**Machine Learning**

There are 3 types of Machine Learning

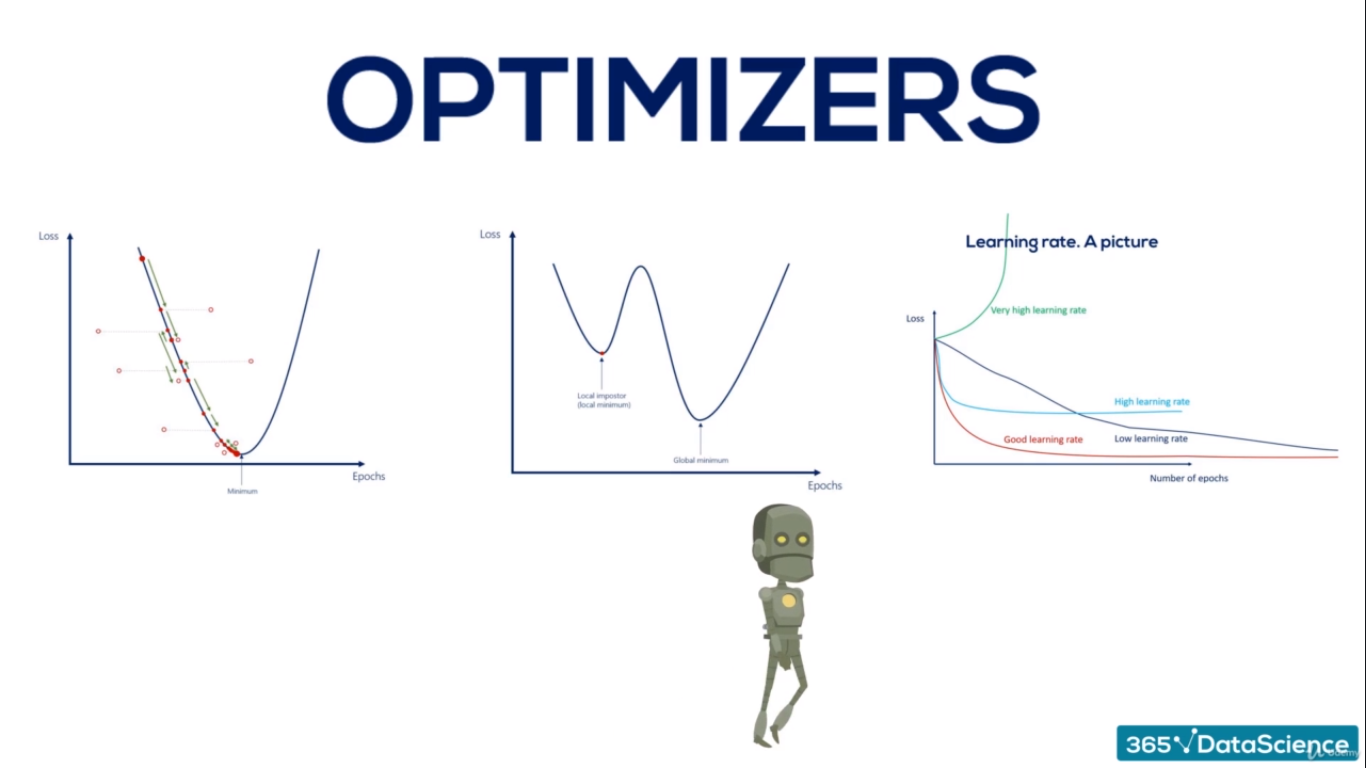
1. Supervised
2. Unsupervised
3. Reinforcement

Data 🡪 Model 🡪 Objective Function 🡪 Optimization Algorithm

Input 🡪 Linear 🡪 Activation Function 🡪 Output or Non-linear

A close up of a sign

Description automatically generated



Artificial intelligence is intelligence shown by machines (opposed to natural (human) intelligence). Machine learning is a way to achieve artificial intelligence. AI is generally wider, and not so well-defined. There’s even a joke that AI is: “whatever hasn’t been done yet”.

Set Goals and Introduce the Framework

Input Output

**Training an Algorithm:**

**Data:-** We must prepare certain amount of data with training. Usually this is a historical data and it’s readily available.

**Model:-** We need to find a model for that and the simplest model is linear regression. Linear Model is just a tip of ice berg. Deep Machine Learning can create a complicated Non-Linear model. They usually fit the data much better.

**Objective Function:-** If our function measureing the prediction error of our model. We would want to minimize the error or minimize the objective Function.

**Optimization Algorithm:-** w1 and w2 are the parameters of the model. For each parameters we find objective function and we will choose the model with the highest predictive power.

Data 🡪 Model 🡪 Objective Function 🡪 Optimization Algorithm

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A close up of a logo

Description automatically generated

**There are 3 types of Machine Learning**

**Supervised learning:-** Classification (Cats or Dogs) or Regression (EUR/USD up or down)

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs.[21] The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and a desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs.[22] An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.[6]

Supervised learning algorithms include classification and regression.[23] Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. Similarity learning is an area of supervised machine learning closely related to regression and classification, but the goal is to learn from examples using a similarity function that measures how similar or related two objects are. It has applications in ranking, recommendation systems, visual identity tracking, face verification, and speaker verification.

In the case of semi-supervised learning algorithms, some of the training examples are missing training labels, but they can nevertheless be used to improve the quality of a model. In weakly supervised learning, the training labels are noisy, limited, or imprecise; however, these labels are often cheaper to obtain, resulting in larger effective training sets.[24]

Unsupervised learning

Main article: Unsupervised learning

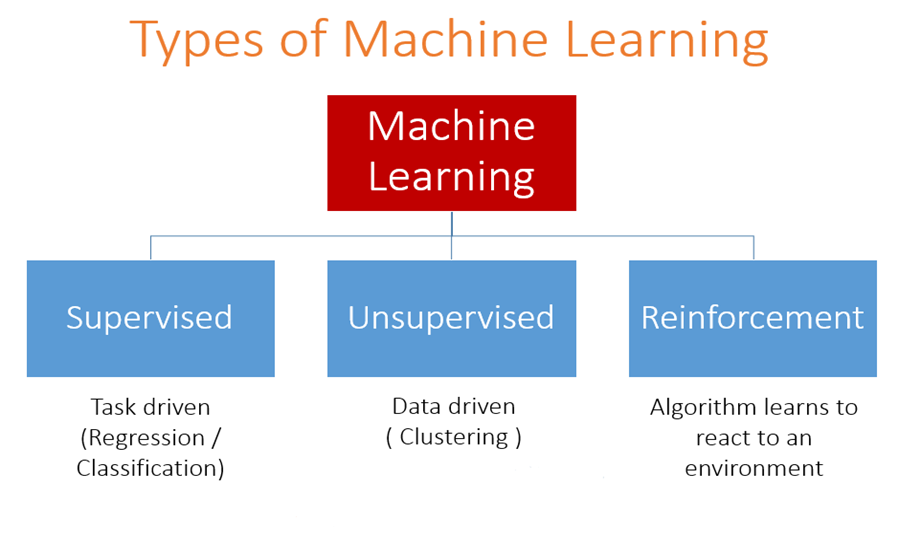
See also: Cluster analysis

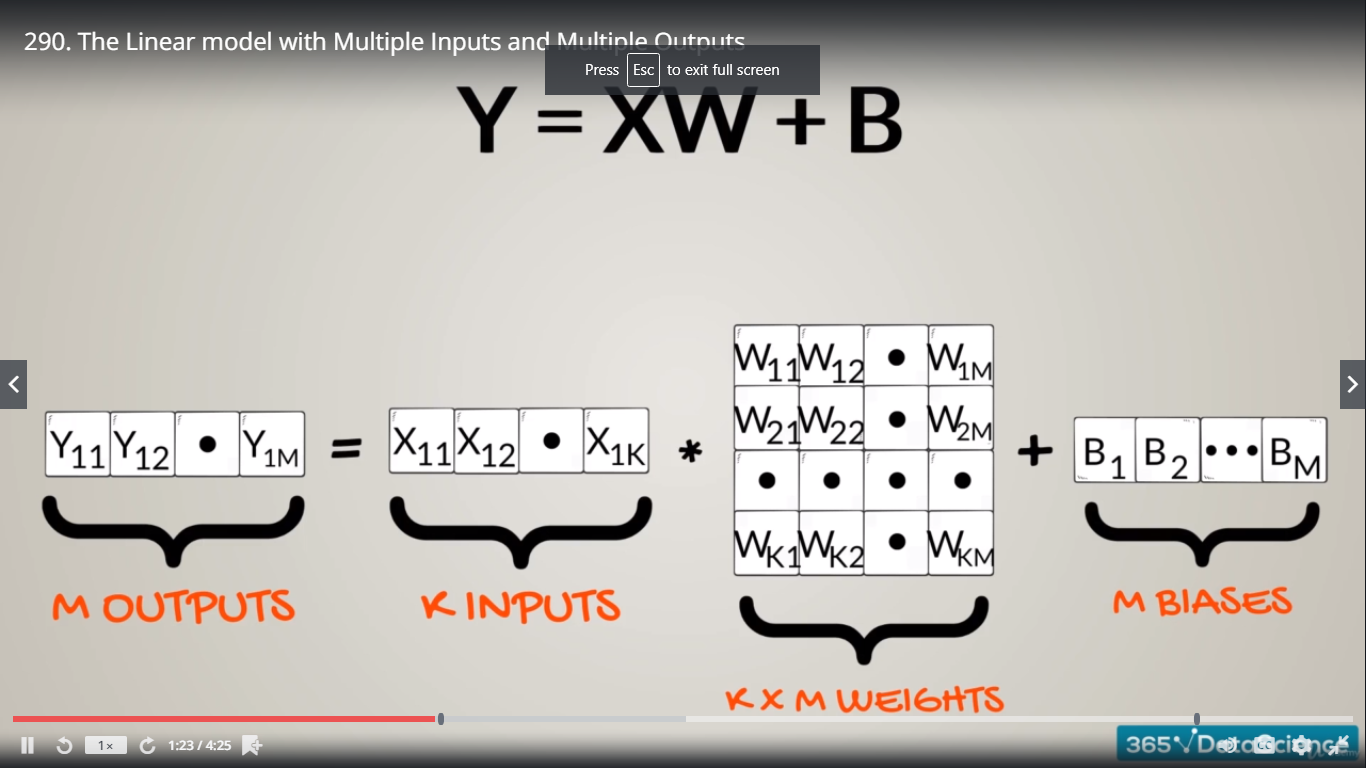
Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms therefore learn from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data. A central application of unsupervised learning is in the field of density estimation in statistics,[25] though unsupervised learning encompasses other domains involving summarizing and explaining data features.

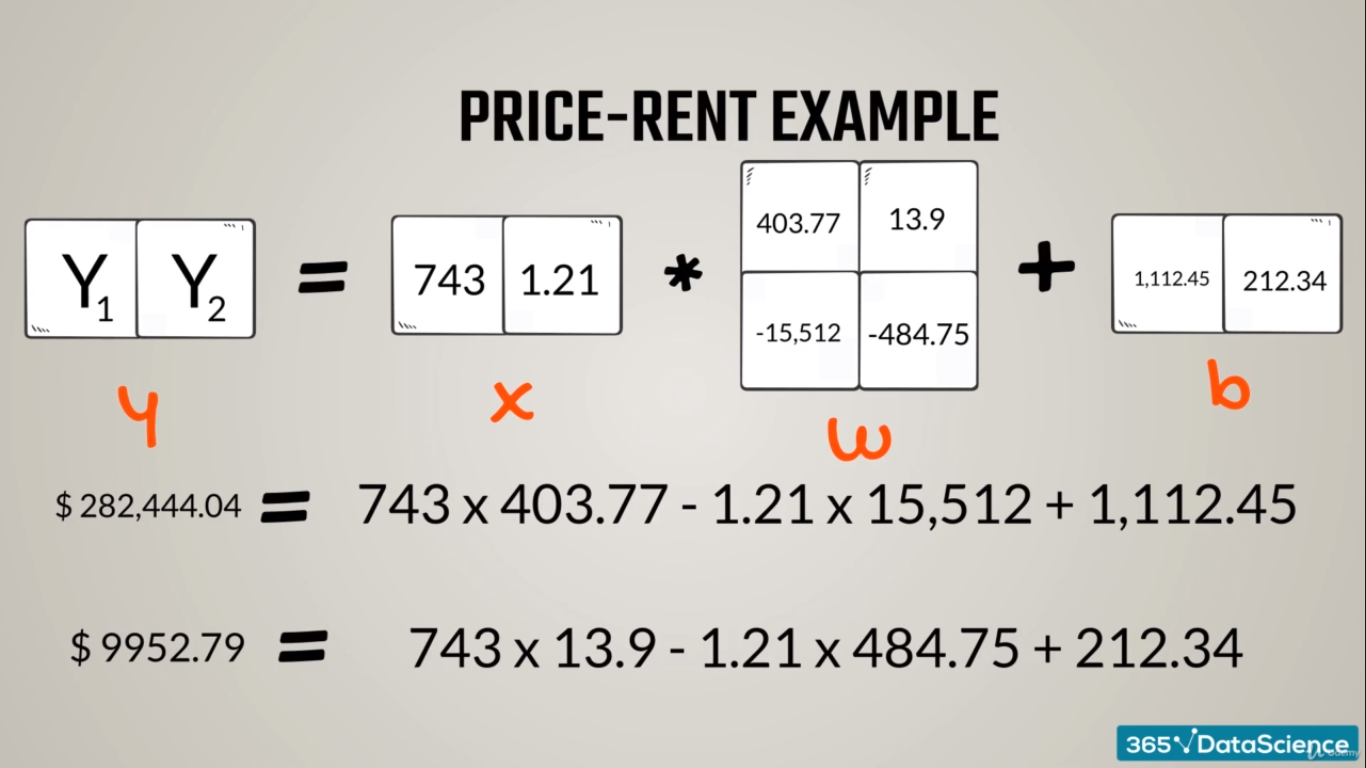
Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to one or more predesignated criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated, for example, by internal compactness, or the similarity between members of the same cluster, and separation, the difference between clusters. Other methods are based on estimated density and graph connectivity.

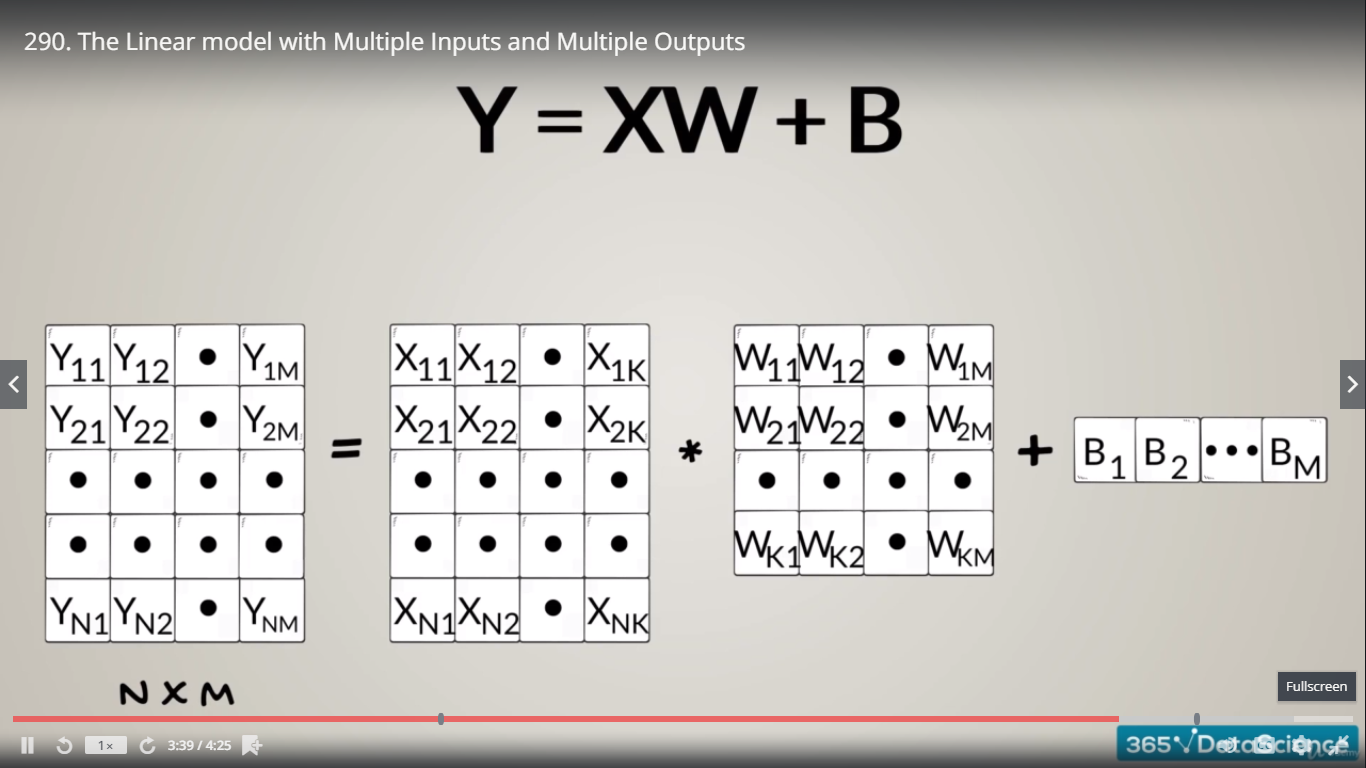
**Reinforcement learning**

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Due to its generality, the field is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In machine learning, the environment is typically represented as a Markov Decision Process (MDP). Many reinforcement learning algorithms use dynamic programming techniques.[26] Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the MDP, and are used when exact models are infeasible. Reinforcement learning algorithms are used in autonomous vehicles or in learning to play a game against a human opponent.









**Objective Function:-** is the measure used to evaluate How well the model’s Output Match the Desired correct values.

1. **Loss Functions (Supervised Learning):-** The lower the loss function, the higher the level of accuracy
2. **Reward Functions (Reinforcement Learning):**- The higher the Reward function, the higher the level of accuracy

Any Function that holds the basis Property:

Higher for Worse Results

Lower for Better Results

Can be a Loss Function

Target – is the desire value at which we are aiming

Loss

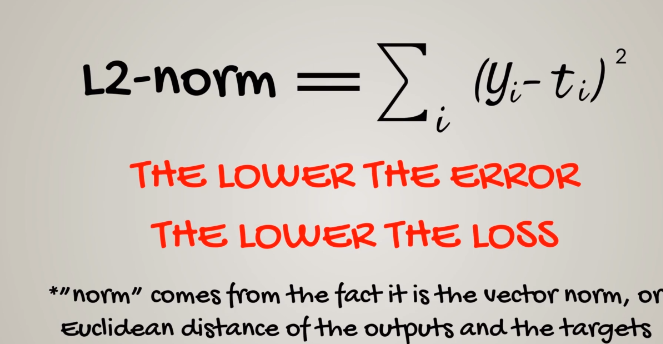
Regression

Classification

L2 – Norm Loss (OLS)

Cross- Entropy

L2-norm = OLS (Ordinary Least Squares)



A picture containing object, gauge

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**Gradient Descent:-** is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to update the parameters of our model.

**Learning Rate:** is a configurable hyperparameter used in the training of neural networks that has a small positive value, often in the range between 0.01 and 1.0. The learning rate controls how quickly the model is adapted to the problem.

**Oscillation:** A repetitive variation around a central value.

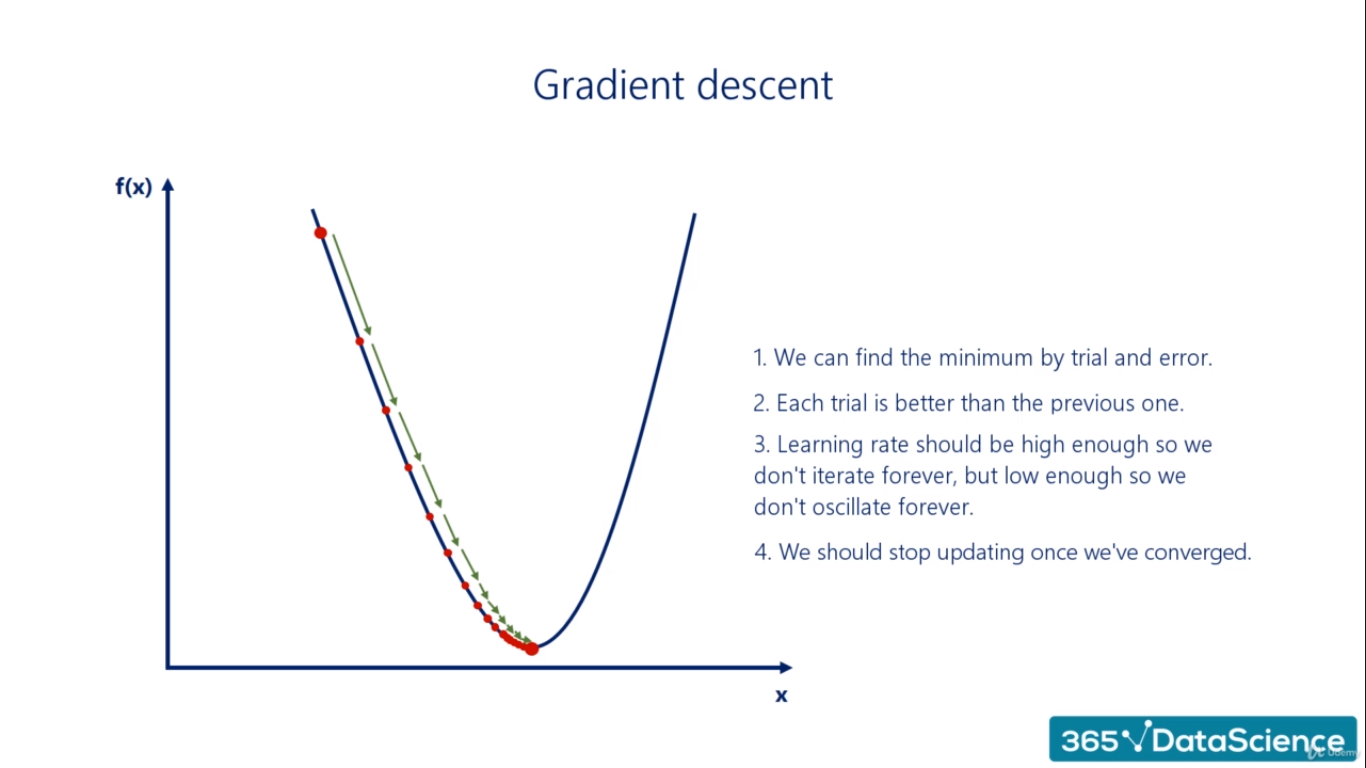
**HIGH enough**, so we can reach the closest minimum in a rational amount of time

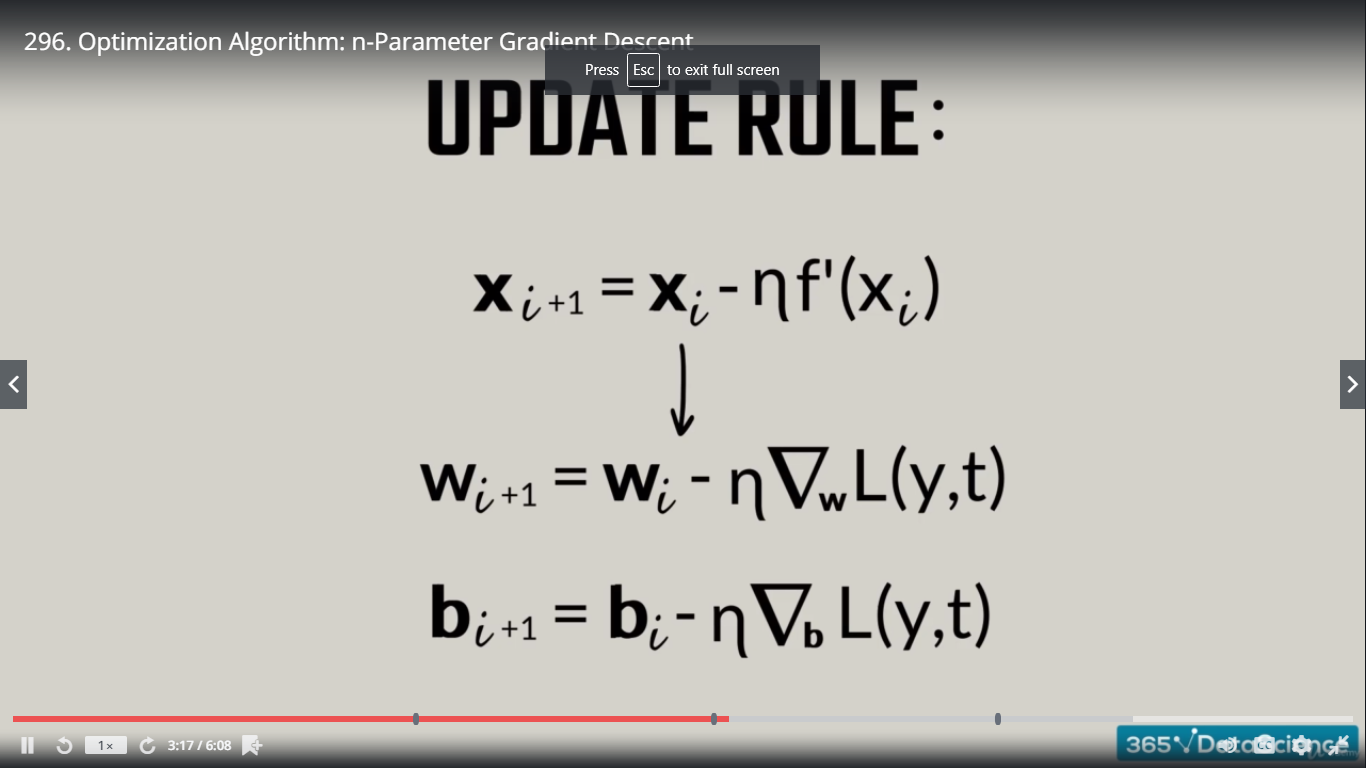
**LOW enough,** so we can’t oscillate around the minimum

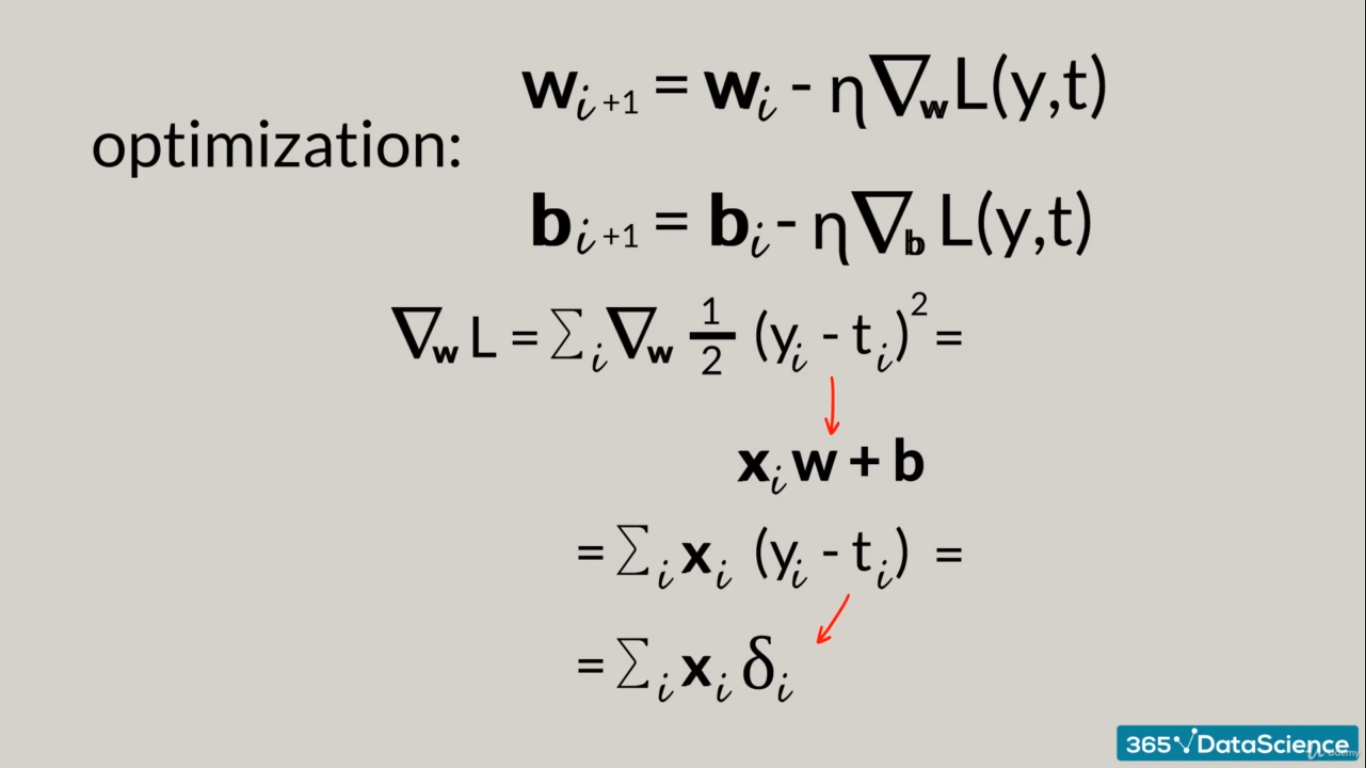
A too high learning rate may cause the loss function to diverge to infinity, instead of finding the minimum.

A picture containing sky, object

Description automatically generated









**How to Install Tensor Flow 2.0**

**Open Anaconda Prompt**

**Conda info -- envs :** to check the environment

|  |  |
| --- | --- |
| Conda create --name py3-TF2.0 python=3 | Create a new environment with the name of TF2.0  Now new environment has been created |
| Conda activate py3-TF2.0 | Activate new environment |
| Conda install tensorlfow |  |
| Pip install –upgrade tensorflow |  |
| Pip install ipykernel | Open Jupyter and check the new Kernel |
| Import tensorflow as tf  Print(tf.\_\_version\_\_)   * 2.0.0 |  |

**Tensorflow History:**

Tensorflow 1.0 (2015): one of the most widely used

2017 : Tensorflow 1.0 is integrated with Keras. At that time PyTorch was also there and all these three languages are Higher – Level packages.

Tensorflow 1.0 and Keras both are Open source.

2019: Tensor flow 2.0 comes into horizon

**Tensorflow 2.0:**

Tensorflow is a deep learning library, developed by Google, that allow us to create fairly complicated models with little coding.

|  |  |
| --- | --- |
| **Keras** | **TF** |
| Higher level than TF1 | Tensorflow 2 is basically Keras |
| Already adopted | Versatile |
| Already loved | Simplied API |
|  | Removed duplicate and deprecated functions |
|  | Added new functions |

**Project -1 Code**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import tensorflow as tf |
| observations = 1000  xs = np.random.uniform(low=-10,high=10,size=(observations,1))  zs = np.random.uniform(low=-10,high=10,size=(observations,1))  generated\_inputs = np.column\_stack((xs,zs))  noise = np.random.uniform(-1,1,(observations,1))  generated\_targets = 2\*xs - 3\*zs + 5 + noise  np.savez('TF\_Intro', inputs=generated\_inputs, targets=generated\_targets) |
| training\_data = np.load('TF\_Intro.npz') |
| input\_size =2  output\_size =1  model = tf.keras.Sequential([  tf.keras.layers.Dense(output\_size)  ])  model.compile(optimizer='sgd', loss='mean\_squared\_error')  model.fit(training\_data['inputs'],training\_data['targets'],epochs=100,verbose=2) |
| model.layers[0].get\_weights() |
| weights = model.layers[0].get\_weights()[0]  weights |
| bias = model.layers[0].get\_weights()[1]  bias |
| model.predict\_on\_batch(training\_data['inputs']).round(1) |
| training\_data['targets'].round(1) |
| plt.plot(np.squeeze(model.predict\_on\_batch(training\_data['inputs'])),np.squeeze(training\_data['targets']))  plt.xlabel('outputs')  plt.ylabel('targets')  plt.show() |

**tf.keras.Sequential():** function that specifies how the model will be laid down('stack layers')

**tf.keras.layers.Dense(output\_size):** takes the inputs provided to the model and calculates the dot product of the inputs and the weights and add the bias \*also applies activation function (optional)

verbose =0 --> No Status, verbose =1 --> Progress Bar, verbose =2 --> stands for one line per epoch

**np.squeeze:-** function is used when we want to remove single-dimensional entries from the shape of an array.



**Deep Net**

**Width:** the number of units (nodes) in layer is often referred to as the width of the layer

**Depth:** the number of hidden layer of a neural network.

We refer to the width and depth (but not only) as hyperparameters.

|  |  |
| --- | --- |
| **Hyperparameters**  **Pre-set by us** | **Parameters**  **Found by optimizing** |
| Width  Depth  Learning Rate (Eata) | Weights (w)  Biases (b) |

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone screen with text

Description automatically generated

Each arrow represents the mathematical transformation of a certain value

Non-linearity don’t change the shape of the expression. Just its linearity.

A screenshot of a social media post

Description automatically generated

**For input Layer to First Hidden Layer**

The weight metrics are 8\*9 so there are 72 weights

Each weights W23: first input 2 indicating from input unit and 3 is indicating from hidden unit.

Non-linearity is added in order to produce 6th unit (for every) of hidden unit.

**Why do we need non-linearities?**

‘Non Linearities’ are needed so we can break the linearity and present more complicated relationships.

Stacking layers are the process of placing one layer after the other in a meaningful way. We cannot stack layers when we only have linear relationships.

A close up of a map

Description automatically generated

Hidden layer is completely useless in this case. We just need to train this neural network.

Two consecutive linear transformations are equivalent to a single one.

A close up of a map

Description automatically generated

Non-linearities 🡪 Stacking layers 🡪 Depth 🡪 Deep Learning

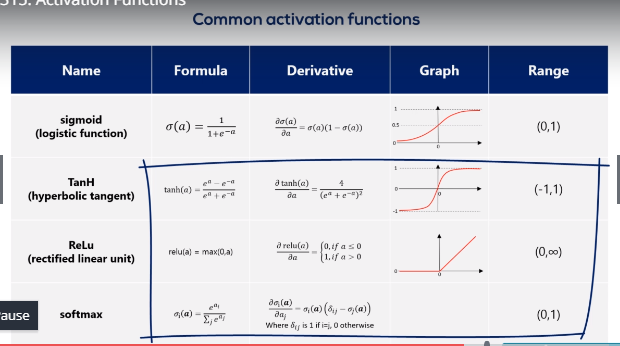
In order to have deep nets and find complex relationships through arbitrary functions, we need non-linearities.

**Activation Functions**

In machine learning context, non-linearities are called activation functions (also called transfer functions).

Activation function transform inputs the outputs of a different kind.

The derivative is an essential part of the gradient descent. When we work with Tensorflow, it’s calculated automatically.



**Softmax**

Softmax graphs changes every time.

Softmax is special. Each element in the output depends on the entire set of elements of the input.

A close up of a map

Description automatically generated

**Properties:**

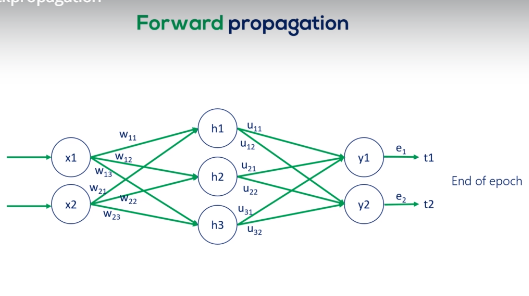
1. Range: (0,1)
2. They always sum up to 1
3. Probabilities : The softmax transformation transforms a bunch of arbitrarily large or small numbers into a valid probability distribution.

**Backpropagation**

Process of optimization consisted of minimizing the loss.

Forward propagation is the process of pushing inputs through the net and the end of each epoch they obtains output and compare with the targets to form the errors and then we backpropagate through partial derivatives and change each parameters so errors of next epochs can minimize.

At the end of each epoch, we backpropagate and change each parameter accordingly.



A close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A close up of a logo

Description automatically generated

We have no targets for the hidden units, so we have no errors.

**The algorithm adjusts:**

The weights that have a bigger contribution to the errors by more;

The weights that have a smaller contribution to the errors by less

Backpropagation is one of the biggest challenges for the speed of an algorithm.

**Training, Validation and Test**

We update the weights and biases for the training set only.

A screenshot of a cell phone

Description automatically generated

A close up of a logo

Description automatically generated

The accuracy that we get by forward propagating the test dataset. Is the accuracy we expect the model to have if we deploy it in real life.

We want to devote as much data as possible to the training dataset, while having enough left to validate and test

Training (80%-70%), Validation (10%-20%) and Test (10%)

What does ‘every now and then’ mean?

Usually, we validate on every epoch and calculate training\_loss and calculate validation\_loss

The accuracy we obtain at this state is the accuracy of the algorithm.

**N – Fold Cross Validation**

Training + Validation and Test

(10000) (1000)

Dataset = 11000

**Side Note:** “Ginormous” datasets have their own problems – being so large, they often have a lot of missing values, We refer to them as being: sparse

A screenshot of a cell phone

Description automatically generated

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Utilized more data  We have a model | Possible overfitted a bit |

**Early Stopping**

We want to ..stop training early.. before we overfit.

1. Preset number of epochs

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Eventually, solves the problem | No Guarantee that the min is reached  Maybe doesn’t minimize at all  Naive |

1. Stop when updates become too smalo

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| We are sure the loss is minimized  Save computing power |  |

1. Validation Set strategy

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| We are sure the loss is minimized  Saves computing power  Prevents overfitting |  |

A screenshot of a cell phone

Description automatically generated

We save computing power by using as few iterations as possible, we don’t iterate uselessly

**Minimal Example:** The minimal example was super simple. A more complicated strategy would be a stretch.

Updates too small:

Validation Set Strategy iterates until we start overfitting.

Best Strategy is to use Updates too small and validation Set Strategy.

A close up of a device

Description automatically generated

**Initialization:** Initialization is the process in which we set the initial values of weights.

Lets initialize our weights and biases in such a way that they are all equal to a constant.

Weights are used in linear combination and then linear combinations are activate and we use sigmoid activator.

A close up of a map

Description automatically generated

If all the inputs for the sigmoid (the linear combination) are in this range. The activation will be linear. Non-linearity are essentials for deep net.

If the values are too high or too low then function values are almost flat.

A static output (only -0s or -1s) is also problematic, as the algorithm doesn’t learn.

A screenshot of a social media post

Description automatically generated

Understanding the difficulty of training deep feedforward neural networks

Xavier Initializer is the default initializer.

Optimization is done by Backpropagation.

A close up of text on a white background

Description automatically generated