas ielāde aizņem vairāk laika nekā to izmantošana apmācībai, tad to var ielādēt paralēli vairākos procesos (**num_workers**).

# 2. Metodoloģija

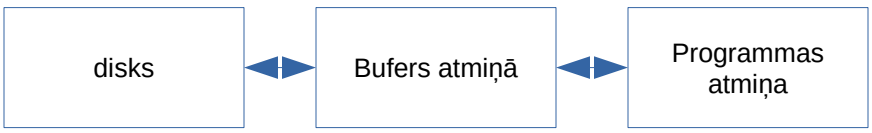
## 2.1. Datu ielādes metodes

### 2.1.1. Failu sistēma

Šī ir visvienkārša metode. Notiek tikai lejupielādēto datu saglabāšana failos uz cietā diska failu sistēmas, no kuras pēc tam notiks ielāde. Bildes tiks iepriekš pārveidotas (samazinātas, apgrieztas) lai netērētu laiku un resursus transformācijai katrā epohā no jauna.

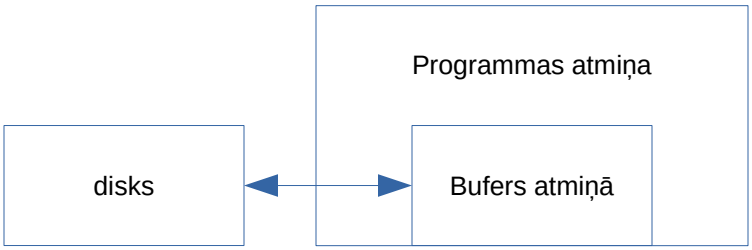
### 2.1.2. NumPy mmap

Mmap (memory map) funkcionālitāte ļāuj programatūrai interpretēt saturu uz diska kā saturu operātīvajā atmiņā. Bez **mmap()** faila ielāde notiek šādi:




Kad programma lasa faila saturu, operētājsistēma vispirms ielāde to saturu buferā pa daļām (lapām) un pēc tam programmas atmiņā. Ja fails ir lielāki par brīvas operatīvās atmiņas apjomu, tad

operētājsistēma sāks izspiest datus uz swap failu, ja tāds vispār ir. Tas krietni palēninās visas sistēmas ātrdarbību. Ja swap faila nav, vai pat tājā vieta izbeidzās, tad apstās kādu programmu, kura mēģinās tājā brīdī pieprāsīt vairāk atmiņas.




**Mmap()** gadījumā bufers tiek iebūvēts programmas atmiņā, izslēdzot lieko datu parraides soli.

Metodei ir arī trūkumi:

- Datiem jāatrodas uz cietā diska lokāli, tie nevar atrasties attālinātos resursos. (Tomēr ir daži risinājumi, lai pieslēgtu attālinātus resursus kā lokālus)
- N-dimensionālo masīvu griezumi var krietni palielināt lasījumu skaitu no diska.

### 2.1.3. CuPy mmap

PyTorch also supports zero-copy data exchange through `DLPack` (see DLPack below):

```
import cupy
import torch

from torch.utils.dlpack import to_dlpack
from torch.utils.dlpack import from_dlpack

# Create a PyTorch tensor.
tx1 = torch.randn(1, 2, 3, 4).cuda()

# Convert it into a DLPack tensor.
dx = to_dlpack(tx1)

# Convert it into a CuPy array.
cx = cupy.from_dlpack(dx)

# Convert it back to a PyTorch tensor.
tx2 = from_dlpack(cx.toDlpack())
```

### 2.1.4. HDF5

Hierarhiskais datu formats tiek domāts liela apjoma datu glabāšanai un organizēšanai. Hierarhijas tiek definētas POSIX-veida sintakša veidā, piemēram “/path/to/dataset”.

### 2.1.5. Zarr

Zarr ir moderna alternatīva NumPy-veidīgo N-dimensionālo masīvu glabāšanai atmiņā, uz diska, ZIP arhīvā, AWS S3 glabātuvē vai key-value datubāzē. Zarr funkcionālitāti var arī paplašināt ar citu

glabātuvi. Tā atbalsta paralēlo lasīšanu un rakstīšanu no dažādiem procesiem. Datu masīvus var organizēt hierarhijās. Tāpat kā HDF5 ir iespēja definēt datu gabala izmēru, ka arī kompresēt datu gabalus, pielagojot to efektīvakai lasīšanai. Šī metode ir noderīga datu lasīšanai no tīkla.

## 2.2. Datu kopas

### 2.2.1. CIFAR10

CIFAR-10 datukopa satur 60 tūsktošus 32x32 krāsainas bildes, kuras ir sadalītas 10 klasēs, katrai klasei atbils 6000 bildes. Datukopa ir sadalīta uz 50000 bildēm apmācībai un 10000 bildēm pārbaudei.

### 2.2.2. Tiny ImageNet

???

### 2.2.3. Images with people wearing masks

Tā ir salīdzinoši augstas izšķirtspējas datu kopa (), kuras attēli tiks samazināti un apgriezti līdz kopējam 512x512 izmēram.

^ Atrast dažādu izmēru attēlu datu kopas

^ 3 dažādu izmēru / kofigurāciju datu kopas

## 2.3. Apmācības protokols

* Jāizdomā metrikas kā noteikt ietekmi datu ielādes metodēm

* Vidējais epocha ātrums sekundēs

# 3. Rezultāti

Datu ielādes metodes PRET datu kopām

# 4. Tālākie pētījumi

# 5. Secinājumi