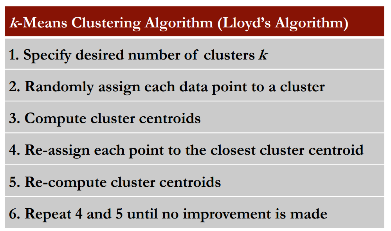
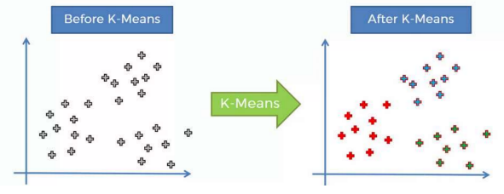
**Introduction to Artificial Intelligence (AI)**

**Week 6 Clustering**

****

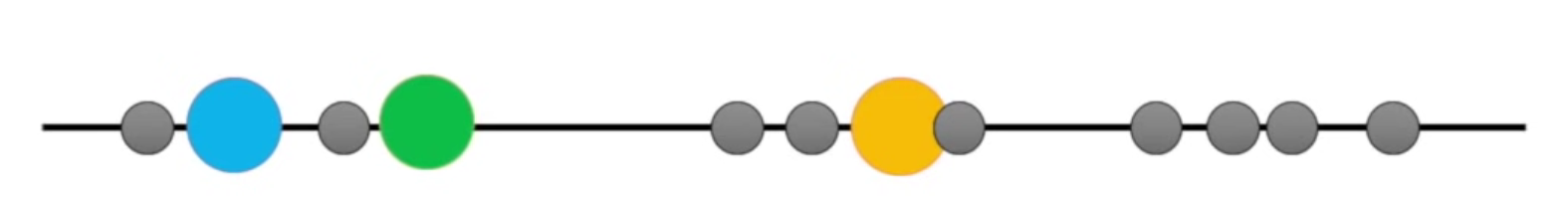
**K-Means Clustering**

We use a computer to identify 3 clusters

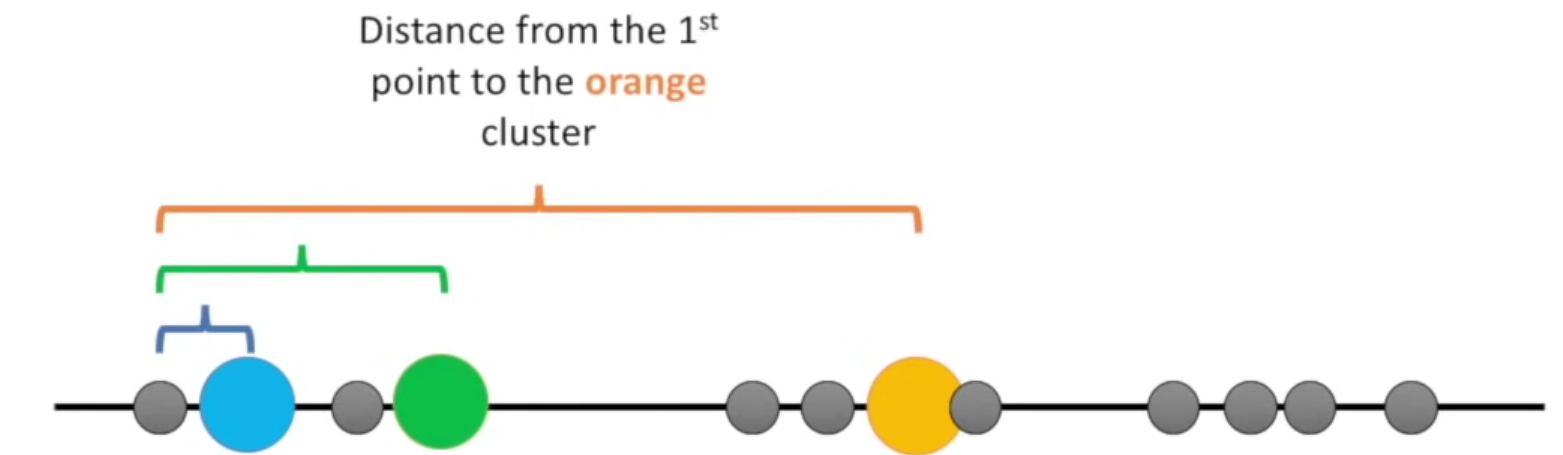
The objective of k-means is to **minimize the intra-class variance**. In simple terms, this means that you want the points within each circle to be as close as possible to the center of that circle. The closer the points are to the center, the more similar they are considered to be. So, by minimizing the distance of each point from the center of its circle, k-means ensures that the points in each cluster are as similar as possible to each other and different from points in other clusters.

**Step 1**: Select the number of clusters you want to identify in your data. This is the K in the K-means clustering (k = 3 means we want to identify 3 clusters).

**Step 2:** Randomly select 3 distinct data points (these are the initial clusters)



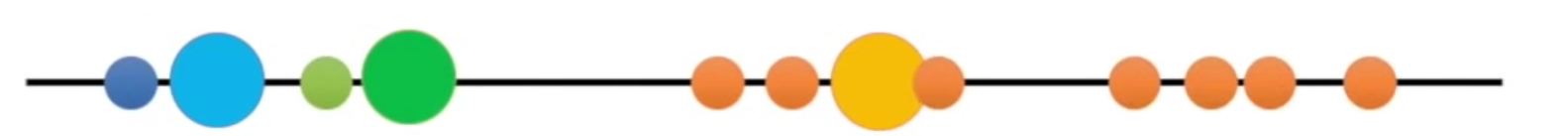
**Step 3:** Measure the distance between the first point and the three initial clusters.



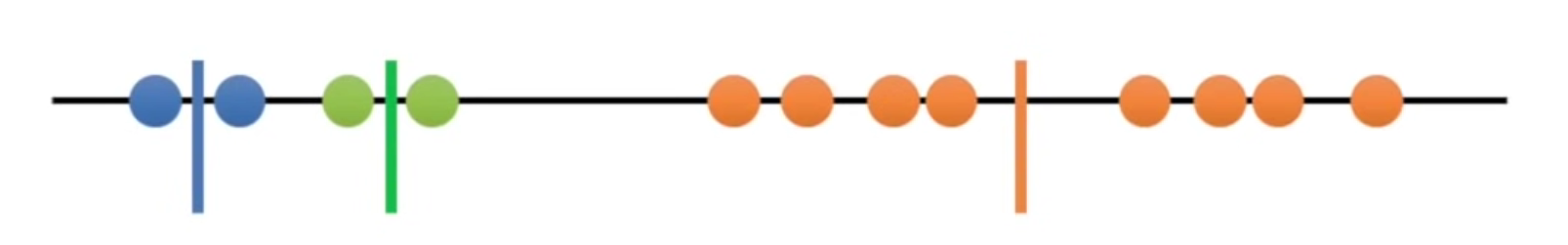
**Step 4:** assign the first point to the nearest cluster. In this case the nearest cluster is the blue cluster



We do this for every cluster:

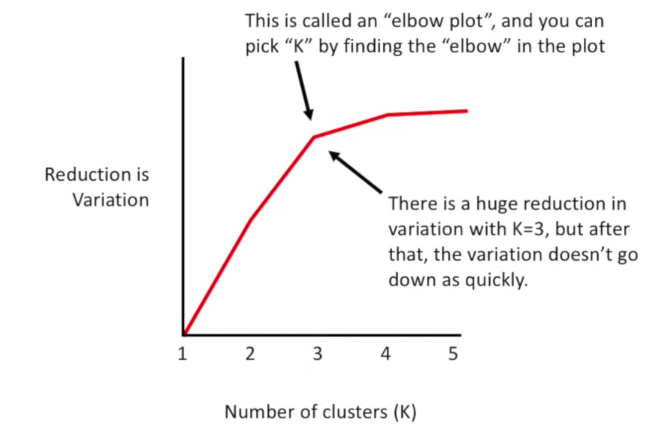


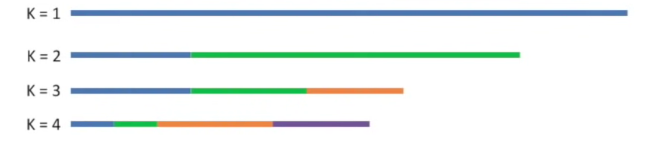
**Step 5:** Calculate the mean of every cluster: (clustering didn’t change so were done)



**Step 6:** Do over and over and over again, till you find right one.

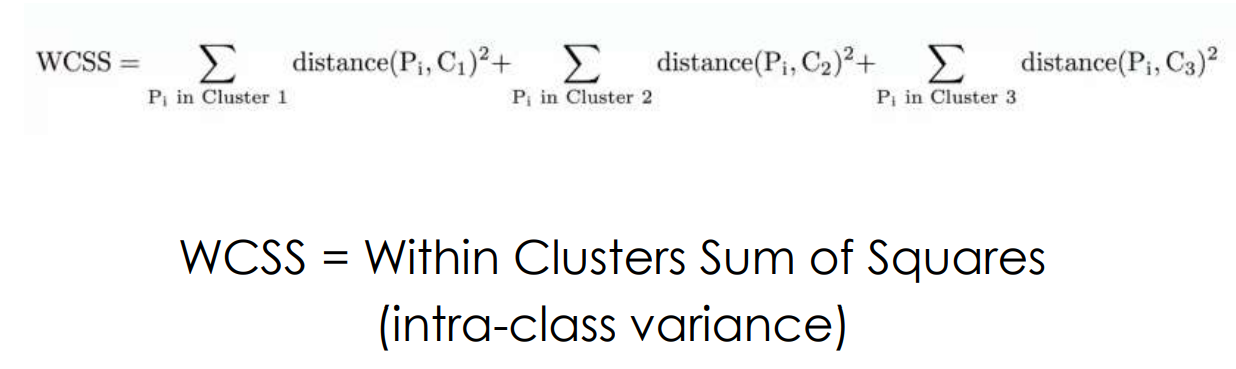
But what number should k be?

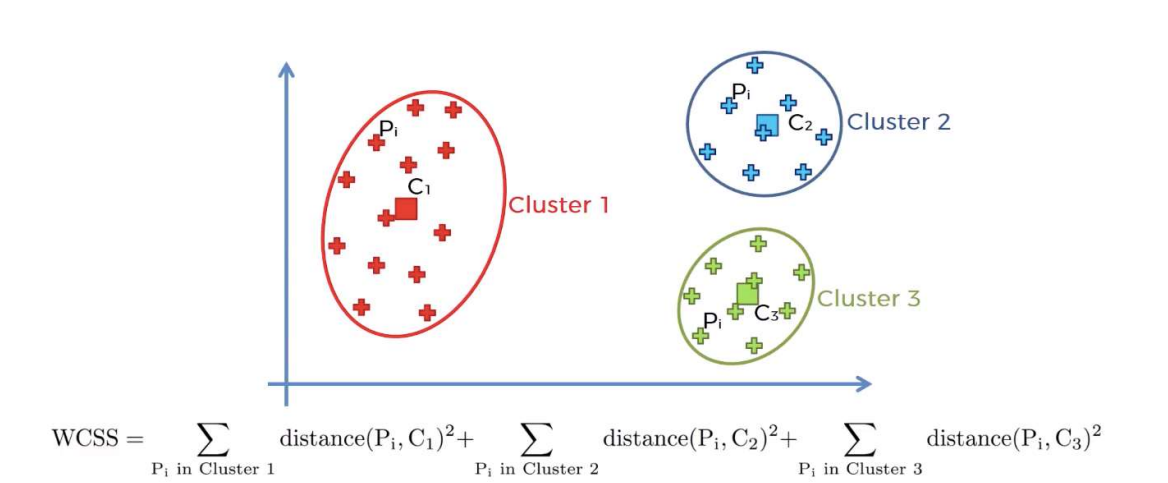
* Try different values for k
  + K = 1 worst case scenario
  + K = 2 is better
  + K = 3 is even better (compare variation to k = 2)
  + K = 4 variation is less then k = 3, so worse.



You find K by finding the elbow in the plot

So if k is not known in advance:





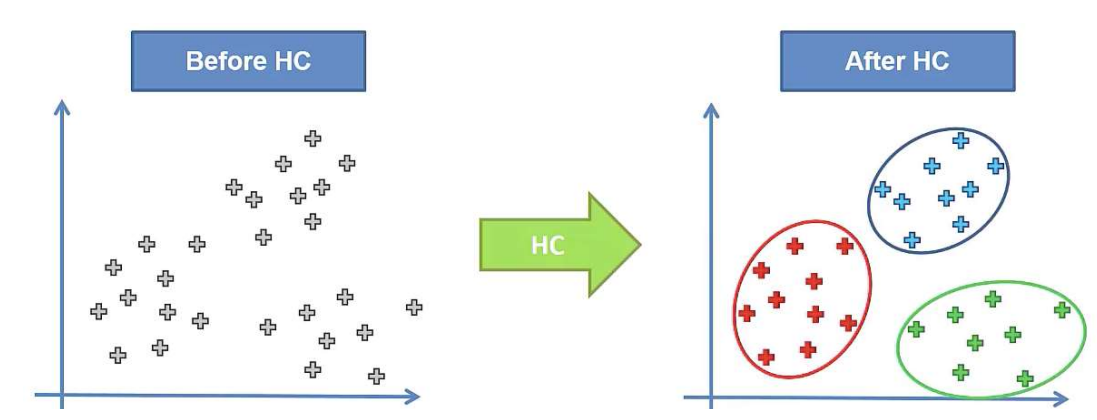
**K-Means++**

K-means++ is an improved version of the standard k-means clustering algorithm. It's specifically designed to choose the initial centers (centroids) for the clusters more effectively, which can lead to better final clustering.

1. **Choose the First Center Randomly:** Start by picking one data point randomly from your dataset. This point becomes the first center of a cluster.
2. **Calculate Distances for Each Point:** For every other data point in your dataset, calculate the distance to the nearest center that you have already chosen. This step is crucial because it sets the stage for how the other centers are selected. The distance D is a measure of how far a point *x* is from the nearest existing center.
3. **Choose New Centers Based on Weighted Probability:** Now, choose the next center, not just randomly, but with a probability weighted by the square of the distance. This means a point is more likely to be chosen as a new center if it's far away from the existing centers. This step helps in spreading out the centers and avoids choosing centers that are too close to each other.
4. **Repeat for Choosing Centers:** Keep repeating step 2 and 3 until you have selected *k* centers. Each time you choose a new center, it influences the probability distribution for choosing the next one, ensuring a good spread of initial centers.
5. **Proceed with Standard K-means:** Once you have all your *k* initial centers, proceed with the standard k-means algorithm. This involves assigning each data point to the nearest center, recalculating the center of each cluster, and iterating this process until the centers stabilize and don't change much anymore.

The main **advantage** of K-means++ is in how it selects the initial centers. By choosing these centers wisely, the algorithm reduces the chances of falling into poor clustering solutions, which can be a problem with standard k-means if the initial centers are chosen poorly. This leads to better, more reliable clustering results.

**Hierarchical Clustering**

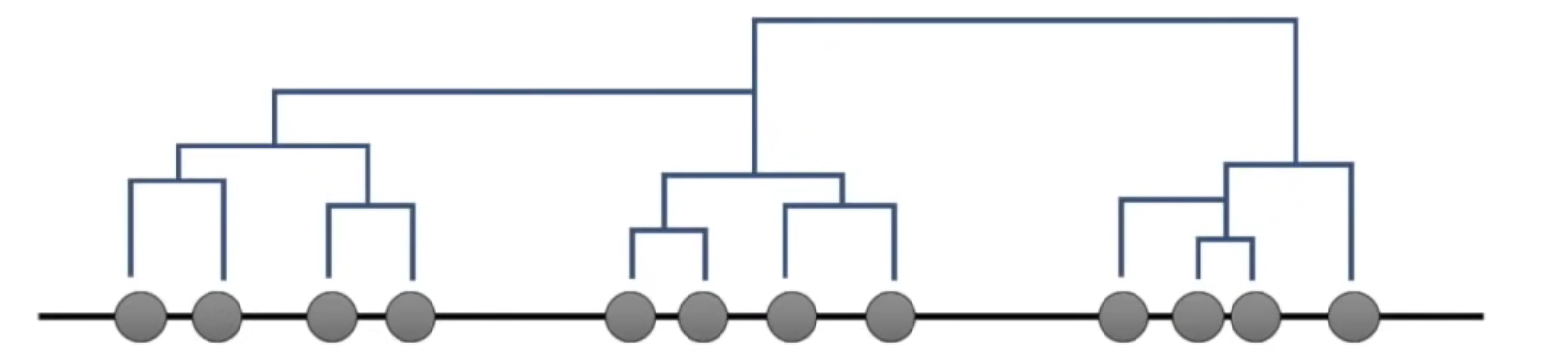


The same as K-means, but the process is different!

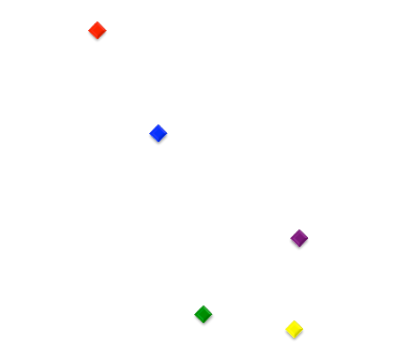
**k-means** clustering specifically tries to put the data into the number of clusters you tell it to. But **hierarchical clustering** just tells you pairwise, what two things are most similar.



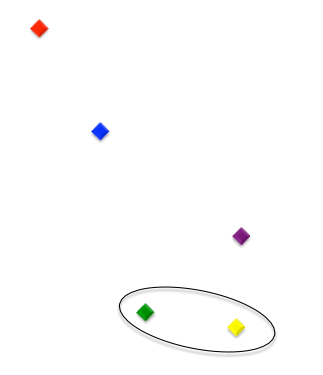
Vs



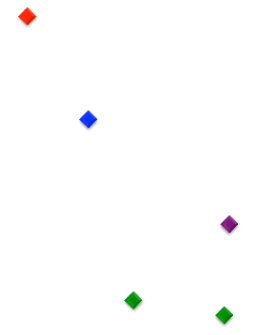
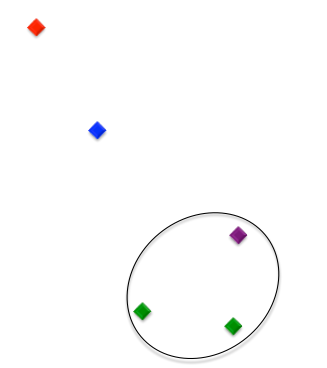
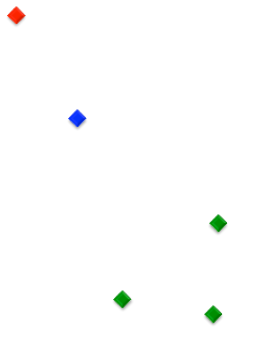
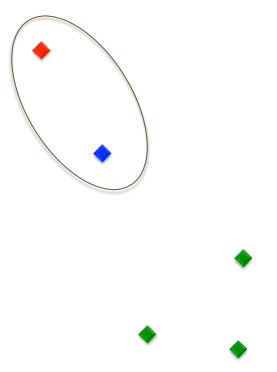
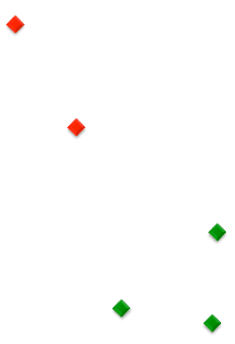
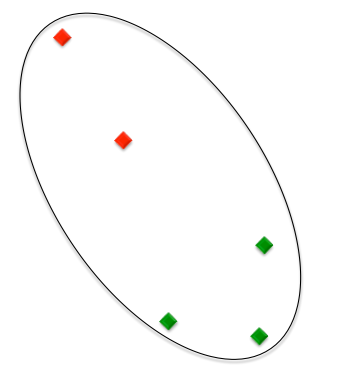
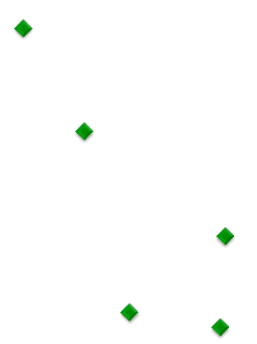
**Step 1**: Make each data point a single-point cluster. (That forms N clusters).

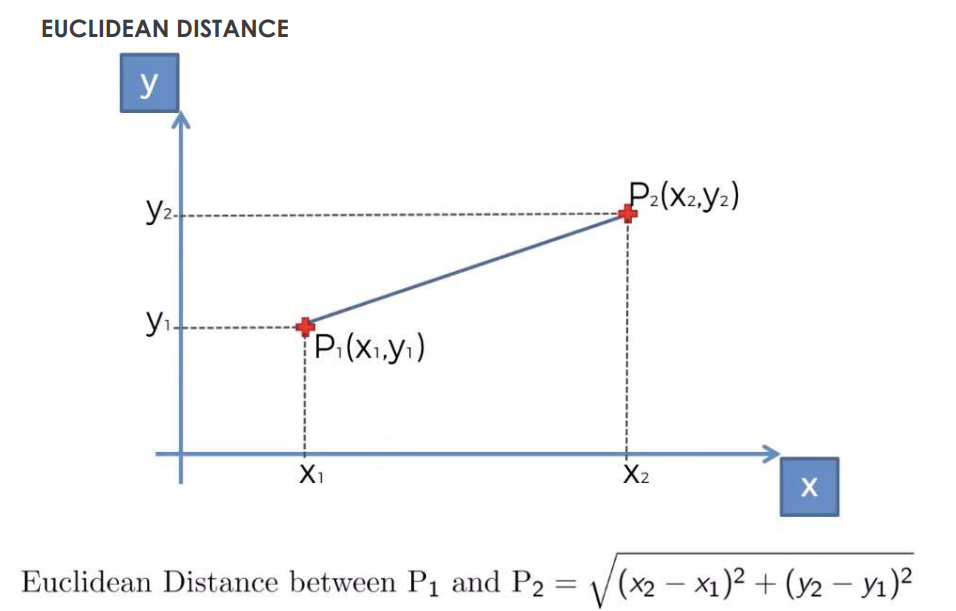


**Step 2:** Take the two closest data points and make them one cluster. (That forms N-1 clusters).



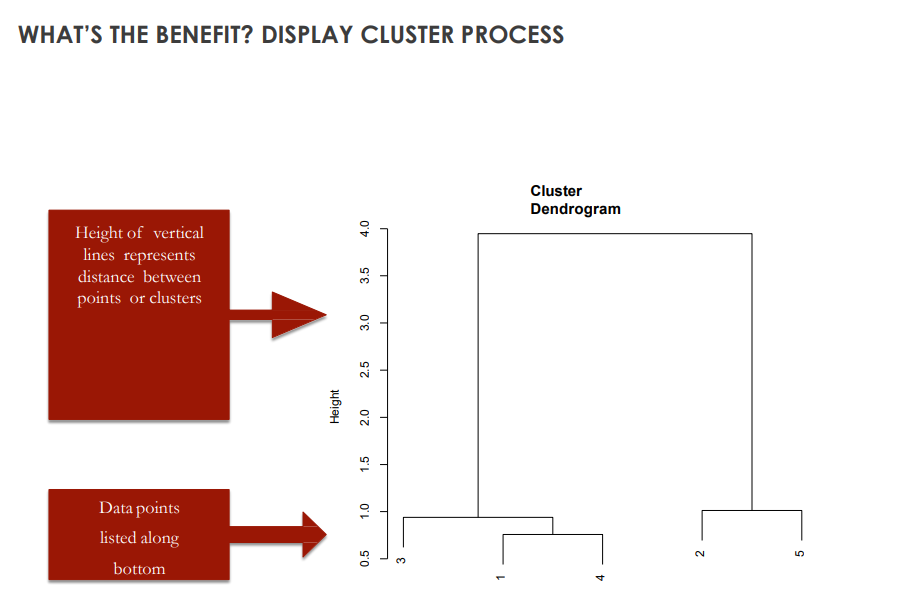
**Step 3:** Take the two closest clusters and make them one cluster. (That forms N-2 clusters). **Step 4:** Repeat step 3 until there is only one cluster.



**Minimum distance:** the minimum distance between clusters is the distance between points that are the closest

**Maximum distance:** the maximum distance between clusters is the distance between points that are the farthest.

**Centroid distance:** it is the distance between centroids of clusters. Centroid is the point that has the average of all data points in each component. (the middle)

**Week 7 Recommendation Systems**

Recommendation systems are essential for various reasons, particularly in the context of digital products and services characterized by a long tail distribution. Key benefits include:

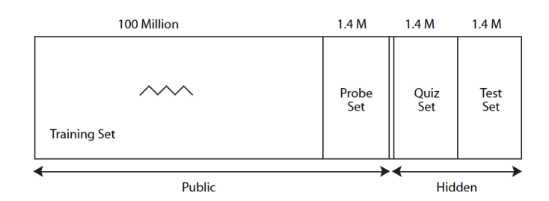
1. **Addressing the Long Tail**: They help in navigating and exploiting the long tail of diverse products or services available online.
2. **Managing Large Data Volumes**: These systems are a solution for sifting through vast amounts of data to identify high-quality content or products.
3. **Reducing Cognitive Load**: By filtering and suggesting, they lessen the cognitive burden on users who would otherwise have to sift through overwhelming choices.
4. **Enhancing User Experience**: Personalized recommendations improve the overall user experience, making it more engaging and user-friendly.
5. **Boosting Loyalty and Sales Volume**: Personalized experiences foster user loyalty and can lead to an increase in sales volume.
6. **Introducing Quality and Relevance**: They ensure that users are exposed to higher quality and more relevant choices.
7. **Aiding in Inventory Management**: These systems can be instrumental in inventory control by highlighting less visible or older items in the catalog.
8. **Broader Impact**: Overall, recommendation systems offer numerous other benefits that contribute significantly to the efficiency and effectiveness of digital platforms.

**Technology Network for Multimedia Streaming**: This refers to the infrastructure and platforms used for streaming multimedia content over the Internet. Examples include services provided by major tech companies like Google and Amazon, as well as Internet Protocol Television (IPTV) systems. This network is the backbone that delivers digital content to users, harnessing advanced technology to stream a vast array of multimedia efficiently.

**Social Network for Recommendation Systems**: This network pertains to the social aspect of recommendation systems. These systems analyze user data, preferences, and social interactions to provide personalized recommendations. An interesting dynamic is noted here: as the social network scales up, encompassing more user data and interactions, it can lead to more refined and individualized recommendations for each user. This paradox of scaling demonstrates that a larger network, while seemingly more impersonal, can enhance personalization in recommendations by leveraging the increased data and connectivity.

A **Figure of Merit** is essentially a benchmark or a set of benchmarks that are used to evaluate and compare the effectiveness of different recommendation systems or the same system over time.

* **Relevance to Tractability**: It refers to the spectrum of metrics from those that are most meaningful for the business or user experience (like customer satisfaction) to those that are most easily calculated and analyzed (like prediction error).
* **Customer Satisfaction**: This is a key metric but is subjective and can be challenging to measure directly and accurately.
* **Prediction Effectiveness**: This is the system's ability to accurately predict how much a user will like an item, which is slightly easier to measure than customer satisfaction.
* **Prediction Error**: This refers to how much the system's predictions deviate from the actual user ratings. It is more quantifiable and includes:
  + **Hamming Distance**: A count of the number of times the predicted rating is different from the actual rating.
  + **Root Mean Square Error (RMSE)**: A standard way to measure the error of a model in predicting quantitative data. The formula calculates the square root of the average squared differences between the predicted and actual ratings. Lower RMSE values indicate better predictive accuracy.
* **Top Few in Rank**: Only the highest-scoring items are recommended, which implies a focus on quality and relevance in the recommendations provided to the user.
* **Ultimate Test**: The final measure of a recommendation system's success is whether users act on the recommendations and their level of satisfaction with the recommended items. This is a holistic measure of performance that encapsulates user engagement and satisfaction.

****The "Figure of Merit" encompasses these metrics to create a comprehensive assessment of a recommendation system's quality and utility.

**Dataset**

* **Training Set**: This is the largest portion of the dataset, consisting of approximately 100 million ratings, which is publicly available. This set is used to train the recommendation algorithms.
* **Probe Set**: This is a smaller subset of approximately 1.4 million ratings and is also public. It shares similar statistical properties with the Test and Quiz sets. Competitors can use this set to test their algorithms (short for "algs" on the slide) and validate their performance before making submissions.
* **Quiz Set**: This consists of another 1.4 million ratings but is hidden from the competitors. Competitors can submit their algorithms to be run against this set to evaluate performance, but they are limited to one submission per day. This set is used for ongoing evaluation during the competition.
* **Test Set**: Also with 1.4 million ratings, this set is hidden and is used for the final evaluation of the algorithms. The RMSE scores from this set determine the final decision in the competition.
* **RMSE Scores**: The Root Mean Square Error (RMSE) scores are a measure of the accuracy of the recommendation algorithms, with lower scores indicating more accurate predictions. These scores are continuously updated on the competition's website (e.g., Netflix Prize) based on the performance on the Quiz Set.
* **Final Decision**: The ultimate winner of the competition is decided based on the comparison of RMSE scores on the Test Set.

**Collaborative Filtering Systems:** Collaborative Filtering Systems are a type of recommendation system that predict a user's interests by collecting preferences from many users. This approach assumes that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a random person. **Top of Form**

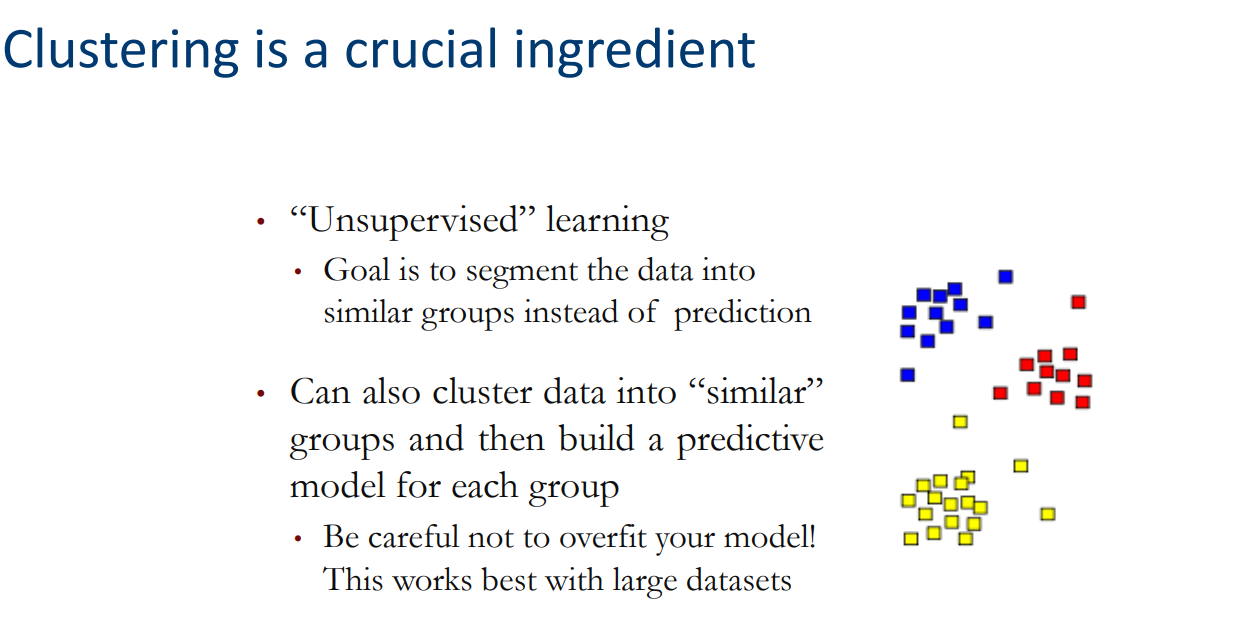
• Can accurately suggest complex items without understanding the nature of the items

• Requires a lot of data about the user to make accurate recommendations

• Millions of items – need lots of computing power

**Content Filtering:** content filtering is based on the characteristics or properties of the items themselves. This includes information like genre, author, specifications, or any other descriptive aspect of the item. It involves creating a user profile based on their preferences and interests, and then matching this profile with item attributes to make recommendations.

* Requires very little data to get started
* Can be limited in scope (recommend only similar items)



**Content Filtering**

**Week 8 AI Ethics**

There are several challenges associated with the use of algorithms, particularly in decision-making processes that impact people's lives:

1. **Lack of Transparency**: Some algorithms operate as "black boxes," meaning their decision-making processes are not transparent or interpretable to users or developers. It's unclear how these algorithms arrive at certain conclusions.
2. **Bias in Big Data**: Algorithms often rely on historical data, which may contain biases, discrimination, or prejudice. If not addressed, these biases can be perpetuated and amplified by the algorithmic decisions.
3. **Ethical Considerations**: Algorithms are increasingly involved in critical decision-making that affects individual lives, such as self-driving cars, university admissions, loan approvals, social credit scores, and the justice system. Ethical considerations arise regarding how these decisions are made and whether they are influenced by underlying biases.
4. **Need for Impartiality and Accountability**: There is a growing demand for algorithms to be impartial and for their developers to be accountable for the decisions the algorithms make.
5. **Transparency and Error Handling**: There's a need to ensure transparency in how algorithms function and a robust mechanism for coping with errors when they occur.

**Algorithmic Discrimination**: This happens when an algorithm consistently and unfairly disadvantages a particular group of people. For example, facial recognition systems have been found to have higher error rates for people of color. An image recognition algorithm misidentifying people of African descent as gorillas is a notorious example of such discrimination. Examples:

1. **Google Search ("Professor Style")**: If a search engine consistently returns images of professors fitting a narrow demographic (e.g., predominantly male and of a specific race), it may be reflecting societal biases or biases within the underlying dataset. This could perpetuate stereotypes and narrow the perception of what a professor looks like.
2. **Photo Editing Filter ("Hotness")**: Filters that alter images to increase 'hotness' or 'attractiveness' might favor certain facial features, skin tones, or body types, reflecting societal beauty standards that can be exclusionary or racially biased.
3. **Blink Detection (Nikon)**: In the past, camera blink detection algorithms have misidentified individuals of Asian descent as blinking because of their eye shape, a clear example of bias stemming from a lack of diverse training data.
4. **Face Recognition**: Facial recognition technologies have been criticized for higher error rates when identifying women and people with darker skin tones. This is often due to the algorithms being trained on datasets that are not sufficiently diverse.
5. **Job Recruiting**: Algorithms used in recruiting may favor applicants based on criteria that reflect historical hiring biases, such as preferring names that sound like they belong to a certain gender or ethnic group, or candidates from certain educational institutions.
6. **Language Translation**: Translation algorithms may default to gender stereotypes, such as associating nurses with females and engineers with males, especially when translating from or into languages with gender-neutral pronouns.
7. **Criminal Sentencing**: Algorithms used to assess the risk of reoffending have been found to exhibit racial biases, potentially affecting sentencing and parole decisions. This could result in harsher sentences for minority groups compared to others for similar offenses.
8. **Language and Translation Bias**: Translation algorithms can also reflect bias. When translating between languages, certain stereotypes or gender roles can be reinforced. For example, the translation of a gender-neutral profession into a language with gendered nouns might default to the male form, reflecting or reinforcing gender biases.

The challenge in addressing algorithmic discrimination lies in the complexity of the machine learning processes and the often opaque nature of the algorithms' decision-making. To mitigate these issues, there is a need for diverse and representative data, regular auditing for bias, algorithmic transparency, and accountability measures to ensure that AI systems perform fairly across different demographics.

**Biased Training Data**: Algorithms learn to make decisions based on the data they are trained on. If the training data is biased, the algorithm will likely perpetuate and amplify those biases. For example, if an image recognition algorithm is trained predominantly on images of women in kitchens, it might incorrectly label men in kitchens as women.

**Historical Bias**: Historical data reflects past prejudices and inequalities. For instance, job recruiting algorithms trained on historical hiring data may favor certain demographics over others simply because those groups have been hired more frequently in the past.

**Representation Bias**: This occurs when the data does not adequately represent the diversity of the real world. If a search algorithm is mostly trained on images of beauty from a narrow cultural perspective, its results for "beautiful" may exclude a wide range of appearances that are considered beautiful in other cultures.The challenge in addressing algorithmic discrimination lies in the complexity of the machine learning processes and the often opaque nature of the algorithms' decision-making. To mitigate these issues, there is a need for diverse and representative data, regular auditing for bias, algorithmic transparency, and accountability measures to ensure that AI systems perform fairly across different demographics.

**AI bias** occurs when an algorithm produces results that systematically favor certain individuals or groups over others, outside of what would be justified by the relevant facts or desired outcomes. This bias typically arises due to certain characteristics of the algorithm's development process, particularly in the data used to train the AI.

**Week 9 Reinforcement Learning**

**Reinforcement learning**

It is a type of machine learning where an agent learns to make decisions by taking actions in an environment to achieve some notion of cumulative reward.

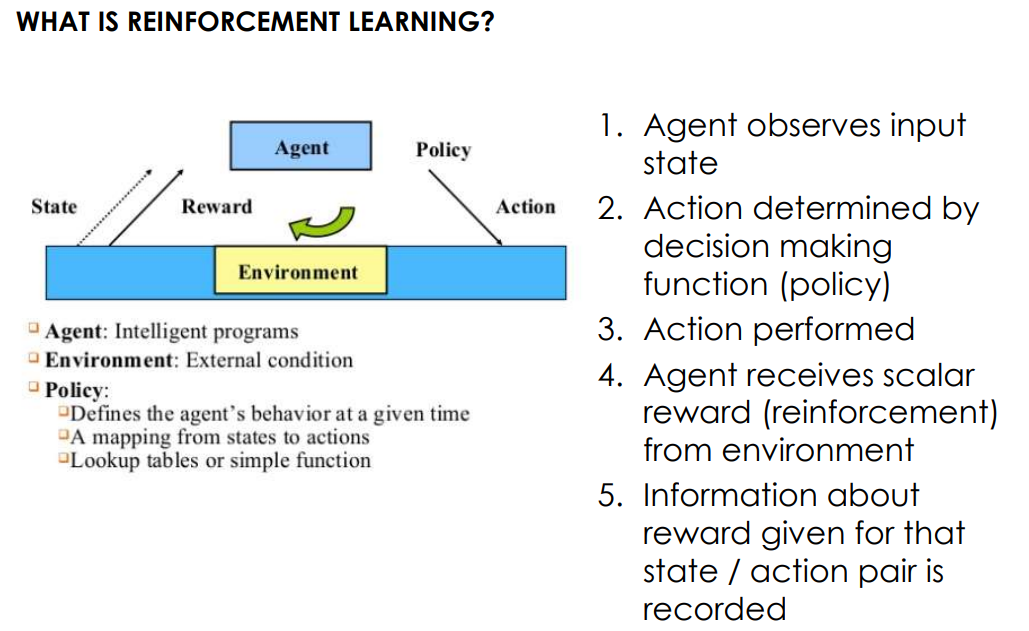
1. **Agent**: This is the learner or decision-maker.
2. **Environment**: The world through which the agent moves, which provides specific situations (states) to the agent.
3. **States**: These are the specific situations or conditions in which the agent finds itself.
4. **Actions**: Choices that the agent can make.
5. **Rewards**: Feedback from the environment, which can be positive (reinforcing the action) or negative (discouraging the action).

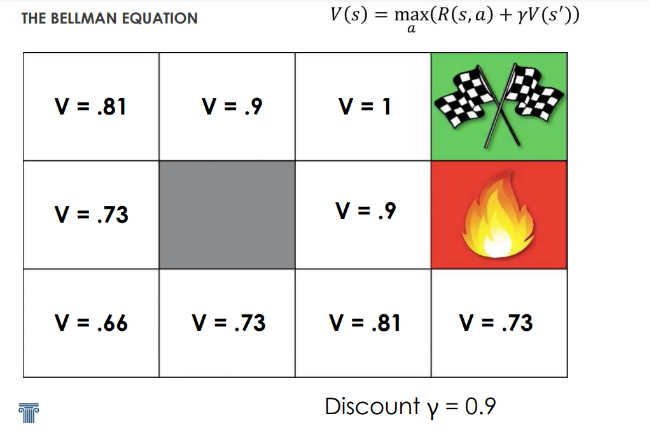
The learning process in RL involves the agent interacting with the environment and receiving rewards for its actions. Here’s how it works:

* The agent starts in a state (the situation it is in within the environment).
* From this state, it can perform various actions. The choice of action may be random at first or based on a current policy (a strategy that the agent follows).
* Upon performing an action, the agent receives a reward and transitions to a new state.
* This reward informs the agent whether the action was beneficial or not.

The goal of the agent is to maximize the cumulative reward it receives over time. To do this, it needs to learn a policy that dictates the best action to take in each state. This policy gets refined as the agent gains more experience through exploration (trying out actions to discover their rewards) and exploitation (using known information to maximize reward).

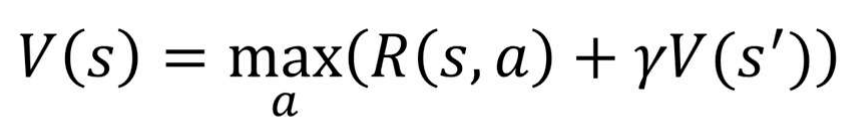
The learning happens through various algorithms, the most common one being Q-learning, where the agent learns a value function that estimates the expected cumulative reward of taking an action in a state and following the best policy thereafter.

In essence, reinforcement learning is like a game where the agent tries different strategies to accumulate the most points (rewards), and as it learns which strategies work best, it becomes better at playing the game (making decisions).

**The Bellman Equation**

**Concepts:**

* s: Store
* a: Action
* R: Reward
* Y: Discount



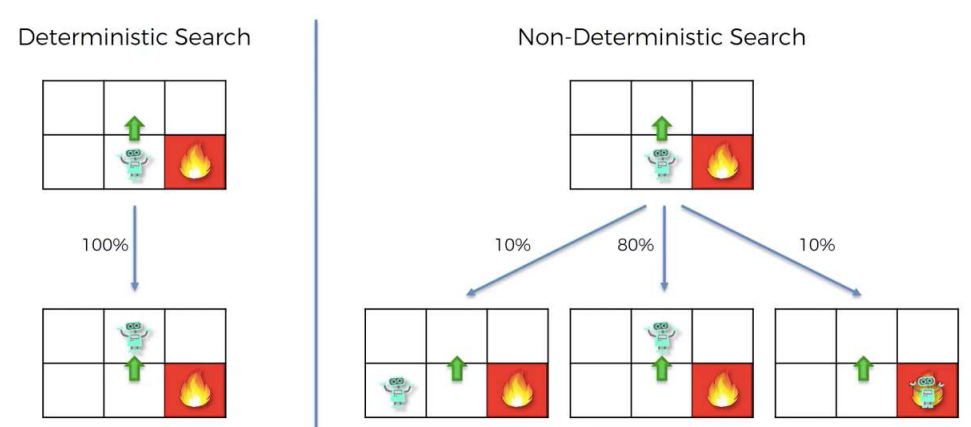
* *V*(*s*) represents the value of a state *s*, which is the expected total amount of reward an agent can expect to accumulate over the future, starting from that state.
* The max *a*​ denotes that we are taking the maximum value over all possible actions *a* from the state *s*. This is the agent's strategy to choose the action that will yield the highest expected reward.
* *R*(*s*, *a*) is the immediate reward received after taking action *a* in state *s*.
* *γ* (gamma) is the discount factor, a number between 0 and 1. This factor discounts future rewards, making them worth less than immediate rewards. It represents the trade-off between immediate and future rewards.
* *V*(*s*′) is the value of the new state ′*s*′ that the agent moves to after taking action *a*. It represents the expected reward from this next state onwards, under the best policy.

So, putting it all together, the equation says: The value of a state *s* is the maximum reward that you can achieve by taking any action *a* in that state, which is the immediate reward *R*(*s*,*a*) plus the discounted value of the state that you end up in after taking that action, *γV*(*s*′).

The Bellman equation thus provides a recursive relationship and is central to many algorithms in reinforcement learning, such as dynamic programming methods, Q-learning, and policy gradient methods. It serves as the basis for updating the value of each state in an iterative process to find the optimal policy that maximizes the cumulative reward.

**Markov Decision Processes (MDP)**An MDP provides a way to model environments for sequential decision-making where an agent must make decisions that have both predictable and unpredictable outcomes. The agent's goal is to make decisions that maximize some cumulative measure of reward over time, despite the randomness in the system's responses to its actions. This model is widely used in various fields, including robotics, economics, and artificial intelligence, particularly in areas like reinforcement learning.

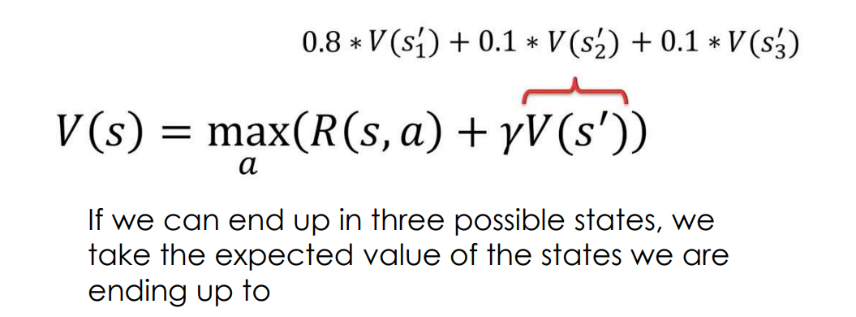
1. **Markov Property**: The Markov property states that the future state of a process depends only on the present state and not on the sequence of events that preceded it. This "memoryless" property means that given the current state, the history of how you arrived there is irrelevant for predicting the future.
2. **Deterministic vs. Non-Deterministic Search**: In a deterministic search, the outcome of each action is certain. For instance, if an agent decides to move up in a grid, it will move up with 100% certainty. In a non-deterministic or stochastic search, each action has a probability of leading to one or more different states. For example, attempting to move up might result in moving up, moving left, or moving right, each with a certain probability.



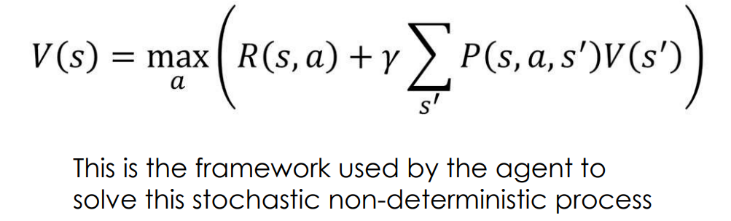
1. **MDP Framework**: An MDP is characterized by a set of states, a set of actions, a transition model (which defines the probabilities of reaching a new state after taking an action in the current state), and a reward function (which gives immediate rewards for taking certain actions in specific states). The objective in an MDP is to find a policy — a mapping from states to actions — that maximizes the expected return, which is typically the cumulative reward over time.
2. **Value Function**: The value function V(s) represents the expected return (cumulative future reward) when starting in state s and following the optimal policy thereafter. It is defined recursively as the maximum expected return for the best action a that can be taken in state s, which is the immediate reward R(s, a) plus the discounted value of the expected next state, summed over all possible next states s'. The discount factor γ (gamma) weighs the importance of future rewards versus immediate rewards.
3. **Bellman Equation**: The Bellman equation for the value function expresses the principle of optimality for MDPs. It states that the value of a state under the optimal policy equals the maximum expected return for the best action in that state. The expected return is the immediate reward plus the discounted value of the next state, considering the transition probabilities P(s, a, s') for landing in state s' after taking action a in state s.
4. **Expected Value**: When actions have stochastic outcomes, the value function takes into account the probability of each potential next state. The expected value is a weighted sum where each possible next state's value is weighted by the probability of transitioning to that state.

**Markov property:**

1. **Memoryless Property**: In a Markov process, the past is irrelevant for predicting the future, as long as you know the current state. This property implies that the history of the process does not provide additional information beyond what is already contained in the current state.
2. **Conditional Probability Distribution**: The probability of transitioning to any future state from the current state can be determined without any knowledge of the past states. The conditional probability of the future state is based solely on the present state.
3. **Dependence on the Present State**: The next state of the process (or the probability distribution of the next state) is determined entirely by the current state. The Markov property simplifies the analysis of stochastic processes since one does not need to consider the complete path taken to reach the current state.
4. **Markov Process**: The way the environment is designed (what happens now does not depend on the past).

**Deterministic Formula**:

This is a simplified version often used in deterministic contexts where each action *a* in state *s* leads to one specific next state ′*s*′ with certainty (hence no summation over next states). The agent looks for the action that yields the highest sum of immediate reward *R*(*s*,*a*) and the discounted value of the resulting next state *V*(*s*′).

**Stochastic Formula**:

This formula calculates the value *V*(*s*) of a state *s* as the maximum reward obtainable by performing any action *a*, plus the sum of the discounted values of all possible next states ′*s*′. The discount factor *γ* weighs the importance of future rewards. *P*(*s*,*a*,*s*′) is the probability of reaching a new state ′*s*′ from the current state *s* by taking action *a*, and *V*(*s*′) is the value of the next state.

Both equations aim to find the action that maximizes the expected return, taking into account both the immediate reward and the potential future rewards. The **stochastic** formula typically leads to a policy that's better suited for environments where outcomes are uncertain because it incorporates a probabilistic understanding of the environment into the decision-making process. In contrast, the **deterministic** formula assumes that the outcome of each action is known and certain, which is not generally the case in real-world scenarios.

We’re in the world of **stochastic** (nondeterministic) search, where we are accounting for random factors!!

**Policy**: A policy is a comprehensive strategy that specifies the best action to take in every possible state of the system. It's like a rulebook that says, "If you find yourself in this situation (state), here's what you should do (action)." Policies are crucial in stochastic environments where uncertainty is involved because they provide a way to handle any situation that might arise. In reinforcement learning and Markov Decision Processes (MDPs), policies are used to achieve long-term goals by maximizing some notion of cumulative reward.

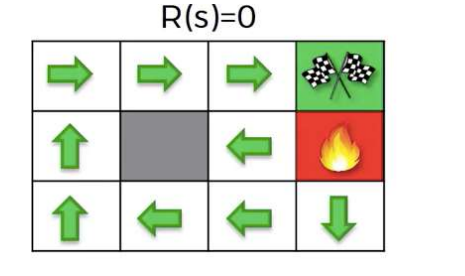
**Plan**: A plan, on the other hand, is a predetermined sequence of actions that lead from an initial state to a goal state. It's specific, linear, and does not account for deviations or changes in the environment. Plans are typically used in deterministic settings where the outcome of actions is known. If an unexpected state is encountered, the plan might need to be reevaluated or entirely reconstructed.

**Living Penalty**:

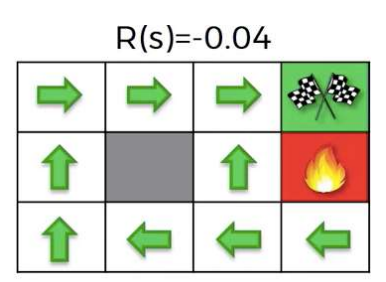
* The concept of "living penalty" (sometimes called "living reward" when it's positive) refers to the reward *R*(*s*,*a*) received immediately after taking an action *a* in state *s*. It can be a penalty (negative reward) to encourage the agent to reach the goal faster. For instance, in a maze, the agent might receive a small penalty at each time step until it reaches the goal to prevent it from wandering indefinitely.

**Rewards During the Journey**:

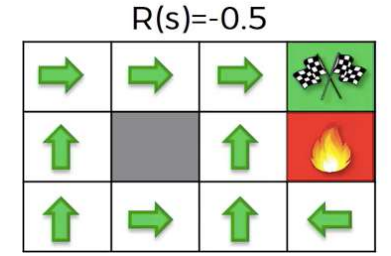
* Traditionally, rewards in games or simulations are given at the end (e.g., winning a game). However, the Bellman equation allows for rewards (or penalties) to be assigned at every step of the process, not just at the end. This incentivizes the agent to not only consider the ultimate goal but also to navigate intermediate states optimally.

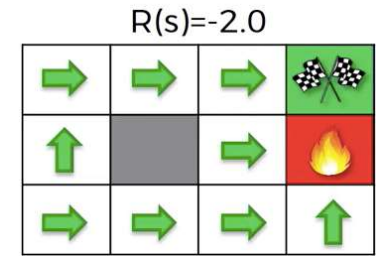
This approach enables more nuanced strategies where the agent must balance the cost of living (penalty for each step taken) against the benefits of exploration and the eventual rewards for reaching specific targets or goals. It's a way to shape the agent's behavior and guide it toward more efficient and effective policies.

**(R(s) = 0)**: This grid represents an environment where there is no living penalty for each step the agent takes. The arrows may represent the optimal path or policy to reach the goal (checkered flag) without any penalty for moving. The agent has no incentive to reach the goal quickly or to avoid non-goal states.



**Top-right grid (R(s) = -0.04)**: Here, each non-goal state has a small living penalty (negative reward) of -0.04. This encourages the agent to reach the goal quickly to avoid accumulating penalties.



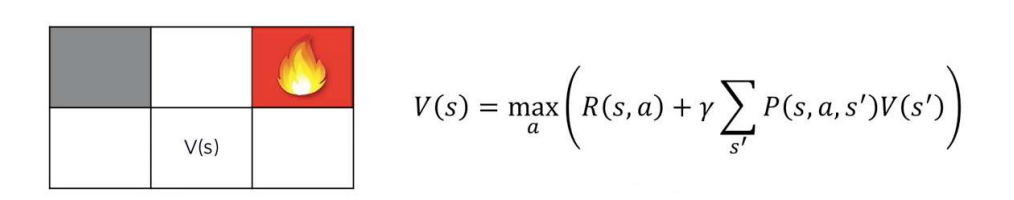
**Bottom-left grid (R(s) = -0.5)**: The living penalty is increased to -0.5 for each non-goal state. This larger penalty provides a stronger incentive for the agent to find the shortest possible path to the goal, as any detour will result in a significantly larger negative reward.

**Bottom-right grid (R(s) = -2.0)**: With an even more severe living penalty of -2.0 for each non-goal state, the agent is heavily penalized for each move that does not lead directly to the goal. This setting makes the agent prioritize the fastest route to the goal above all else to minimize the negative rewards.

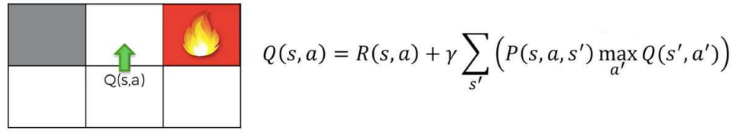
**Q-Learning**

Q-Learning is a model-free reinforcement learning algorithm used to find the optimal action-selection policy for any given finite Markov decision process (MDP). It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter.

1. The equation without Q(s,a) in it represents the value function. It maximizes over all possible actions the sum of immediate rewards R(s,a) plus the discounted value of the state resulting from that action.



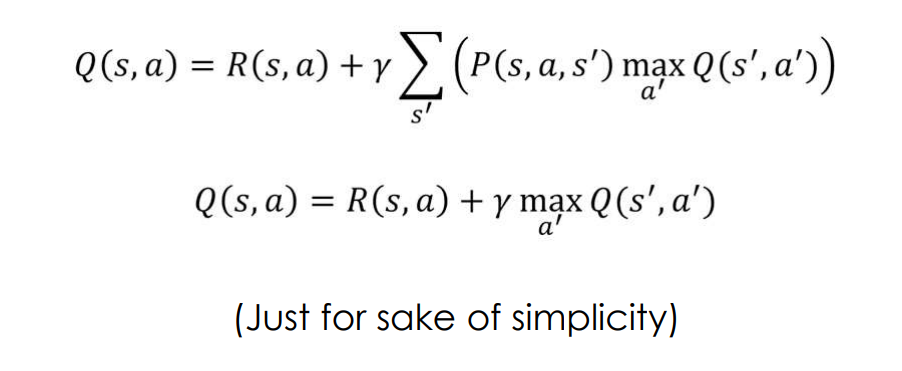
1. The second equation with Q(s,a) represents the action-value function for Q-learning. Instead of just taking the max over actions, it calculates the expected value if we take action a in state s. It's considering the immediate reward plus the discounted maximum future reward that can be obtained from the next state s' we end up in after taking action a.

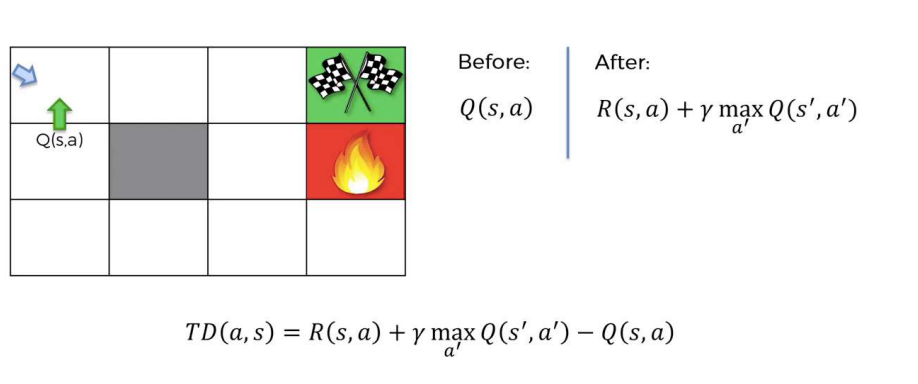


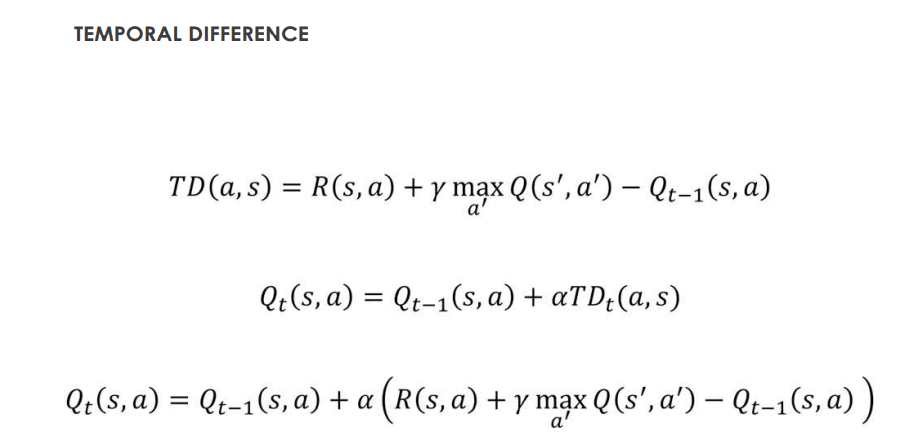
**Temporal Difference**

**Deterministic Search:** In a deterministic search, the outcome is predictable and consistent every time the search is conducted given the same input or state. There is no element of randomness or probability in the process. When an algorithm is deterministic, it means that its behavior can be determined from its initial state and inputs, and it will always perform the same way and produce the same output when given the same initial conditions.

**Non-Deterministic Search:** Non-deterministic search involves randomness or probability. The outcome is not predictable with certainty, and different runs of the algorithm may yield different outcomes, even with the same input. In such searches, there may be several possible next steps from any given state, and the algorithm may include a random choice among these next steps.



****



**Week 10 Artificial Neural Networks (ANN)**

**Neural network**

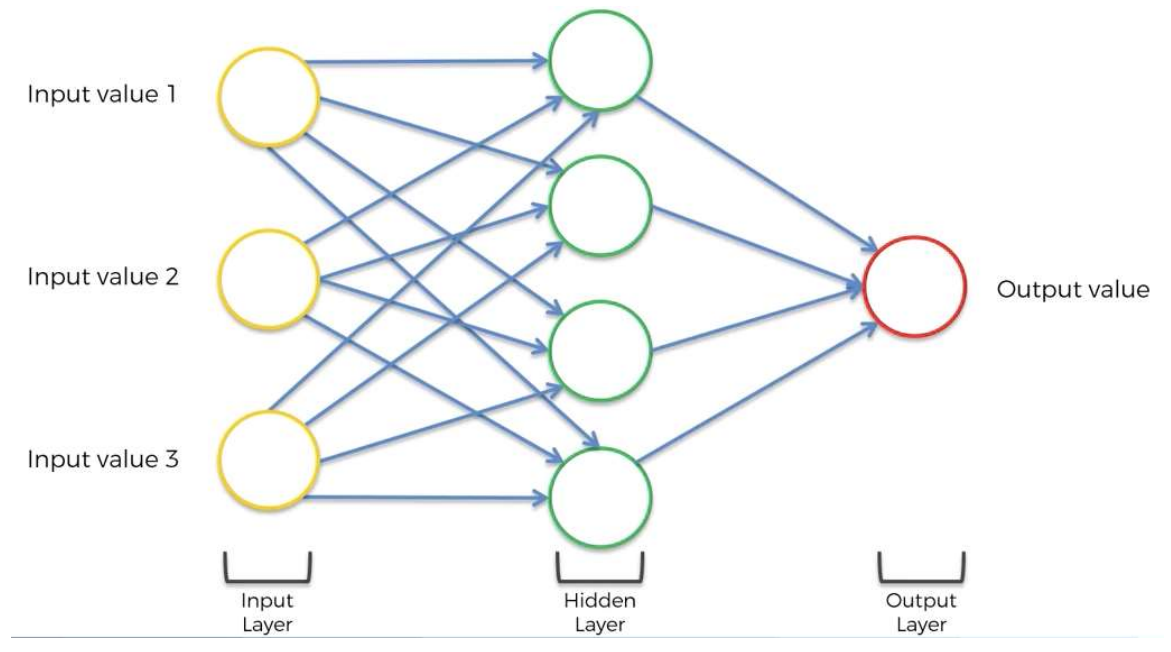
A neural network is a computational system inspired by the structure, processing method, and learning ability of the human brain. It consists of interconnected units or nodes called artificial neurons, which are modeled after biological neurons. These artificial neurons are organized in layers:

1. **Input Layer:** Receives the initial data for processing, much like sensory input in biological systems.
2. **Hidden Layers:** Intermediate layers of neurons that apply computations through their interconnected neurons to transform the input into something that the output layer can use. There can be one or multiple hidden layers, and the "deep" in deep learning refers to networks with many hidden layers.
3. **Output Layer:** Delivers the final output produced by the network, such as a classification or prediction.

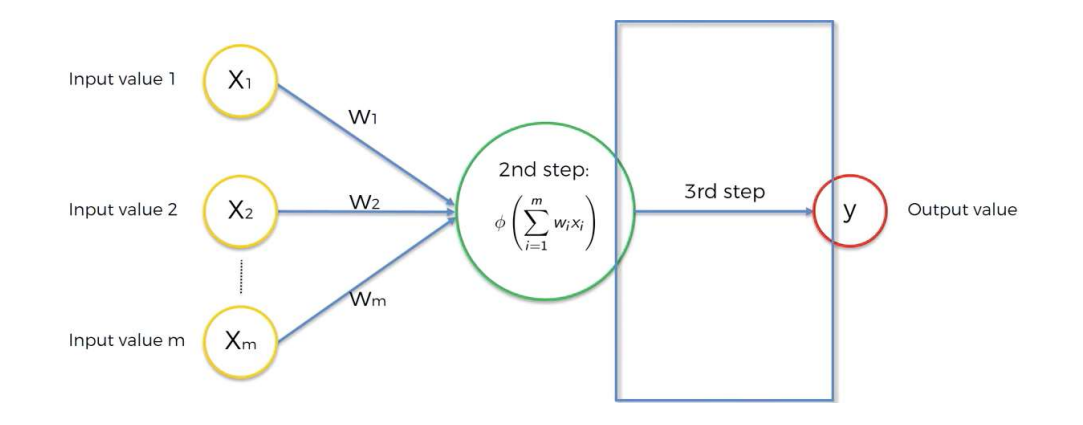
Each connection between neurons has an associated weight, which is adjusted during the training process. The neurons apply a weighted sum of their inputs and pass this through an activation function that determines whether and to what extent the signal should affect the next layer.

The learning process involves adjusting these weights based on the error of the output compared to the expected result, a process known as backpropagation combined with an optimization algorithm like gradient descent.

Neural networks are used in machine learning applications where traditional linear algorithms fall short, particularly in complex tasks such as image recognition, speech recognition, and natural language processing. They have the ability to model and process non-linear relationships between inputs and outputs in parallel, making them highly flexible and powerful for a wide range of tasks.

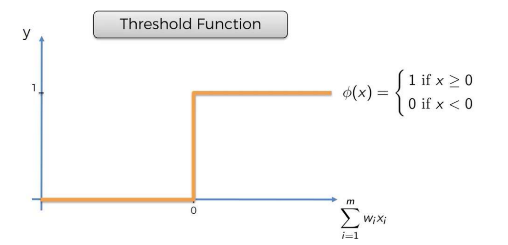


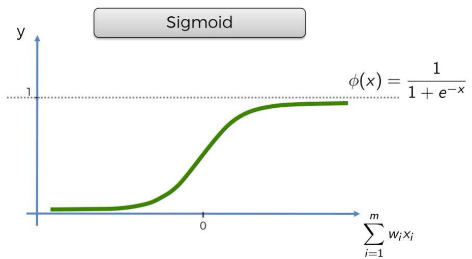
**The Neuron**

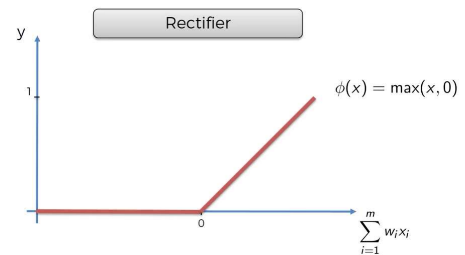


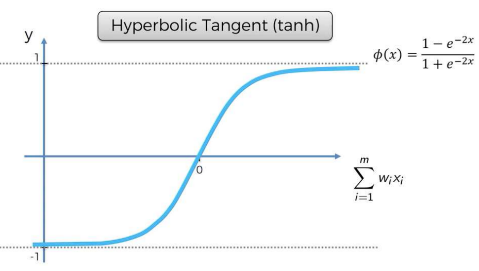
1. **Input Weighting**: Each input value *Xi*​ is multiplied by a corresponding weight *Wi*​. These weights are parameters that the neural network learns during the training process. Each weight signifies the importance of its corresponding input to the neuron's output.
2. **Summation and Activation Function**: The products of the weights and inputs are summed together to form a weighted sum, ∑*i*=1*m*​*Wi*​*Xi*​, where *m* is the number of inputs. This sum is then passed through an activation function *ϕ*, which is a non-linear function that maps the input signals to an output signal. The activation function is crucial for the network's ability to handle non-linear problems.
3. **Output Generation**: The result of the activation function is the neuron's output value *y*. This output can then be used as an input to subsequent layers in a neural network, or it can be the final output of the network if this is an output layer neuron.

**The Activation Function**

**Threshold Activation Function:**  
The threshold activation function is a type of activation function where the output is binary, based on whether the input value is above or below a certain threshold. If the input (sum of weighted inputs) is greater than or equal to zero, the output is 1; otherwise, the output is 0.

**Sigmoid Activation Function:**  
The sigmoid activation function produces an S-shaped curve. This function outputs a value between 0 and 1, which is useful for problems where we need to predict probabilities.

**Rectifier Activation Function:**  
An activation function, also known as the Rectified Linear Unit (ReLU) outputs the input directly if it is positive, otherwise, it outputs zero.

**Hyperbolic Tangent Activation Function:**  
The hyperbolic tangent (tanh) activation function outputs values in a range between -1 and 1. It is similar to the sigmoid but can output negative values as well, making it useful in situations where the model needs to predict two classes that are not only different but opposite.

the choice of activation functions when dealing with a neural network that outputs a binary dependent variable. In the context of neural networks, a binary dependent variable means that the output is either 0 or 1. This is common in classification tasks where, for instance, an email is classified as either spam (1) or not spam (0).

1. **Threshold Activation Function**: This function is a step function that assigns a 1 if the input (the weighted sum of inputs, ∑*i*=1*m*​*wi*​*xi*​) is greater than or equal to 0, and assigns a 0 if the input is less than 0. It's a very simple and clear-cut way to make a binary decision.
2. **Sigmoid Activation Function**: Unlike the threshold function, which has a sharp transition, the sigmoid function smoothly increases from 0 to 1 and thus provides a probability-like output. For any given input to the function, the sigmoid will output a value between 0 and 1, which can be interpreted as the probability of the input belonging to a particular class. For example, a sigmoid output of 0.8 could be interpreted as an 80% chance of the input being classified as 1 (or "true" in a true/false scenario).

The key difference between the two is that the threshold function gives a hard classification, while the sigmoid function provides a soft classification that can be interpreted as a probability. The choice between these two functions would depend on whether you need a hard decision boundary (use threshold) or if you want to interpret the outputs as probabilities (use sigmoid).

**How do neural networks work?**  
Neural networks are a series of algorithms that attempt to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Here is a simplified overview of how they work:

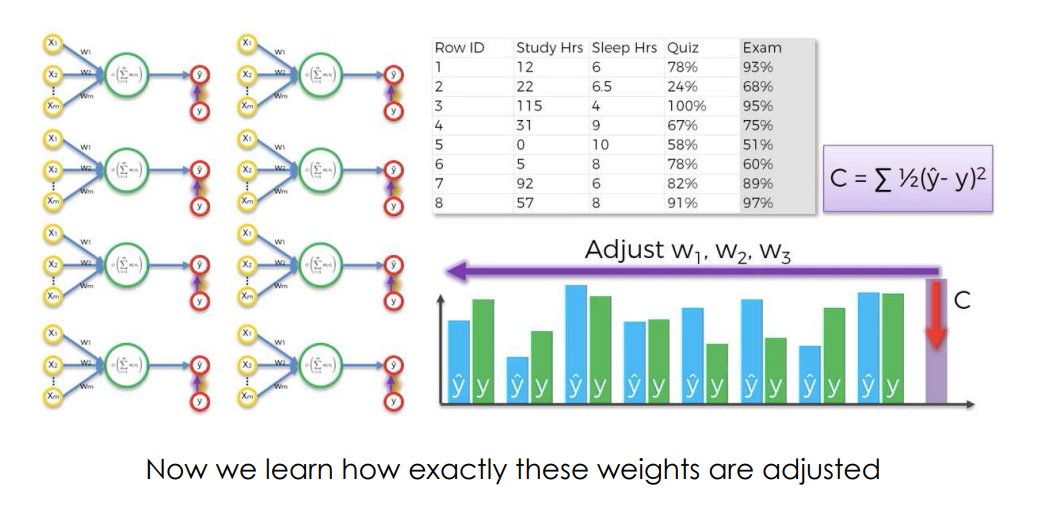
1. **Structure of a Neural Network**:
   * **Input Layer**: The first layer receives the input features. It's directly fed from the data you're trying to model or classify.
   * **Hidden Layers**: These layers perform computations using weighted inputs from the previous layer and send their output to the next layer. The complexity of the network often depends on the number of hidden layers and neurons within them.
   * **Output Layer**: The final layer produces the predictions or classifications of the network.
2. **Forward Propagation**:
   * Each neuron in a layer receives input from the neurons of the previous layer. This input is a sum that is weighted by the importance the network currently assigns to this input.
   * Each neuron applies an activation function to this sum. The choice of activation function depends on the nature of the data and the desired output (e.g., ReLU, sigmoid, tanh).
   * The output of each neuron is then passed on to the next layer.
3. **Loss Function**:
   * After forward propagation, the network produces an output. This output is evaluated using a loss function, which measures the difference between the network's prediction and the actual data.
   * The goal of training a neural network is to minimize this loss function.
4. **Backpropagation**:
   * Backpropagation is the key to learning in a neural network. It involves calculating the gradient of the loss function with respect to each weight by the chain rule, moving backwards from the output layer to the input layer.
   * Backpropagation provides a way for the network to adjust its weights to minimize the loss by making small changes in the direction that most reduces the error.
5. **Learning Rate and Optimization**:
   * The learning rate is a hyperparameter that controls how much the weights of the network are adjusted with respect to the loss gradient. Too high a learning rate might overshoot the minimum, and too low might take too long to converge or get stuck in a local minimum.
   * An optimization algorithm, like stochastic gradient descent (SGD) or Adam, is used to update the weights iteratively in the direction that reduces the loss.
6. **Iteration and Convergence**:
   * The process of forward propagation, loss calculation, backpropagation, and weight adjustment is repeated many times across multiple epochs, which are full passes through the entire dataset.
   * Over time, the network weights converge to values that minimize the loss function, ideally to the point where the network's predictions are accurate or the changes in loss are negligibly small.
7. **Generalization**:
   * After training, the neural network should be able to generalize from the training data to make accurate predictions on new, unseen data. This ability is crucial for the network to be useful in practical applications.

Neural networks can be very complex, with many layers (deep learning) and numerous neurons, leading to a vast number of weights that need to be trained. The actual functioning of a neural network can get much more mathematically complex, especially for different types of neural networks (convolutional, recurrent, etc.), but the above steps give a high-level understanding of the basic workflow.

**How do Neural Networks learn?**

1. **Input and Weight Assignment**: Neural networks receive inputs (X1, X2, ..., Xm), and each of these inputs is associated with a weight (W1, W2, ..., Wm). These weights are parameters that the neural network will learn to adjust during training to make accurate predictions.
2. **Weighted Sum and Activation**: Inputs are multiplied by their respective weights and summed together to form a weighted sum ∑*wi*​*xi*​). This sum is then passed through an activation function (denoted as ϕ), such as sigmoid or ReLU, to produce a non-linear output, which is necessary for the network to model complex patterns.
3. **Prediction and Actual Value**: The output from the activation function is the predicted value (^*y*^​) of the network. This is then compared with the actual value (y) to assess the network's performance.
4. **Cost Function**: The difference between the predicted value (^*y*^​) and the actual value (y) is measured using a cost function, often the mean squared error 2*C*=21​(*y*^​−*y*)2). The cost function quantifies the error of the network’s predictions.
5. **Backpropagation**: This is a key step where the neural network learns from its errors. The gradient of the cost function with respect to each weight is computed, and the weights are updated accordingly. The update typically involves subtracting a fraction of the gradient from the weight, which is determined by the learning rate.
6. **Learning Over Multiple Rows**: When the training involves multiple data points, the network repeats the prediction and update process for each instance. The weights are adjusted based on the cumulative error across all training instances.
7. **Iteration**: The network undergoes multiple iterations or epochs, where it processes the entire dataset, makes predictions, calculates error, and updates weights.
8. **Convergence**: The process of iteration continues until the network’s performance no longer significantly improves, indicating that it has learned to approximate the function that maps inputs to outputs as closely as possible given the network architecture and data. This is often determined by a small change in the cost function between epochs or based on performance on a validation dataset.
9. **Dealing with Multiple Rows**: In practice, neural networks deal with multiple data points simultaneously. This is shown in the last image where the network is trained on multiple rows of data. Each row represents a different instance with various input features like "Study Hours", "Sleep Hours", "Quiz Score", and the target variable "Exam Score".
10. **Repeating the Process**: The final image indicates that the learning process is repeated multiple times, like with a single row, but scaled up for the entire dataset. The goal is to minimize the cost function across all instances. This iterative process of forward propagation (making predictions), calculating the cost, and then using backpropagation to update the weights is repeated for many epochs.

**Gradient Descent**



Gradient descent is a first-order iterative optimization algorithm used to find the minimum of a function. Here's a detailed explanation, especially in the context of training neural networks, which seems to be the focus of your upcoming exam:

**Understanding Gradient Descent:**

1. **The Objective**: The goal of gradient descent is to minimize a cost function. In the context of neural networks, the cost function measures the difference between the network’s predictions and the actual targets.
2. **The Cost Function**: Typically, a cost function like Mean Squared Error (MSE) 2*C*=21​(*y*^​−*y*)2 is used, where ^*y*^​ is the predicted output from the network, and *y* is the actual value. The factor 1221​ is often included to simplify the derivative.
3. **The Gradient**: The gradient is a vector that contains all the partial derivatives of the cost function with respect to each weight in the network. It points in the direction of the steepest ascent on the cost function surface.
4. **Descent**: To minimize the cost, we need to go in the opposite direction of the gradient because we are looking for the lowest point (minimum) of the cost function.

**The Process:**

1. **Initialization**: We start with random weights and calculate the cost.
2. **Compute the Gradient**: Calculate how much each weight contributes to the error by finding the partial derivatives of the cost function with respect to each weight.
3. **Update the Weights**: Adjust the weights in the opposite direction of the gradient. This is where the term "descent" comes in. The update rule is: *Wnew*​=*Wold*​−*α*×∇*C* Here, *Wold*​ are the current weights, *α* is the learning rate, and ∇*C* is the gradient of the cost function.
4. **Learning Rate**: The learning rate *α* determines the size of the steps we take towards the minimum. If it’s too large, we may overshoot the minimum; if it's too small, the descent may be very slow or get stuck in a local minimum.
5. **Iteration**: Repeat the process of calculating the gradient and updating the weights until the cost function stops decreasing significantly.

**Example of Gradient Descent:**

Imagine you're training a neural network to predict exam scores based on hours studied (X1) and hours slept (X2). The initial random weights are W1 and W2. You choose MSE as your cost function.

* After the first forward pass, you get a predicted score ^*y*^​ that is different from the actual score *y*.
* You calculate the cost *C* using MSE.
* Then, compute the gradient of *C* with respect to W1 and W2.
* Suppose the gradient vector is ∇*C*=[2,3], and your learning rate *α* is 0.01.
* The weights update would be:
  + *W*1*new*​=*W*1*old*​−0.01×2
  + *W*2*new*​=*W*2*old*​−0.01×3
* This adjustment is made to reduce the cost, and the process is iterated through the entire training dataset.

**Visualizing Gradient Descent:**

The images you've provided show the cost function in various forms. Gradient descent can be visualized as a ball rolling down the surface of the cost function to the lowest point. Each step taken by the ball is analogous to an iteration in gradient descent, where the ball "steps" in the direction that leads most directly downhill.

In the context of your exam, understanding gradient descent involves recognizing it as an optimization algorithm that allows neural networks to adjust their weights based on the calculated gradient to reduce the prediction error. The learning rate and the calculation of the gradient are crucial aspects that govern how effectively and quickly the network learns.

**Stochastic Gradient Descent**

Stochastic Gradient Descent (SGD) is a variant of the gradient descent optimization algorithm that is used for training a wide variety of machine learning models, especially large-scale neural networks. Here's a detailed explanation:

**Understanding Stochastic Gradient Descent (SGD):**

1. **Stochastic Nature**: Unlike standard gradient descent, which uses the entire dataset to compute the gradient of the cost function at each step (batch gradient descent), SGD updates the model's weights using only a single sample or a small batch of samples, chosen randomly.
2. **Objective**: The goal of SGD remains the same as gradient descent: to minimize the cost function. However, it aims to do this more efficiently, especially when dealing with large datasets.
3. **The Cost Function**: The cost function in SGD is computed for each training example or small batch, rather than the entire dataset. If the cost function is MSE, then for a single training example it would be 2*C*=21​(*y*^​−*y*)2.
4. **Gradient Computation**: The gradient of the cost function with respect to the weights is computed using only the current sample or batch. This is a noisy estimate of the gradient over the entire dataset, but it is computationally much faster.
5. **Weights Update**: Weights are updated incrementally after evaluating the cost function for each sample or batch, which allows for much faster convergence, especially when the dataset is large.

**The Process:**

1. **Initialization**: The weights are initialized randomly.
2. **Random Sample or Batch**: At each iteration, a sample or a small batch of samples is chosen randomly from the dataset.
3. **Compute the Gradient**: The gradient is computed using just the chosen sample(s), which means that the gradient calculation is much faster.
4. **Update the Weights**: The weights are updated using this stochastic gradient: *Wnew*​=*Wold*​−*α*×∇*C* Here, *Wold*​ are the current weights, �*α* is the learning rate, and ∇*C* is the gradient computed on the current sample or batch.
5. **Repeat**: This process is repeated for many iterations, where each iteration may involve a different random sample or batch.

**Example of Stochastic Gradient Descent:**

Let's continue with the neural network example for predicting exam scores based on hours studied and hours slept. In SGD, instead of using all student data to update weights, you use data from just one student at a time.

* You pick a student's data randomly and make a prediction *y*^​.
* Compute the cost *C* for that single example.
* Calculate the gradient of *C* with respect to W1 and W2 for that student's data.
* If the gradient vector is ∇*C*=[2,3] and your learning rate *α* is 0.01, you update W1 and W2 as before but based only on this single example.
* In the next iteration, you choose another random student's data and repeat the process.

**Benefits of SGD:**

* **Efficiency**: SGD is much more computationally efficient than batch gradient descent, especially for large datasets.
* **Frequent Updates**: Frequent updates with a high variance cause the objective function to fluctuate heavily, which can help to escape local minima.
* **Large Datasets**: It is well-suited for problems with a very large number of training examples where holding the entire dataset in memory would be impractical.

**Considerations for SGD:**

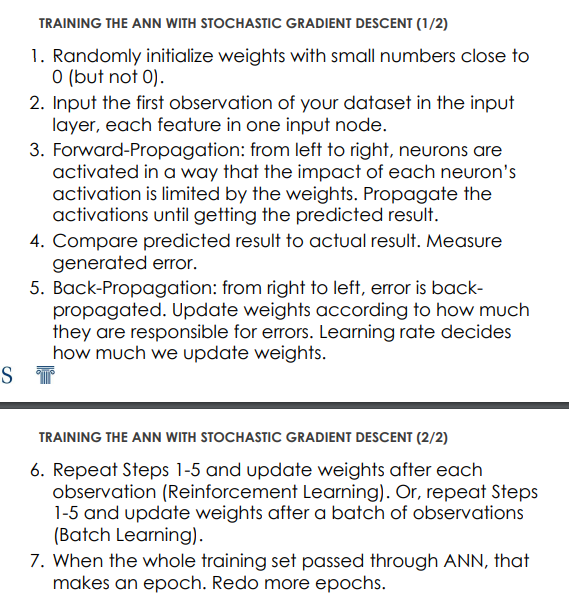
* **Learning Rate**: Choosing an appropriate learning rate can be more critical for SGD since the gradient estimate is noisier than in batch gradient descent.
* **Convergence**: While the path to convergence can be noisier for SGD, it often converges faster in practice for large datasets.
* **Mini-Batch SGD**: Often a compromise between batch gradient descent and stochastic is used, called mini-batch SGD, where the gradient is computed on a small batch of data.

**Forward Propagation**

Forward propagation is the process of moving inputs through the neural network to obtain an output.

**Backpropagation**

Backpropagation is the process of adjusting the network's weights and biases based on the error of the output.



**Week 11 Convolutional Neural Networks (CNN)**

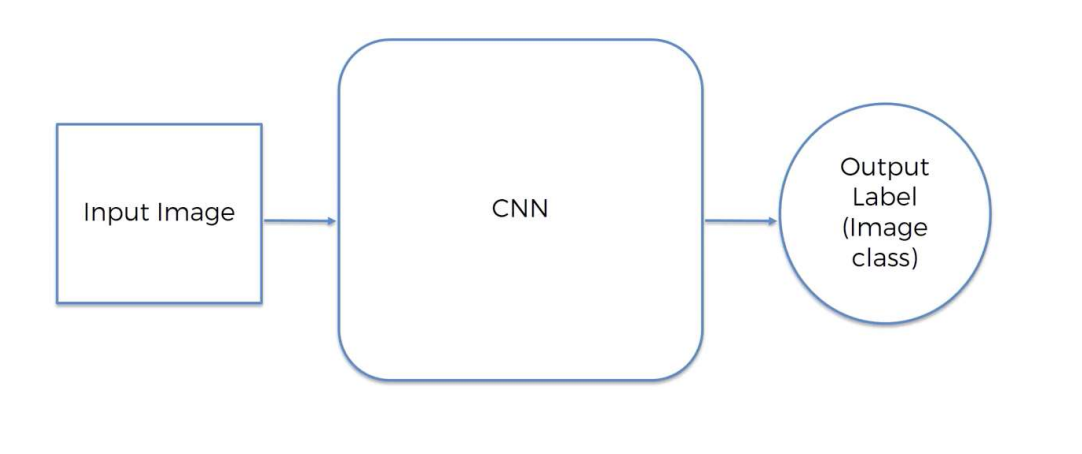
**What are Convolutional Neural Networks?**

Convolutional Neural Networks (CNNs) are a type of artificial neural network that's particularly effective for analyzing visual imagery. They're designed to automatically and adaptively learn spatial hierarchies of features from input images.

Imagine if you had a special pair of glasses that could help you recognize and understand pictures. These glasses could look at a photo and pick out all the important parts, like finding where the dogs are, identifying all the balls, or telling you where the trees stand. Convolutional Neural Networks are like those smart glasses for computers. They help computers see and understand pictures by learning about the important parts and paying attention to them.

1. **Eyes for Computers**: CNNs are like eyes for computers, giving them the power to see and recognize images.
2. **Filters**: They use filters to scan through an image and pick up patterns like lines, shapes, and colors, just like when you're looking for a specific object in a Where's Waldo picture.
3. **Layer by Layer**: They stack layers of filters on top of each other. The first layer might find simple things like edges, and the layers after that combine these simple things to find more complicated stuff, like how different shapes make up a bicycle.
4. **Remembering**: Once a CNN has learned what different things look like in pictures, it can recognize those things in new pictures it's never seen before, just like you can recognize your friend in a crowd because you remember what they look like.

CNNs are great for tasks where recognizing shapes, sizes, and textures is important, which is why they're so popular for things like self-driving cars, face recognition, and reading handwritten letters and numbers.



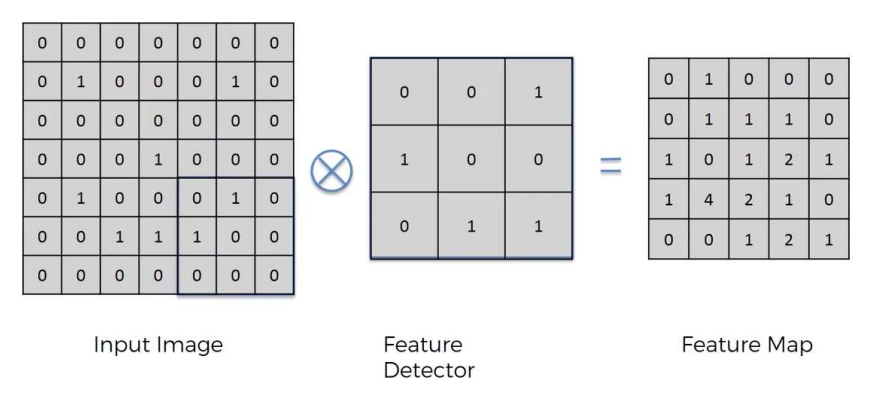
• Step 1 – **Convolution Operation**

**Purpose:** The main goal of the convolution operation is to extract features from the input image. It helps in detecting edges, textures, gradients, and other important visual aspects that are essential for understanding an image.

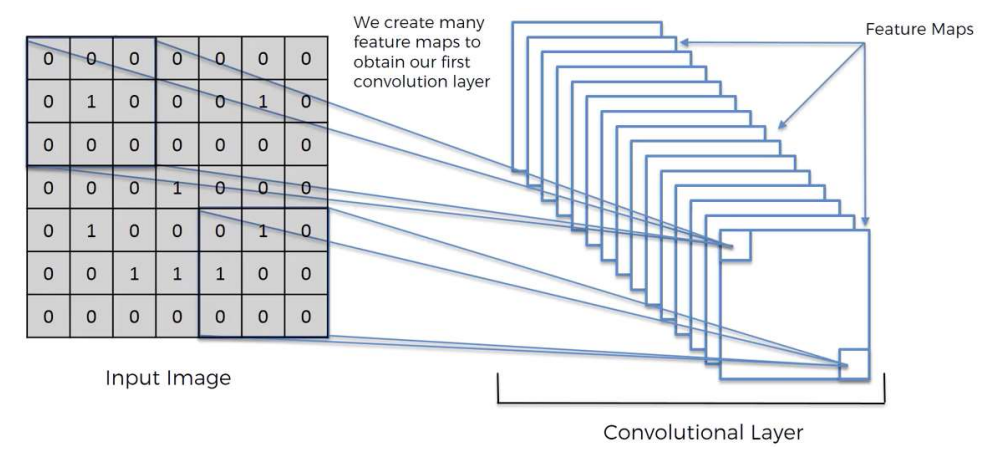
**Components: Input Image:** This is the raw image that you input into the network. It could be in grayscale or color (RGB) **Filter (or Kernel):** A small matrix used to detect specific features like edges, lines, etc. This filter is slid over the input image.

**Process of Convolution: Sliding the Filter:** The filter is applied to the input image by sliding it across the image, one pixel at a time. **Element-wise Multiplication:** For each position of the filter on the image, an element-wise multiplication is performed between the values in the filter and the corresponding values in the image. **Summation:** After the multiplication, the products are summed up to get a single number. This sum forms one element in the output feature map. **Stride:** This refers to the number of pixels by which we slide the filter across the image. A stride of 1 moves the filter one pixel at a time.

**Feature Map: Output:** The result of the convolution operation is a feature map (or activation map), which represents certain features detected in the image. The dimensions of this feature map depend on the size of the filter, the stride, and the padding (if used).

****

**Role in Feature Detection: Edge Detection:** In the initial layers, convolutional filters usually detect simple features like edges or colors. **Complex Features:** As we progress deeper into the network, these convolutional layers start to combine simple features to detect more complex patterns.

* The convolution operation is crucial because it allows CNNs to make sense of the visual information in an image by breaking it down into features that can be analyzed and used for tasks like image classification, object detection, and more. 
* **ReLU Layer:** The ReLU function is defined as *f*(*x*)=*max*(0,*x*). In simple terms, it replaces all negative pixel values in the feature map with zero. The output is the same as the input if the input is positive, but it's zero if the input is negative. **Main reason**: Increase non-linearity in our CNN (images are highly non-linear**)**

• Step 2 – **Max Pooling**

Max Pooling is the second step often used in Convolutional Neural Networks (CNNs) after the convolution operation and the application of an activation function like ReLU. Here's an explanation of Max Pooling and its role in CNNs:

**1. Purpose of Max Pooling:**

* **Dimensionality Reduction:** Max Pooling reduces the dimensions (height and width) of the input feature maps, making the computations more manageable.
* **Feature Extraction:** It extracts the most prominent features from the feature map (like edges, textures), while discarding less useful information.

**2. How Max Pooling Works:**

* **Pooling Window:** A window (often 2x2 size) slides over the input feature map.
* **Stride:** The window moves by a certain number of pixels (called the stride, usually the same size as the window) across the feature map.
* **Maximum Value Extraction:** In each position, the maximum value within the window is selected and forms a new, reduced feature map.



**3. Benefits of Max Pooling:**

* **Reduces Overfitting:** By downsampling the feature maps, max pooling helps in reducing overfitting by providing an abstracted form of the features. Reducing size is important, as we are also reducing the parameters and thus preventing overfitting (removing information is a good thing!)
* **Invariance to Small Translations:** Max pooling provides the network with a degree of invariance to small translations, rotations, and scalings.
* **Reduces Computational Load:** By reducing the size of the feature maps, it decreases the number of parameters and computations, leading to faster processing.

**4. Max Pooling vs. Average Pooling:**

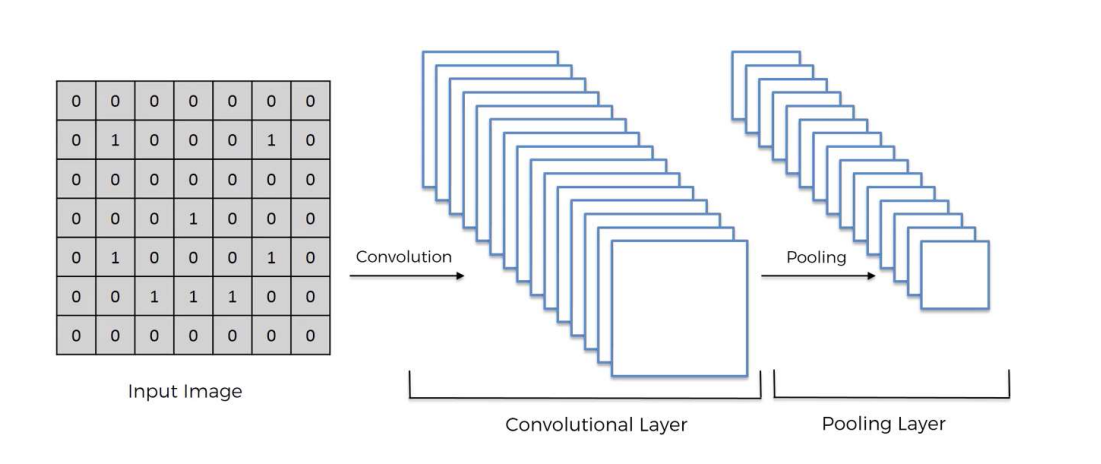
* **Max Pooling:** Chooses the maximum value from each window. It is more commonly used because it tends to extract sharp features like edges.
* **Average Pooling:** Calculates the average of the values in each window. It is less commonly used as it tends to blur the features.

**5. Application in CNNs:**

* **Follows Convolution Layer:** Max Pooling is typically applied after a convolution layer that has already applied an activation function like ReLU.
* **Hierarchy of Features:** As the network deepens, max pooling layers help in building a hierarchy of more complex and abstract features.

**6. Considerations:**

* **Loss of Information:** While max pooling is effective in reducing dimensions and computational load, it can lead to a loss of finer details in the image.
* **Fixed Pooling Size and Stride:** The size of the pooling window and the stride are hyperparameters that need to be set. They can affect the network's performance and its ability to learn features.



• Step 3 – **Flattening**

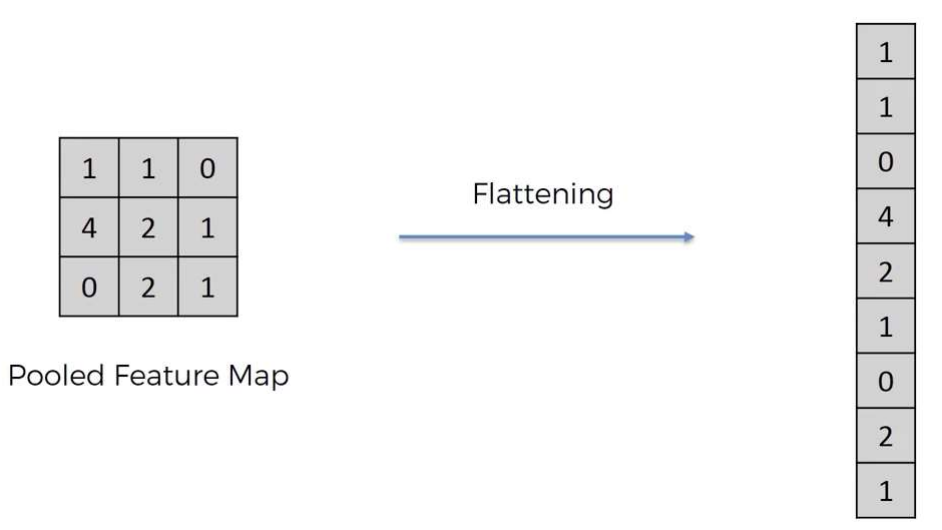
Flattening is the third key step in the process of building a Convolutional Neural Network (CNN). After the initial layers of convolution and pooling have processed the input image, flattening prepares the data for the final fully connected layers. Here's a breakdown of what happens during the flattening step:

**1. Purpose of Flattening:**

* **Preparation for Fully Connected Layers:** The main goal of flattening is to transform the processed 2D feature maps into a 1D vector. This format is necessary for the fully connected layers (also known as dense layers) that follow.

**2. Process of Flattening:**

* **Conversion to 1D Vector:** Flattening takes the entire 2D array (or 3D if considering multiple channels) of the feature map and converts it into a single long vector.
* **Maintaining Feature Information:** The process ensures that all the learned features from the convolution and pooling layers are preserved but restructured into a format suitable for classification.



**3. Why Flattening is Important:**

* **Compatibility with Dense Layers:** Fully connected layers, which often form the tail end of a CNN, require input data in a flat, 1D form. They are designed to perform classification based on the features extracted in previous layers.
* **Combining Features:** By flattening the data, the network can combine features learned in different parts of the image (which were processed by different filters) for final classification or regression tasks.

**4. Role in a CNN:**

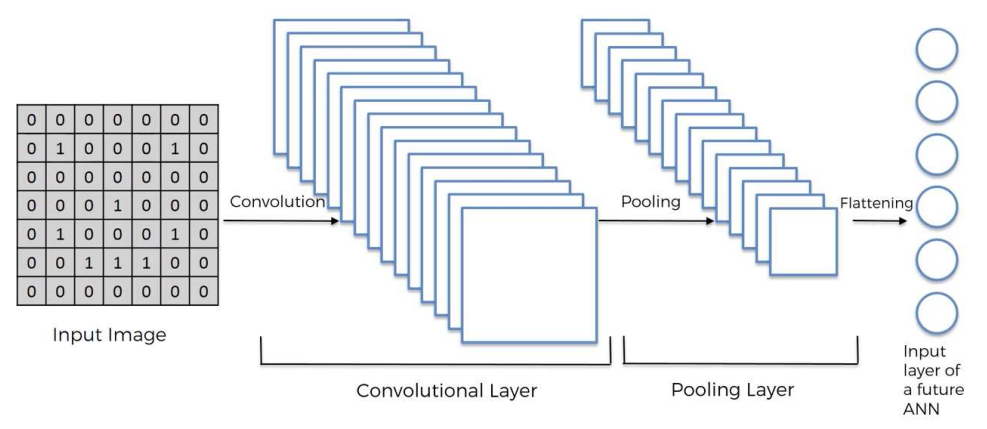
* **Bridge Between CNN and MLP:** Flattening acts as a bridge between the convolutional/pooling part of the CNN (which handles feature detection) and the fully connected layers (which handle the high-level reasoning or classification).
* **End of Spatial Processing:** Once the data is flattened, the spatial relationships between pixels are no longer directly maintained. The network transitions from focusing on extracting spatial features to making decisions based on those features.

**5. Considerations:**

* **High-Dimensional Vector:** Depending on the size of the feature maps and the number of filters, the resulting flattened vector can be very long, leading to a large number of parameters in the subsequent dense layers. This can increase computational complexity and the risk of overfitting.
* **Data Loss Concerns:** While flattening itself does not lose information, the transition to 1D means that spatial hierarchies and local structures are not explicitly represented anymore. The network relies on the dense layers to appropriately interpret these features.

**6. Subsequent Steps:**

* **Fully Connected Layers:** After flattening, one or more fully connected layers follow. These layers are traditional neural network layers where every input is connected to every neuron in the layer, leading to the final output such as a classification.



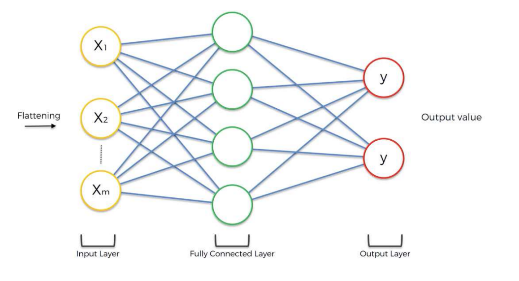
• Step 4 – **Full Connection**

Step 4 in the process of building a Convolutional Neural Network (CNN) involves establishing Full Connection, also known as Fully Connected (FC) layers. This step is critical in integrating the features extracted by previous layers to make final predictions or classifications. Let's explore this step in detail:

**1. Definition and Purpose:**

* **Fully Connected Layers:** These are traditional neural network layers where each neuron is connected to every neuron in the previous layer. This is different from convolutional layers where neurons are connected to only a small region of the previous layer.
* **Integration of Features:** The purpose of fully connected layers is to use the high-level features (represented by the flattened vector from the previous step) to make predictions or classify the data.

**2. Process of Full Connection:**

* **Input from Flattened Layer:** The input to the fully connected layers is the flattened vector obtained from the preceding layers (convolutional and pooling layers).
* **Weighted Sum and Activation:** Each neuron in a fully connected layer computes a weighted sum of its inputs, adds a bias, and then applies an activation function. This could be sigmoid, softmax, ReLU, etc., depending on the task.

**3. Role in a CNN:**

* **Decision Making:** Fully connected layers are where the network makes decisions based on the extracted and combined features. For instance, in classification tasks, these layers determine the class of the input image.
* **Final Layers in CNN:** They usually form the ending layers of a CNN architecture. The last fully connected layer often has the same number of neurons as the number of output classes in a classification task.

**4. Benefits and Importance:**

* **Learning Non-linear Combinations of Features:** Fully connected layers are capable of learning complex non-linear combinations of the high-level features extracted by the convolutional and pooling layers.
* **Versatility:** They can be used for various tasks such as classification, regression, and more, depending on the final activation function and the structure of the network.

**5. Activation Functions:**

* **Softmax for Classification:** In classification tasks, the softmax activation function is often used in the final layer to provide a probability distribution over different classes.
* **Sigmoid or ReLU:** For binary classification or regression tasks, sigmoid or ReLU might be used.

**6. Considerations:**

* **Overfitting Risk:** Fully connected layers typically have a large number of parameters. This can lead to overfitting, especially if the network has too many neurons or layers without proper regularization or dropout techniques.
* **Computational Intensity:** These layers can be computationally intensive due to the high number of parameters and connections, especially in deep networks with large input sizes.

**7. Subsequent Steps:**

* **Output Layer:** The final fully connected layer acts as the output layer. Its design (like the number of neurons and the type of activation function) depends on the specific task, like multi-class classification, binary classification, or regression.

