Greetings everyone, this is Yash Kumar from CSE IInd Sem. Today I am here to sort of teach you guys about Machine Learning, Artificial Intelligence, and what’s the deal that goes inside to make this one of the biggest evolving branches of Tech World that makes it so popular and useful in the current era. We’ll try to keep it in length so that it is as much close to Layman as possible and not a bit overwhelming, but at the same time how it transcodes to the Tech Language. Now, the idea is for you all to have the basic knowledge of computers, how they work, what goes behind the computation and reasoning, and how we as programmers should be manipulating it to our advantage.

**WHAT IS ARTIFICIAL INTELLIGENCE**

Artificial Intelligence is a heavily debated hot topic in the recent tech industry. Unlike arithmetic and constrained computing. To call out on the meaning, AI in Layman’s terms is the scope of computing derived from uncertainty in the scope of a material paradigm. It requires the constructs utilized to dictate the flow of output as per rules, but not categorize it to a field. What it actually means is that, let’s say, we get a question 2 + 2. For a regular calculator – it’ll directly convert it to e ^log2 and then perform hereditary differentiation. However in a decision dictated scenario, what basically happens is you take a couple of data from the predetermined base, based on which you try to decipher what will be for the follow up. Like, let’s say you got two phones at the moment. Then theres various deciding factors like the price, it’s specifications, the brand value, and most importantly, your personal bias. This can be well translated in the form of mathematical expressions, that would express how the output should flow. Conditionally tho, with every step of recurring calculation, all the inputs fed to the system change, and as per that, the output everytime would show an inclination towards a particular property of the phone itself. The AI although isn’t limited to these basic constraints, it can predict weather, optimal conditions for a space mission, real-time navigation based on several factors – like the one used in SpaceX’s falcon heavy, balance stock markets pricing, automate vehicles and many more. Quantum Computers are the ones that are actually able to exploit AI to its mass scale as they work with the help of real world uncertainty and bias factors like qubit transitions, spectrum analysis algorithm and many more.

**MACHINE LEARNING**

Machine Learning or ML as it is abbreviated, is a small yet very prominent subset of Artificial Intelligence that tends to bring logical computation as close to the masses as possible. From a mathematical point of view, it is called Statistical Learning as well. It involves learning algorithms that utilize prior information and make them more potent compared to their previous handling of the same. Machine Learning tends to take data information and convert it to algebraic/transformation equations. These equations then interpret various forms of preprocessed data to learn from it, and then make use of it in the real world to generate precise outcomes for similar situations. A practical common usage is found in food sorting industries on a high grade. They have image scanners making use of hyperspectral data that is used to analyze defective fruits, big fruits and so on placed on rails with actuators that moves the whole product and categorizes them.

Types of Machine Learning -

1. Supervised

2. Unsupervised

3. Reinforcement

Supervised learning is the form of ML where we provide data to the algorithm and the output relevant to the corresponding data index. The model then learns from the data fed to it so far, and tries to produce output as close to the relevant one as possible. The most basic example would be a student in a classroom. The teacher shows him a couple of questions, and the answers corresponding to those questions. The student is then supposed to decipher the logic behind obtaining that answer, and learns from it. After which the student is given raw questions, and based on what he or she has learned, should find the output which should be closest to the actual answer. Now, like in a real world scenario, the output cannot always be correct, just like the answers of the student. Another example would be a weather forecast media just like the ones we discussed a while ago. Let’s say we feed it three data – temperature, humidity and wind speed. Then we give it the corresponding weather forecast on that day. Based on this, the model learns at what temperature and humidity rate and wind speed should the weather be like and, when queried with a similar data sample, it would give the near-perfect answer.   
This sample data fed to the algorithm along with the answer is called training data. The model trains over it and then gives output. Training basically means developing a symbiotic expression between the data and the answers fed to the model. This expression then is then utilized for after-training data.

Unsupervised – In unsupervised learning, we give the algorithm input, however there’s no corresponding output fed to the model. The model itself is supposed to find some sort of relevance in between the data fed so far, and based on that it should categorize the output in all possible distinct types. For example, let’s say we have a model to which we feed a set of images of cats and dogs. Now what we would anticipate the cats to be automatically placed in the cats category and the dogs to be placed in the dogs category. In supervised learning, we would tell the model to do so based on the training output data we feed to them to derive a correlation. However, in scenarios where we don’t have a sample data to train the model like in supervised learning, or when we don’t know all possible categories of data we fed to the model, in such cases, unsupervised learning is very helpful. We could simply ask the model to split the query dataset of images into two categories based on visual similarities. We would get two groups of data one of which containing cats and the other one containing dogs – and then the user can simply label the group of pictures containing cat images as cats and the other group as dogs. Unsupervised learning is thus very helpful when the prerogative is to split the dataset into a specific number of unknown categories. This method is called *clustering*.

Reinforcement – The model is made to act like a progression handle with every time it performs a task. Reinforcement Learning makes use of trial and error to find out which actions serves the objective the best. Let’s compare this model to a 10 year old kid in school. He’s promised to be given chocolates if he finishes his homework for the day. Once he completes the homework, as promised, he gets his treat. He’s later on promised a bike when he gets the best score in the coming SATs, which when he accomplishes, he gets it. This step of using positive reinforcement as a correction utility for managing a model is called **progression handling**.

Supervised Leanring is the most common and widely used way of implementation of machine learning to a particular problem these days.

**NEURAL NETWORKS**

To begin with, lets get a reference from Biology. Neurons as we know are the brain cells which are the elementary scope of information transfer of all sorts throughout the body for coordination, control, conscience, behavior and many others. A single neuron however, isn’t capable of conducting/performing all these operations by itself. So, what the neuron does is it receives an impulse, based on which it creates a Na+ and Cl- concentration around and inside itself. Measuring the + or – concentration tells about the bias or the inclination of the neuron towards a particular output. When these neurons work together as threads or nodes to contain several feeds and give out a bias based on the aggregate inclination of every node, it is called neural network.  
  
If we translate that to Computer’s terminology, a neural network is nothing but a model of several nodes (each having a dependency compliance based on probability between the region 0 and 1). The probability of 1 ascertains the node is compliant to be positive, i.e., holds pure inclination, and the probability of 0 ascertains the node is declined to a negative range, i.e., it holds 0 sign of action.

When these neurons are placed together, and made to work on a challenge, like let’s say finding the sum of 2 + 2 (an example in reference to deep neural networks). The hidden nodes (or the neurons working between input feeder and output resolver) they start ranging it within the limit of +2 to -2 for the second operand. Then the first operand is passed to each node with a specific batch size and a probability ratio, or constraint. These intermediate nodes work over each other, like a cricket match and obtain the aggregate bias among 2 or 4 (depending on the batch size provided). This aggregate data from each node exit end is then used by an algorithm called Map Reduce to calculate a resultant bias by making use of matrix integration and then solving linear equations by cramer’s rule. This gives the end result or a single bias value which is then translated to a boolean, be it true or false. Use cases involve signal processing, pattern recognition, etc.

COMING BACK TO SUPERVISED LEARNING :

Since we’ll deal this whole video with the perspective of supervised learning, let’s understand its types...

1. Regression

2. Classification

Classification supervised learning models just provide outputs in subtypes or categorize, like for a coffee maker, classifying how good or bad the coffee is made.

Regression yields a numerical value asserting boolean or value figure, for example, for a robot that solves rubik’s cube, a matrix of which patterns are tackled, and how many more moves are required.

Coming to the 4 important aspects that constitute the whole ML experience :

1. Data

2. Model

3. Objective Function

4. Optimization Algorithm

Let’s begin with a basic example,

let’s say y is a function of x such that   
f(x) = y

Now, it is to find out the relation between x and y by providing it with many pairs of x and y observations.

Let’s start with Linear Model,

the basic formula or the intercept function as we recall from coordinate geometry from class 9, is given as

f(x) = xw + b

which looks something like y = mx + c

where m is the slope (or the coefficient) and c is the intercept.

This model can although be defined in multiple ways, however to keep the scope of this illustration concise and easy to understand, let’s consider this one.

In machine learning, the equation is translated to y = xw + b

where w is called the weight of the input, and b is the bias

Let’s look into these terminologies and understand what these are with the help of an example.

We have to figure out the price of an apartment from a pre trained model.

The formula is xw + b which gives the output or the price y

let’s say the size of the apartment is x = 763 sq feet.

Now, the cost per sq km changes every now and then, and could vary depending on the season, region, etc. This is considered as weight of the model.

Another thing to consider is the locality, the apartment could be close to a beach, which increases it’s price, or it could contain glass windows with a clear ocean view, or it could be very bad apartment with dark surroundings and a ivy vine lurking on the windows, which significantly drops it’s price.

Let’s convert it into a mathematical expression onw.

By formula y = xw + b

let the cost sq per feet be 300 rupees. Since the apartment is in congested location in Rajanukunte, we’ll have lower bias or lower attraction points. So, let’s say the cost margin deplets by 1000 rupees or the owner gives us a discount of 1000 rupees.

Then cost = 763 \* 300 – 1000 = 227900

Let’s consider another apartment in Goa of the same size, close to the beach. Now rate per sq feet in goa is let’s say as high as 550 rupees.

And since the apartment has an ocean facing view, the bias or the overcharge adds upto around 5000 rupees extra. In that case, the cost of the apartment is given by

cost = 763 \* 550 + 5000 = 424650

This is how the model works. Now based on these multiple observation/training data, the model develops it’s own weights and bias, and depending on the locality you’re searching for, or the aesthetics of the apartment you’re buying, the model will implement +- weights and/biases to comprehend for the near-actual costs.

Another example would be working of a stock exchange market. How the prices go up, how a model can tackle it, and would help predict the best stocks to invest your money in.

Make sure, the higher the number of parameters fed to the algorithm, the more accurate predictions it would give.

It is given by y = x1w1 + x2w2 + b1 + b2

Similarly, weights and biases upto n numbers can be added or as defined by the model, forced to be extracted.

So, that clears our concept of data and model

Now, diving into **OBJECTIVE FUNCTION**

This is like an evaluation scheme of how well or poorly the model has performed. They’re split into two types – loss function and reward function (deprecated)

The lower the loss function means the better performance by the model. Loss functions are also called cost functions.

Let’s say we train a model to recognize an image of a dog, and then give it the parameters like height, color, shape, maybe even species, and stuffs, and then apply the loss function. If we get the loss output like 120, or 80, this means the model performed poorly, and is not able to properly recognize whether the image fed compares to a dog or a cat.   
If the loss function is low like 10, 5, or sometimes even 0, it means the model’s accuracy rate is extremely high.

Reward functions although deprecated perform just the opposite. If the reward output is higher, it means the model is more accurate, and vice versa. They’re mostly used in reinforcement learnings, like area mapping robots and similar stuffs as discussed prior.

A common example would be, a game of dropping balls, instead of bouncing it, the model has to pllay it the number of times it touches the ground, the lesser number of times it touches the ground, the better. That is evaluated by loss function, as higher number of touches to the ground would be a bad thing, and as programmed – would result in losing the game.

Now, there’s not any one type of loss function, there’re several. However for beginners as well as for common pracitce models, there are two famous loss functions used namely

L2 norm, and cross entropy.

**L2 norm** : This is also called squared loss. This loss function is used for regression type. Let’s start with an algorithm that’ll predict to what percentage an image fed to a pre-trained model is similar to a specific category. Mathematically, it’s given by

Σi (yi – ti)2

which gives the summation of square of differences between the output value and the target value. When visualized (we’ll talk about that later on in maybe another session of this video tutorial), that when visualizaed, Euclidean distance between the plotted target and output vector is called norm, and since it is linear summation squared, it’s called L2 norm. The lower the sum is, the lower the error, hence the better the model...

**Cross-entropy** : Cross entropy is a loss function used to calculate accuracy deviation in classification supervised models rather than regression. What it means is that, it will give a percentage deviation of originality between how the output should be categorized to how it actually is. Mathematically, it is given as

L(y, t) = -Σi ti ln yi

This gives the summation of negative product of target vector to the natural log of output vector.

Illustrating further with an example, let’s consider a classification of datasets into three categories – A, B, C images. For a provided testCase,

t = [0, 1, 0]

The image representation sequence from the target vector implies, it’s not the letter A, it’s B, it’s not the letter C.

Now, for the image of B, let’s say the output vector we obtain is given by

y = [0.6, 0.7, 0.2]

These vectors show the probability for the feeded image to have a 60% chance of being the letter A, 70% chance of being the letter B, and 20% chance of being the letter C.

So, if we calculate cross-entropy is given as

L(y, t) = -0 x ln 0.6 – 1 x ln 0.7 – 0 x ln 0.2

This when calcultated gives us 0.35667

Let’s consider yet another image of the letter B, the target vector remains same, i.d., t = [0, 1, 0]

But, the output vector is somewhat y = [0.2, 0.8, 0.1]

Now, from this, if we calculate cross-entropy, we get

L(y, t) = -0 x ln 0.2 – 1 x ln 0.8 – 0 x ln 0.1

This when calculated gives us 0.22314

Now, as we already know, the lower the loss, the more accurate the model is. Hence, from model’s output vector, it’s evident that the second image is much closer to being the letter B than the first image.

**OPTIMIZATION ALGORITHM**

The easiest way to understand this is a sort of software controller that handles the output vector by varying the values of weights and bias in a simple linear model. The rate at which these values are varied is managed by an algorithm called **GRADIENT DESCENT**. There are several gradient descent algos, we’ll be using SGD or (Stochastic Gradient Descent) for further understanding, as it’s easy to understand, and comparatively useful at the same time). Stochastic basically means that the probability figures could be determined by the previous link modulation but could be not be predicted to 0 figure precision or 100% accuracy.

Gradient descent works by multi variant generalization of divergence of the function the model works on.

Let’s say, we obtain a function y = 5x2 + 3x – 4

First derivative = 10x + 3

Let’s choose an arbitrary number x0 = 2

Then, carrying on, we get x1, x2 and so on using the formlua

x(i+1) = xi – ηf’(xi)

So, x1 = 4 – η[10 \* 4 + 3] = 4 – η43

And similarly, further values can be calculated.

Here, η is called the learning rate. Using this update rule, we can find x2, x3 and so on.

After a certain ratio of iteration, the gradient output deviation would collapse to 0, id est, the output itself will be the same for subsequent iterations. That’s the point where the function is minimized to it’s lowest figure.

Hence, at that point, x(i + 1) = xi will be the statement, id est, the derivative equates to 0, hence the learning rate deprecates to null. NOTE that the learning rate for a model cannot be 0, hence it’s said to be nullified.

The speed of minimization depends on the value of η. The lesser the value of η, the larger number of iterations is required to equate x(i+1) to xi.

NOTE that if the learning step is higher than the deviation per iteration, it may never reach the exact value, and will continue to sit at the value before the target output with the same loss gradient. This state is called oscillation of min. The model bounces around the minimum, but because of the step being so large, it never reaches that value. Hence, it’s a good idea to keep the learning rate high enough so as to minimize the iteration count to reach the desired vector, yet not so high so that it oscillates around the value. Mathematically, it is given by

x(i+1) – xi > step

For SGD, normally the step is taken to a lowest minimum of 0.001 so as to avoid possible innane iterations that’ll consume resources for no reason.

For basic examples, you can feed this data to an excel sheet, and apply the formula, and then you can try various η values to test yourself whether it reaches the minimum for a given polynomial or oscillates around the vector indefinitely.

Similar to this, you can n-parameter gradient descent working over multiple steps that helps reduce iteration limits, but at the same time, requires much precise banking to each input constraint.

If you’ve studied Rayleigh’s power method for calculating eigen vectors in 1st sem, it is very similar to that.

**UNDERFITTING AND OVERFITTING**

As you start coding your model and implementing your own share of manifested ideas, you might come across these terminologies.

To understand these, you must have the basic understanding of noise. So, what is noise. Although without a proper data visualization, it’s not really the best way to learn noise, but we’ll try our best to figure out what it is and why it’s important to understand it for datapoint vectors.

Noise is basically randomness or irrelevant observations in a dataset that tends to throw off the accuracy of the model. A very easy to understand example would be, let’s say there’s a model to calculate the sum of two numbers. It’s training over a dataset that holds the operands and the final sum. However, there’s some points, where the sum equals the products. It develops weights and bias for that equation frame as well. And when fed similar data, it might yield random irrelevant output. This random irrelevant output when visualized on a 2D plot plane as datapoints, is called noise.

Let’s say the model trains over the structure {op1, op2, out} and it trains on {2, 2, 4}  
It can derive both sum and product relation between these two. This adds a random bias and weight to the model, or a deformed learning. Now, when the model faces a testData of vector {4, 4}, it tends to generate more probability for 8 since it mostly trained on sum, however it will also show some noise for 4 \* 4 = 16. (It will be much more clearer when we explain it with the help of visualization tools in the next session).

Now, an underfitted model :- Broadly, underfitting means the training is sustantially bad, and the model is unable to properly figure out a proper relationship between all the training data and their corresponding outputs. It manages to understand only a couple of relationships, and it’s outputs are hence not so accurate. In such cases, if visualized on a 2D graph, the model would tend to lose many data points and would hence prove ineffective as it would have very high loss function. Example – if you try to converge the datapoints of a DLM into a linear model, it does happen to capture data points, but misses on most. Hence, it does provide the answer, but doesn’t understand the actual logic and would necessarily have very high cost.

Overfitting – On the other hand, some models train so hard, that they even represent a linearity with harmonic datapoint plots. For example, if a model is to train to just find the sum, then it would even capture some innane noise like I discussed priorly about the 4\*4 times too. These models tend to incorporate and establish a mathematical relationship across all possible datapoints present. Hence are called overfitted models. Terminologically, the model trained so well that it missed the whole point of the issue, and thus modelled the noise as well. In these cases, even if the lost is very low with an accuracy of 99%, it tends to throw wrong results.

Finding the point where the model isn’t underfitted, or overfitted is called bias-variance tradeoff.

**PURSUING MACHINE LEARNING**

Machine Learning has just begun, and needless to say with the power this tech possesses, it’s going to ramp up the world in the days to come. For anyone with experience in computational programming for over 1-1.5 years, he or she can easily take up machine learning to understand the logistics behind it and start working with all sorts of algorithm. Some primary things you’ll need to have full control over is calculus, linear algebra, efficient programming in a high level language, and ofcourse some documentation.

There are some keypoints I believe that associate with your eagerness to learn Machine Learning :

1. A formidable imaginative mind to create something that is yet to be realized.

2. Translate any sort of problems into codes that’ll do the work for you. An excellent example to this would be Facebook’s source code transcoder.

**CLEARING UP**

At this point, if you have command over a good language like Python, Java, or Golang (prefereably as chosen for tensorflow), and you go through the documentation once, you can pretty much code anything. Ofcourse to learn to optimize your model better, you’ll have to further learn different cost functions, more creative ways to bypass over/underfitting models, more powerful optimization algorithms, and their functioning and choose which one is applicable for which problem.

To finish up, I would like to say, unlike other types of programming, where if you imagine something and you know the tech associated with it, it’s just a matter of coding, Machine Learning gives you that difference which leaps beyond your level of creativity. You can reinvent it, create something that isn’t bound to be constrained by the computational limits of your system, but reaches far beyond that. Do note that we have just scratched the head of this giant, it’s a deep ocean as you dive deep within.