DATAHACKER

Neural Network and Deep Learning

datahacker.rs

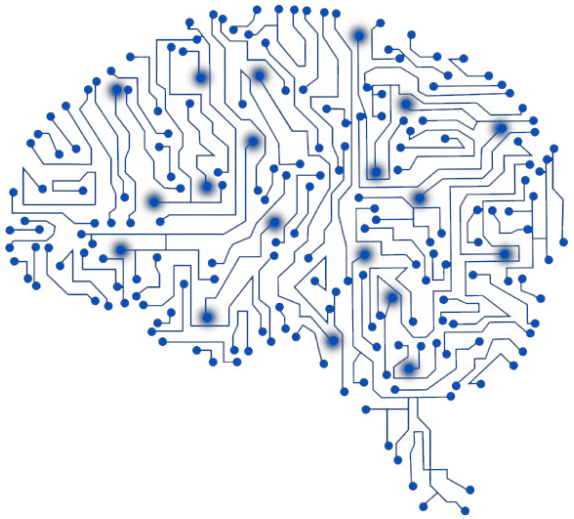
DATAHACKER

What is Deep Learning?

001 – What is Deep Learning?

DATAHACKER

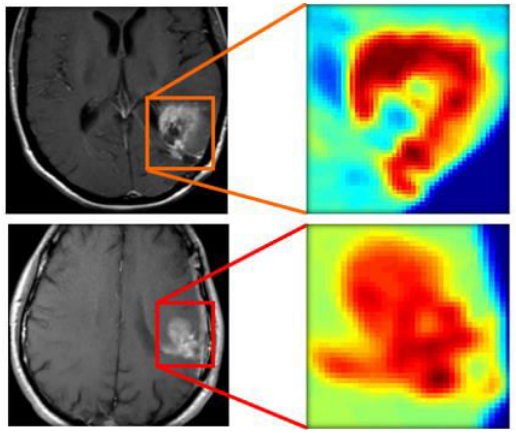
- Deep Learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making.
- Deep Learning transformed internet businesses like web search and advertising.
- Deep Learning is enabling brand new products and businesses.

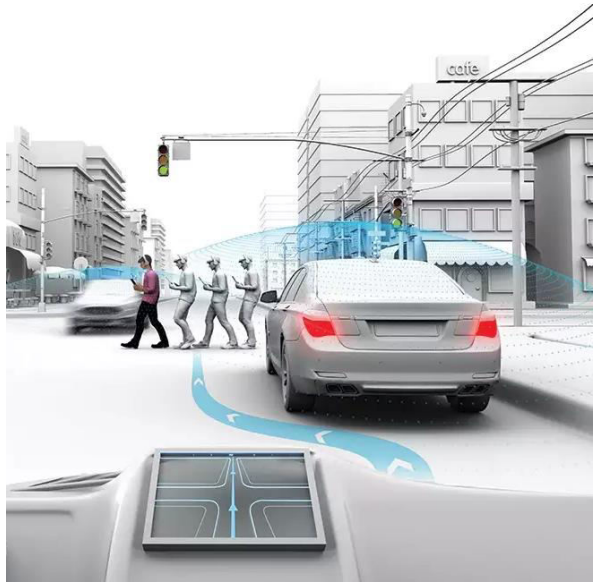


001 – What is Deep Learning?

DATAHACKER

- Deep Learning uses:
  - Medicine (reading x-ray images)
  - Self driving car
  - Agriculture (precision agriculture)
  - And many more...





001 – What is Deep Learning?

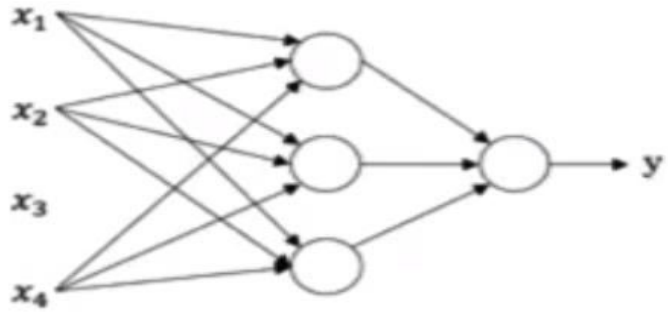
DATAHACKER

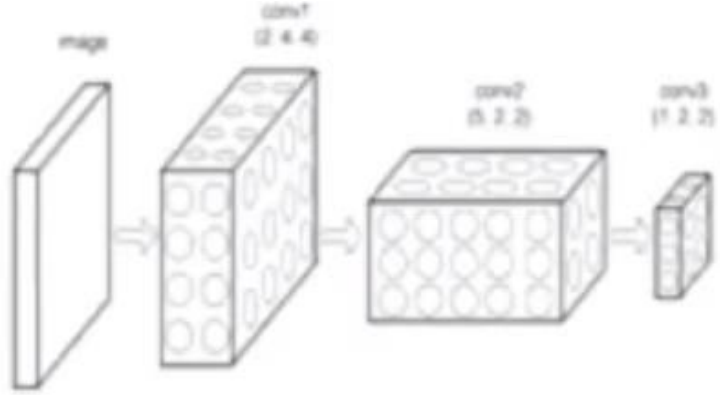
- Opportunity to create an amazing world powered by AI.
- AI is the new electricity
- Electricity transformed every major industry, from manufacturing to healthcare and many more.
- AI is on path to bring about an equally big transformation.
- Big part of rapid rise of AI is driven by developments in deep learning.
- One of the highly sought out skills.

001 – What is Deep Learning?

DATAHACKER

- What will we learn?
  - Neural Networks and Deep Learning
    - You will learn how to build a neural network including a deep neural network
    - How to train it on data.
    - In the end, you will be able to build a deep neural network to recognize cats.
  - Improving Deep Neural Networks
    - Hyperparameter tuning, Regularization, Optimization, how to diagnose bias and variance.
  - Structuring your Machine Learning project
    - The strategy has changed in the era of deep learning.
    - A split of train and test
    - End-to-end deep learning
  - Convolutional Neural Networks
    - Often applied to images
  - Natural Language Processing
    - Building sequence models
    - Applying them to sequence data
    - Natural language is just a sequence of words





DATAHACKER

Neural Network

datahacker.rs

002 - Neural Network

DATAHACKER

- Deep learning refers to training neural networks sometimes very large neural networks.
- A Neural Network, is a technology built to **simulate the activity of the human brain**. Specifically, pattern recognition and the passage of input through various layers of simulated neural connections.
- Inspired by human mind
- Neurons are **activated** by **some** input
- Massively used for problems of classification and regression
- Archived amazing results in the field
- Not so new, developed in 1940s, 1950s and 1960s

002 - Neural Network

DATAHACKER

Where can neural networks be applied.

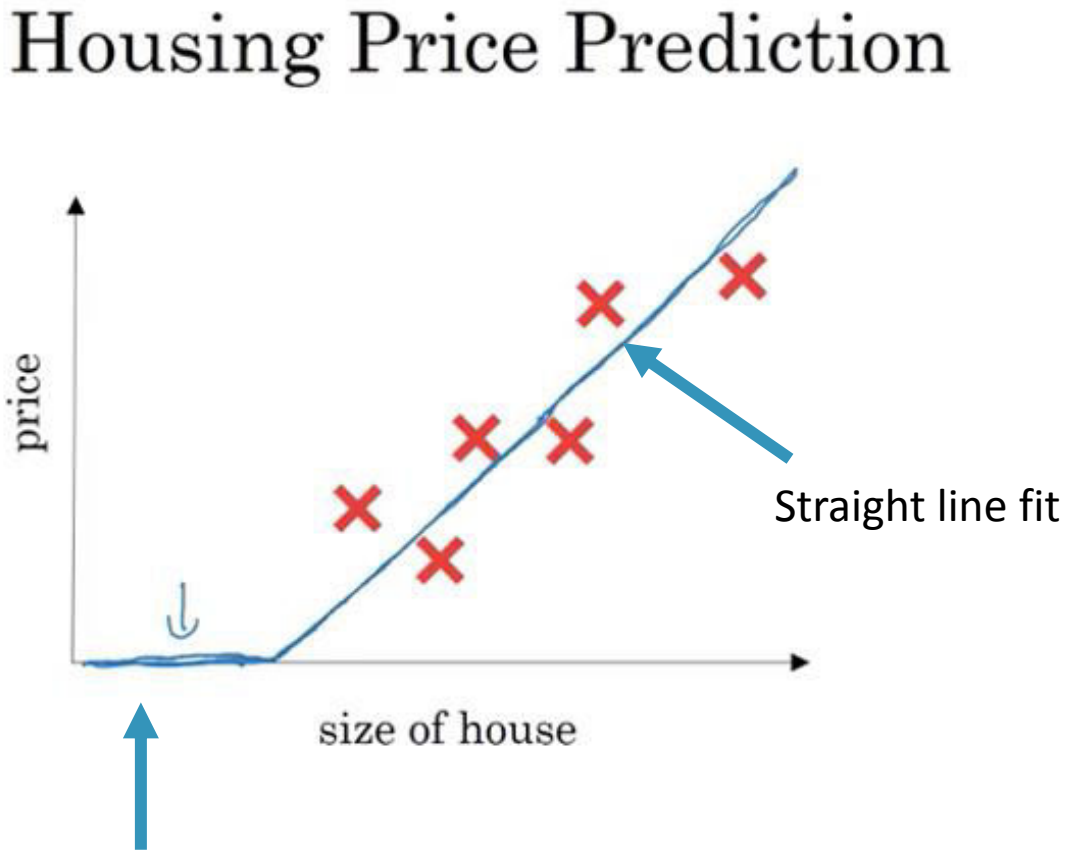
Speech Recognition	Object Recognition on Images	Driving Cars (Googles self-driving car)
House Price Prediction	Image Processing (Face Recognition)	Advertisement (Whether a user would click on an advertisement {0, 1 })

002 - Neural Network

DATAHACKER

Housing Price Prediction.

We have a dataset with six houses, and we know the size of these houses in square feet or square meters and the price. Now we want to fit a function to predict the price of a house as a function of the size, so we apply a straight line to our graph.

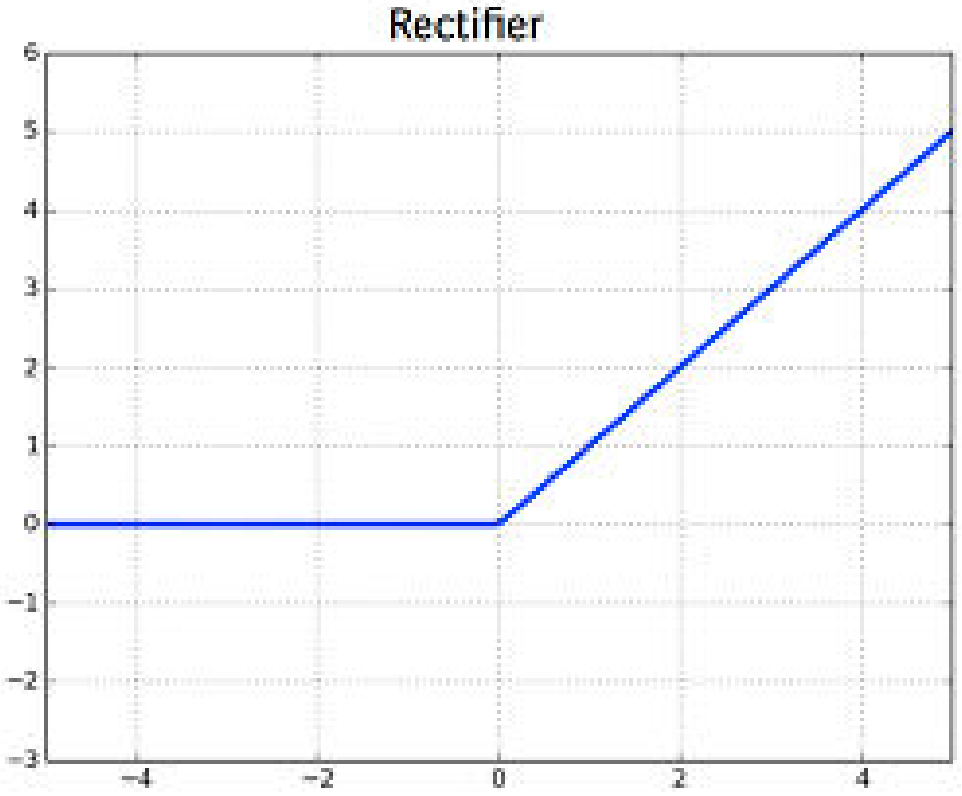


Since we know that the price can never be negative, we bent the curve so it ends up been 0.

002 - Neural Network

DATAHACKER

ReLU (Rectified Linear Activation)

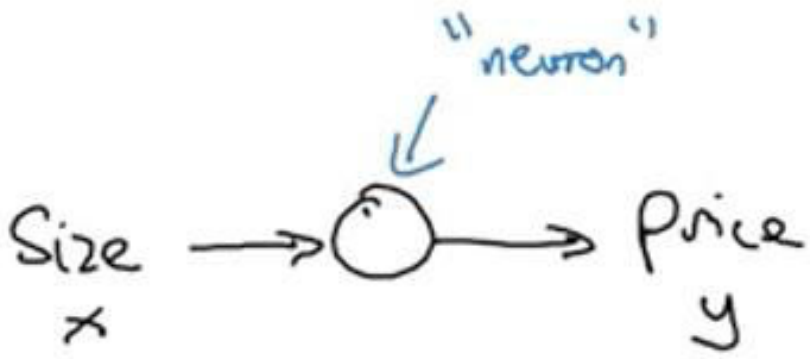


$$RELU(x) = \begin{cases} 0 \; if \; x < 0 \\ x \; if \; x >= 0 \end{cases}$$

002 - Neural Network

DATAHACKER

- This is a tiny little neural network and larger neural network are formed by taking many of these single neurons and stacking them together.



X (size of the house) goes into this node this little circle (hidden unit) and then outputs Y, which is the price of the house. This little circle is a single neuron in a neural network implementation.

002 - Neural Network

DATAHACKER

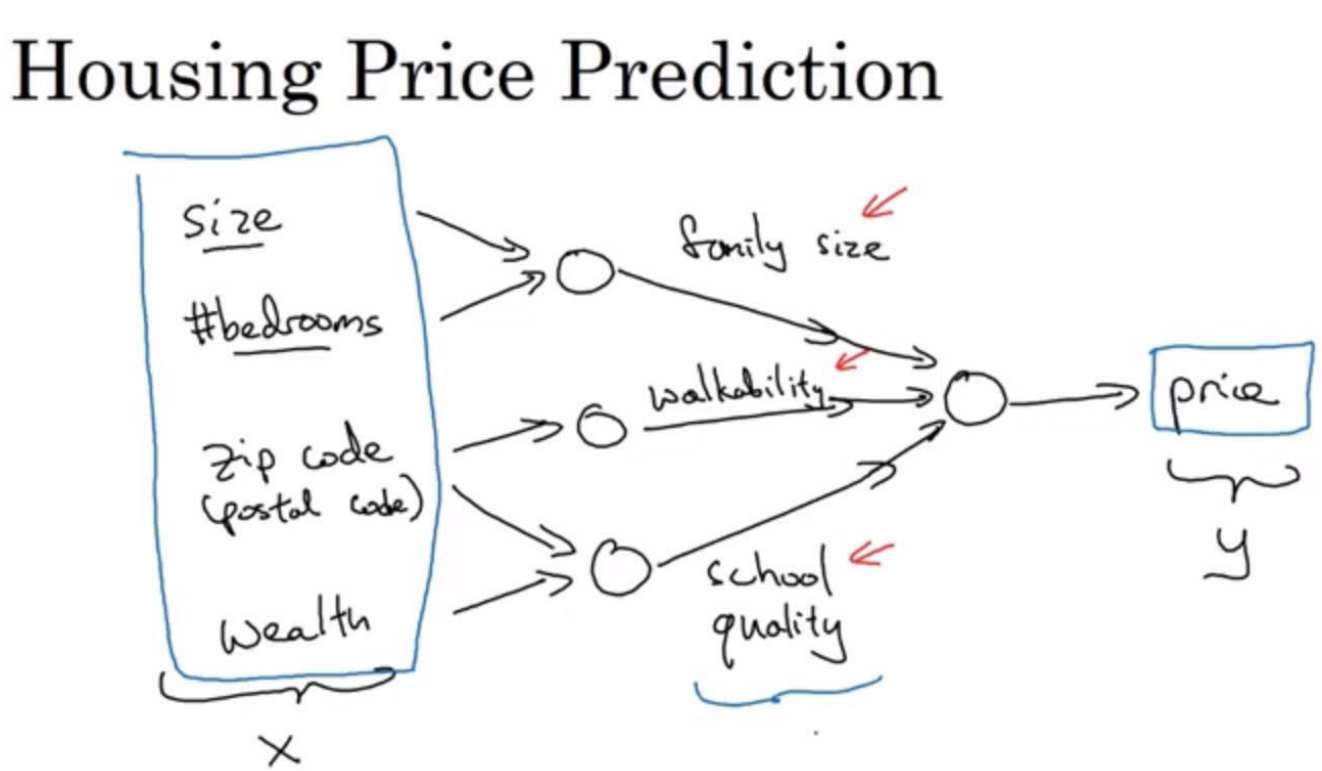
- Now we have know other things about these house other than just the size {features}

  - number of bed rooms

  - zip code & postal code { country }

  - wealth { school quality, grocery store }

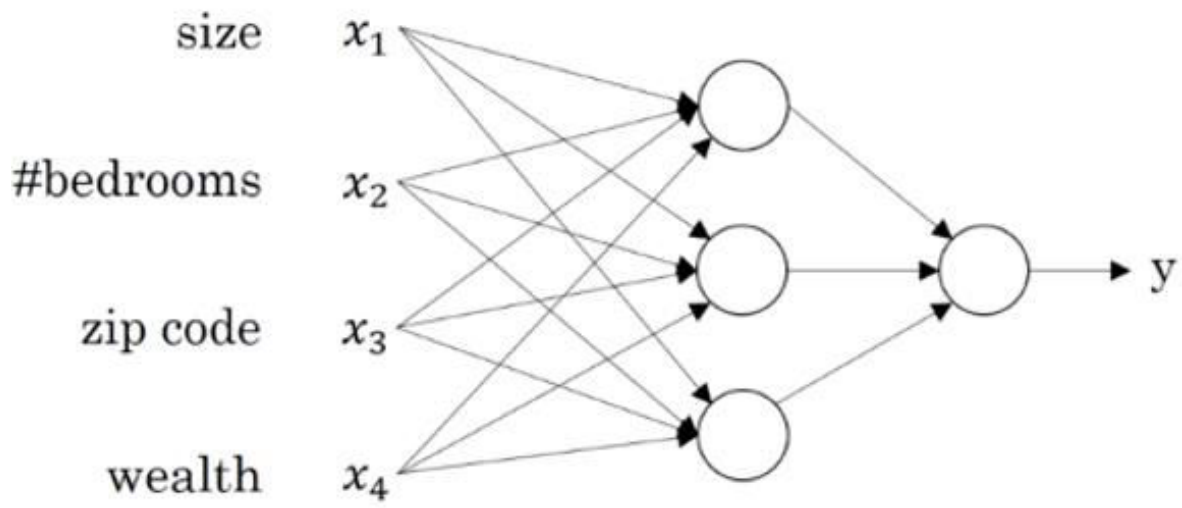
- The family size, walkability and school quality helps us predict the price.



002 - Neural Network

DATAHACKER

- In our example, “ X ” are all of these four inputs and “ Y ” is the price we are trying to predict. So by stacking together a few of the single neurons we now have a slightly larger network.
- The magic of a neural network is that when you implement it, you need to give it just the inputs “ X “ and the output “ Y ” for a number of examples in your training set and all these things in the middle, they will figure out by itself.



DATAHACKER

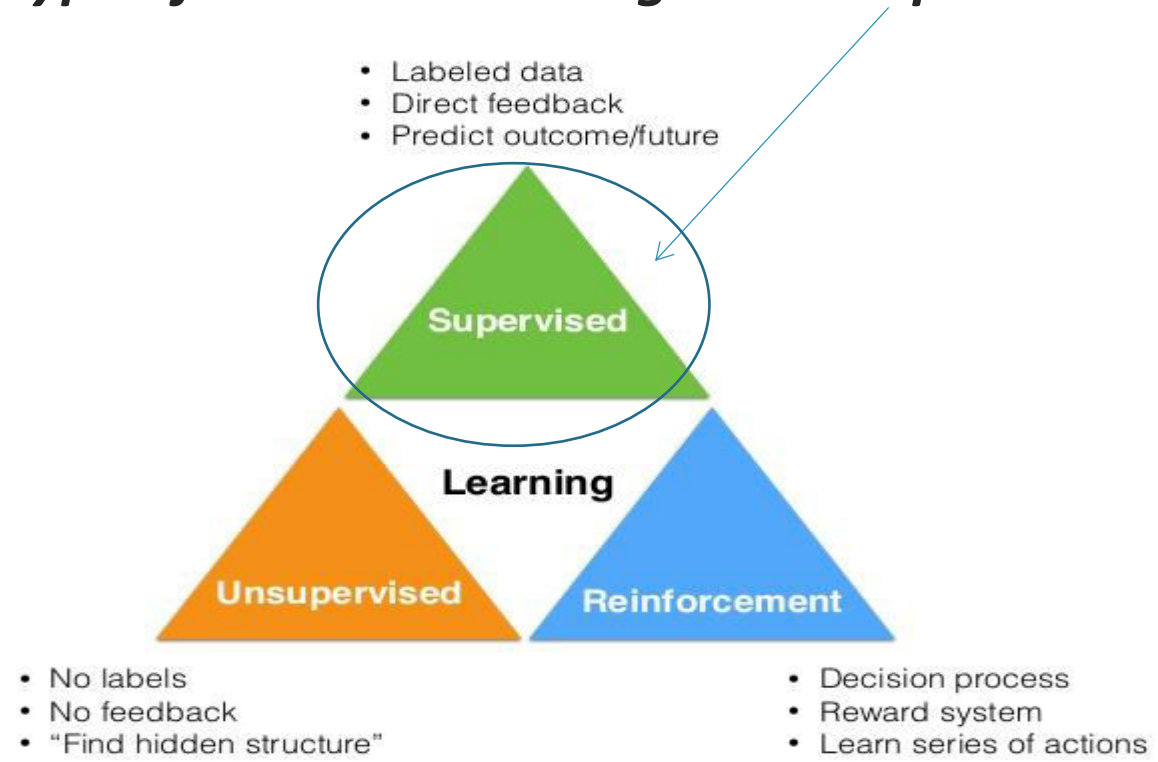
Supervised Learning with Neural Networks

datahacker.rs

003 - Supervised Learning with Neural Networks

DATAHACKER

- There's been a lot of hype about neural networks and perhaps some of that hype is justified given how well they're working.
- But it turns out that so far almost all the economic value created by a neural networks has been through ***one type of machine learning called supervised learning.***



003 - Supervised Learning with Neural Networks

DATAHACKER

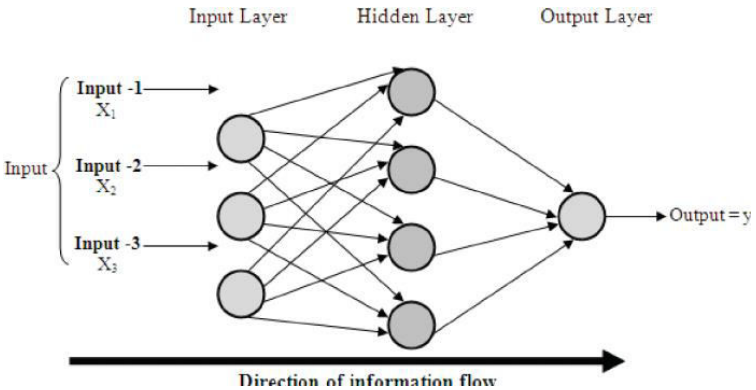
- You have some **input X** and you want to learn the function mapping to some **output Y.**
- So for example just now we saw the housing price prediction application where you input some features of a home and try to output or estimate the price Y.
- **The single most lucrative application of deep learning today is online advertising**. Neural networks have gotten very good at predicting whether or not you click on an ad and by showing you and showing users the ads that you're most likely to click on.
- **You might input an image and want to output an index from one to a thousand** trying to tell you if this picture it might be any one of a thousand different images so you might use that for **photo tagging**.
- The recent progress **in Speech recognition** has also been very exciting where you can **now input an audio clip** to, a neural network can have it **output a text transcript**.
- **Machine translation** has also made huge strikes thanks to deep learning where now you can have a neural network **input an English sentence** and directly **output their Chinese sentence**.
- **In Autonomous driving** you might input then say a picture of what's in front of your car as well as some information from a radar and based on that maybe in your network can be trained to tell you the position of the other cars on the road so **this becomes a key component in autonomous driving systems.**

Input(x)	Output(y)	Application	
Home features	*Price*	***Real Estate***	Standard Neural Network
Ad, user info	*Click on ad?(0/1)*	***Online Advertising***	
Image	*Object (1,….,1000)*	***Photo tagging***	Convolutional Neural Network
Audio	*Text transcript*	***Speech recognition***	Recurrent Neural Network
English	*Chinese*	***Machine translation***	
Image, Radar info	*Position of other cars*	***Autonomous driving***	Custom / Hybrid Neural Network

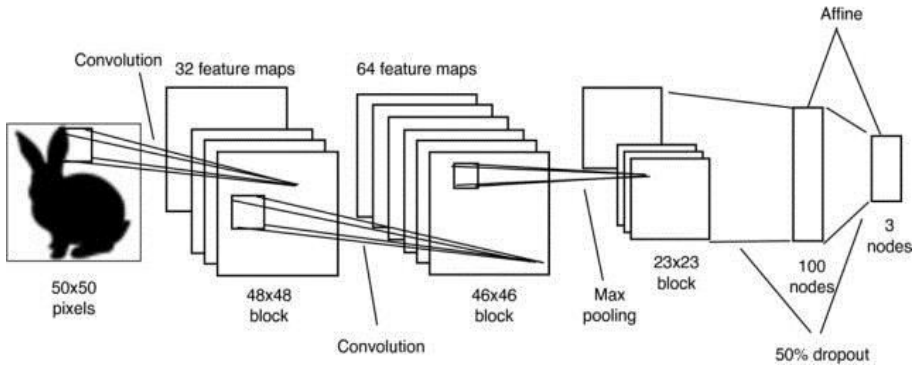
003 - Supervised Learning with Neural Networks

DATAHACKER

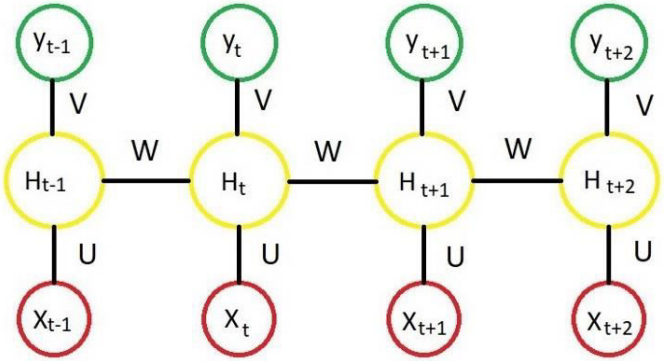
Neural Network examples



Standard Neural Network (SNN)



Convolutional Neural Network (CNN)



U = Weight vector for Hidden layer
V = Weight vector for Output layer
W = Same weight vector for different Timesteps
X = Word vector for Input word
y = Word vector for Output word

Recurrent Neural Network (RNN)

003 - Supervised Learning with Neural Networks

DATAHACKER

Structured Data

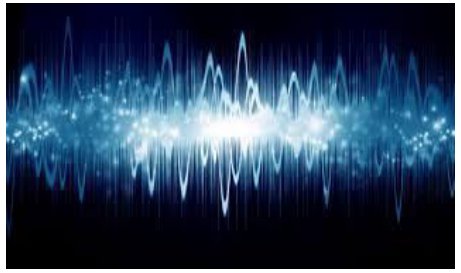
- **databases** of data with a columns.
- each of the features such as size of a house on upper bedrooms or the age of a user has a **very well-defined meaning**.

Size	#bedrooms	Price(1000$s)
2104	3	400
1600	4	540

User Age	Ad Id	Click
41	93242	0
18	71244	1

Unstructured Data

- refers to things like **raw audio, images or text**.
- it's been much harder for computers to make sense of unstructured data compared to structured data.



Audio

Image

Do you mind if I maybe come over? I'm sorry I don't know if it's going to get better soon

Text

003 - Supervised Learning with Neural Networks

DATAHACKER

- **Thanks to neural networks computers are now much better and interpreting unstructured data as well compared to just a few years ago.** This creates opportunities for many new exciting applications that use speech recognition, image recognition, natural language processing on text much more than it was possible even just two or three years .
- Creating has also been **on structured data such as much better advertising systems , much better private recommendations** and just **a much better ability to process the giant databases** that many companies have to make accurate predictions from them .
- So **Neural Networks have transformed Supervised Learning** and are creating tremendous economic value. It turns out that the basic technical ideas behind Neural Networks have mostly been around sometimes for many decades , so why is it then that they're only just now taking off and working so well.

DATAHACKER

The Rise of Deep Learning

datahacker.rs

004 – The Rise of Deep Learning

DATAHACKER

- If the basic technical idea behind deep learning neural networks has been around for decades why are they only now taking off?

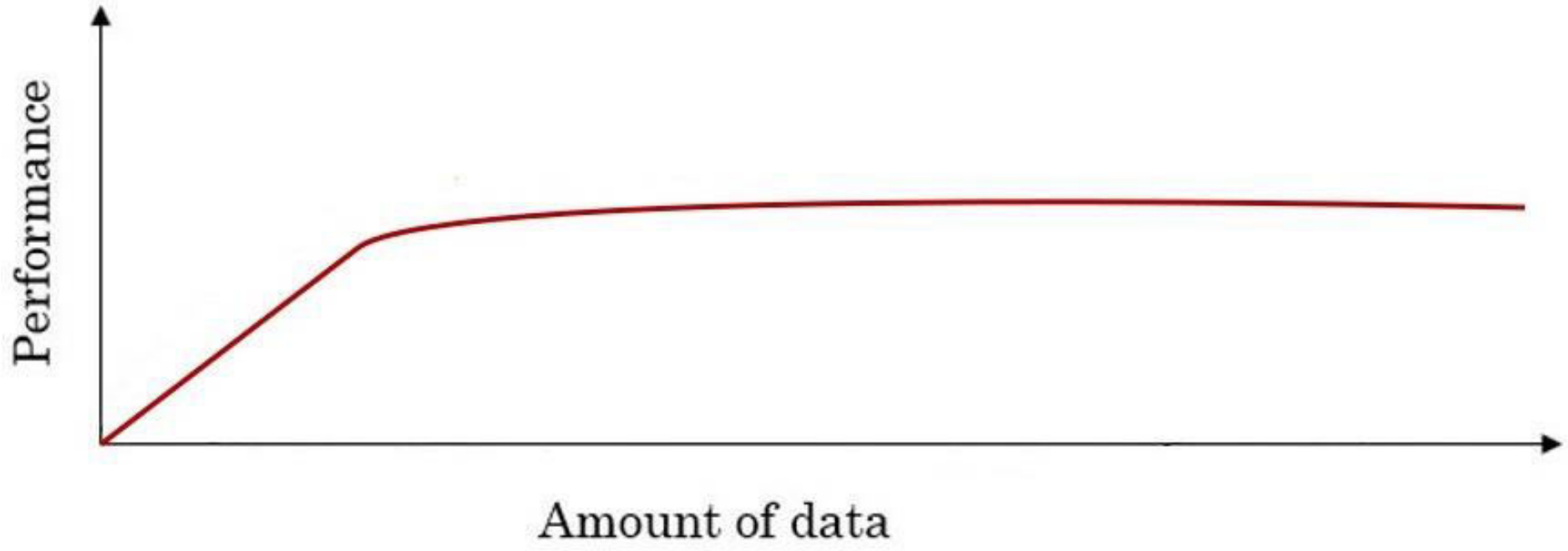
- To answer this question, we plot a figure where on the x-axis we plot the amount of labeled data we have for a task, and on the y-axis we plot the performance of our learning algorithm (accuracy).

- For example, we want to measure accuracy of our spam classifier or the accuracy of our neural net for figuring out the position of other cars for our self-driving car.

004 – The Rise of Deep Learning

DATAHACKER

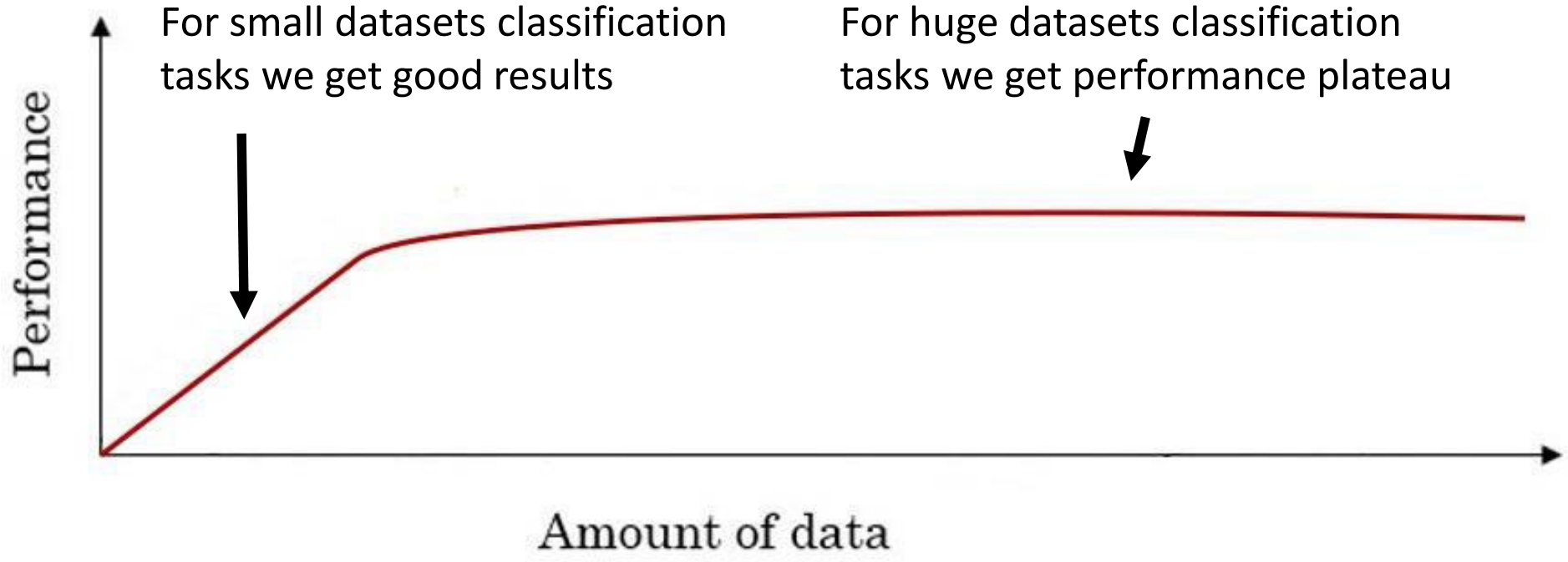
- If we plot the performance of a traditional learning algorithms such as Support Vector Machine or Logistic Regression as a function we will get the following curve:



004 – The Rise of Deep Learning

DATAHACKER

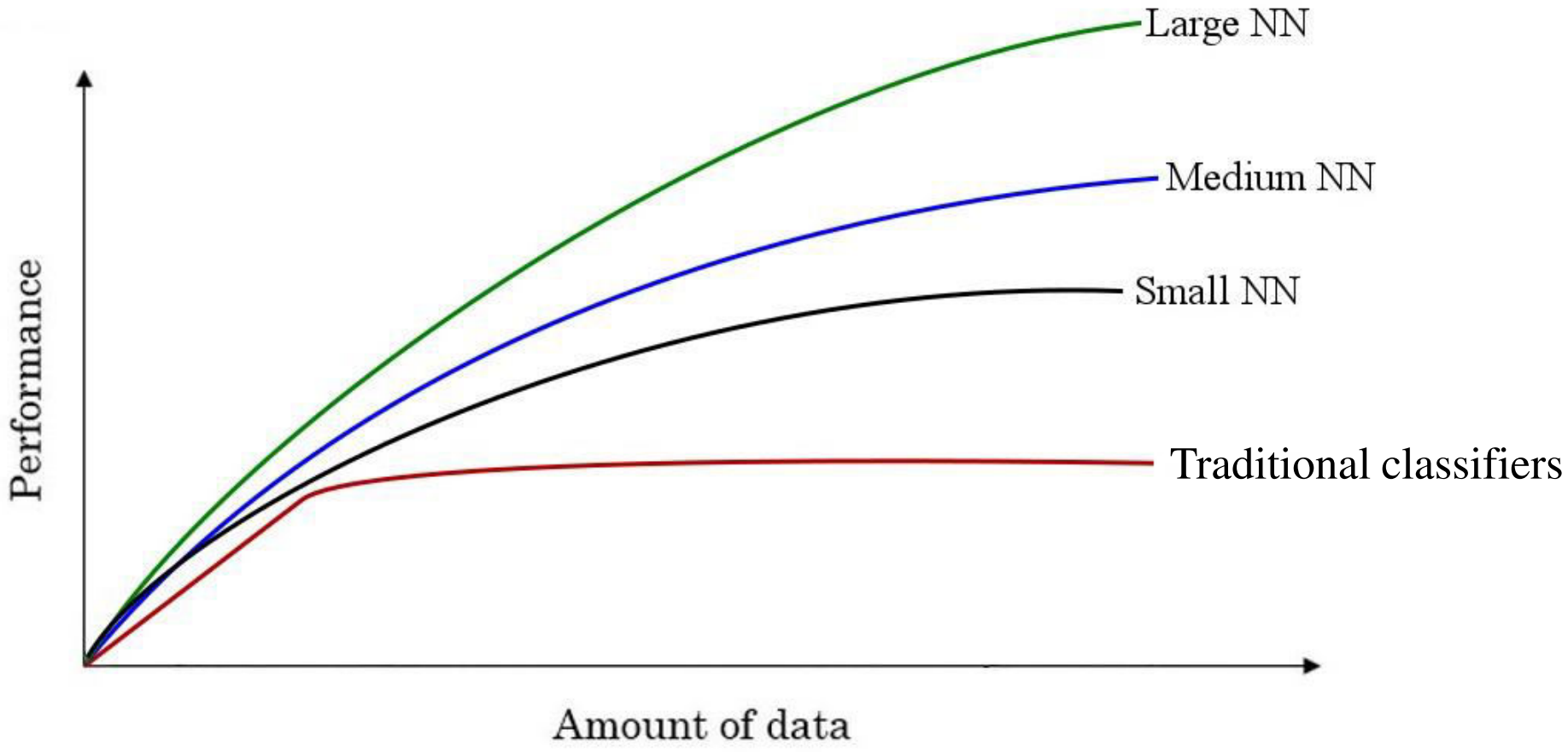
- If we plot the performance of a traditional learning algorithms such as Support Vector Machine or Logistic Regression as a function we will get the following curve:



004 – The Rise of Deep Learning

DATAHACKER

- How to overcome performance plateau problem?

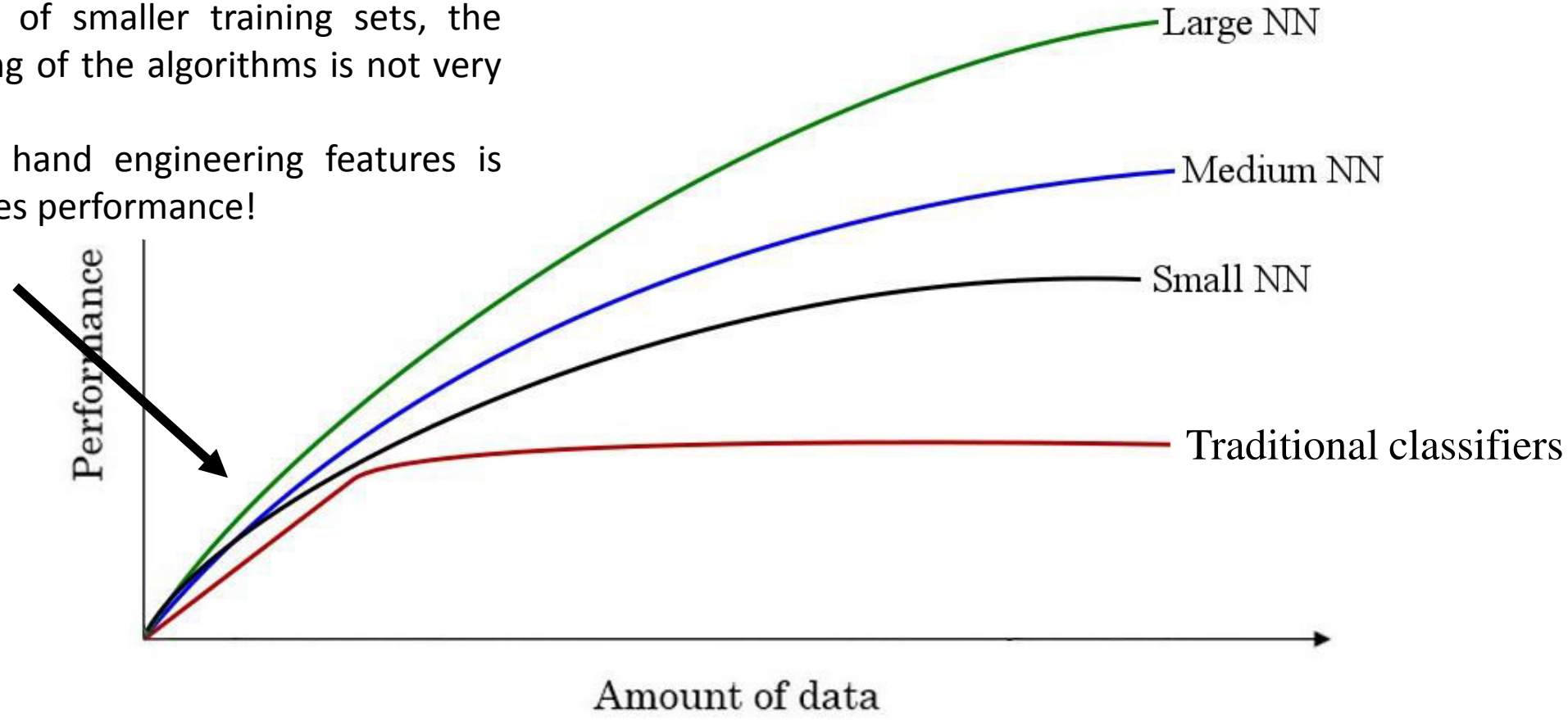


004 – The Rise of Deep Learning

DATAHACKER

- How to overcome performance plateau problem?

In this regime of smaller training sets, the relative ordering of the algorithms is not very well defined.
Often, skill at hand engineering features is what determines performance!



004 – The Rise of Deep Learning

DATAHACKER

- Over the last 20 years we accumulated a lot more data for lot of applications than traditional learning algorithms were able to effectively take advantage of.

- Enormous amount of human activity is now in the digital realm were we spend so much time on a internet.

- And with neural networks it turns out that if you train a very large neural network then its performance often keeps getting better and better.

004 – The Rise of Deep Learning

DATAHACKER

- In the early days in the modern rise of deep learning it was a scale of data and a scale of computation constraints which limited our ability to train very large neural networks on either a CPU or GPU.

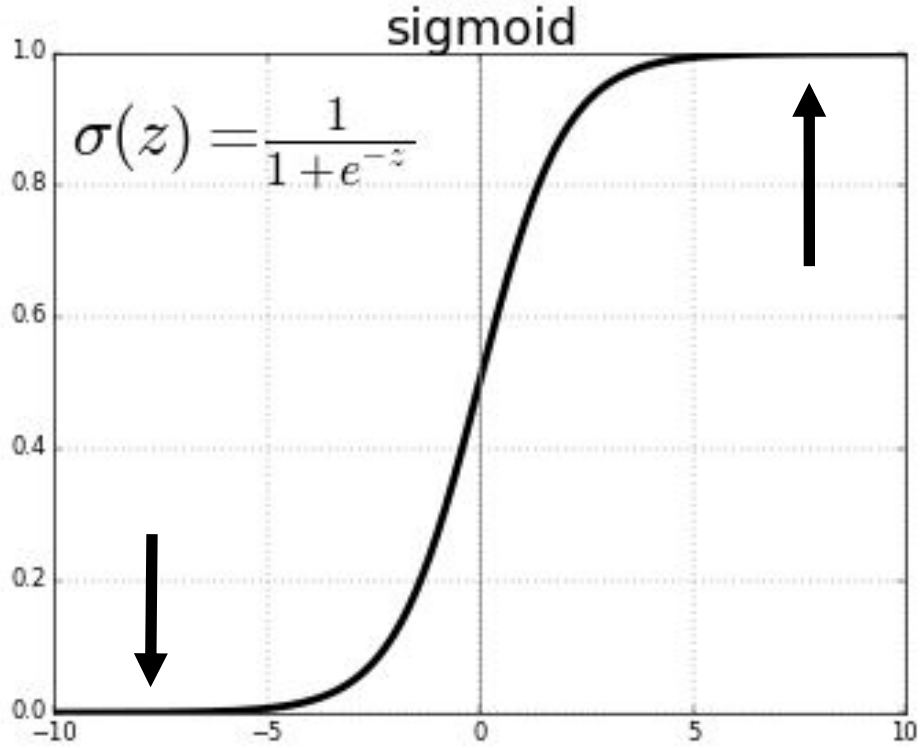
- But relatively recently there have been tremendous algorithmic innovations all of them trying to make neural networks run much faster.

- For example, switching from a sigmoid function to a ReLU function has been one of the huge breakthroughs in neural networks with large impact on their's performance.

004 – The Rise of Deep Learning

DATAHACKER

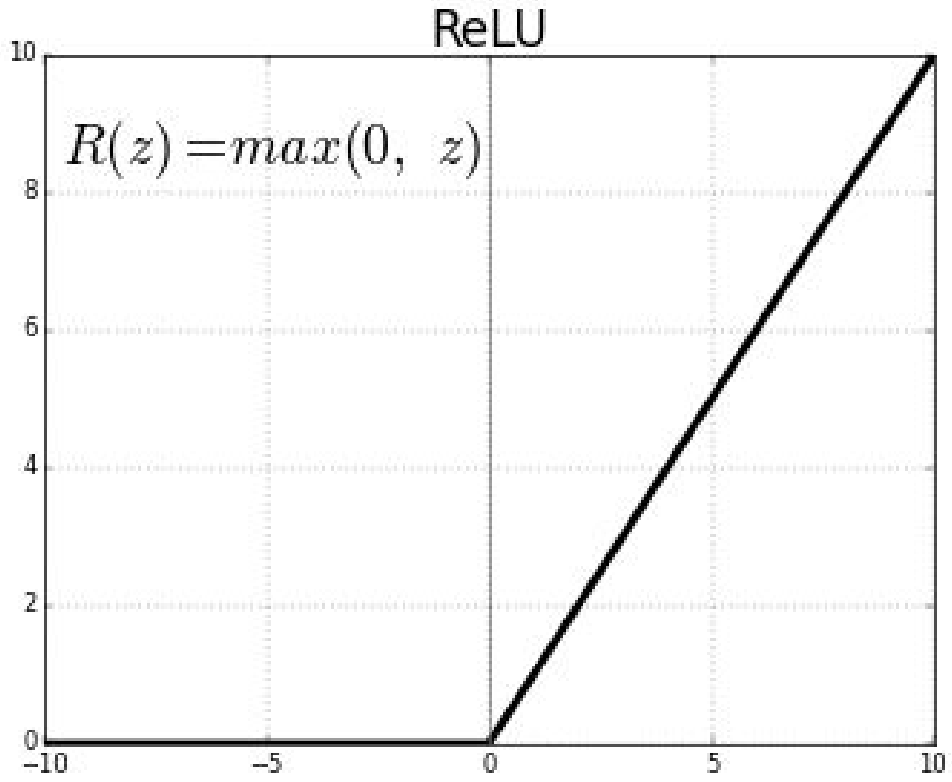
- Let's look in more detail why switching from sigmoid to ReLU function did so well on NN performace.



- One of the problems of using sigmoid functions in machine learning arrises in these regions.
- Here the slope of the function with a gradient is nearly zero that is learning becomes really slow when gradient descent is implemented.
- The gradient is zero and the parameters change very slowly yielding to very slow learning.

004 – The Rise of Deep Learning

DATAHACKER



- For the Rectified Linear Unit function, the gradient is equal to 1 for all positive values of input.
- The gradient is much less likely to gradually shrink to 0, and the slope of the line on the left is 0.
- At the end, just by switching from the sigmoid function to the RELU function has made an algorithm called gradient descent to work much faster.

004 – The Rise of Deep Learning

DATAHACKER

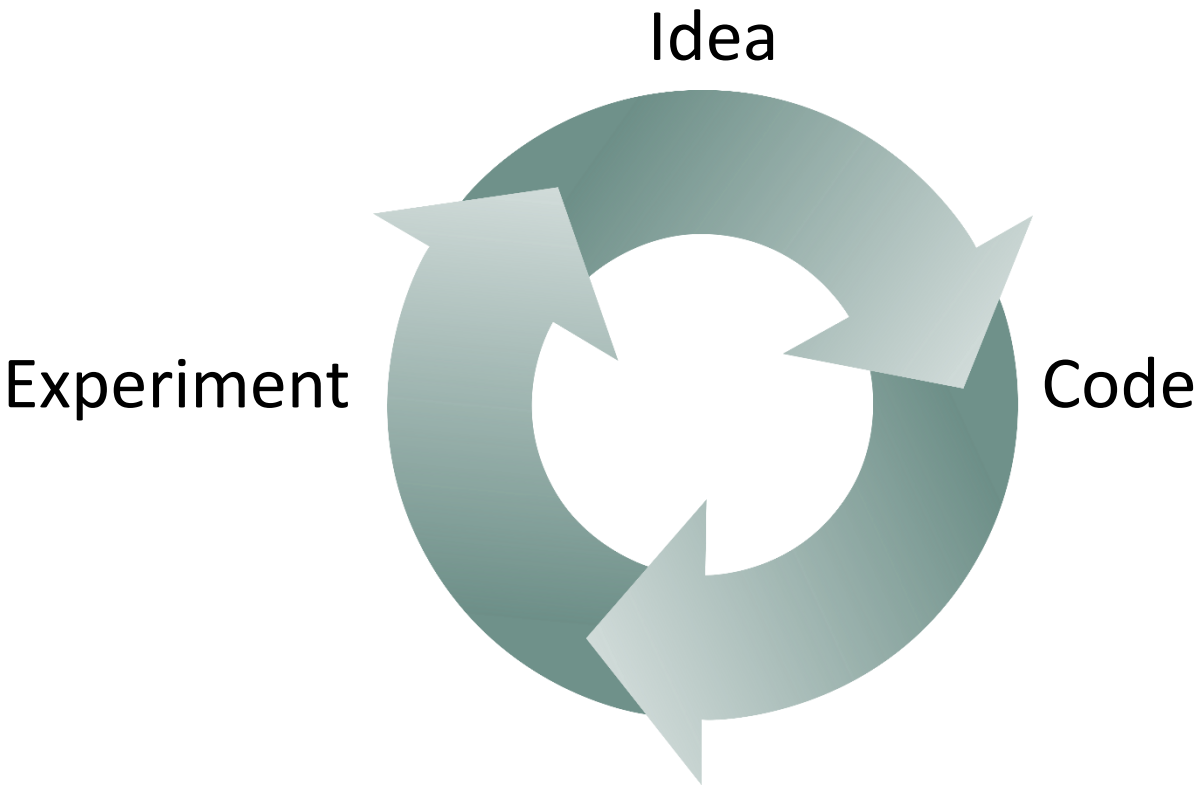
- Often you have an idea for a neural network architecture and you want to implement it in code.

- Fast computation is important because the process of training neural network is very iterative and can be time consuming.

- Implementing your idea then lets you run an experiment which tells you how well your neural network does. Then, by looking at it, you go back to change the details of your neural network and then you go around this circle over and over.

004 – The Rise of Deep Learning

DATAHACKER



004 – The Rise of Deep Learning

DATAHACKER

- Entire deep learning research community has been incredible at just inventing new algorithms and making nonstop progress on that front.

- These are some of the forces powering the rise of deep learning and the good news are that these forces are still working powerfully to make deep learning even better.

- Ability of the very large neural networks from computation point of view will keep on getting better, as society will continue throwing off more and more digital data.

DATAHACKER

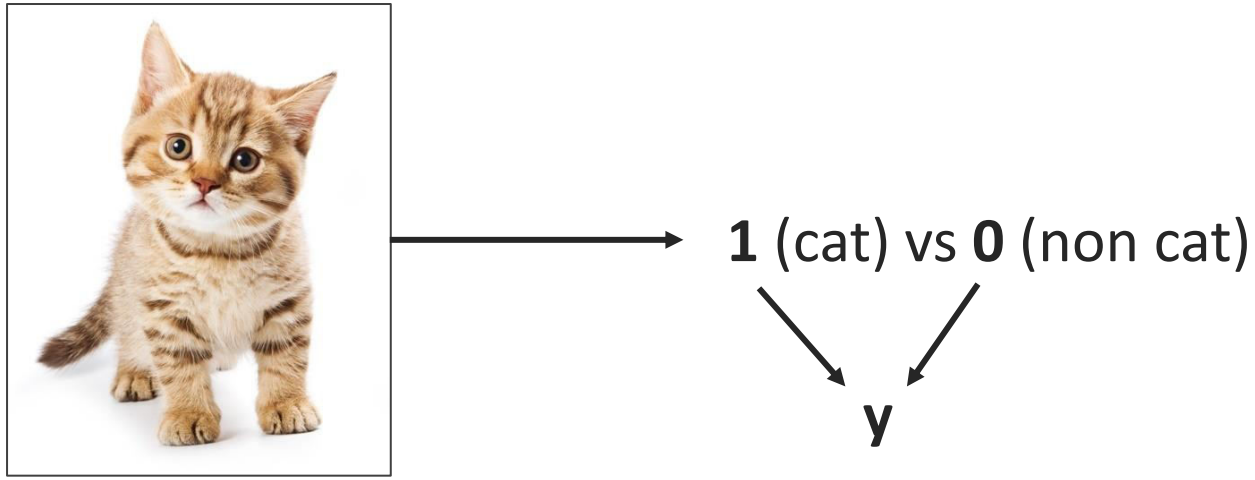
Binary Classification

datahacker.rs

005 - Binary Classification

DATAHACKER

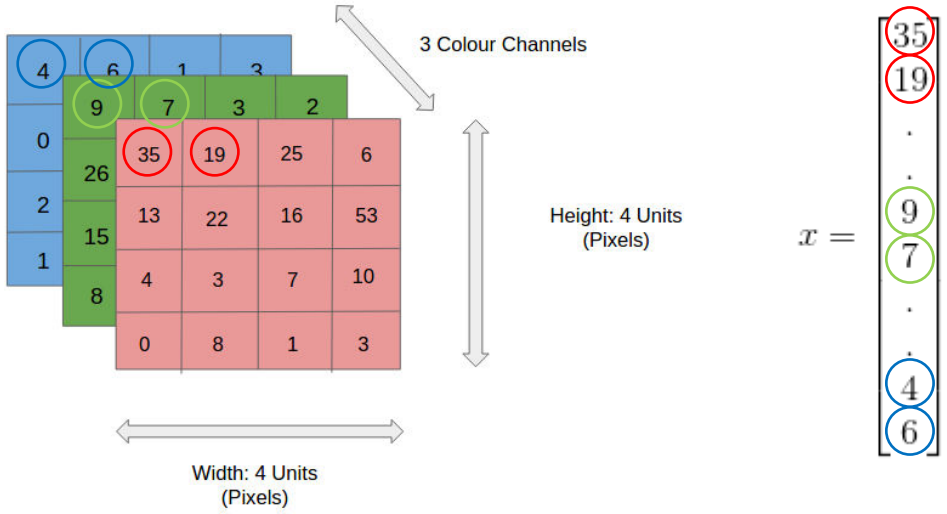
- Logistic regression is an algorithm for **binary classification**
- Example of a **binary classification** problem:
  - you have an **input (x)** of an image and
  - the **output (y)** is a label to recognize the image, 1 (cat) vs 0 (non cat)
- In binary classification our goal is to learn a classifier that can input an image represented by its feature vector X and predict whether the corresponding label y is 1 or 0, that is whether this is a cat image or a non cat image



005 - Binary Classification

DATAHACKER

- **Image representation in computer**: The computer stores 3 separate matrices corresponding to the red, green and blue color channels image
- If the input image is **64 by 64 pixel**, then you would have three 64 by 64 matrices corresponding to the red, green and blue pixel intensity values for your image.
- **64 by 64 image -** the **total dimension** of this vector **X** will be: n_x**=64*64*3=12288**



005 - Binary Classification

DATAHACKER

Notation

- Single training example is represented by (x, y) , where $x \in \mathbb{R}^{n_x}$, label $y \in \{0, 1\}$
- Training set **m,** training examples: $\{ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)}) \}$
- All **training examples**:

$$X = \begin{bmatrix} \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ x^{(1)} & x^{(2)} & \ldots & x^{(m)} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \end{bmatrix} \begin{matrix} \\ \\ \uparrow \\ n_x \\ \downarrow \\ \\ \end{matrix}$$

$$\longleftarrow \quad m \quad \longrightarrow$$

- **Output**:

$$y = \begin{bmatrix} y^{(1)}, & y^{(2)}, & \ldots & , & y^{(m)} \end{bmatrix}$$

- $\boldsymbol{X}$ is $\boldsymbol{n_x}$ x $\boldsymbol{m}$ dimensional matrix: $X \in \mathbb{R}^{n_x x m}$
- Python command for finding the shape of X matrix is *X.shape = (n,m)*
- $\boldsymbol{y}$ is $\boldsymbol{1}$x $\boldsymbol{m}$ dimensional matrix: $y \in \mathbb{R}^{1xm}$
- Python command for finding the shape of y is *y.shape = (1,m)*

DATAHACKER

Logistic Regression

datahacker.rs

006 – Logistic Regression

DATAHACKER

- Logistic regression is a **supervised learning** algorithm that we can use when the output labels are all either: **0** or **1**.
- This is, so called, **binary classification** problem.
- An input feature vector ***x*** may be corresponding to an image that we want to recognize as either a **cat picture (1)** or **not a cat picture (0)**.
- That is, we want an algorithm that can output the prediction ($\hat{y}$) which is an estimate of ***y***:

$$\hat{y} = P(y = 1|x)$$

$$x \in \mathbb{R}^{n_x}$$

$$\text{Parameters} : \omega \in R^{n_x}, b \in \mathbb{R}$$

006 – Logistic Regression

DATAHACKER

- More formally, we want *y* to be the probability of the chance that *y* is equal to 1, given the input features *x*.
- In other words, if *x* is a picture, we want $\hat{y}$ to tell us what is the chance that this is a cat picture.
- An *x* is an *nx* dimensional vector.
- The parameters of logistic regression will be *w* which is also an *nx* dimensional vector together with *b* which is just a real number.
- Given an input x and the parameters *w* and *b,* how do we generate the output $\hat{y}$?
- One thing we could try, that **doesn't work**, would be to have: $\hat{y} = w^T x + b$

006 – Logistic Regression

DATAHACKER

- $\hat{y} = w^T x + b$ is a **linear function** of the input *x* and in fact this is what we **use** if we were doing **linear regression**.
- However, **this isn't a very good algorithm for binary classification**, because we want $\hat{y}$ to be the chance that *y* is equal to 1, so $\hat{y}$ should be **between 0** and **1**.
- It is difficult to enforce, because $w^T x + b$ can be much bigger than 1 or can even be negative which doesn't make sense for a probability that we want to be between 0 and 1.
- So, in logistic regression our output is going to be the **Sigmoid function**.

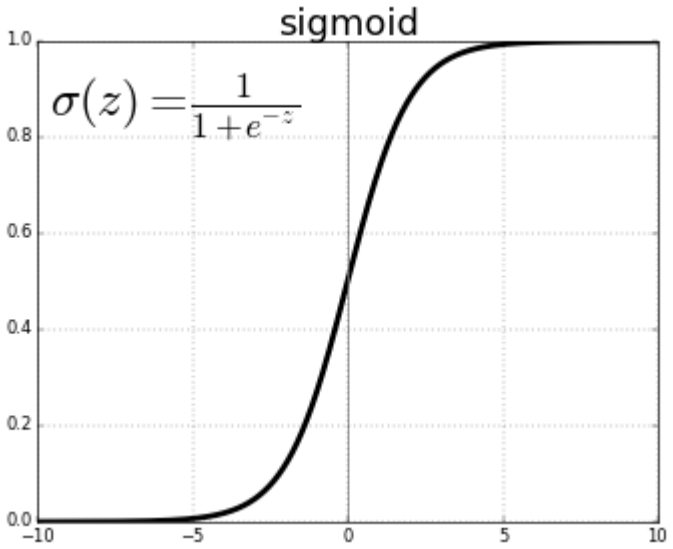
006 – Logistic Regression

DATAHACKER

Sigmoid Function of z

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- It goes smoothly from 0 up to 1



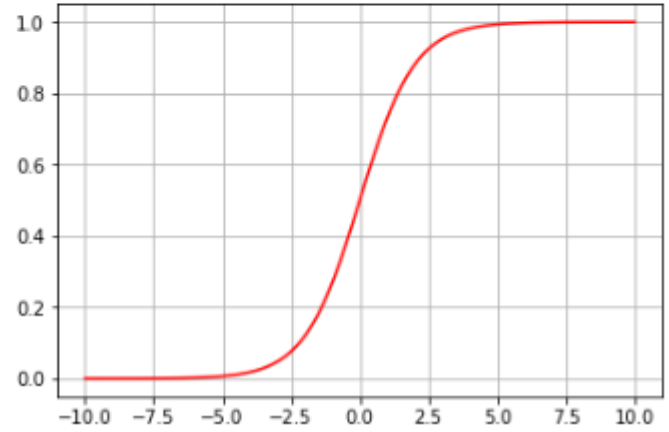
Sigmoid Function in Python

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

In [2]: z = np.linspace(-10,10,100)

In [3]: sigmoid =1/(1+np.exp(-z))

In [4]: plt.plot(z, sigmoid,'r')
        plt.grid();
```



006 – Logistic Regression

DATAHACKER

- We are going to use ***z*** to denote this quantity $w^T x + b$
- Then, we have: $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$
- If ***z*** is **large**: $\sigma(z) = \frac{1}{1+0} \approx 1$
- If ***z*** is **large negative number**: $\sigma(z) = \frac{1}{1+\infty} \approx 0$
- When we implement logistic regression, our job is to try to **learn parameters w** and **b**, so that $\hat{y}$ becomes a good estimate of the **chance of *y* being equal to 1**.

DATAHACKER

Logistic Regression Cost Function

datahacker.rs

007 – Logistic Regression Cost Function

DATAHACKER

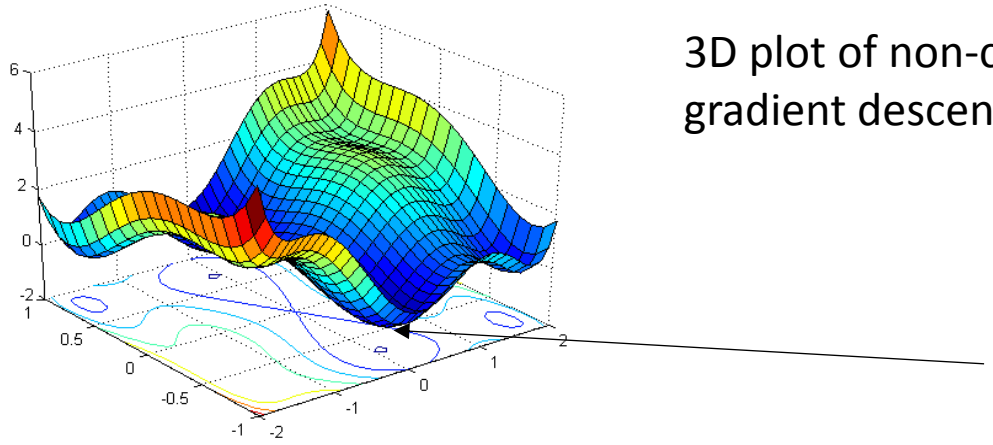
- To train parameters W and b of your logistic regression model you need to define a cost function.
- Given a training set of m training examples, we want to find parameters W and b so that $\hat{y}$ is as close to y (ground truth).
- We will use (I) superscript to index different training examples.
- We use loss (error) function (L) to measure how well out algorithm is doing. Loss function is applied only to single training sample.

007 – Logistic Regression Cost Function

DATAHACKER

- Squared Error $L(\hat{y}, y) = \frac{1}{2}(\hat{y}, y)^2$

  - We don't use this one because it makes gradient descent not work well so in logistic regression optimization problem becomes non-convex.



3D plot of non-convex gradient descent

Global minimum

007 – Logistic Regression Cost Function

DATAHACKER

- We will use following loss function: $L(\hat{y}, y) = -(y log \hat{y} + (1 - y) log(1 - \hat{y}))$
  - It will give us convex optimization problem and it is much easier to otimize.
  - Y and ŷ are sigmoid function so they can never bi bigger than 1 or less then 0
  - If y = 1:
    - L(ŷ, y) = - log ŷ
    - Want log ŷ large, want ŷ large (as close to 1 as possible)
  - If y = 0:
    - L(ŷ, y) = - log (1 - ŷ)
    - Want log (1 - ŷ) large, want ŷ small (as close to 0 as possible)

007 – Logistic Regression Cost Function

DATAHACKER

- Cost function (how well are you doing on the entire training set):
  - J(W,b) = $\frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} \hat{y}^{(i)} log\hat{y^{(i)}} + (1 - y^{(i)}) log(1 - \hat{y}^{(i)})$
  - J is average of sum of loss functions of all parameters.
  - Cost function is cost of all parameters.
  - We will try to find W and b that minimize the overall cost function J.

DATAHACKER

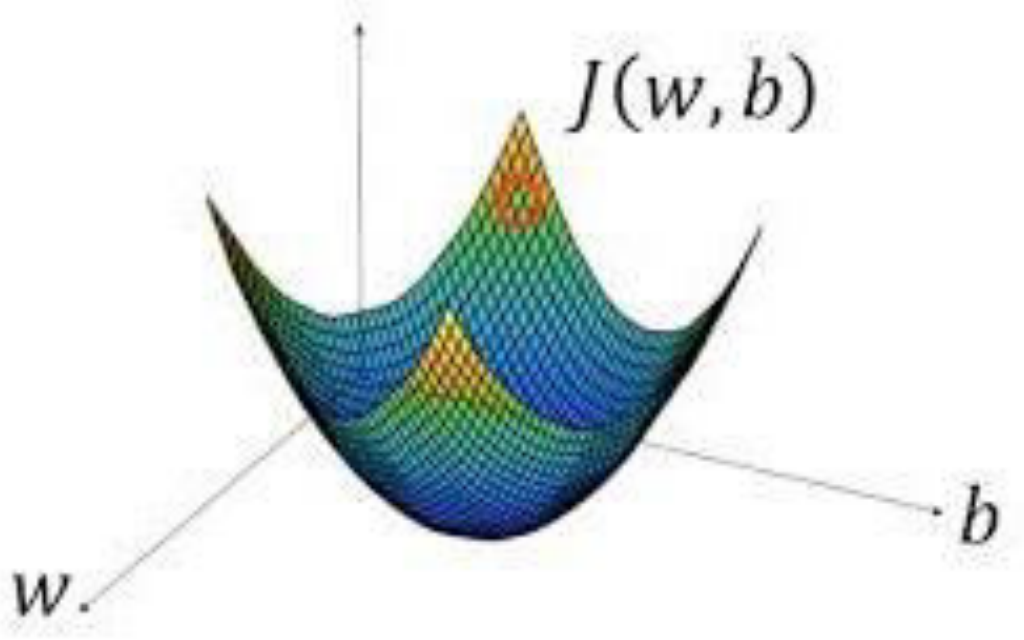
Gradient Descent

datahacker.rs

008 – Gradient Descent

DATAHACKER

- Gradient Descent is a process of trying to minimize the error function.
- Gradient descent algorithm is been used to train and also learn the parameters W on your training set.
- In order to learn the set of parameters W and B it seems natural that we want to find W and B that make the cost function J (W, b) as small.



008 – Gradient Descent

DATAHACKER

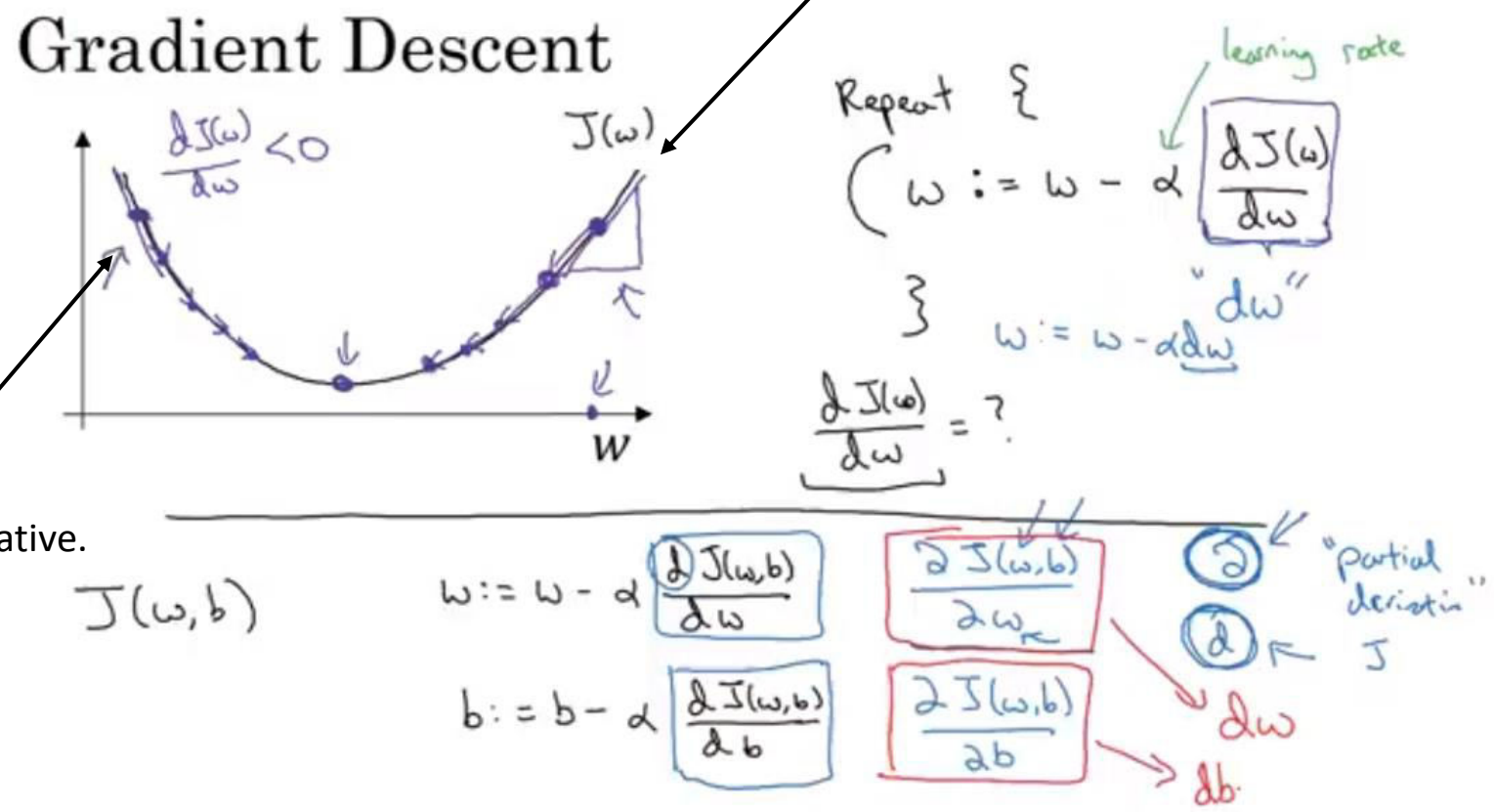
How does Gradient Descent Work

- Gradient Descent starts at an initial point and began taking steps in the steepest downhill direction.
- After a single step, you might end up there because it's trying to take a step downhill in the direction of steepest descent or quickly down low as possible.
- So after one and two iterations of gradient descent, you might end up with the third iteration and so on, until eventually, hopefully you converge to this global optimum or get to something close to the global optimum.

008 – Gradient Descent

DATAHACKER

- When derivative is positive.



- When derivative is negative.

- Derivative is the slope of a function at the point.

- The learning rate controls how big a step we take on each iteration of gradient descent
- Derivative is basically the update of the change you want to make to the parameters W
- Note : in our code ‘ dw ’ will be used to represent this derivative term.

008 – Gradient Descent

DATAHACKER

Explanation for the graph.

- Here the derivative is positive, " W " gets updated as " W " minus a learning rate times the derivative. We know the derivative is positive, so we end up subtracting from W and taking a step to the left. Here, gradient descent would make your algorithm slowly decrease the parameter, if you have started off with this large value of W.
- Here the derivative is negative, the gradient descent update would subtract alpha times a negative number and so we end up slowly increasing W and end up you're making W bigger and bigger with successive iterations of gradient descent
- So that hopefully whether you initialize on the left or on the right, gradient descent would move you towards this global minimum.

008 – Gradient Descent

DATAHACKER

These equations are equivalent.

$$w = w - \alpha * \frac{\mathrm{d}J(w,b)}{\mathrm{d}w} \qquad w = w - \alpha * \frac{\partial J(w,b)}{\partial w}$$

$$b = b - \alpha * \frac{\mathrm{d}J(w,b)}{\mathrm{d}b} \qquad b = b - \alpha * \frac{\partial J(w,b)}{\partial b}$$

Partial Derivatives - ∂

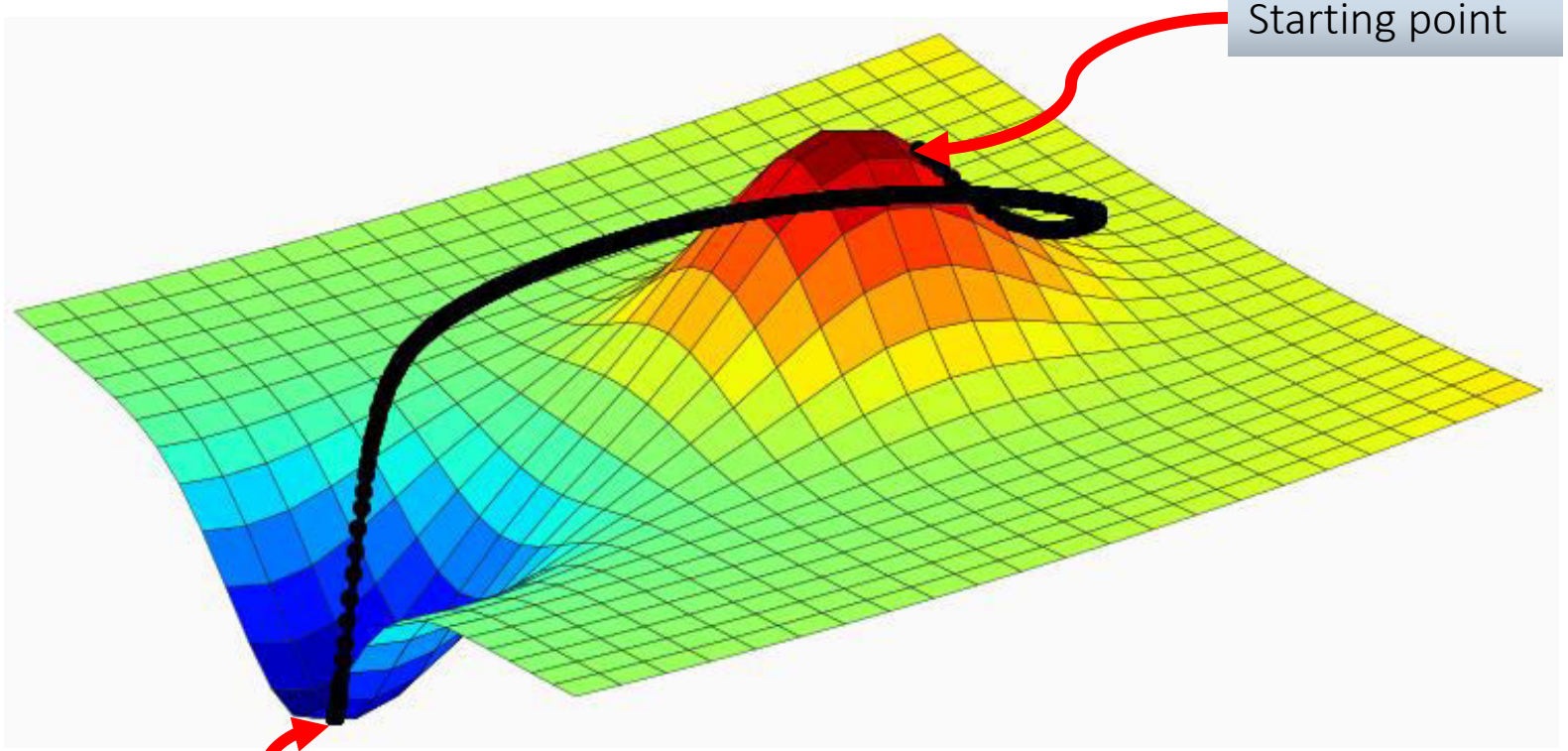
RULE : If ‘ J ’ is a function of two variables, instead of using the lower case ‘d’, we use the partial derivative symbol.

008 – Gradient Descent

DATAHACKER

Visualization in 3D

We can see a picture of the gradient descent algorithm in action, in a 3D visualization, were it is taking baby steps in order and hopefully to reach it's global optimum.



008 – Gradient Descent

DATAHACKER

- By using the function gradient descent we have a **general** direction in which to go to reduce the error
- **Convex functions** have a **single global optimum**
- **Non-Convex functions** have **different local optima.**
- **Optimization algorithm** tries to alter model parameters **“** $\boldsymbol{W}$ " and get a smaller error
- Lots of calculation is based on Linear Algebra
- That means we can write lots of things in a matrix notation
- This allows us to **parallelize,** it quite a lot on a GPU

DATAHACKER

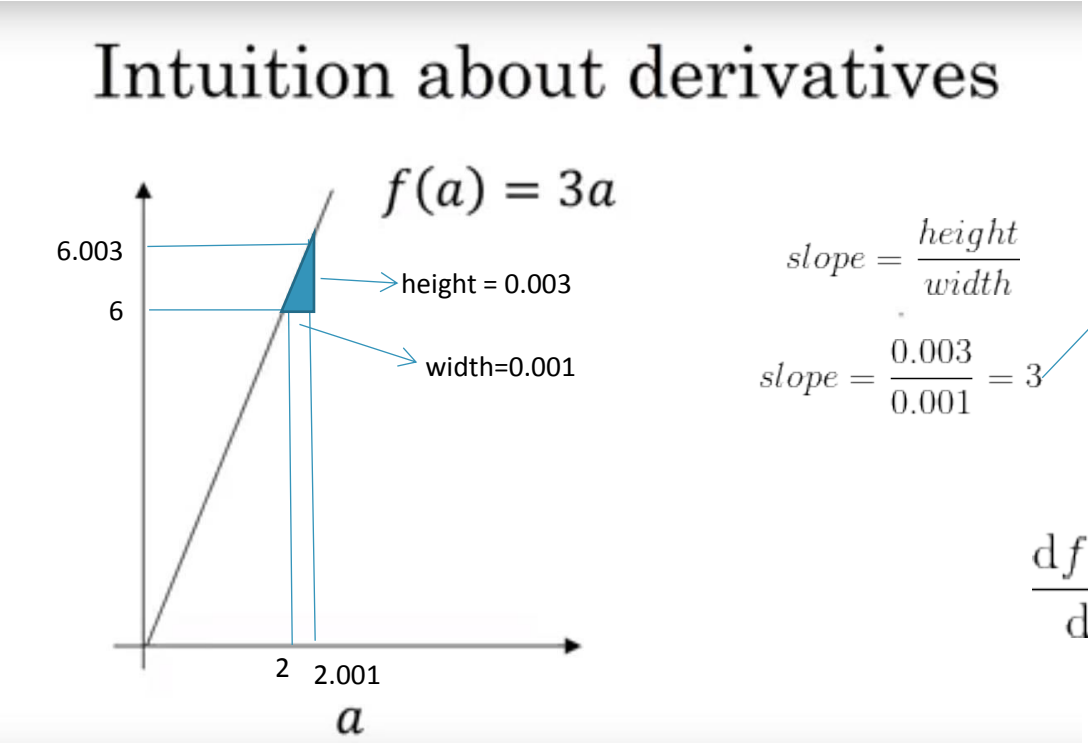
Derivatives

datahacker.rs

009 - Derivatives

DATAHACKER

- We should open up the box and peer a little bit further into the details of calculus but really all you need is an intuitive understanding of this in order to build and successfully apply Deep Learning algorithms.
- The term derivative basically means slope of a function.
- Here the function **f(a) = 3a** is plotted so it's just a straight line to gain intuition about derivatives.
- Slope is a friendlier way to describe the concept of derivative, so **whenever you hear derivative just think slope of the function** and more formally the **slope is defined as the height divided by the width** of this little triangle that we have in blue.



$$slope = \frac{height}{width}$$

$$slope = \frac{0.003}{0.001} = 3$$

$$a = 2, f(a) = 6$$
$$a = 2.001, f(a) = 6.003$$

slope (derivative) of f(a) for a = 2 and a = 2.001 **is 3**

$$a = 5, f(a) = 15$$
$$a = 5.001, f(a) = 15.003$$

slope (derivative) of f(a) for a = 5 and a = 5.001 **is also 3**

$$\frac{\mathrm{d}f(a)}{\mathrm{d}a} = 3 = \frac{\mathrm{d}}{\mathrm{d}a}f(a)$$

Whenever ***a*** is nudged by 0.1 or 0.001 or 0.00…0001, **slope will always be 3 for function *f(a) = 3a***

009 - Derivatives

DATAHACKER

- It is explained for derivatives what happens that we nudge the variable ***a*** by 0.001.
- If you want a formal mathematical definition of the derivatives, derivatives are defined with an even smaller value of how much you nudge a to the right so it is not 0.001, it is not 0.00001 and so on one, it's sort of even smallerthan that.
- The formal definition of derivative says what have you nudge ***a*** to the right by an infinitely tiny tiny amount for f(a), a go up three times as much as whatever was the tiny amount.
- One property of the derivative is that no matter where you take the slope of this function, it is equal to 3 whether **a** = 2 or ***a*** = 5, the slope of this function is **equal to 3**.
- **This function has the same slope everywhere** and one way to see that is that wherever you draw little triangle like that blue in the previous slide, the height divided by the width always has a ratio of three to one.

Definition: Derivative of $f(x)$ at $x = a$

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}, \text{ or}$$

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$$

DATAHACKER

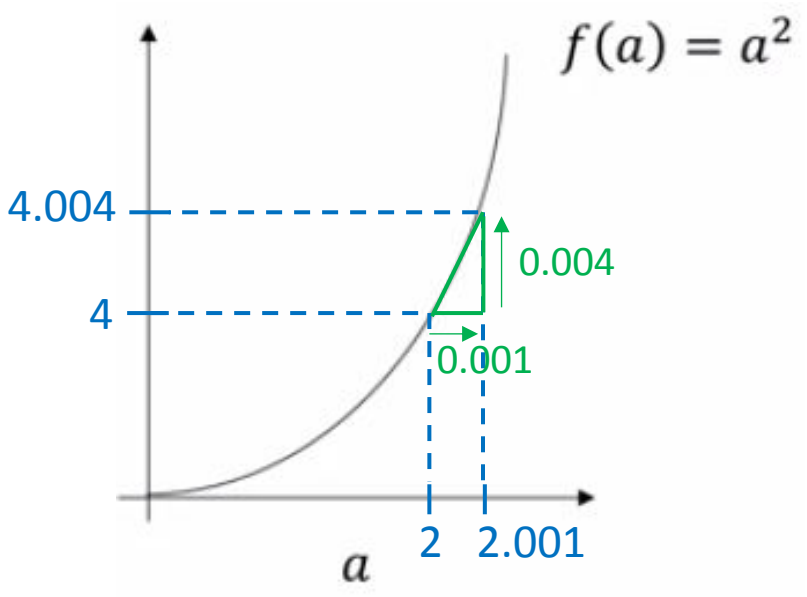
More Derivatives Example

datahacker.rs

010 – More Derivatives Example

DATAHACKER

Intuition about Derivatives



$$a = 2 \qquad f(a) = 4$$

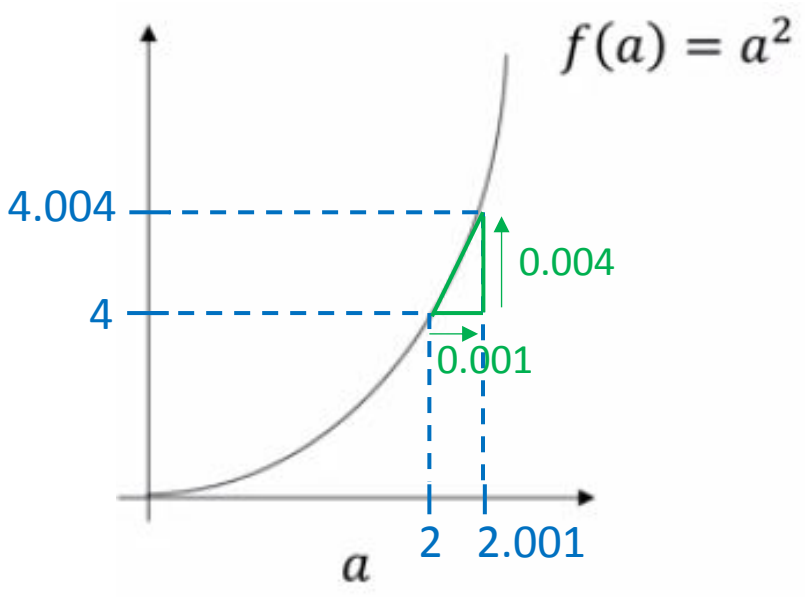
$$a = 2.001 \qquad f(a) \approx 4.004$$

- We have plotted the function **f(a) = a²**, taking a look at the point **a=2**, **a²** or **f(a) = 4**.
- If we move a slightly to the right, **a=2.001**, **f(a)** which is **a²** is going to be approximately **4.004**.
- What this means is when **a=2**?
- What we're saying is that if **a=2**, then **f(a) = 4** (here the x and y axis are not drawn to scale).
- If we move **a** to **2.001**, then **f(a)** becomes roughly **4.004**.
- So if we draw a little triangle, this means is that if we move it to the right by **0.001**, **f(a)** goes up four times as much by **0.004**.

010 – More Derivatives Example

DATAHACKER

Intuition about Derivatives



$$a = 2 \qquad f(a) = 4$$

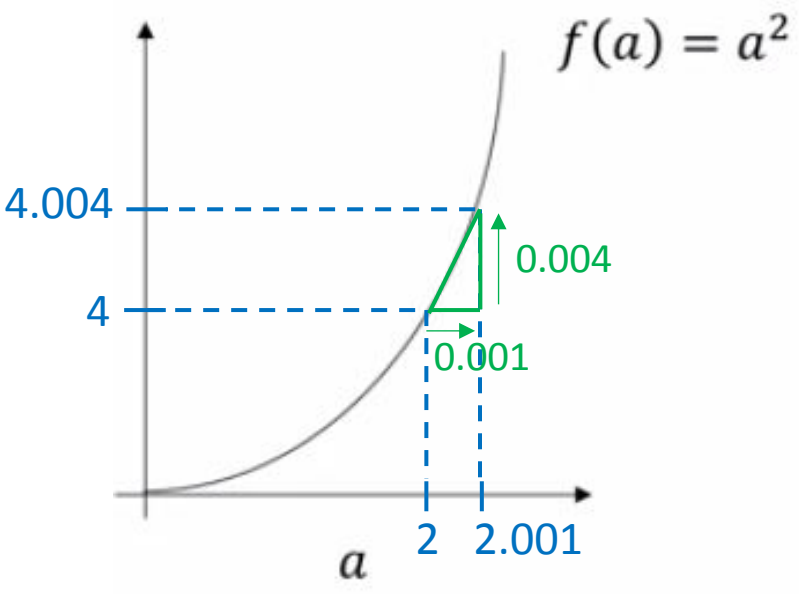
$$a = 2.001 \qquad f(a) \approx 4.004$$

- In the language of calculus, we say that a **slope** that is the **derivative** of **f(a)** at **a=2** is **4**
- Calculus notation: $\frac{\mathrm{d}}{\mathrm{d}a} f(a) = 4, \text{ when } a = 2$
- We say that **d/da** of **f(a) = 4** when **a=2.**
- Now one thing about this function **f(a) = a²** is that **the slope is different for different values of a**.

010 – More Derivatives Example

DATAHACKER

Intuition about Derivatives



- If **a=5**, then **a^2=25**, so that's **f(a).**
- If we move **a** to the right again, it's tiny little change to **a**, now **a=5.001**, then **f(a)** will be approximately **25.010**.
- So what we see is that by moving **a** up by 0.001, **f(a)** goes up ten times as much 0.010.
- We have that: $$\frac{\mathrm{d}}{\mathrm{d}a} f(a) = 10, \text{ when } a = 5$$
- When **a=5** because **f(a)** goes up ten times as much as **a** does when we make a tiny little nudge to **a**.

DATAHACKER

Computation Graph

datahacker.rs

011 – Computation Graph

DATAHACKER

- The computations of a neural network are organized in terms of
  - a **forward pass** or a **forward propagation step**, in which we compute the output of the neural network,
  - followed by a **backward pass** or **back propagation step**, which we use to compute gradients or compute derivatives.
- The **computation graph** explains why it is organized this way.
- In order to illustrate the computation graph, let's use a simpler example than logistic regression or a full blown neural network.

011 – Computation Graph

DATAHACKER

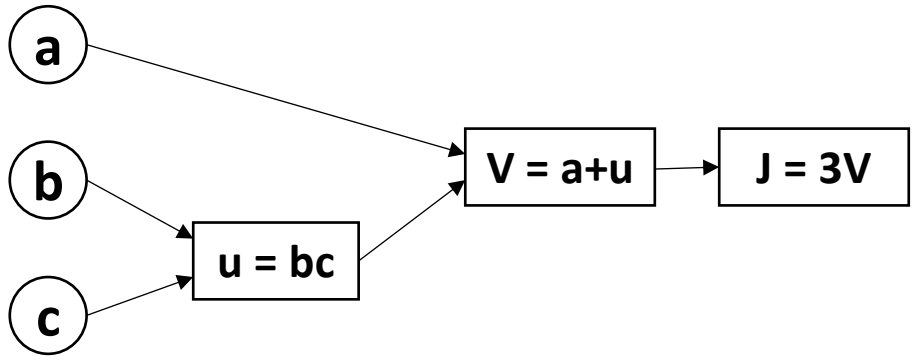
- Let's say that we're trying to compute a **function J**, which is a function of **three variables a, b,** and **c** and let's say that **function J** is **3(a+bc)**.
- Computing this function actually has three distinct steps:
  1. Compute **bc** and store it in the variable **u,** so **u=bc**
  2. Compute **V=a u,**
  3. Output **J** is **3V**.

J(a,b,c)=3(a+bc)

u=bc

V=a+u

J=3V



011 – Computation Graph

DATAHACKER

- So, the computation graph comes in handy when there is some distinguished or some special output variable, such as J in this case, that you want to optimize.
- And in the case of a logistic regression, J is of course the cost function that we're trying to minimize.
- And what we're seeing in this example is that, through a left-to-right pass, you can compute the value of J.

DATAHACKER

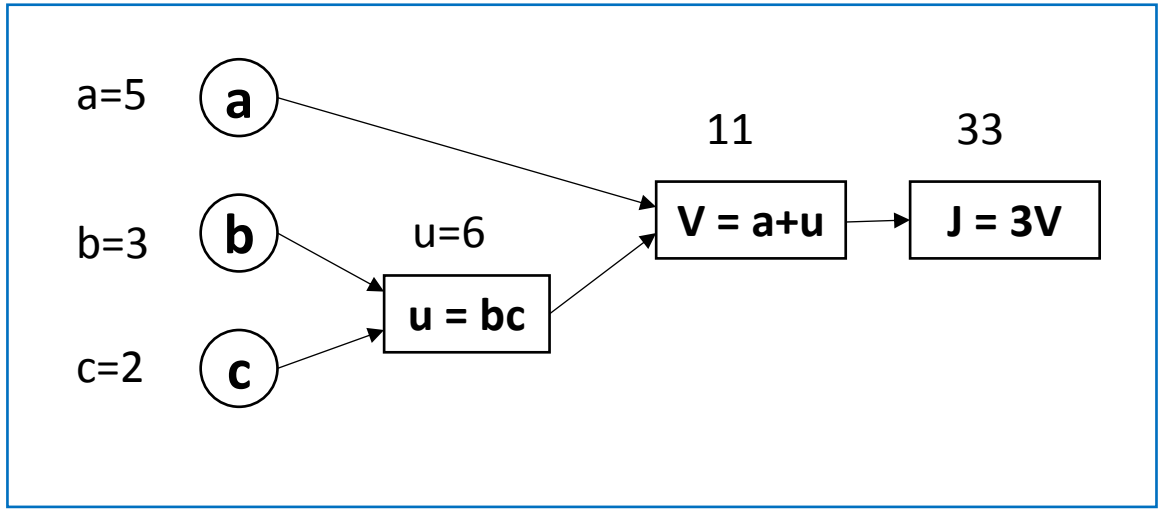
Derivatives with a Computation Graph

datahacker.rs

012 – Derivatives with a Computation Graph

DATAHACKER

- We have learned how we can use a **computation graph to compute the function J**.
- How we can use a computation graph to figure out **derivative calculations of the function J**.
- Using a computation graph we want to compute the **derivative of J** with **respect to v**.



012 – Derivatives with a Computation Graph

DATAHACKER

- So what is that?
- If we were take this value of **v** and change it a little bit, how would the value of **J** change?

$$J = 3v$$

$$v = 11 \mapsto 11.001$$

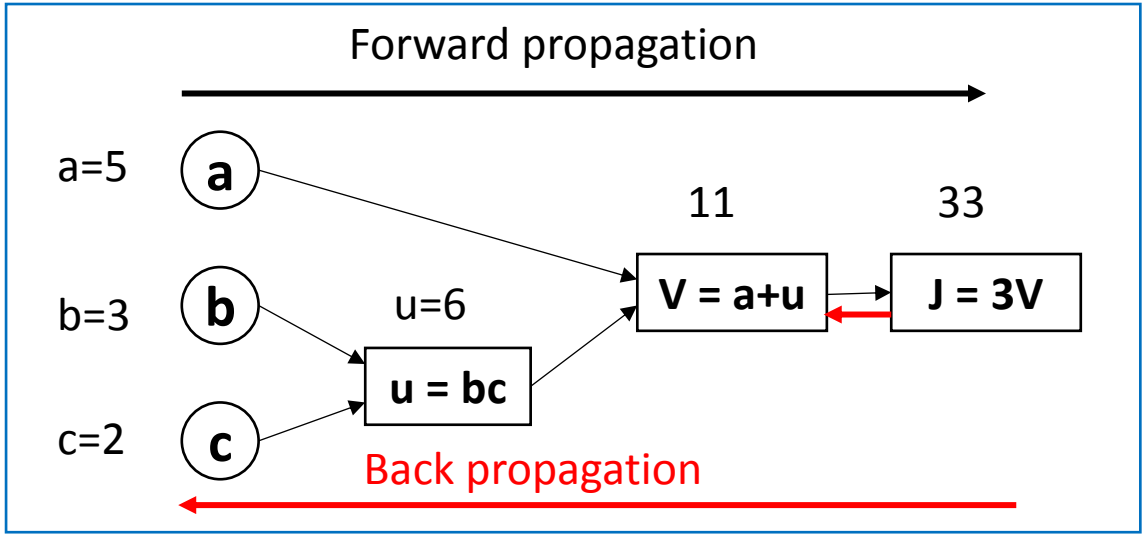
$$J = 33 \mapsto 33.003$$

$$\frac{\mathrm{dJ}}{\mathrm{d}v} = 3$$

- Analogous example

$$f(a) = 3a \quad \frac{\mathrm{df}}{\mathrm{d}a} = 3$$

- Calculating dJ/dv is a one step of **back propagation.**



012 – Derivatives with a Computation Graph

DATAHACKER

- What is dJ/da?
- That is, if we increase **a** from 5 to 5.001, **v** will increase to 11.001 and **J** will increase to 13.003
- So the increase to **J** is three times the increase to a so that means this derivative is equal to 3.

$$a = 5 \mapsto 5.001$$

$$v = 11 \mapsto 11.001$$

$$J = 13 \mapsto 13.003$$

$$\frac{\mathrm{dJ}}{\mathrm{d}a} = \frac{\mathrm{dJ}}{\mathrm{d}v}\frac{\mathrm{dv}}{\mathrm{d}a}$$

$$\frac{\mathrm{dJ}}{\mathrm{d}a} = 3$$

012 – Derivatives with a Computation Graph

DATAHACKER

- One way to break this down is to say that if you change ***a*** then that would change ***v*** and through changing ***v*** that would change *J.*
- By increasing **a**, how much is **v** increased? This is determined by **dv/da**.
- Then, the change in **v** will cause the value of **J** to also increase. This is called a **chain rule** in calculus:

$$\frac{dJ}{da} = \frac{dJ}{dv}\frac{dv}{da}$$

012 – Derivatives with a Computation Graph

DATAHACKER

- In this course we will use the following notation:

$$\frac{\mathrm{d}FinalOutputVar}{\mathrm{d}Var} --> "\ dVar\ "$$

- So, in our code will use:

$$\frac{\mathrm{d}J}{\mathrm{d}v} --> "\ dv\ "$$

$$\frac{\mathrm{d}J}{\mathrm{d}a} --> "\ da\ "$$

012 – Derivatives with a Computation Graph

DATAHACKER

- What is the derivative dJ/du?

$$\frac{dJ}{du} = ?$$

$$u = 6 \mapsto 6.001$$

$$v = 11 \mapsto 11.001$$

$$J = 33 \mapsto 33.003$$

$$\frac{dJ}{du} = \frac{dJ}{dv}\frac{dv}{da} = 3 \cdot 1$$

- A question when working with derivatives is: How much do we need to **change b** in order to **maximize/minimize J**? What is the slope when we change b?
- When we **change b**, we will **change u**, and **u will affect J**.

$$\frac{dJ}{db} = ?$$

$$b = 3 \mapsto 3.001$$

$$u = 6 \mapsto 6.002$$

$$J = 33 \mapsto 33.006$$

$$\frac{du}{db} = 2$$

$$\frac{dJ}{db} = \frac{dJ}{du}\frac{du}{db} = 3 \cdot 2 = 6$$

012 – Derivatives with a Computation Graph

DATAHACKER

- What is the derivative dJ/dc?

$$\frac{dJ}{dc} = ?$$

$$c = 2 \mapsto 2.001$$

$$u = 6 \mapsto 6.003$$

$$J = 33 \mapsto 33.009$$

$$\frac{du}{dc} = 3$$

$$\frac{dJ}{dc} = \frac{dJ}{du}\frac{du}{dc} = 3 \cdot 3 = 9$$

DATAHACKER

Logistic Regression Derivatives

datahacker.rs

013 - Logistic Regression Derivatives

DATAHACKER

Logistic Regression Recap

- **Z** - Logistic Regression Formula

$$z = w^T x + b$$

- **Y** predictions

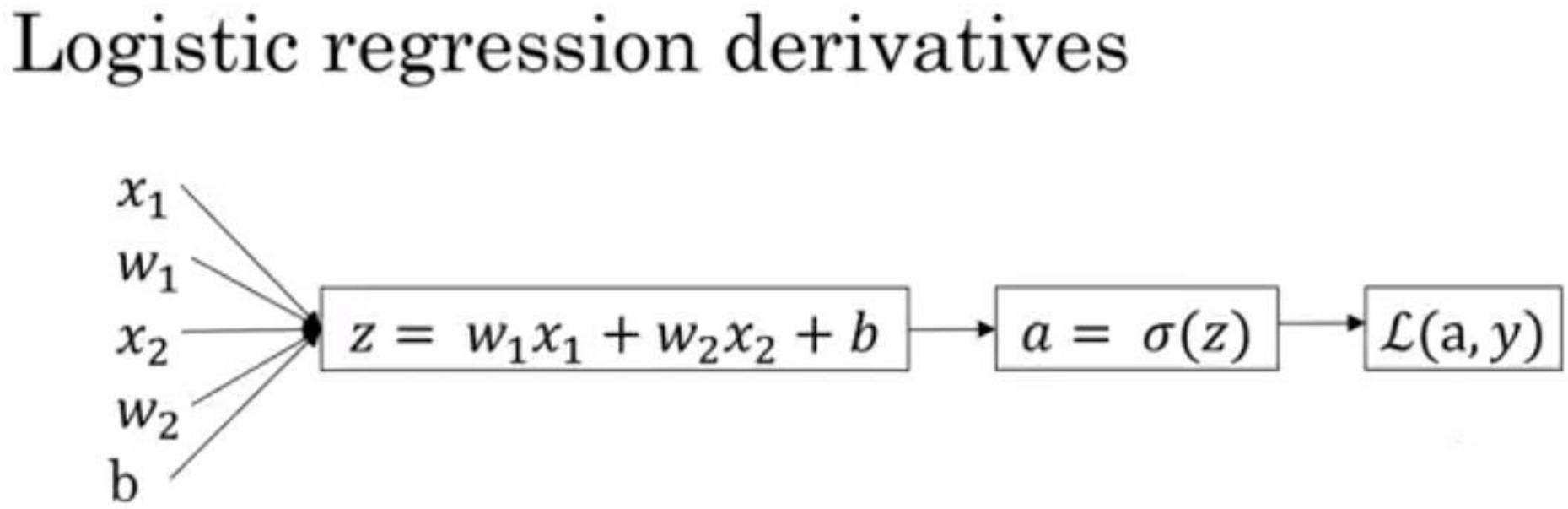
$$\hat{y} = a = \sigma(z)$$

- Loss Function where **a** is the output of logistic regression and **y** is the ground truth label

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

013 - Logistic Regression Derivatives

DATAHACKER



013 - Logistic Regression Derivatives

DATAHACKER

Explanation for the diagram

- In this example we have only two features **x1** and **x2.** So in order to compute **Z,** we'll need to input **w1, w2** and **B** in addition to the feature values **x1** and **x2 :**

$$z = w1x1 + w2x2 + b$$

- Then we compute **y-hat** or **a** equals **Sigma of Z :** $\hat{y} = \sigma(Z)$ or $a = \sigma(Z)$

- Finally we compute our **loss function :** $\mathcal{L}(a, y)$

013 - Logistic Regression Derivatives

DATAHACKER

- In logistic regression all what we want to do is to modify the parameters **W** and **B** in order to reduce this loss.
- First we have to compute the loss on a single training using the forward propagation steps.
- Then we go backwards to compute the derivatives.

013 - Logistic Regression Derivatives

DATAHACKER

How do we compute the derivatives.

- Compute the derivative of $\frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}a}$ and in code we would denote this as $da = \frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}a}$

  Note when writing code I assigned $\frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}a}$ to "**da**"

  $$da = -\frac{y}{a} + \frac{1-y}{1-a}$$

- Having computed "**da**", we can go backwards to compute "**dz**"

$$dz = \frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}z}$$

$$dz = \frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}a} * \frac{\mathrm{d}a}{\mathrm{d}z}$$

$$dz = a - y$$

Since we know that :

$$\frac{\mathrm{d}a}{\mathrm{d}z} = a(1-a)$$

$$\frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}a} = -\frac{y}{a} + \frac{1-y}{1-a}$$

013 - Logistic Regression Derivatives

DATAHACKER

- The final step in back propagation is to go back to compute how much we need to change **w** and **b**

$$dw1 = \frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}w1} = x1 * dz \qquad dw2 = \frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}w2} = x2 * dz \qquad db = \frac{\mathrm{d}\mathcal{L}(a,y)}{\mathrm{d}b} = dz$$

- so if you want to do gradient descent with respect to just this one example, we would do the following by performing this updates.

$$w1 = w1 - \alpha * dw1 \qquad w2 = w2 - \alpha * dw2 \qquad b = b - \alpha * db$$

DATAHACKER

Gradient Descent on m Training Examples

014 – Gradient Descent on m Training Examples

DATAHACKER

- The derivative respect to w1 of the overall cost function is also going to be the average of derivatives respect to w1 of the individual loss terms.

$$\frac{\mathrm{d}}{\mathrm{d}w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\mathrm{d}}{\mathrm{d}w_1} L(a^{(i)}, y^{(i)})$$

- Previously we have already shown how to compute this term for a single training sample:

$$dw_1^{(i)} = \frac{\mathrm{d}}{\mathrm{d}w_1} L(a^{(i)}, y^{(i)})$$

- We need to compute this derivative on the previous training example and average them and we will get the overall gradient that we can use to implement gradient descent.

014 – Gradient Descent on m Training Examples

DATAHACKER

- Let's wrap this up into a concrete algorithm and what you should implement to get logistic regression with gradient descent working. We do this calculation assuming that we have two features so that n = 2, otherwise we do this for dw1, dw2, dw2...

$$J = 0; dw_1 = 0; dw_2 = 0; db = 0$$

For i = 1 to m

$$Z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(Z^{(i)})$$
$$J+ = -[y^{(i)} log a^{(i)} + (1 - y^{(i)}) log(1 - a^{(i)}))]$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw_1+ = x_1^{(i)} dz^{(i)}$$
$$dw_2+ = x_2^{(i)} dz^{(i)}$$
$$db+ = dz^{(i)}$$

n

014 – Gradient Descent on m Training Examples

DATAHACKER

- After we leave for loop we need to divede all by m because we are computing averages:

$$J/ = m;$$
$$dw_1/ = m;$$
$$dw_2/ = m;$$
$$db/ = m;$$

- With all of these calculations, we computed derivative of the cost function J with respect to parameters w1, w2, and b.

014 – Gradient Descent on m Training Examples

DATAHACKER

- We're using dw1 and dw2 and db as accumulators.

$$dw_1 = \frac{\mathrm{dJ}}{\mathrm{d}w_1}; dw_2 = \frac{\mathrm{dJ}}{\mathrm{d}w_2}; db = \frac{\mathrm{dJ}}{\mathrm{d}b}$$

- We use them to sum the entire training set.
- After all this calculations to implement one step of gradient descent we implement

$$w_1 := w_1 - \alpha dw_1$$
$$w_2 := w_2 - \alpha dw_2$$
$$b := b - \alpha db$$

014 – Gradient Descent on m Training Examples

DATAHACKER

- It turns out there are two weaknesses with calculation as we've implemented in here.
- To implement logistic regression this way we need to write two for loops (loop over m training samples and loop over all the features).
- When we are implementing deep learning algorithms we find that explicit for loops make our algorithms run less efficiency so if we want to scale to bigger data sets we must avoid them.
- For that we use vectorization.

DATAHACKER

Vectorization

datahacker.rs

015 - Vectorization

DATAHACKER

- **Vectorization** is a style of computer programming where operations are **applied to whole arrays instead of individual elements**
- Basically the art of **getting rid of explicit for loops** in your code.
- Deep learning algorithms tend to shine when training on large datasets, so it's important that your code runs quickly. Otherwise, your code might take a long time to get your result.
- Vectorization can significantly **speed up your code**.
- The rule of thumb to remember is whenever **possible avoid using explicit for loops in your code.**
- In the next slides, I would be doing a comparison in time using a vectorized and non-vectorized implementation to compute the dot products of two arrays.

015 - Vectorization

DATAHACKER

Vectorized Implementation.

In [1]:
```
import numpy as np
```

In [2]:
```
a = np.random.rand(1000000)
b = np.random.rand(1000000)
```

In [3]:
```
# Vectorized Implementation
```

In [4]:
```
%%timeit
c = np.dot(a, b)
```

```
597 µs ± 72.6 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Vectorized Implementation : 597 nanoseconds

015 - Vectorization

DATAHACKER

Non-Vectorized Implementation.

In [1]:
```
import numpy as np
```

In [2]:
```
a = np.random.rand(1000000)
b = np.random.rand(1000000)
```

In [3]:
```
# Non-Vectorized Implementation
```

In [4]:
```
%%timeit
c = 0
for i in range(1000000):
    c += a[i] * b[i]
```

```
405 ms ± 67 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Vectorized Implementation : 405 milliseconds

015 - Vectorization

DATAHACKER

- Our results clarifies that having a vectorized implementation in your code, does actually make your code run faster.
- The **vectorized Implementation** took **597 nanoseconds** (Better)
- While, the **non-vectorized implementation** took **407 milliseconds**.
- NumPy is the fundamental package for scientific computing in Python.

DATAHACKER

More Vectorization Examples

datahacker.rs

016 – More Vectorization Examples

DATAHACKER

- When we are programming neural networks or logistic regression we should avoid explicit for loops.
- It isn't always possible but when we can we should use built-in functions or find some other ways to compute.
- Some examples:
  - Non vectorized:

$$u = AV$$
$$u_i = \sum_j A_{ij} V_j$$
$$u = np.zeros((n,1))$$
$$for\ i\ ...$$
$$\quad for\ i\ ...$$
$$\quad\quad u[i] += A[i][j] * V[j]$$

Vectorized:

U=np.dot(A,V)

016 – More Vectorization Examples

DATAHACKER

- Let's apply the exponential operation on every element of matrix/vector.
- Non vectorized:

$$
\begin{aligned}
&v = [v_1, v_2, ..., v_m] \\
&u = [ev_1, ev_2, ..., ev_m] \\
&u = np.zeros((n, 1)) \\
&for\ i\ in\ range(n): \\
&\quad u[i] = math.exp(v[i])
\end{aligned}
$$

- Python library numpy has many functions that help us with vectorization.
- Vectorized:

```
Import numpy as np
U = np.exp(v)
```

016 – More Vectorization Examples

DATAHACKER

- Some other numpy vectorization functions:
  - Np.log
  - Np.abs
  - Np.maximum
  - V**2
  - 1/v

016 – More Vectorization Examples

DATAHACKER

- Let's apply this on our logistic regression gradient descent to remove our for loops.

$$
\begin{aligned}
&J = 0, dw = np.zeros((nx, 1)), db = 0 \\
&for\ i = 1\ to\ m: \\
&\quad z^{(i)} = w^T x^{(i)} + b \\
&\quad a^{(i)} = \alpha(z^{(i)}) \\
&\quad J+ = -[y^{(i)} log\hat{y}^{(i)} + (1 - y^i) log(1 - \hat{y}^{(i)})] \\
&\quad dz^{(i)} = a^{(i)}(1 - a^{(i)}) \\
&\quad dw+ = x^{(i)} dz^{(i)} \\
&\quad db+ = dz^{(i)} \\
&J = J/m, dw/ = m, db = db/m
\end{aligned}
$$

- We still have one for loop but code is running much faster.

DATAHACKER

Vectorizing Logistic Regression

datahacker.rs

017 - Vectorizing Logistic Regression

DATAHACKER

- We are going to learn how to vectorize our implementation of logistic regression
- We want to process an entire training set that has implemented a single iteration of gradient descent with respect to the entire training set without using even a single explicit for loop.
- First we need to compute the forward propagation step of logistic regression, by computing this below :

$$z^{(1)} = w^T x^{(1)} + b \qquad z^{(2)} = w^T x^{(2)} + b \qquad z^{(3)} = w^T x^{(3)} + b$$
$$a^{(1)} = \sigma(z^{(1)}) \qquad a^{(2)} = \sigma(z^{(2)}) \qquad a^{(3)} = \sigma(z^{(3)})$$

- First we need to compute Z, then compute also the activation function.
- For M training examples, you have to carry out the forward propagation step that is to compute these predictions on all M training example.

017 - Vectorizing Logistic Regression

DATAHACKER

- First we stack together in different columns our training inputs and assign them to the variable $X = [x^{(1)}, x^{(2)}, ..., x^{(n)}]$ which is an Nx by M dimensional matrix.
- w^T (w transpose) would be a row vector.
- b is a 1 by M matrix or an M dimensional row vector.
- Next we can compute our Z as a 1 by M matrix

$$Z = [z^1, z^2, ..., z^n] = w^T * X + [b, b, ..., b] = [w^T x^1 + b, w^T x^2 + b, ..., w^T x^n + b]$$

- In python :

$$Z = np.dot(w.T, X) + b$$

017 - Vectorizing Logistic Regression

DATAHACKER

- Next we want to find a way to compute A all at the same time.

$$A = [a^{(1)}, a^{(2)}, ..., a^{(n)}] = \sigma(Z)$$

- Formula :

$$\sigma = \frac{1}{1 + e^{-z}}$$

- In python :

```
In [1]: import math

        def sigmoid(x):
            return 1 / (1 + math.exp(-x))
```

```
In [2]: sigmoid(0.458)
```

```
Out[2]: 0.6125396134409151
```

DATAHACKER

Vectorizing Logistic Regression's Gradient Computation

datahacker.rs

018 – Vectorizing Logistic Regression's Gradient Computation

DATAHACKER

- We are going to learn how to compute all samples at a same time.
- We need to define new variable dZ which will contain all dz values stacked horizontally in 1 by m matrix.

$$dZ = [dz^{(1)}, dz^{(2)}, ..., dz^{(m)}]$$
$$dZ = A - Y = [a^{(1)} - y^{(1)}, a^{(2)} - y^{(2)}, ...]$$

- In the previous example, we have gotten rid of one for loop but the second one (for loop over m training samples) still remained.

018 – Vectorizing Logistic Regression's Gradient Computation

DATAHACKER

- Now we will vectorize all. Starting with vectorization of db and dw.

$$db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)} \qquad dw = \frac{1}{m} X dZ^T = \frac{1}{m}[x^{(1)}dz^{(1)} + ... + x^{(m)}dz^{(m)}]$$

- In python:

$$db = \frac{1}{m} np.sum(dZ) \qquad dw = \frac{1}{m} X dZ^T$$

018 – Vectorizing Logistic Regression's Gradient Computation

DATAHACKER

- Let's pull it all together to actually implement logistic regression.

$$Z = np.dot(w.T, X) + b$$
$$A = \alpha(Z)$$
$$dZ = A - Y$$
$$dw = \frac{1}{m} X\, dZ^T$$
$$db = \frac{1}{m} np.sum(dZ)$$
$$w := w - \alpha dw$$
$$b := b - \alpha db$$

018 – Vectorizing Logistic Regression's Gradient Computation

DATAHACKER

- With this, we implemented single iteration of logistic regression.
- If we want to implement logistic regression over a number of iteration we need for loop.
- We can't get rid of that for loop.