DEFINITIONS

Machine Learning: study of algorithms that improve their performance P at some task T with experience E

Performance: refers to how well a ML model is able to accomplish its task. We must define an evaluation of the performance (metric) according to the designed task (Task: classification → Metric: accuracy)

Task: refers to a specific problem or objective that a machine learning model is designed to solve or accomplish (ex. Classification, regression, clustering…)

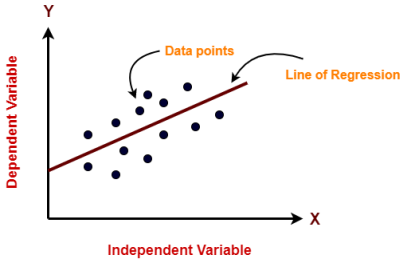
Experience: two types of experiences are generically faced which are supervised and unsupervised. In the supervised case each sample x is associated with a label y and the model tries to predict y by estimating p(y|x). In the unsupervised case each sample x has no label and the model tries to learn the probability distribution p(x)

Samples: also known as examples, are collections of features that have been quantitatively measured from some object or event that we want the ML system to process.

Dataset: The set of samples is also known as dataset. We assume that each sample is independent and identically distributed. We can distinguish two types of datasets: labeled and unlabeled. In the labeled dataset, each sample is associated with a label (or class) it belongs to. In the unlabeled dataset there's no label. The first case (labeled dataset) involves supervise learning, the second case involves unsupervised learning.

The dataset, which dimensions are here expressed as N, is going to be split into training set (which size is often 0.8*N), validation set (which size is often 0.1*N) and test set (which size is often 0.1*N). All these sets have different aims: the training set is the part where the model learns the relationships within data and updates the learnable parameters in order to minimize some loss function; the validation set is used to understand which model is the best for that given task and data, to finetune the hyperparameters (which value is not fixed), to prevent overfitting, to measure the performances and to ensure that the model is likely to generalize well to unseen data; the test set is used to test what the model has learnt during training on unseen samples.

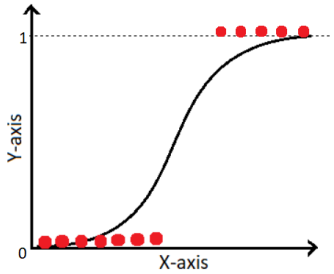
Linear Regression:



It is a fundamental statistical and machine learning technique used for modeling the relationship between a dependent variable (also called the target or outcome, y) and one or more independent variables (also called predictors or features, x) by fitting a linear equation to the observed data.

Given n samples x, $x \in \mathbb{R}^n$, the predicted y ($\hat{y}$) is defined as $\hat{y} = w^T x$. The vector $w, w \in \mathbb{R}^n$, contains values also called weights which can be updated through the learning process in order to find the line fitting best the samples. To achieve this goal a loss function is used, the Mean Squared Error (MSE) defined as: $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$. The reduction of this loss is done computing $\nabla_w MSE = 0$. The update step is done on the training set.

Logistic Regression:



It is a statistical and machine learning technique used for binary classification, which means it's used to predict one of two possible outcomes or classes (usually denoted as 0 and 1) based on one or more independent variables. Logistic regression models the probability that a given input belongs to one of the two classes. To do this, it uses the logistic (sigmoid) function, which maps any real-valued number to a value between 0 and 1. The logistic function is an S-shaped curve (as shown in the figure).

The logistic function is $P(y = 1 \mid x) = \frac{1}{1+e^{-z}}$ where z is the combination of independent variables x and their associated weights w. The sigmoid function outputs a value between 0 and 1 so a threshold is fixed to classify a sample as 0 or 1. To get better results the learning process updates the weights in order to minimize a loss function, the Binary Cross Entropy (BCE):

$$\mathcal{J}(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^T\mathbf{x}_i) + (1-y_i)\log\left(1-\sigma(\mathbf{w}^T\mathbf{x}_i)\right)$$

Minimizing this loss is not easy as in linear regression, $\nabla_w J(w) = 0$ is not tractable. So the Gradient Descent algorithm is used to update weights: $w_t = w_{t-1} - \alpha \frac{\partial J}{\partial w}$ where α is the learning rate (an hyperparameter).

Gradient Descent: it is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.

These are the steps of the algorithm:

Initialize weights randomly w $\sim N(0,\ \sigma^2)$

Loop until convergence:

Comput Gradient $\frac{\partial J(w)}{\partial w}$

Update weights $w_t = w_{t-1} - \alpha \frac{\partial J}{\partial w}$

Return weigths

The number of times this algorithm is repeated is called **epochs**.

Two main types of GD are used: stachastic gradient descent, SGD, (were weights are updated at each sample) and batch gradient descent, BGD, (where weights are averaged updated after an entire batch (group of k samples) has been seen).

Advantages of SGD: Efficiency, Convergence Speed, Generalization (since it introduces noise), Regularization Effect, Memory Efficiency, Generalization

Disadvantages of SGD: High Variance in Updates, Noisy Updates, Learning Rate Tuning, Not Guaranteed to Converge, Hyperparameter Sensitivity, Difficulty in Mini-Batch Selection

Advantages of BGD: Stable Convergence, Deterministic Updates, Smooth Trajectory, Efficient Use of Hardware, Easier Hyperparameter Tuning, Well-Behaved Learning Rate, Parallelization, Stability, Regularization

Disadvantages of BGD: Computational Intensity, Slow Convergence on Large Datasets, Memory Usage, Limited Parallelism, Sensitivity to Outliers, Fixed Learning Rate, Lack of Online Learning, Large Initial Memory Requirements, Potential Overfitting

Batch Size (Hyperparameter): The batch size determines how many data samples are used in each iteration of training. The choice of batch size can impact training speed, memory requirements, and the convergence behavior of your model.

- **Large** batch size: faster training, may converge to a suboptimal solution or overfit the training data, more stable gradient, may have smoother loss curves but can converge to less optimal solutions.
- **Small** batch size: slower training, better generalization, introduce more randomness in the gradient estimates, may exhibit more erratic loss curves

Learning Rate (Hyperparamter): it determines the step size at which the model's parameters are updated during training. If the learning rate is too high or too low, it can significantly impact the model's convergence and performance. The optimal learning rate can vary depending on the dataset, model architecture, and the specific optimization algorithm used. Strategies for setting the learning rate: manual tuning, Learning Rate Schedules, Grid Search and Random Search, Adaptive Learning Rate Methods (…)

Decay Learning Rate: Decaying the learning rate during training is a common technique in machine learning to improve convergence and training stability. The idea behind learning rate decay is to start with a relatively high learning rate to make large updates early in training and then gradually reduce it as training progresses. This approach helps the model converge faster at the beginning and fine-tune its parameters more carefully as it gets closer to a minimum of the loss function.

Generalization: it is a concept that refers to the ability of the model to perform well on unseen data. So the performance on the test set gives a measure of the model ability to generalize.

Overfitting: it is a phenomenon occurring when the model performs too well on training data, since it captures noise and random fluctuations in the data rather than the underlying patterns and relationships, but poorly on the test set. So the model doesn't generalize well. Overfitting can happen when: the model is too complex (too many degrees of freedom), data is insufficient (with limited data the model memorizes easily the samples), the number of features is high (they can be irrelevant or noisy), no regularization technique is applied, the data is noised.

There exist some approaches to prevent (or address) overfitting: simplify the model, feature selection, regularization, cross validation, increase training data, ensemble methods, early stopping, data preprocessing.

Underfitting: it is the opposite of the overfitting. The model is too simple to capture the underlying patterns in the training data. It performs bad both in the training and test set. Underfitting occurs when: data is insufficient, features have been selected in a wrong way, training is inadequate, inappropriate data preprocessing.

There exist some approaches to prevent (or address) underfitting: Increase Model Complexity, Feature Engineering, More Training Data, Regularization, Hyperparameter Tuning, Data Augmentation, Ensemble Methods, Revisit Data Preprocessing

Model capacity: also referred to as model complexity, is a crucial concept in machine learning. It represents the ability of a machine learning model to represent complex relationships in data.

Regularization: it is a technique used in machine learning and statistical modeling to prevent overfitting and improve the generalization ability of models. ("any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error")

Dropout (2012): it is a regularization technique consisting in "turning off" a fraction of neurons (the drop out rate is an hyperparameter, typically set from 0.2 to 0.5). This method aims to prevent overfitting since random neurons don't allow informations to pass through next (previous) layers in forward (backward) pass when dropped out. This randomness encourages the network to be more robust and prevents it from relying too heavily on any specific neurons. Because dropout is applied stochastically, the model effectively learns to make predictions by averaging over many different subnetworks. This ensemble effect helps the model generalize better to unseen data. During the testing or inference phase, dropout is typically turned off, and all neurons are active. This ensures that the model's predictions are deterministic and consistent. So the key benefits of dropout are: prevent overfitting, regularization (reducing the model's capacity to memorize the training data), ensemble learning (trains an ensemble of neural networks with shared parameters), Reducing Sensitivity to Noise.

Early Stopping: it is a regularization technique consisting in stopping the learning process of the model before the maximum number of epochs (or convergence) is reached. This occurs when a certain metric, typically validation loss (or accuracy), doesn't decrase (or increase) its value for a specified number of consecutive epochs, so when the performance starts to degrade. Once the learning processi s stopped, the model's parameters from the epoch where the validation performance was the best are typically saved.

This method helps preventing overfitting (prevents the model from continuing to learn from the training data when it starts to overfit.), improving efficiency (can save computation time and resources because it avoids training for a fixed number of epochs, which may be unnecessary) and generalization (the model has learned relevant features from the training data and can generalize well).

The effectiveness of early stopping depends on selecting appropriate hyperparameters, such as the patience (the number of epochs to wait for improvement before stopping) and the choice of metric to monitor.

Data Augmentation: it is a technique used to increase dataset size by adding new samples. These new samples are obtained by applying various transformations or perturbations to the original data (in images: cropping, resizing, scaling, flipping, rotating, brightness or contrast adjustments, adding noise, …).

Data Augmentation allows to:

- add diversity in the dataset, making the model more robust and reducing the risk of overfitting
- Leverage Limited Data, often good results can be achieved if the training is done on a big amount of data
- improve generalization
- reduce overfitting

Batch Normalization: it is a popular technique in deep learning, primarily used in neural networks, to improve training stability, accelerate convergence, and make deep networks easier to train. During training, BatchNorm normalizes the activations (output values) of a layer by subtracting the mean and dividing by the standard deviation, independently for each feature (neuron), within each mini-batch of data. So it introduces two learnable parameters per feature, gamma, denoted as γ and shift, denoted as β.

Its formula is: $\frac{x_i-\mu}{\sqrt{\sigma^2+\varepsilon}} \cdot \gamma + \beta$

Benefits: Stabilized Training, Reduced Internal Covariate Shift, Regularization Effect, Increased Learning Rates, Independence from Initialization

Function estimation: it refers to the process of approximating an underlying mathematical function f with another function f' parametrized by ϑ. Estimating f means estimating ϑ.

Bias: it refers to the error introduced by approximating a real-world problem (which may be complex) with a simplified model. High bias models are typically too simple and unable to capture the underlying patterns in the data. Its formula is:

$\text{Bias}(\hat{\theta}) = E(\hat{\theta}) - \theta$, where $E(\hat{\theta})$ is Expected value (mean) of the estimator $\hat{\theta}$ and θ is True (population) parameter value.

Variance: it refers to the error introduced by the model's sensitivity to small fluctuations in the training data. High variance models are usually very complex and can fit the training data extremely well causing overfitting.

- Underfitting: high bias, low variance
- Overfitting: low bias, high variance

Bias-Variance Tradeoff: the goal in machine learning is to strike a balance between bias and variance to achieve good generalization. The bias-variance trade-off suggests that as you decrease bias (by increasing model complexity), you typically increase variance, and vice versa.

Neural Network: Most of the times several issues occur when treating more complex tasks:

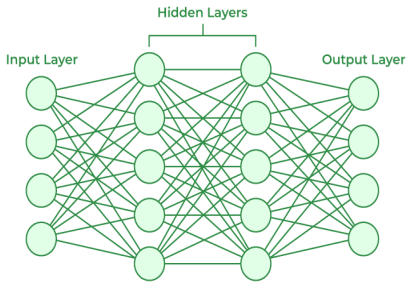
- relationships within data are not linear
- data type is complex (image, audio, text…)
- data is high dimensional
- data can evolve

To address them, neural networks have been widely used, showing in most of cases state-of-the-art results. Neural networks introduce some advantages as they can capture deep, non-linear and complex relations in data, they suit lots of data type, they automatically learn hierarchical feature representation, they handle large dataset with a lots of parameters, (…).

Neural networks also require careful tuning, sufficient data, and computational resources.

The structure and functions of a neural network are similar to the biological neurons, since both artificial neural networks and biological brains:

- use individual processing units to perform computations
- rely on connections between processing units
- involve the transmission of signals
- are capable of learning and adapting and generalization
- can be organized into hierarchical structures
- can perform parallel processing



The components of a neural network are: the **input layer** (a vector xϵR^n), a number of **hidden layers** with m neurons, and the **output layer**. The connections linking each neuron of the layer to those of the next one represent the weights (a matrix W$\epsilon \mathbb{R}^{n\times m}$).

So we have already some decisions to make (number of hidden layers, number of units per hidden layers, activation function). Network depth is a hyperparameter that requires careful tuning and experimentation.

The number of hidden layer influences the model complexity and capacity (risk of overfitting), the computational resources (increasing the number of parameters). Simpler models are preferable to complex ones when their performance are similar. A good practice is starting with shallow networks and then increase the complexity.

Decision Boundaries: they refer to the regions in the input space that separate different classes or categories of data points. These boundaries determine how the network assigns inputs to specific output classes. Complex networks provide more accurate decision boundaries. They are influenced by the architecture, activation functions, and training data.

Activation functions: they are mathematical functions used in artificial neural networks to introduce non-linearity into the network. These non-linearities are essential for enabling neural networks to learn complex patterns and relationships in data. Activation functions determine whether a neuron should be activated (i.e., produce an output) or not, based on its input. There are several types of activation functions and each one influences the neural network behaviour. Generally, choosing the right activation function depends on the problem at hand, the architecture of the neural network, and empirical performance on validation data.

The most popular activation functions are:

- **Sigmoid function**: $f(x) = \frac{1}{1+e^{-x}}$, often used in the output layer of binary classification models, where the goal is to produce a probability between 0 and 1. They squash input values to the range (0, 1) and are smooth and differentiable.
- **Tanh:** $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, similar to sigmoid functions but have a range of (-1, 1). They are commonly used in hidden layers of neural networks.
- **ReLU**: $f(x) = max(0, x)$, one of the most widely used activation functions. It replaces all negative input values with zero, introducing sparsity and non-linearity
- **Softmax**: $f(x)_i = \frac{e^{x_i}}{\sum_{J=1}^{n} e^{x_j}}$, typically used in the output layer of multi-class classification models. It normalizes the inputs into a probability distribution over multiple classes, ensuring that the outputs sum to 1.

Losses: loss functions (also known as cost functions or objective functions) are crucial components used to quantify how well a machine learning model's predictions match the true target values (ground truth). The choice of an appropriate loss function depends on the type of problem being solved, such as regression, classification, generative modeling, or reinforcement learning.

Forward pass: During the forward pass, input data is fed through the neural network from the input layer to the output layer. Each layer performs two main operations:

- Weighted Sum: For each neuron in a layer, it calculates a weighted sum of its inputs, which are the outputs from the previous layer, and adds a bias term. This is represented as $z=w \cdot x+b$, where z is the weighted sum, w is the weight vector, x is the input vector, and b is the bias.
- Activation Function: The weighted sum z is then passed through an activation function $f(z)$ to introduce non-linearity and determine the neuron's output.

After the forward pass, the network's output is compared to the ground truth (target) to calculate a loss or error. The choice of loss function depends on the specific task.

Backpropagation: it involves computing the gradients of the loss with respect to the network's parameters (weights and biases) for each layer. This process is carried out in reverse order, starting from the output layer and moving backward through the network's layers. Backpropagation relies on the chain rule of calculus, which allows us to calculate the gradient of the loss with respect to the parameters of a layer by combining the gradients from subsequent layers. Once the gradients are computed for all layers, the network's parameters are updated using an optimization algorithm, most commonly gradient descent, in order to minimize the loss function, so the parameters are adjusted in the direction that reduces the loss (the opposite direction with respect to the gradient of the loss which points to the maximum).

Vanishing/Exploding Gradient: The vanishing gradient and exploding gradient problems are challenges that can occur during the training of deep neural networks. The vanishing gradient problem occurs when gradients become extremely small as they are propagated backward through the layers of a deep neural network. This often happens when activation functions squash their input into a small range, such as the sigmoid or hyperbolic tangent (tanh) functions. When gradients are very small, weight updates during training become negligible, leading to slow convergence or causing the model to get stuck in a poor local minimum. The exploding gradient problem is the opposite of the vanishing gradient problem. It occurs when gradients grow exponentially as they are propagated backward through the layers of a deep neural network. Exploding gradients can lead to numerical instability during training, causing weight updates to become very large.

Parameter Tying and Parameter Sharing: techniques used in deep learning and machine learning to reduce the number of learnable parameters in a neural network while maintaining certain relationships or constraints between them. Parameter tying involves enforcing constraints on a set of model parameters so that they share the same values. In other words, the parameters are tied together or linked in some way. Parameter sharing refers to the practice of using the same set of parameters for multiple parts of a model, such as different layers or branches. This encourages the model to learn shared representations or features across different parts of the input data.

Benefits: Reduce Model Size, Improved Generalization, Inductive Bias

Optimization: it is the process of training a neural network to minimize a specific objective function or loss function. The goal is to find the set of model parameters (weights and biases) that result in the best possible predictions on the training data. Most common optimization algorithms are GD, SGD, Momentum. The last one introduces modification to the standard SGD by involving a new vector of velocities to the updates. Momentum maintains a velocity vector v that accumulates the gradients of past iterations. The velocity vector is initialized to zero. At each iteration, the velocity vector v is updated as follows:

$v \leftarrow \beta v + \alpha \nabla J(\vartheta)$ where v is the velocity vector, β is the momentum coefficient (common values are in the range of [0.8, 0.99]), a hyperparameter between 0 and 1, α is the learning rate, another hyperparameter, $\nabla J(\vartheta)$ is the gradient of the objective function with respect to the model parameters ϑ. The model parameters ϑ are updated using the velocity vector v, as follows: $\vartheta \leftarrow \vartheta - v$.

Benefits of momentum: accelarate convergence, Dampening Effect, Combination with Other Optimizers, Hyperparameter Tuning.

Adaptive Learning Rate Algorithms: they automatically adjust the learning rate during training. These algorithms help address the challenge of choosing an appropriate learning rate and can accelerate convergence and improve training stability. Some common algorithms: Adagrad, Adam, RMSProp, AdaDelta…

Transfer Learning: it is a machine learning and deep learning technique where a model trained on one task is adapted for use on a different but related task. It involves reusing the knowledge or representations learned from one task to improve the performance of a model on another, often more specific, task. This technique makes the learning process faster and needs less data (and less labeled data) for the specific task by reusing its previous knowledge. It also improves generalization.

- *What* to transfer: what is common between the source and the target
- *When* to transfer: There can be scenarios where transferring knowledge for the sake of it may make matters worse than improving anything (negative transfer)
- *How* to transfer: changes to existing algorithms and different techniques

Naming the dataset where the model has been previously trained as "*source domain*" and the dataset where the model is applied on with transfer learning as "*target domain*" we can distinguish different types of transfer learning based on the datasets' characteristics:

- Inductive Transfer Learning: transfer learning is inductive when labeled data are available in the target domain
- Transductive Transfer Learning: transfer learning is transductive when labeled data are available only in the source domain
- Unsupervised Transfer Learning: transfer learning is unsupervised when no labeled data is available

Another definition of TL is: *Given a source domain D_S, a corresponding source task T_S, as well as a target domain D_T and a target task T_T, the objective of transfer learning now is to enable us to learn the target conditional probability distribution $P(Y_T|X_T)$ in D_T with the information gained from D_S and T_S where $D_S \neq D_T$ or $T_S \neq T_T$. In most cases, a limited number of labeled target examples, which is exponentially smaller than the number of labeled source examples are assumed to be available.*

This definition allows to define some other important concepts:

Cross-Lingual Adaptation: it occurs when $x_S \neq x_T$ (feature spaces of source and target domains are different) (ex. Two documents are written in different languages)

Domain-Adaptation: it occurs when $P(X_S) \neq P(X_T)$ (the marginal probability distributions of source and target domain are different) (ex. Two documents discuss different topics)

Class Imbalance: $P(Y_S|X_S) \neq P(Y_T|X_T)$ (The conditional probability distributions of the source and target tasks are different) (ex. source and target documents are unbalanced with regard to their classes)

Label Transfer: $\mathcal{Y}_S \neq \mathcal{Y}_T$ The label spaces between the two tasks are different (ex. documents need to be assigned different labels in the target task)

Applications: Model Zoo, HuggingFace, NeuralDBs, …

Siamese Neural Network: it is a class of neural network containing two identical subnetworks which have same parameters and weights. Each subnetwork takes one of the input data points, and produces a fixed-dimensional embedding or representation. The representations generated by the twin networks are then compared to determine the similarity between the two input data points.

Neural Collaborative Filtering: Recommended systems' main component is collaborative filtering. **Matrix factorization** is the most used variation of Collaborative filtering. It uses a fixed inner product of the user-item matrix to learn user-item interactions. **Neural Collaborative Filtering** replaces the user-item inner product with a neural architecture.

The problem is: Given a set of users U = {*u* = 1, …, U}, a set of items I = {*i* = 1, …, I}, and a log of the users' past preferences of items O = (*u, i, y*), our goal is to recommend to each user u a ranked list of items that will maximize her/his satisfaction. *y* can be either 1(Case-1: Observed entries) or 0 (Case-2: Unobserved entries).

The recommendation algorithms estimates the scores of unobserved entries in *Y*, which are used for ranking the items. They calculate:

$$\hat{y}_{ui} = f(u, i|\Theta)$$

y(u,i): predicted score for interaction between user and item

theta: model parameters

f(Interaction Function): maps model parameters to the predicted score

In order to calculate ***theta,*** an objective function needs to be optimized. The 2 most popular loss functions for the recommendation system are a pointwise and pairwise loss.

Self-Supervision: it is a machine learning paradigm where a model learns to make predictions or generate labels from unlabeled data without human annotation. Instead of relying on external labels, self-supervised learning leverages inherent structures or patterns in the data to create proxy tasks, allowing the model to learn useful representations.

Contrastive Learning: The fundamental idea behind contrastive learning is to encourage similar data points to be closer together in a learned feature space while pushing dissimilar data points farther apart. This approach can help the model capture meaningful patterns and structure in the data.

The primary objective of contrastive learning is to learn a similarity metric or **embedding space** where similar data points have representations that are close in this space, while dissimilar data points have representations that are far apart. It formulates learning as a binary classification problem, where the model distinguishes between positive pairs (similar data points) and negative pairs (dissimilar data points). Contrastive learning requires forming pairs of data points for training. Each pair consists of an anchor data point and another data point, which can be either similar (positive pair) or dissimilar (negative pair). Contrastive learning often uses Siamese neural network architectures where Each subnetwork takes one data point from a pair as input and produces an embedding (a vector representation) for that data point.

In contrastive learning, pairs of data points are formed, consisting of an anchor (A) and a positive (P) or negative (N) data point. The loss function encourages the anchor to be close to the positive and distant from the negative.

Typically the triplet loss and noise contrastive estimation are used as losses.

Some important components of contrastive learning are: data augmentation (for creating noise versions of itself to feed into the loss as positive samples), large batch size (the loss function can cover a diverse enough

collection of negative samples), hard negative mining (involves selecting hard negative pairs with high initial similarity scores for training and these hard negative pairs are then used in the contrastive loss function to emphasize the model's mistakes and encourage better separation between the anchor and hard negatives)

Hard negative mining accelerates the training process by focusing on challenging negative pairs that help the model learn more effectively.

It leads to improved convergence and better representations, as the model becomes more robust to difficult negative examples.

Some issues with contrastive learning are: Negative Sampling (Selecting too-easy negatives can lead to a trivial solution, while selecting too-hard negatives can make training unstable), Hyperparameter Sensitivity, Large Memory and Computation, Data Augmentation Strategies, Negative Pair Imbalance, …

Image based SSL: is a machine learning paradigm in which neural networks are trained to learn meaningful representations from unlabeled image data.

Ex. Rotation Prediction, Jigsaw Puzzle Solving, Distortion, Relative Position, Colorization…

SimCLR - Simple Contrastive Learning Representation: is a self-supervised learning framework developed for training deep neural networks to learn meaningful representations from unlabeled data. SimCLR employs a contrastive learning approach. The core idea is to maximize the similarity between positive pairs (samples from the same data instance) while minimizing the similarity between negative pairs (samples from different instances). SimCLR uses data augmentation techniques to create positive and negative pairs. For each data sample, you create two augmented versions with two different transformations belonging to the same family, treating them as positive pairs. Other samples in the batch act as negative pairs. So if we have N samples, we get 2N samples when applying two transformations on the N data points. In turn each pair is treated as positive samples and the remaining 2(N-1) are negatives.

This encourages the model to learn useful features invariant to various transformations.

A common choice for the architecture is a deep neural network, often based on convolutional neural networks (CNNs) like ResNet or a similar architecture. The network takes input images and maps them to a feature space. The loss used is the contrastive loss which encourages the model to make the embeddings of positive pairs more similar while making the embeddings of negative pairs more dissimilar. SimCLR learns to map input data to a feature space where similar data points are clustered together. The model is trained to maximize a similarity metric between positive pairs and minimize it for negative pairs. Training is done through backpropagation.

BYOL – Bootstrap Your Own Latent: it is a self-supervised learning method designed to learn useful representations from data without the need for explicit labels. BYOL uses two neural networks, each consisting of an encoder. These encoders are often based on architectures like ResNet. One of the encoders is referred to as the "online" encoder, and the other is the "target" encoder. BYOL employs data augmentation techniques to create augmented views of the input data. Each data sample is augmented twice to create two different views. The key idea is to create positive pairs from the two views of the same data sample. One view is processed by the online encoder, and the other is processed by the target encoder. The representations obtained from these two encoders are treated as positive pairs. BYOL doesn't explicitly create negative pairs or use a contrastive loss function. Instead, it leverages a technique called "target network momentum updating" to implicitly create negative-like pairs.

The target encoder is updated slowly over time using a moving average of the online encoder's weights. This moving average helps in creating a contrastive effect by making the target encoder lag behind the

online encoder. As a result, the online and target encoders effectively behave like positive and negative pairs.

BYOL uses a loss function that encourages the online and target encoders to produce similar representations for the two views of the same data sample. This loss is often a cosine similarity loss.

During training, the network learns to map input data to a shared latent space where the representations of the two views from the same data sample are similar.

Barlow Twins: it is another self-supervised learning method for training deep neural networks, specifically designed to learn useful representations from unlabeled data. The core idea behind Barlow Twins is to encourage the neural network to reduce redundancy in the learned representations. Redundancy here means that multiple neurons in the network encode similar information. By reducing redundancy, the network is forced to capture more informative and unique features. Barlow Twins uses a pair of identical neural network encoders, often based on architectures like ResNet. These dual encoders process different views of the same input data. The key innovation in Barlow Twins is the use of a cross-covariance matrix. The network computes the cross-covariance matrix of the feature representations produced by the two encoders. This matrix captures the relationships between the features learned by each encoder. Barlow Twins introduces a novel loss function based on the cross-covariance matrix. The loss function encourages the off-diagonal elements of the cross-covariance matrix to be close to zero, effectively minimizing redundancy between the two encoders' representations. The diagonal elements of the matrix are regularized to prevent them from becoming too small. The method introduces a regularization hyperparameter called "λ" that controls the trade-off between preserving informative features and reducing redundancy. Tuning this hyperparameter is essential to achieving optimal results.

Deep Metric Learning: it is a subfield of machine learning and deep learning that focuses on learning similarity or distance metrics directly from data. The goal of deep metric learning is to train a neural network to embed data points into a continuous vector space in such a way that similar data points are closer to each other, while dissimilar ones are farther apart.

Meta-Learning: it is a concept relying on the learning to learn paradigm. It is a subfield of machine learning that focuses on training models or algorithms to learn from and adapt to new tasks quickly and effectively. The primary idea behind meta-learning is to enable models to acquire knowledge or prior experience from a set of related tasks and then apply this knowledge to perform well on new, unseen tasks. The adaptation process, essentially a mini-learning session, happens during test but with a limited exposure to the new task configurations. The tasks can be any well-defined family of machine learning problems: supervised learning, reinforcement learning, etc. There are 3 main approaches to meta-learning: metric based, model based and optimization based.

A good meta-learning model should be trained over a variety of learning tasks and optimized for the best performance on a distribution of tasks, including potentially unseen ones. Each task is associated with a dataset, containing both feature vectors and true labels. The optimal model parameter are:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D})}[\mathcal{L}_\theta(\mathcal{D})]$$

It looks the same as other tasks but here each sample is a dataset not a single point

The training process is done in order to get :

$$\theta = \arg\max_{\theta} E_{L\subset\mathcal{L}}[E_{S^L\subset\mathcal{D},B^L\subset\mathcal{D}}[\sum_{(x,y)\in B^L} P_\theta(x, y, S^L)]]$$

S_L is the support set, $S_L \subset \mathcal{D}$

B_L is a training batch $B_L \subset \mathcal{D}$

Both of them only contain data points with labels belonging to the sampled label set L, $y \in L, \forall (x,y) \in S^L \times B^L$

The final optimization uses the mini-batch B^L to compute the loss and update the model parameters through backpropagation, in the same way as how we use it in the supervised learning.

Difference between meta learning and transfer learning: Meta-learning focuses on the ability to adapt to new tasks, while transfer learning aims to improve performance on a specific target task by transferring knowledge from a related source task. rather than fine-tuning according to one down-stream task, it optimizes the model to be good at many, if not all.

Difference between meta learning and supervised: Supervised learning focuses on solving specific tasks using labeled data, while meta-learning aims to equip models with the ability to learn quickly from a variety of tasks.

Learner and Meta-Learner: The "learner" refers to the primary model or algorithm that is responsible for solving a specific task. This is the model that receives input data, processes it, and produces predictions or outputs. In a meta-learning context, there are typically multiple "learner" models, each associated with a specific task. The "meta-learner" is a higher-level model or algorithm that learns how to adapt the "learner" models to new tasks quickly and effectively. It is responsible for acquiring knowledge or representations from the "learner" models. The meta-learner's primary role is to generalize across multiple tasks and learn patterns or strategies that can be applied to new, unseen tasks. It can be viewed as a model that learns how to learn.

Metric-Based ML: The key idea behind metric-based meta-learning is to train a model to learn a similarity metric or distance metric that can measure the similarity between data points. This learned metric is then used to make predictions on new, unseen tasks or data points.

Architectures and model used in this approach are: siamese neural network, matching networks, relation networks, prototypical networks, …

Model-Based ML: the goal is to learn an initialization of a neural network model in such a way that it can quickly fine-tune its weights on new tasks with a small number of training examples. This approach is particularly useful in scenarios where it is expensive or impractical to collect large amounts of task-specific data.

Architectures and model used in this approach are: neural turing machines,

Optimization-Based ML: the goal is to develop models that can quickly adapt to new tasks or optimization problems by fine-tuning their optimization procedures, which can lead to more efficient and effective learning.

Architectures and model used in this approach are: LSTM meta learner, Model Agnostic Meta Learner (MAML), …