Summary

- Abstract Patterns and Features
  - Features
    - type
      - Raw quantitive data
        - numerical data, boolean values, vectors, and even images
      - Direct features
        - Edge detection, detected circles/ellipses, Spectrograms
      - Abstract features
        - Region textures, **Moments**
    - problem
      - insufficient training data
      - unrepresentative training data
      - irrelevant features
      - poor quality data
  - Abstract features: Moments
    - Calculating Moments

$$m_{pq} = \sum_x \sum_y x^p y^q$$

      -
      - m = moment
      - $moment_{pq}$ the pq^{th} moment
      - x,y = pixel
    - normalized moment

$$M_{pq} = \sum (x - \bar{x})^p (y - \bar{y})^q$$

      -

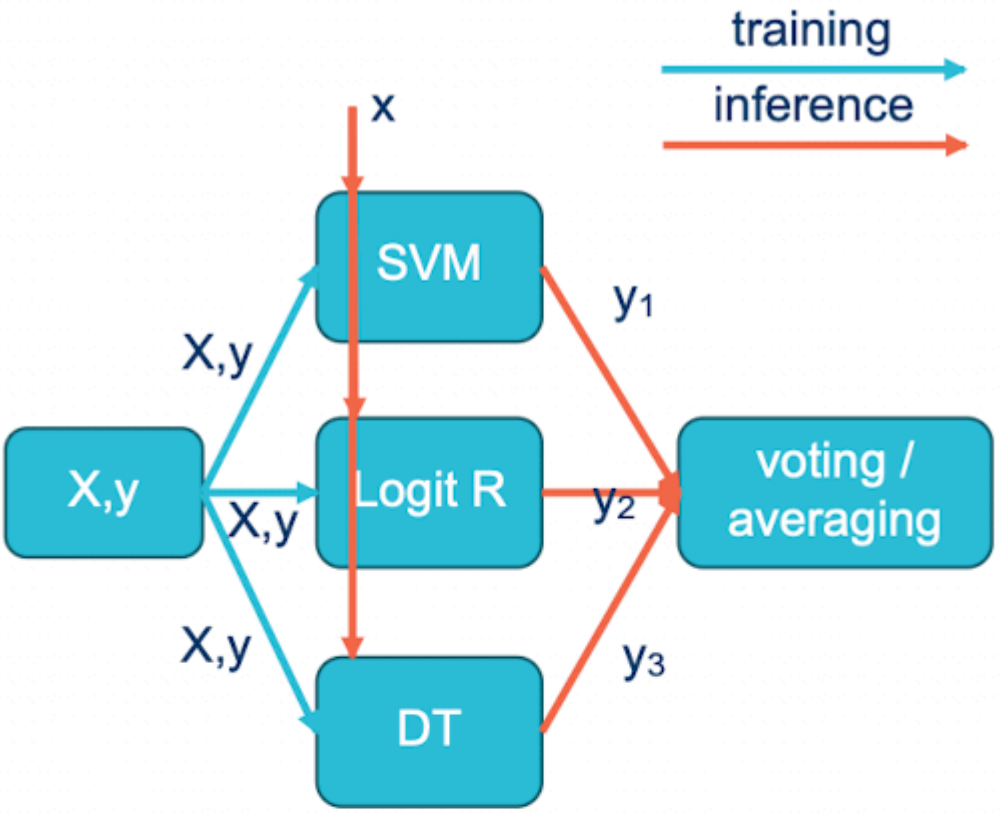
- Classification and trees
  - Classification
    - supervised learning
    - predict any categorical - multiple-choice target feature

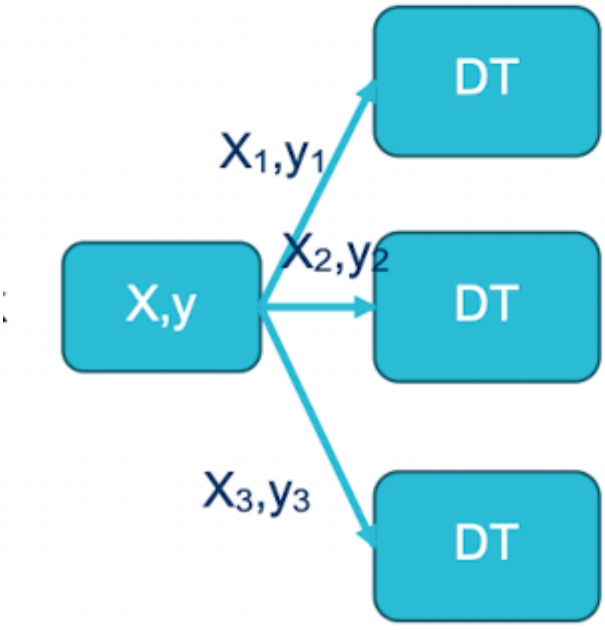
- Data preparation
  - Splitting data into train set and test set
  - Data Cleaning/ imputation
    - `np.nan` or `pd.NA` --> filled or dropping
    - duplication --> dropping or keeping
  - Data preparation
    - Categorical values -> encode, one-hot encoding, some algorithms
    - Scaling/Standardisation of numerical features (PCA, SVM)
    - for undersampling or oversampling -> imbalanced learn package
  - Pipelines
    - composed by a sequence of transformers that transform the input for the next transformer or for the final predictor
    - used for easily repeated and tuned
- Scikit-learn (sklearn) design
  - estimators
    - `fit()` for supervised learning
    - `set_params()` and `get_params()` how to know and set the hyperparameters
  - transformers
    - `transform(x)` : e.g., like using `pca.fit(x)` and `fit_transform()`
  - predictors: `predict(x)` , provide predictions from observations
  - pipeline
    - a set of transformers and a final predictor
    - `pipeline.fit(x)` : invoke transformation step
    - `pinpeline.predict(x)` : invoke transformation on each step and predict on the final predictor
- classification algorithm
  - binary algorithms
    - Support Vector Machine (SVM)
    - Logistic Regression
    - Stochastic Gradient Descent
  - Multiclass algorithms
    - Naive Bayes
    - Decision Tree

    ** binary algorithms can be useful for multiclass problems with M label
  - one-vs-one strategy
    - distinguish every couple of labels
    - M(M-1)/2 -> N/M**2
  - one-vs-all strategy
    - distinguish class from non-class

        - predictors, each trained on a full dataset
- model evaluation
    - how good is the resulting model
    - overfitting or not
        - overfitting and underfitting
            - overfitting -> It memorizes the training set and is not good at generalizing to new inputs
            - underfitting -> not powerful enough to learn/explain the training data (low training accuracy)
        - increasing the model complexity by reducing bias and increasing variance, called bias-variance tradeoff
        - hyperparameters and cross-validation
            - e.g., in decision tree models, fix a maximum depth or number of leaves
            - hyperparameter should be evaluated using a portion of the training dataset called the validation set
    - Metrics
        - binary classifier accuracy = (TP + TN) / (TP + TN + FP + FN)
        - precision = TP / (TP + FP)
        - Recall:
            - For positive class (Sensitivity) = TP / (TP + FN)
            - For negative class (Sensitivity) = TN / (TN + FP)
        - F1 score: TP / (TP + (FP + FN) / 2)
- Decision Trees
    - idea: recursively split dataset on an attribute-condition-value -> ensemble
- Ensemble models

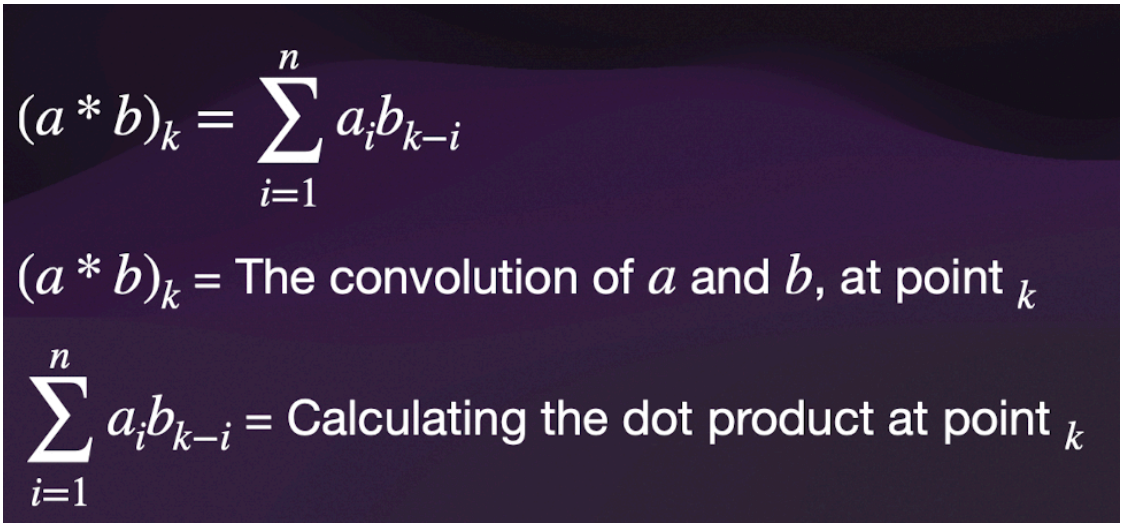
- aggregating multiple trained models to improve predictions



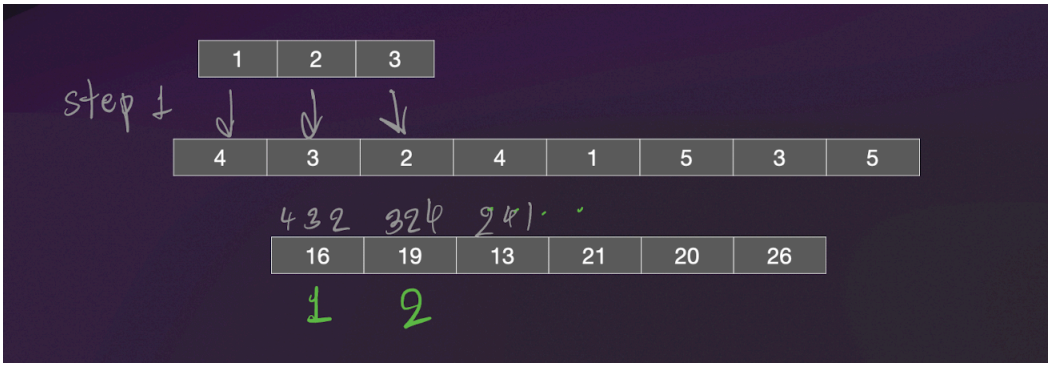


  - voting classifier
  - average model irregularities -> decrease variance, reduce overfitting
  - parallelize train
  - example:
    - `VotingClassifier`
    - `BaggingClassifier` for patches and subspaces
    - `RandomForest` : random patches with decision trees

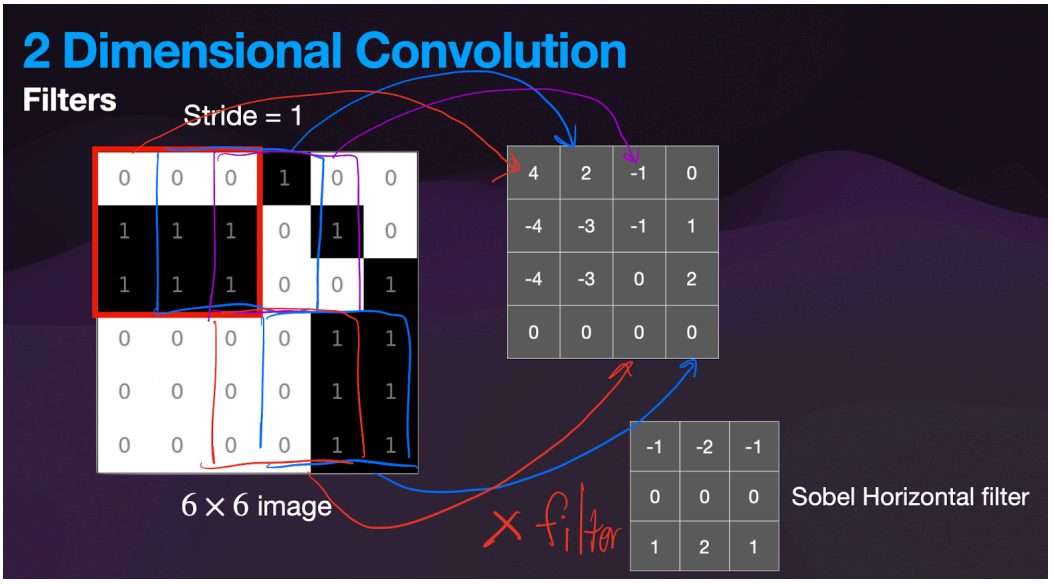
    - `Boosting` : Train predictors sequentially
    - `Adaboost` : increase the weight of misclassified instances, uniform weights
    - `Gradient Tree Boosting`
- Convolutional Neural Networks (CNN)
  - Convolutional layer

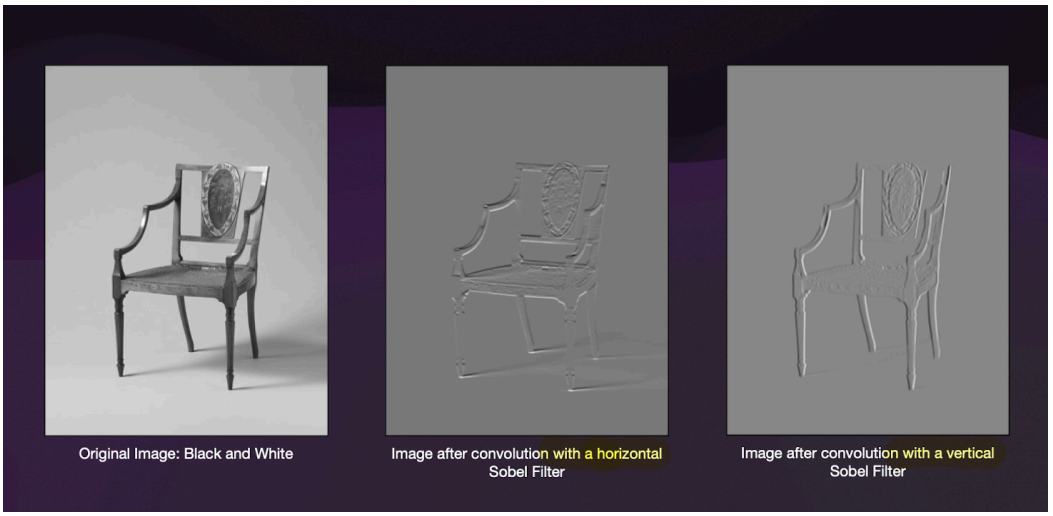


    - 1 Dimensional

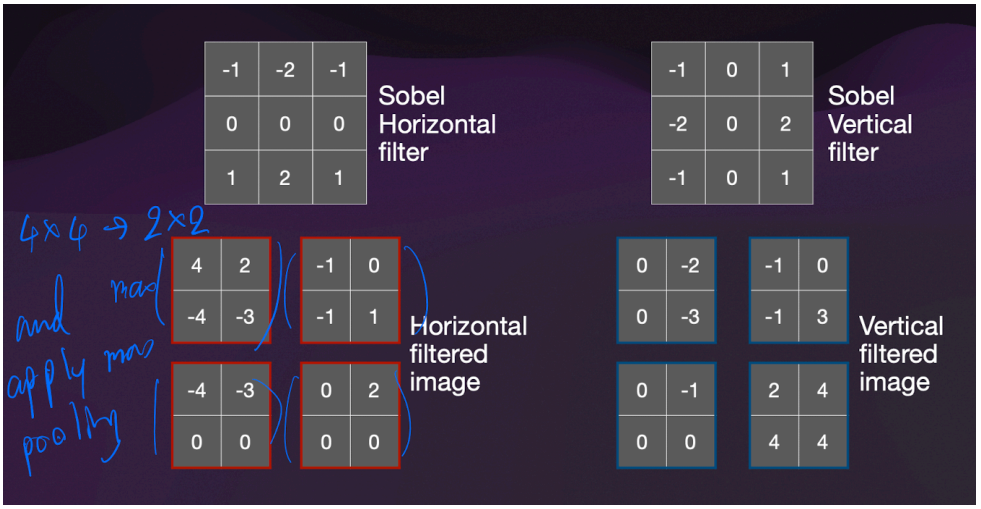


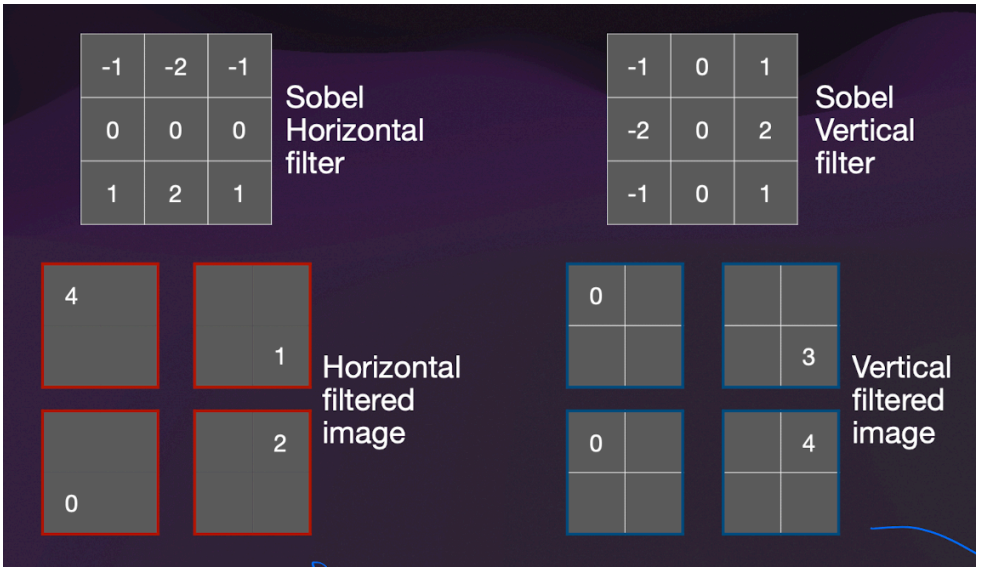
    - 2 Dimensional



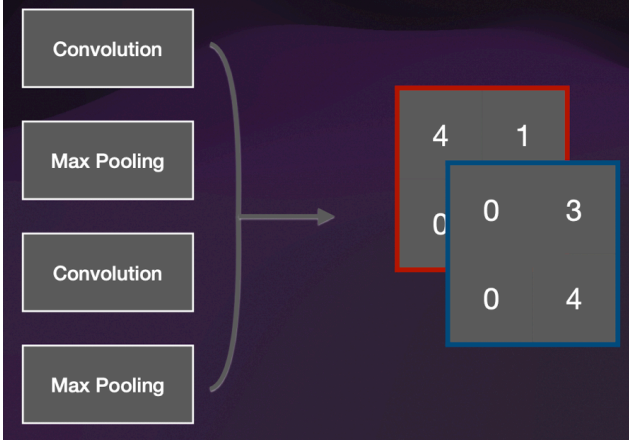


- Pooling Layer
  - Max Pooling





Multiple Convolution Layers

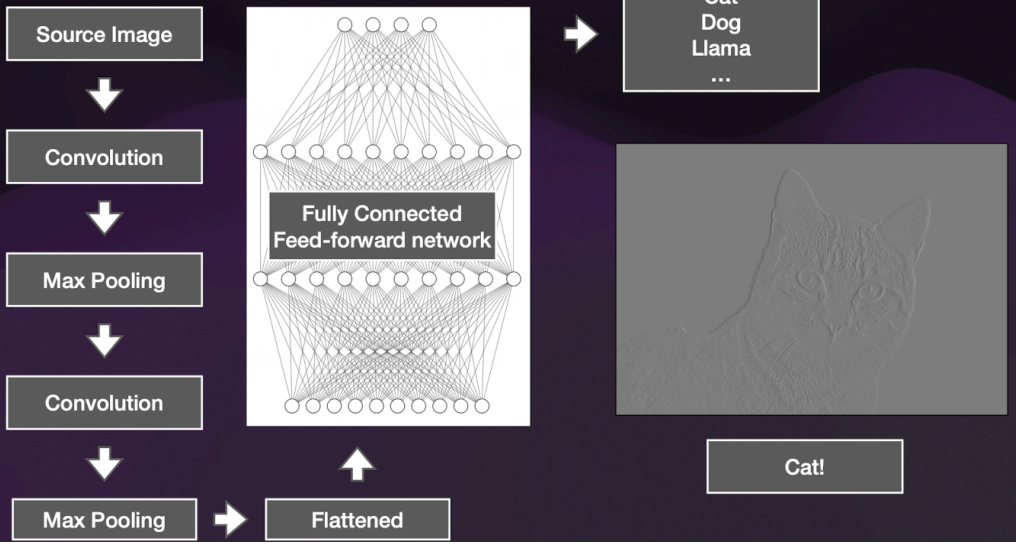


Smaller than the original image

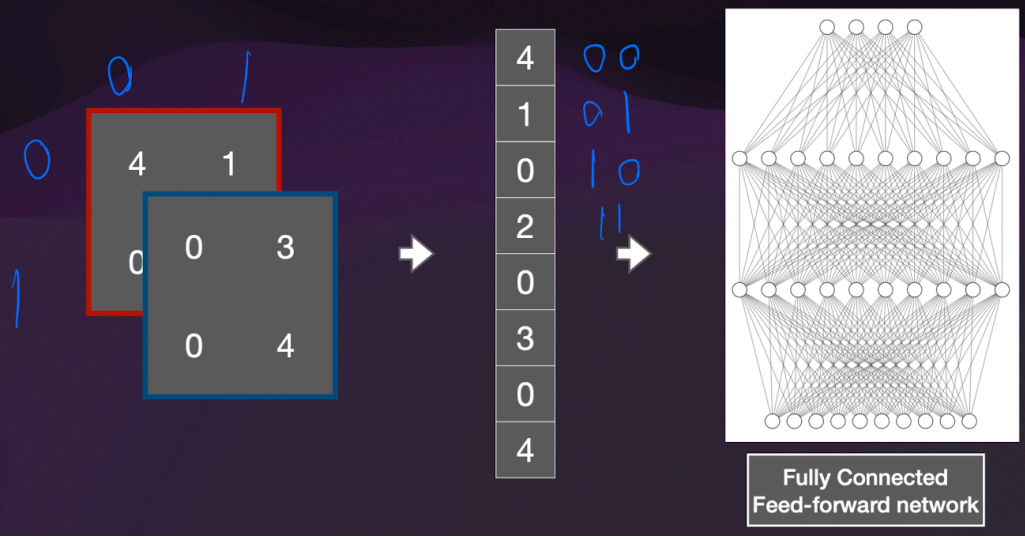
The number of channels is the number of filters

Subsequent layers look for combinations of features from previous layers!

The Whole CNN

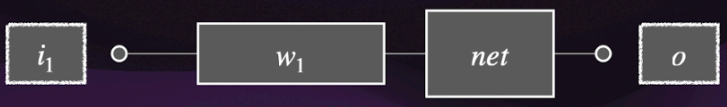


Flattening



Single Input Neuron

NB: i_1 not Boolean in this example



$$net = i_1 \times w_1$$

$$o = 1, \text{ if } net \geq \theta$$

$$o = 0, \text{ if } net < \theta$$

For threshold θ, may rewrite as:

$$net = i_1 \times w_1 - \theta$$

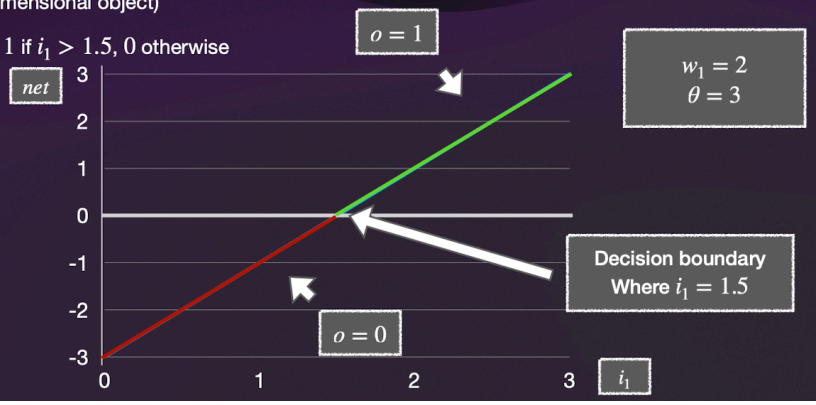
$$o = 1, \text{ if } net \geq 0$$

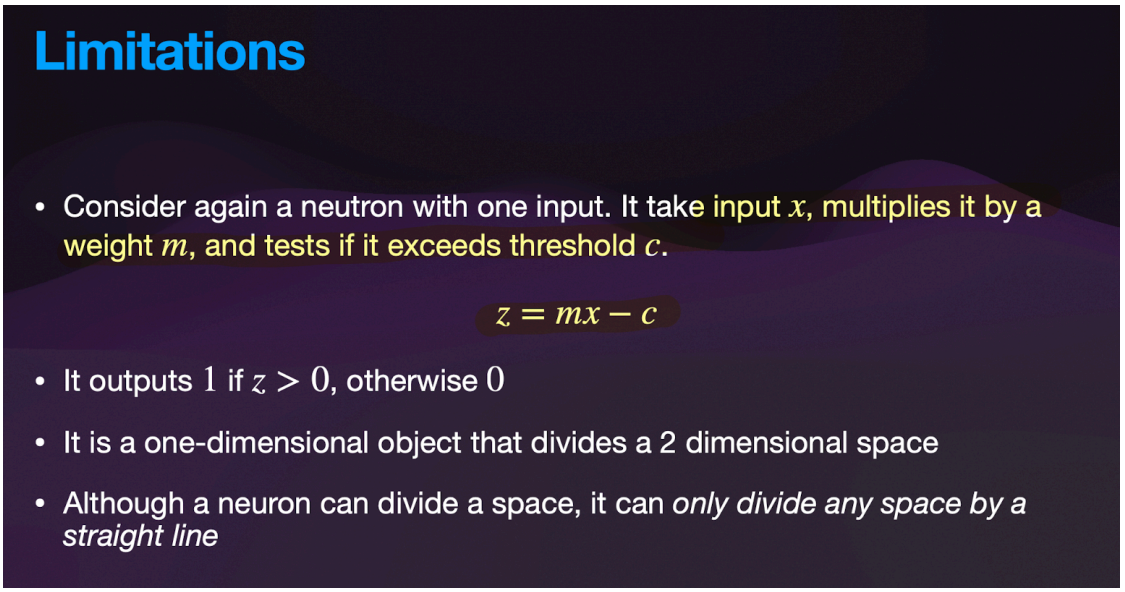
$$o = 0, \text{ if } net < 0$$

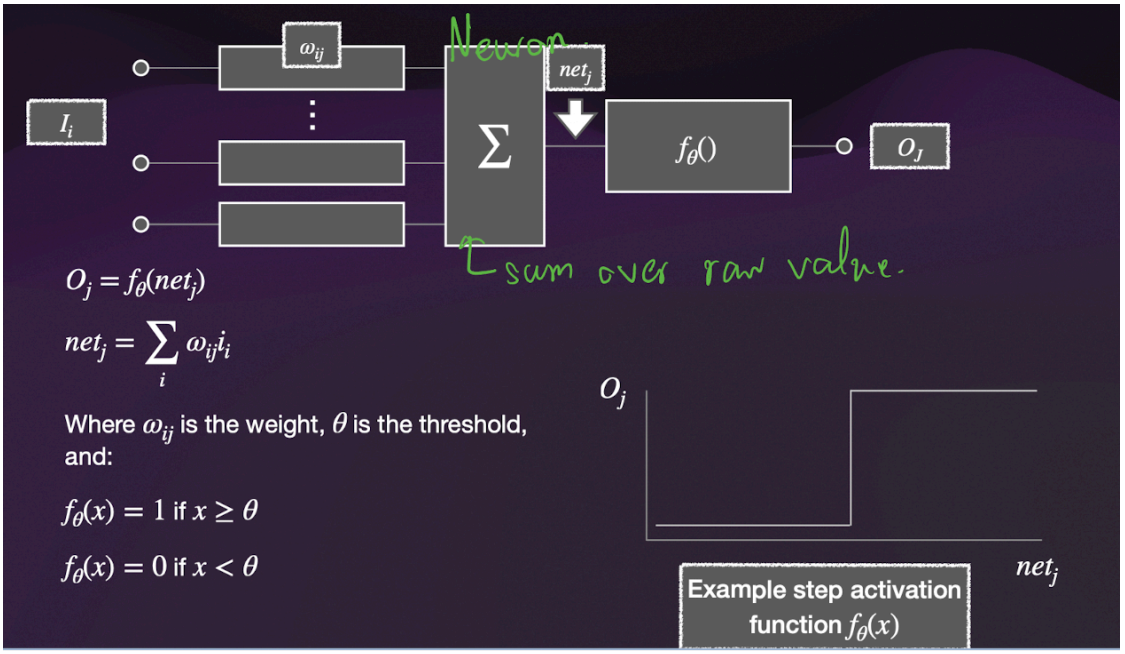
Analysis

- Decision Boundary: Where straight line formed by *net* equations meets straight line formed by i_1 axis
  - It is a single point (0 dimensional object)
  - Original example: $o = 1$ if $i_1 > 1.5$, 0 otherwise

i_1	*net*
0	-3
1	-1
2	1
3	3







- Reinforcement Learning
  - 5 elements
    - Environments
      - Farama Gymnasium = OpenAI Gym
        - a collection of simulations and environments
        - to simulate step by step whilst connecting to an agent
        - hug variety
        - use a tabular-based Q-learning
          - Q-learning
            - **a reinforcement learning algorithm that finds an optimal action-selection policy for any finite Markov decision process (MDP)**
            - A Markov model is a stochastic model that describes a sequence of events where the probability of each event depends on the previous event. It's used in probability theory, statistics, and decision analysis

- Applications e.g.,
  - **Weather**
    A Markov model can be used to predict the weather, for example, by modeling the probability of a sunny day being followed by another sunny day.
  - **Healthcare**
    A Markov model can be used to model the health of patients over time, including the progression of disease and the effects of medical interventions.
- Q-Table
  - States (S_t) with Q-Value
    - Q-Value: showing that the algorithm thinks it would be the best decision
- Update Formula

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot max_a Q(s',a') - Q(s,a)]$$

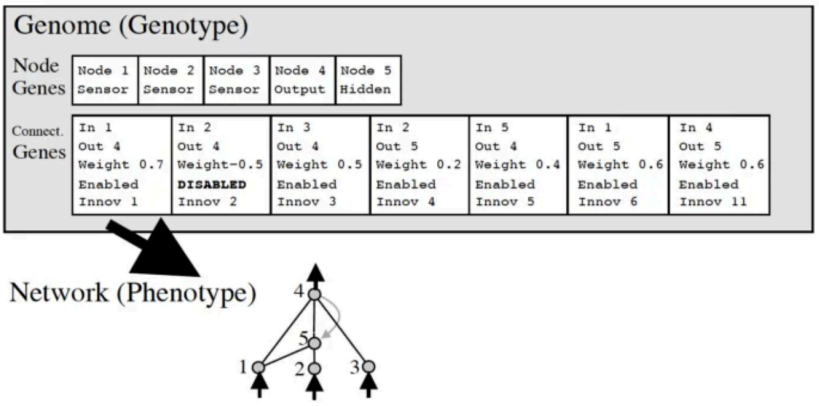
- $Q(s,a)$: The current Q-value for s and a
- r: The immediate reward for taking action a in state s
- α: The learning rate
- s': The next state after taking action a in state s
- a': The next action chosen in s'
- $max_a Q(s',a')$: the maximum possible Q-Value from all possible actions in the next state s

State	Q-Values
1	[0.735, 0.773, 0.773, 0.735]
2	[0.735, 0, 0.814, 0.773]
3	[0.773, 0.857, 0.773, 0.814]
4	[0.814, 0, 0.773, 0.773]
5	[0.773, 0.814, 0, 0.735]
6	[0, 0, 0, 0]
7	[0, 0.902, 0.902, 0]
...	...

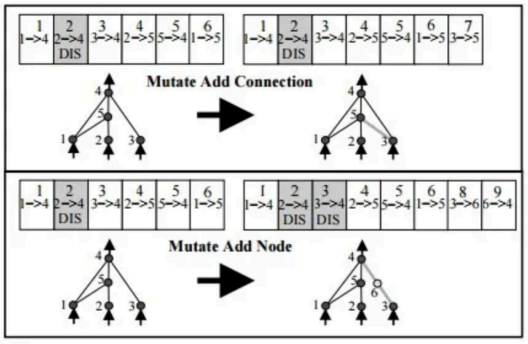
  - utilise epsilon-*greedy Q-learning*, a well-known reinforcement learning algorithm
  - (SARSA) State-Action-Reward-State-Action
- cr. https://gymnasium.farama.org/introduction/basic_usage/
  - States (S_t) - the current state of the scenario (a value or array of value)
  - Actions (A_t) - An array representing some forces to be applied in the simulation
  - Rewards (R_t) - A value representing how the system performed
  - Agents - The system that reads the states
- Policies - set the observation space, action space, and reward.
- Basic Neural Network
- Credit assignment - every task of agent has reward and reduction.

- Genetic algorithms (GA) and Neuroevolution
  - fitness function - measuring the quality of each solution
  - method
    - Selection
      - Roulette-whee;

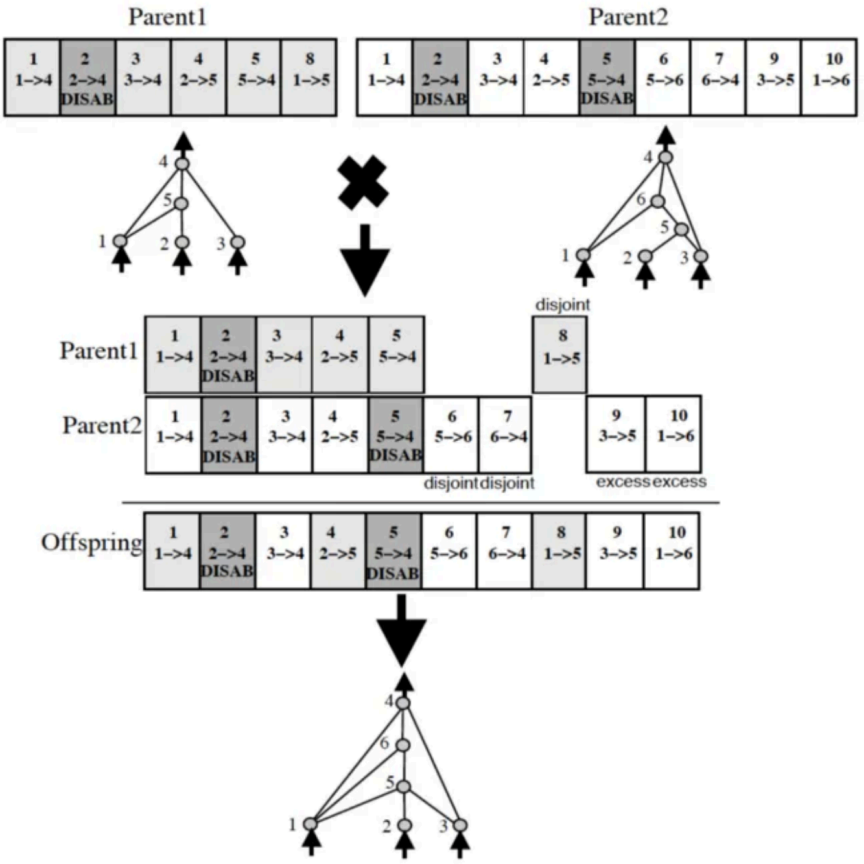
        - Tournament-based
        - Truncation
    - Replication
        - One-point crossover
        - Multipoint crossover
        - Uniform crossover
    - Mutation
    - Elitism
- Applying GA to NN for
    - evolving NN weights
    - alternative to backpropagation based on gradients
        - backpropagation - technique of train
        - gradient range of environment
- NeuroEvolution of Augmenting Topologies (NEAT), used for
    - is an evolutionary algorithm that creates artificial neural networks.
    - Method
        - encoding
            - Mutation - weight mutation



1. Add a connection between two unconnected nodes.
2. Add a node between two nodes that are already connected, disabling the old connection & creating two new ones to preserve the structure of the network.
3. Change the weight of a connection.



- Crossover
  - First of all, we need to select a dominant parent. This is the parent that will specify the structure of the child's network.
  - Secondly, we find all of the connections shared by both parents. You'd think this would be difficult, but since we have innovation numbers, we can just take each connection whose innovation number appears in both parent networks.
  - For any connection shared by both parents, we randomly give the child one of the connections. This ensures that the weights of any shared connection in the child network will randomly come from either parent.
  - Finally, for any connections not shared by both networks, the child inherits them from the dominant parent.



- Niching
  - how to manage the evolution of topologies and weights at the same time
  - defending the compatibility threshold by $c_1 \triangle G + x_2 \triangle W$, Average of G and Average of W