WHY DO WE NEED TO USE MORE THAN ONE LAYER IN NNs? IN OTHER WORDS, WHAT IS THE ADVANTAGE OF A MULTI-LAYER NN OVER A PERCEPTRON?

A single layer NN, such as the Perceptron, is a binary classifier which acts as a linear model, computing an output based on a linear combination of its inputs. Such architecture is not able to model nonlinear relationships: for example, the XOR function cannot be represented by any linear classifier. Multi-layer NNs overcome this limitation by composing multiple simpler functions. It a multi-layer NN is important to have at least one layer that uses a non-linear function, otherwise if all layers have a linear function the overall function of the network would be equivalent to a linear transformation. Considering the XOR problem, a simple feed-forward NN with one hidden layer containing a non-linear function can solve it. Together with the advantage of the representation of more complex function, multi-layer NNs offer several advantages: they are often more efficient since they can use fewer units than a shallow NN, they have improved generalization, indeed multi-layer NNs tend to generalize better to unseen data; and also, multi-layer NNs can learn hierarchical representation of the data thanks to deep layers.

WHAT IS THE XOR PROBLEM IN THE CONTEXT OF NNs? DISCUSS HOW THE RESOLUTION OF THE XOR PROBLEM HAS INFLUENCED THE DESIGN OF DEEP LEARNING ARCHITECTURES

The XOR problem is an example in NNs that illustrates the limitations of simple, linear models. Indeed, since the function is not linearly separable (the function outputs 1 when exactly one input is 1, and 0 otherwise), a linear and simple model can’t learn this function. If we want to solve this problem we must add at least one hidden layer and use a non-linear activation function (a NN with one hidden layer and RELu as activation function is able to learn the XOR and solve this problem). This highlighted the need of higher depth and non-linearity in the design of deep NNs, demonstrating that deep NNs are more expressive.

WHAT’S THE DIFFERENCE BETWEEN BACKPROPAGATION AND SGD?

Backpropagation is a phase of the algorithm used to train a NN. The training of a NN is divided into 3 phases: forward phase, backpropagation phase and optimization, i.e. updating weights. This last phase is performed by optimization algorithms such as SGD. Hence, backpropagation is a phase of the training process, while SGD is an optimization algorithm used for the update of the weights. The goal of the backpropagation is to compute the gradient of the loss function with respect to the parameters in an efficient way using the chain rule. In order to do this, the backpropagation starts from the last layer, and it goes backward computing the derivatives of the loss functions that compose the gradients. Once the gradient has been computed, an optimization algorithm can use it to minimize the loss function. In particular, the SGD minimizes the loss function using only one random example of the training set to approximate the gradient, making the update quicker.

HOW CAN THE MAXIMUM LIKELIHOOD PRINCIPLE BE APPLIED TO SELECT THE MOST APPROPRIATE ACTIVATION FUNCTION FOR NEURONS IN A NN?

The ML principle is not typically applied to select the activation function, but it guides the choice of the cost function of a NN, and consequently, the appropriate activation function for the output layer, depending on the type of the variable the network is trying to predict. By assuming a probability distribution for the network’s output, ML helps define the loss function and the negative log-likelihood of the predicted output, which in turn determines the correct activation function for the output neurons. For example, the ML says that: for binary classification, assuming a Bernoulli distribution leads to using the sigmoid activation with binary cross-entropy loss; for multi-class classification, assuming Multinoulli distribution leads to using SoftMax with categorical cross-entropy loss; while for regression tasks, assuming a Gaussian distribution leads to using a linear activation with mean squared error loss. Hence, the ML principle guides the selection of an activation function for the output layer, while the selection of an activation function for hidden layers is based on empirical performance and optimization behaviour and thus is typically chosen through validation.

EXPLAIN WHAT’S DROPOUT, HOW IT’S IMPLEMENTED IN BOTH TRAINING AND EVALUATION, AND WHY DOES IT ACT AS A REGULARIZATION

Dropout is a regularization technique used to reduce overfitting by randomly “dropping” units (i.e. setting them to zero) during training applying a binary mask. This prevents the network from becoming overly reliant on specific neurons and encourages the learning of more robust and generalizable features. During the training phase dropout randomly removes a subset of neurons in the input and hidden layers by applying a binary mask (the probability that a unit is or is not in the network is an hyperparameters). This mask changes with each training step and is applied independently for each unit, and only the retained units participate in forward and backward passes. During evaluation, dropout is turned off, and the full network is used. To compensate for the dropped unit during training, weights are scaled by the dropout probability. This process acts as regularization because it improves generalization and reduces overfitting by making the network more robust and less sensitive to specific weights or features.

EXPLAIN WHAT'S REGULARIZATION, WHY IT'S IMPORTANT, AND GIVES SOME EXAMPLES OF REGULARIZATION TECHNIQUES

Regularization refers to a collection of techniques designed to improve a model’s generalization ability, i.e. it’s performance on unseen data, by reducing overfitting. Overfitting occurs when a model performs well on training, but it’s not able to generalize when tested on unseen data. Regularization addresses this by intentionally introducing constraints or modifications during training, preventing the model from becoming overly complex. In addition to reducing overfitting, regularization is crucial in several situations, like when the models are very large, or when the training data is limited, or when the problems are ill-posed. There are many effective regularization techniques like L2 regularization which adds a penalty based on the square of the weights, or L1 regularization which encourages sparsity. Other techniques are dropout, early stopping, data augmentation, noise injection, batch normalization, sparse representation, ensemble methods, etc.

EXPLAIN WHAT ARE SEMI-SUPERVISED LEARNING, TRANSFER LEARNING, MULTI-TASK LEARNING, SELF-SUPERVISED LEARNING

All these kinds of learning are paradigms of modern deep learning which aim to make better use of available data or improve model generalization. Semi-supervised learning is used when we have only a small amount of labelled data but a large pool of unlabelled data, and instead of solely rely on labelled data, the model learns also from unlabelled data. Transfer learning focuses on reusing knowledge learned from one task to improve performance on a different but correlated task. Multi-task learning takes a different approach: instead of training one task at a time, it trains multiple tasks simultaneously sharing parts of the model across all tasks while keeping some task-specific components. This approach encourages the model to learn general features that are useful for multiple tasks. Self-supervised learning, instead, is used when no labelled data are available, thus the idea is to create artificial labels from the data itself by designing a pretext task like for example predicting missing parts of an image.

WHAT ARE THE MAIN PROBLEMS IN OPTIMASING DEEP NNs? WHY ARE THEY A PROBLEM FOR OPTIMIZATION ALGORITHMS? HOW IS IT POSSIBLE TO AVOID SUCH PROBLEMS?

Optimization refers to the task of finding the parameters of a model that minimize a chosen cost function to achieve a good performance on unseen data. Optimizing deep NNs is difficult due to the cost functions that are involved which typically are non-convex functions. This non-convexity leads to several problems: Ill-conditioning (i.e. the hessian matrix can be ill-conditioned, hence the condition number is big and the function’s curvature varies greatly), local minima and saddle points, cliffs and exploding/vanishing gradient (i.e. the function can have extremely steep regions and the function can become very small) and inexact gradient (i.e. optimization algorithms often have not access to the exact gradient). Ill-conditioning is a problem for first order methods because it causes the algorithm to waste time moving back and forth across directions of high curvature, but the problem can be solved by using second order methods. Local minima can trap optimization algorithms preventing them from reaching parameters with lower cost, however as long as the loss function value is not much different from that in the global minima, they are not a problem. Saddle points are characterized by a mixture of positive and negative eigenvalues of the hessian, and they are a problem for second order methods. To solve this problem, we can use hessian-free optimization methods. Cliffs and exploding gradient leads to excessively large parameters while the vanishing gradient problem leads to excessively small parameters. These problems can be solved by using gradient clipping, hessian-free optimization, implementing LSTM or GRU units or reservoir computing architectures. Finally, inexact gradient problem can be solved by a proper parameters initialization.

EXPLAIN THE MAIN CHARACTERISTICS AND DIFFERENCES AMONG: GD, MINI-BATCH SGD, ADAGRAD, RMS PROP AND ADAM

GD is a batch gradient method that processes the entire training set simultaneously, thus it computes the gradient of the loss function on the whole training set and then update the weights. By doing this, GD computes the exact gradient of the loss. Unlike GD, mini-batch SGD computes the gradient of the loss function on a subset of the training examples called mini-batch and then updates the weights. Adagrad adapts the learning rates of model parameters inversely proportional to the squared root of the sum of all historical squared values of the gradient of each parameter. RMSprop addresses the aggressive learning rate decay of Adagrad using an exponential moving average of squared gradients instead of the cumulative sum. Adam can be seen as the combination of RMSprop with momentum; thus, it uses adaptive learning rates for each parameter.

WHAT ARE LOCAL MINIMA AND SADDLE POINTS, AND WHY ARE THEM A PROBLEM FOR OPTIMIZATION ALGORITHMS?

Local minima are points where the function value is lower than at all neighbouring points making no longer possible to decrease the function’s values with infinitesimal steps. They represent a problem because optimization algorithms aim to follow the negative gradient to decrease the function value, but in these points the gradient is zero and thus it does not provide any information about the direction to follow. This can cause the algorithms to stop at these points even if they are not global minima (desired points), but if the NN is large enough, most of local minima might have a low-cost function value, making them suitable. Saddle points are another critical point where the gradient is zero, but unlike local minima, they have neighbours that are both higher and lower the saddle point itself. They represent a problem for both first and second order methods. First order methods get stuck in these points since they are not able to find the correct direction, while second order methods are attracted by critical points and thus, they can jump from a saddle point to another saddle point.

DESCRIBE THE PROCESS OF CORRECTLY ESTIMATING THE TRUE ERROR OF A DEEP NN. DISCUSS THE CHALLENGES INVOLVED IN THIS ESTIMATION AND THE METHODOLOGIES USED TO ADDRESS THESE CHALLENGES

Estimating the true error of a deep NN is essential to evaluate its ability to generalize to unseen data. The true error represents how often the model will mistake on new unseen data. However, since the distribution, from which we drawn new examples, is unknown, the true error cannot be computed directly, introducing several challenges. Since our aim is to minimize the true error, but it is not computable, we minimize the empirical error (empirical risk minimization). However, this process may not be the best option for achieving low true error. The estimation of true error has several challenges like overfitting (the model achieves low error on training data, but the true error is too large), model capacity, computational cost since evaluating the model on the entire dataset to compute the exact gradient is very expensive, hyperparameter choices and bias. To address these challenges, the standard methodology is to divide the data into three sets: training set (for learning model parameters), validation set (for hyperparameters tuning and model selection) and test set (for final performance evaluation). If the dataset has a low number of examples we can use k-cross validation, which divides the dataset into k subsets, and the model is trained k times. In each trial a subset serves as test set and the other k-1 serve as training set. Other techniques to overcome these challenges are regularization, hyperparameter tuning and debugging strategies.

HOW CAN ONE APPROPRIATELY SELECT AN APPROPRIATE NUMBER OF HIDDEN LAYERS AND THE CORRESPONDING ACTIVATION FUNCTION IN A DEEP NN? WHY IS IT CRUCIAL TO ACCURATELY EVALUATE THE MODEL'S PERFORMANCE, AND HOW CAN THIS BE ACHIEVED?

While a single layer can theoretically approximate any continuous function, deeper networks often offer practical advantages such as more efficient parameter usage, better generalization and hierarchical feature learning. However, deeper networks come with challenges such as difficulty in the training process, vanishing or exploding gradient, and in general optimization issues. Thus, if we want to find the optimal depth of the network we must empirically experiments different depths, and we must follow validation performances. As for the depth, also for the activation function choice, which introduces non-linearity in the model, we must look at evaluation performances of different types of activation functions, since the choice of the correct one depends on the specific task we want to solve. Hence, a correct evaluation ensures that the model will perform well on unseen data, and it also prevents overfitting/underfitting, implementation errors, parameters tuning and it helps us to adjust the architecture. If we want to accurately evaluate our model, we can use appropriate metrics (accuracy, F1-score, …) split the data, model selection phase (train several architectures), use cross-validation if data is limited, debug and monitor.

EXPLAIN WHAT ARE THE PROBLEMS OF INITIALIZING ALL THE WEIGTHS OF A NEURAL NETWORK TO THE SAME CONSTANT VALUE

Initializing all the weights of a NN to the same constant, creates a critical issue known as the symmetry problem which compromises the network’s ability to learn. The problem is that, with all weights initialized equally, all the neurons in the same layer will behave identically and they will update their weights in the same way, failing to break the symmetry problem. Thus, they will learn all the same feature, limiting the model’s expressive power. To train a NN effectively, each neuron must learn to extract different features, and for this reason the weights should be initialized randomly, breaking the symmetry problem.

WHAT IS THE APPROXIMATION THEOREM?

The approximation theorem states that a feedforward network with a linear output layer and at least one hidden layer using a “squashing” activation function can approximate any measurable function with any desired non-zero amount of error, provided the network has enough hidden units. In simpler terms, the theorem means that regardless of what function we are trying to learn, we know that a large multi-layer perceptron will be able to represent this function. However, the theorem comes with some challenges: size of the network, indeed the theorem does not state how much large the network should be; learnability, indeed the theorem does not guarantee that the network will be able to learn the target function.

WHAT IS THE FREE LUNCH THEOREM?

The No Free Lunch Theorem states that when averaged over all possible data generating distributions (i.e. all possible problems), every classification algorithm has the same error rate when classifying previously unobserved points. This means that no machine learning algorithm is universally any better than any other. The theorem implies that to achieve good performance, we must design a machine learning algorithm able to perform well on our task. As the theorem says that there is not the best algorithm, there is no best form of regularization.

DETAIL THE MAIN PROPERTIES OF CONVOLUTIONAL OPERATOR AND EXPLAIN, FOR EACH ONE, THE DIFFERENCE COMPARED TO A FULLY CONNECTED LAYER

The convolutional operator is fundamental component of CNNs, especially effective for processing data with grid-like structures such as images. Unlike fully connected layers, conventional layers possess distinct structural and computational properties. The main property is sparse interactions, which means that each output unit is influenced by only a small local region of the input (receptive field) thanks to the use of small filters (kernels). This represents the first difference with a fully connected layers where all inputs are connected to all outputs. Another key property, that a fully connected layer does not have, is equivariance to translation, which means that if the input is shifted, the output shifts as well, preserving the spatial structure. Other properties are parameter sharing, which means that the same set of filters is applied across different spatial locations of the input allowing the model to detect the same feature regardless of its position (fully connected layers assign unique weights to each input-output pair), and the ability of the convolutional operator to handle inputs of variable size, while fully connected layer can only handle fixed-size inputs.

DESCRIBE WHAT IS A GRAPH CONVOLUTIONAL NETWORK AND WHAT ARE ITS MAIN PROPERTIES

A Graph Convolutional Network is a specialized type of NN designed to work directly on graph-structured data. The main goal of a GCN is to learn meaningful representations for each node by aggregating and transforming features from its neighbours, layer by layer. As the name suggests, this kind of network uses the convolution operator, adapted to the irregular structure of graphs. Using the convolution operator, the GCN inherits its main properties such as sparse interactions (a node’s representation depends only on its neighbours), permutation equivariance and invariance (respect of the permutation of nodes, if we reorder the nodes of the input graph, the output reorder the nodes accordingly), parameter sharing (use of the same set of weights across all nodes, regardless of their position in the graph).

WHAT IS A GRAPH CONVOLUTIONAL NN? EXPLAIN HOW IT IS FORMULATED IN THE GRAPH SPECTRAL DOMAIN AND IN THE NODE DOMAIN

A GCN is a NN architecture specifically designed to work with data represented as graphs. The main goal of a GCN is to learn meaningful representations for each node by aggregating and transforming features from its neighbours, layer by layer. As the name suggests, this architecture uses the convolutional operator adequately adapted to the irregular structure of graphs. GCNs can be formulated in two different domains: spectral domain and node domain. In the spectral domain, graph convolution is defined based on the graph Fourier transform, where the convolution corresponds to the element wise multiplication of Fourier-transformed signals and the filter. Instead, the node domain can be seen as an approximation and implementation, through Chebyshev filters, of the spectral formulation since the operators are more intuitive and efficient.

IN THE CONTEXT OF SEQUENTIAL TRANSDUCTIONS, GIVE THE DEFINITION OF CAUSALITY AND DISCUSS HOW THIS CONCEPT IS IMPLEMENTED IN RNN. ARE ALL RNN ARCHITECTURES CAUSAL?

In the context of sequential transduction, causality is defined as the property where the output at time t does not depend on future inputs (at time t+1, t+2 …), thus a transduction T() is said to be causal if this condition holds. The meaning behind this condition is that the model has no access to future information but only to current or previous states. RNNs are a type of model used for sequential learning, and they often employ a recursive state representation where the input at time t (o(t)) depends on the current hidden state (h(t), which represent the history of past inputs) and potentially the current input (x(t)). A recursive state representation in a RNN exists only if the transduction T is causal, but not all RNN architectures are strictly causal.

GIVE THE DEFINITION OF SEQUENTIAL TRASDUCTION, EXPLAINING THE CONCEPT OF MEMORY, CAUSALITY AND RECURSIVE STATE REPRESENTATION.

A sequential transduction is defined as a function that maps an input sequence with labels from a space X to an output sequence with labels from a space O. This can be denoted as T: T\* -> O\*, where X\* is the set of all sequences with labels in X. The concept of memory refers to how much of the past the model needs to consider in order to generate the current output. If a transduction has finite memory of size k, it means that the output at time t depends only on the current input and the previous k inputs. A transduction is said to be causal if the input at time t depends only on the current or previous inputs and not on the future ones. To implement the concept of causality, models often use a recursive state representation, i.e. the model has a hidden state h(t) that keep tack the history of past inputs up to time t. The current state is hence updated based on the state h(t) and the current input, and the output is computed using a function g on h(t) and the current input.

DISCUSS THE DIFFERENCES BETWEEN BACKPROPAGATION TT AND REAL TIME RECURRENT LEARNING

Backpropagation TT and RTRL are two algorithms designed to compute gradients in RNNs, BPTT operated by unfolding the RNN over time into a deep feedforward network and then it performs the forward pass across the entire sequence and then applies backpropagation backward through these time steps waiting till the end before computing the gradient. This approach has time complexity that scales with both the number of neurons and the sequence length. Instead, RTRL calculates the gradient during the forward pass. As each input arrives, the algorithm incrementally updates the partial derivatives of the hidden state w.r.t. the model’s parameters. This approach comes with computational cost; indeed, the time complexity scales cubically with the number of hidden units. In summary, BPTT is more efficient for batch training on fixed length sequences, while RTRL is more suitable for online learning.

EXPLAIN WHAT A RESERVOIR COMPUTING NETWORK IS. WHAT ARE THE MAIN FEATURES AND PROPERTIES THAT SUCH A MODEL OWNS?

A reservoir computing network is a type of RNN architecture designed to process sequential data in an efficient and simple way. Unlike standard RNNs, which adjust their weights during learning, reservoir computing models (echo state networks or liquid state machines) only train the output weights, while the rest of the network (the so called “reservoir”) remain fixed after the initialization. By doing this the reservoir wants to avoid the instability and complexity often encountered during training. To ensure the reservoir produces useful dynamics, it must satisfy the echo state property, i.e. the influence of any previous input should vanish over time, helping the network to remain stable and not “remember” input forever. Moreover, a good reservoir should be big, and it should be sparse and randomly connected.

DEFINE RNN. EXPLAIN HOW IT IS USED FOR AND HOW IT CAN BE TRAINED

A RNN is a type of NN architecture specifically designed to handle sequential data. Unlike traditional feedforward NNs, RNNs have cyclic connections in their computational graph, allowing information to persist across time steps. This is achieved through a hidden state that is updated recursively based on the previous hidden state and the current input. The hidden state serves as a form of memory, summarizing relevant information from past sequences. The training phase of RNNs usually involves algorithms such as BPTT and RTRL. However this phase comes with some issues such as exploding gradient and vanishing gradient, which make long time dependencies difficult to learn. To solve these problems, we can use some gated RNN variants like LSTMs and GRUs, which are architecture that used gating mechanisms to control the flow of information and help preserve gradient over long dependencies.

DEFINE A BIDIRECTIONAL RNN

A BRNN is a type of RNN designed for tasks where the desired output at a given time step may depend on future inputs in the sequence, as well as past and present inputs. The dependency of the current input on past and future sequences means that this kind of RNN architecture does not implement the concept of causality. A BRNN combines two RNNs, one that processes the input forward in time, while the other one processes the same input backward in time. This combination produces a hidden state that summarizes past history and a hidden state that summarizes future history. The output is then computed by combining both forward and backward hidden states at time t.

FORMALLY DEFINE A SHALLOW RNN WITH FEEDBACK FROM OUTPUT. EXPLAIN ALSO HOW TEACHER FORCING CAN BE IMPLEMENTED

A shallow RNN with feedback from output is a type of RNN in which the computation of the hidden state at each time step not only depends on the current input and the previous hidden state but also on the output from the previous time step. Teacher forcing is a training technique used to improve the stability and effectiveness of training in RNNs with output feedback. Instead of feeding the model’s predicted output into the next step, teacher forcing substitute the true output from the training data. This ensures the model is trained using correct context, preventing the compounding of prediction errors during sequence generation.

INTRODUCE THE LSTM UNIT AND EXPLAIN THE ROLE OF ITS COMPONENTS

LSTM units are a specialized type of RNN designed to overcome the vanishing gradient problem of traditional RNN. LSTM overcomes this by adding a cell state (memory of LSMT designed to retain information across time) with self-loops and gating mechanism that enable the model to decide what information to keep, forget and output over time. Indeed, LSMT implements three gates: forget gate, which determines how much of the previous memory should be retained or discarded, input gates, which regulates how much new information should be added to the memory, and output gate, which decides how much of the updated cell state should be output. Each of these gates uses sigmoid functions.

DESCRIBE IN DETAIL THE DIFFERENCE BETWEEN LSTM AND GRU

LSTMs and GRUs are two types of RNNs specifically designed to handle limitations of standard RNNs such as exploding and vanishing gradient. Both architectures address these limitations by adding gating mechanisms that regulate the flow of information through the network. LSTM introduces a cell state and three gated units: input gate, which controls what new information is added to the cell state, forget gate, which decides how much of past information should be discarded, and output gate, which determines how much of the current memory is used to produce the output. In contrast, GRU do not maintain separate cell state, instead it operates directly on the hidden state. Moreover, GRU uses two gates: update gate, which combines the roles of input and forget gate of LSTM, and the reset gate, which determines how much of the past should be ignored when computing the new state. This architecture difference results in a less complexity of GRUs compared to LSTMs

DESCRIBE WHAT TRANSFORMERS ARE IN DEEP LEARNING DISCUSSING THEIR KEY PROPERTIES

Transformers are a type of NN architecture particularly suited for processing sequential data and widely used for tasks involving language. Transformers use an encoder-decoder structure where the encoder processes the input sequence into a contextualized representation, while the decoder uses that representation to generate the output sequence. Both are composed by repeated blocks containing two main components: attention mechanism and feedforward network. There are different types of attention mechanisms. Self-attention mechanism which allows the model to weight the importance of different elements in a sequence enabling each token to consider the entire sequence at once and to process data sequentially. Multi-head attention mechanism where multi attention heads work in parallel, each learning to focus on different parts of the sequence. Moreover, transformers incorporate positional encoding which makes the model understands the order of tokens in a sequence. However, transformers come with computational and memory limitations, in fact the computational and memory cost scales quadratically with the sequence length. In addition, transformers require large dataset to perform well.

DEFINE IN DETAIL THE ATTENTION MECHANISM IN A TRANSFORMER. MOREOVER, EXPLAIN THE DIFFERENCE BETWEEN SELF-ATTENTION AND CROSS-ATTENTION

The attention mechanism has been introduced by the transformer models, where it enables them to dynamically focus on different parts of an input sequence when producing outputs. In formal terms, the attention mechanism operates on queries, keys and values, and each output is a weighted sum of the values where the weights are computed by a function that uses the queries and the keys. The transformers specifically use the dot product as function to compute the weights. To allow models to attend to information from different representation subspaces, it has been introduced the multi-head attention which replicates the attention process multiple times in parallel. There two types of attention. self-attention, where the queries, keys and values all come from the same sequence, allowing the model to learn dependencies between different positions within a single input. When we use self-attention, in the encoder it allows each token to attend to every other token, while in the decoder is masked to prevent attending to future tokens. In contrast, cross-attention involves queries from the decoder and keys and values from the encoder, allowing outputs’ token to reference relevant parts of the input.

DESCRIBE GPT MODELS AND LIST THEIR PROPERTIES

Generative Pretrained Transformers are a class of deep learning models specifically designed for processing and generating sequential data. They are based on transformer architecture, but unlike the encoder-decoder structure as transformer, GPT models use only the decoder stack. As main property, GPTs leverage masked self-attention, allowing each token to attend only to previous tokens in sequence. This means that the output is generated one word at a time coherently. Their structure includes multi-head self-attention and feedforward network. In addition, as transformers, GPTs us positional encoding. The training of GPT models is divided into two phases: unsupervised pre-training where the model learns to predict the next word, and supervised fine-tuning where the model can be fine-tuned on specific tasks using labelled data. A particularity of GPT models is their ability to perform few-shots learning (0-1 shot learning).

WHICH ARE THE MAIN DIFFERENCES BETWEEN TRANSFORMERS AND GPT MODELS?

Transformer is a general purpose NN architecture designed for processing sequential data. In its structure includes an encoder and a decoder stack making it particularly powerful for sequence-to-sequence tasks. Each layer of encoder and decoder stacks integrates multi-head self-attention and feedforwards layers. In contrast, GPT models represent a specific adaption of the transformer architecture that uses only the decoder stack modified with masked self-attention ensuring that each token in the sequence can only attend to previous tokens. Another difference is the training process; while transformers can be trained in different way depending on the task, GPTs follow a specific procedure that include an unsupervised pre-training phase (predict the next token given the post ones) and then a supervised fine-tuning on a specific task.

DEFINE WHAT AN AUTOENCODER IS

An autoencoder is a typer of NN designed to learn efficient representations of data by attempting to reconstruct its input at the output. It is composed of two main parts: an encoder, which compresses the input into a latent representation, and a decoder, which reconstructs the original output from the latent representation. Obviously, the objective of the training phase of these models is to minimize the difference between the original input and the reconstruction. Since, simply learning how to perfectly copy the input is not useful, the model is encouraged to learn salient features by adding some constraints that avoid the model learning trivial features. Based on the type of constraints or regularizations techniques used, there are different types of autoencoders: undercomplete autoencoders, sparse autoencoders, denoising autoencoders, contractive autoencoders, VAEs. Usually, autoencoders are used for dimensionality reduction, feature learning and generative modelling.

DEFINE A CONTRACTIVE AUTOENCODER. EXPLAIN THE DIFFERENCES WITH RESPECT TO UNDERCOMPLETE AUTOENCODER

A contractive autoencoder is a type of overcomplete autoencoder that learns robust representation by introducing a regularization term (Frobenius norm) that penalizes the sensitivity of the encoder to small changes in the input. This makes the model learn more stable and invariant features, focusing on capturing the structure of the data. The differences between overcomplete and undercomplete autoencoder lie in how they enforce useful features learning. In fact, a undercomplete autoencoder imposes structural constraint by using a hidden layer that has fewer dimensions than the input, forcing the model to compress the input itself and focusing only on the most essential information. In contrast, a contractive autoencoder can use equal or even higher dimensional hidden layer, but it introduces a regularization penalty. In short, undercomplete autoencoders compress the data by reducing dimensionality, while contractive autoencoders introduce a regularization term.

DEFINE A SPARSE AUTOENCODER. EXPLAIN THE DIFFERENCES WITH RESPECT TO A UNDERCOMPLETE AUTOENCODER

A sparse autoencoder is type of overcomplete autoencoder (may have hidden layer with equal or higher dimension than the input) that learns useful representations by enforcing sparsity in its hidden layer. This is done by adding a regularization penalty, such as L1, to the loss function which encourages most of the hidden units to be inactive for any given input. The idea is to push the network to activate only a few neurons per input, leading it to discover distinctive and informative features. In contrast, an undercomplete autoencoder relies on architectural constraints, specifically it has hidden layer smaller than the input forcing the network to compress the image and learns more salient features.

DEFINE A DENOISING AUTOENCODER. EXPLAIN THE DIFFERENCES WITH RESPECT TO A UNDERCOMPLETE AUTOENCODER

A denoising autoencoder is a type of overcomplete autoencoder that is trained to reconstruct clean inputs from corrupted versions. In fact, during training, noise Is added to the original input data, and the autoencoder is trained to reconstruct the original input from its corrupted version. Thus, the objective is to minimize the difference between the reconstruction and the original input, typically using a loss like the squared error. In contrast, an undercomplete autoencoder imposes structural constraints. It has hidden layer that has smaller dimension than the input, forcing the model to compress the input itself and trying to learn more salient features. Hence, the difference lies in how the two kinds of autoencoders are constrained: undercomplete ones have structural constraints, while denoising ones rely on training objective constraints.

DEFINE AN UNDERCOMPLETE AUTOENCODERS AND LIST SOME EXAMPLES OF THIS KIND OF ARCHITECTURE

An undercomplete autoencoder is a type of NN designed to learn compressed representations of data by limiting the size of its hidden layer (called code layer) to be smaller than the input dimension. This structural bottleneck forces the network to prioritize and capture only the most important or salient features of the input. The model’s decoder, thus, is encourages to reconstruct the input from its compressed hidden representation, minimizing the reconstruction loss. There are several examples of undercomplete autoencoders: linear autoencoders, that behaves similarly to PCA, are linear and use mean squared loss, shallow autoencoders, that have a single hidden layer, deep autoencoders, that have multiple hidden layers.

WHAT IS THE DIFFERENCE BETWEEN PROBABILISTIC MODELS REPRESENTED BY A DIRECTED GRAPH AND BY AN UNDIRECTED GRAPH? EXPLAIN THE PROS AND CONS OF EACH REPRESENTATION

A directed graphical model uses arrows to indicate the direction of dependence between variables (using arrows); thus, each variable is modelled through its conditional probability given its parents in the graph. This approach naturally reflects causal relationships, and it allows for efficient sampling using ancestral sampling and when each node has few parents, they offer parameter efficiency and computational scalability. However, this approach struggles with conditional sampling, and it can be lossy in terms of the conditional independence information. In contrast, an undirected graphical model represents symmetric relationships between variables, indeed there’s no notion of direction. This makes these model more suitable for cases where causality is unclear or bidirectional. These models can represent independence structures; they also provide flexibility in factor representation. However, with these kinds of models, compute the partition function is often intractable. Moreover, sampling is computationally intensive, requiring methods like Gibbs sampling.

DEFINE IN DETAIL THE CONTRASTIVE DIVERGENCE (CD) ALGORITHM. IN ADDITION, EXPLAIN WHAT IS THE DIFFERENCE AND MOTIVATION FOR SUCH DIFFERENCE, BETWEEN CD AND PERSISTENT CD

Contrastive Divergence is an algorithm for training undirected probabilistic models (such as RBMs), when computing the exact gradient is infeasible due the intractability of the partition function. CD provides an efficient approximation to this gradient by focusing on the difference between two expectations: one under data distribution and one under model distribution. In practice, the algorithm works as follows: it starts by initializing a Markov chain using actual data samples, then, instead of running the chain until convergence, it performs only a few steps, and the gradient is approximated using the statistics from the initial data and the short-run samples. However, this process has a main limitation: the gradient approximation that does not explore the full distribution. This problem is addressed by Persistent CD which changes how the samples are generated when running the chain. In fact, instead of reinitializing the chain from data every time, PCD maintains a persistent set of samples across training interactions. This change leads to an improve exploration of the distribution.

FORMALLY DEFINE ENERGY BASED MODELS. EXPLAIN WHY THEY ARE USEFUL AND GIVES AT LEAST ONE EXAMPLE OF ENERGY BASED MODEL

Energy-based models (EBMs) are a type of structured probabilistic model, specifically an undirected model, used to describe probability distribution. EBM defines an unnormalized probability distribution using an energy function, and to obtain a valid probability distribution, this unnormalized probability must be normalized by a partition function, defined as a sum or integral over all possible values of x. EBMs are useful because they guarantee non-zero probabilities, the energy function can be any real-valued function, when the energy function is composed of multiple additive terms, each term can be seen as an enforcement of a soft constraint, the gradient of the log-partition function can be approximated using Monte Carlo methods. An example of EBM is Restricted Boltzmann Machines (RBMs) which consists of two layers of units: visible units and hidden units, and the connection are only between the visible units and the hidden units.

EXPLAIN IN DETAIL WHAT IS THE ROLE OF MONTE CARLO CHAINS IN THE TRAINING OF A STOCHASTIC NN. GIVE AN EXAMPLE OF A NEURAL NETWORK MODEL WHERE MONTE CARLO CHAIN ARE USED

Monte Carlo chains, in particular Markov Chain Monte Carlo (MCMC) methods, are training methods used when we are using stochastic NNs (probabilistic models) where distributions are complex and difficult to sample from directly. They are used to approximate expectation and gradients that are otherwise intractable due to the presence of unknow normalizing constants. An example of NN where we can use MCMC methods is RBMs where training requires computing the gradient of the log-likelihood (the partition function sums over an exponential number of configurations) which is infeasible. To overcome this problem, we can use Gibbs sampling, which approximates the distribution of the model and estimates the gradient.

WHAT IS A MONTE CARLO CHAIN? EXPLAIN WHY IT IS AN IMPORTANT CONCEPT. DESCRIBE AN ALGORITHM FOR SAMPLING FROM A MONTE CARLO CHAIN

A Monte Carlo Chain, also known as MCMC method, is a technique used to sample from complex probability distributions when direct sampling is computationally infeasible. The idea is to construct a Markov Chain (sequence of random variables) where each new sample depends only on the current state, according to a defined transition distribution. As the chain progresses over many steps, it converges to a stationary distribution that ideally matches the target distribution. This concept is important particularly when we are dealing with undirected probabilistic models, since these models define distributions that are intractable to normalize, and thus MCMCs are used to approximate the expectations, such as the gradients. A widely used MCMC method is Gibbs sampling, which starts initializing the variables randomly, then it iterates over each variable, and for each variable it samples a new value from its conditional distribution given the current values of all other variables. Then the method updates the variable with the new value and repeat the process. After a burn-in period, the chain’s state approximates a sample from the desired distribution.

DISCUSS IN DETAIL WHAT ARE THE PROBLEMS WHEN RUNNING A MARKOV CHAIN

One of the main issues when running a Markov Chain is the unknow convergence time (burn-in). While the theory guarantees that a properly constructed Markov Chain will converge to the target distribution, in practice it’s nearly impossible to know how many steps the process requires. Moreover, even if the chain has reached the equilibrium, there’s another problem, i.e. correlated samples. In fact, since each new sample depends on the previous one, consecutive samples may not be statistically independent. Another problem, more common in high-dimensional or structured distributions, is slow mixing, i.e. the chain that may get stuck in one region and fail to explore others effectively. Finally, a further complication is the intractability of posterior distribution. Indeed, computing exact posterior distribution is often impossible due to high complexity, and thus, even if MCMC methods provide a practical way to approximate it, poor choices in the number of steps or learning rate can lead to instable training or convergence to inaccurate models.

EXPLAIN WHY IMPORTANCE SAMPLING IS USEFUL

Importance sampling is a Monte Carlo technique used to approximate expectations, integrals or sums that are otherwise computationally intractable. It is useful because if we use it, we can perform these approximations even when direct sampling from the target distribution is not feasible. In practice important sampling allow to compute expectations under a difficult distribution by instead sampling from a simpler and more tractable one (proposal distribution) and then rewriting the sample to account for this difference. Moreover, each sample is assigned a weight proportional to the ratio of the two distributions. By using this technique, we can reduce the variance of gradients in optimization, accelerate the training of large models, approximate the partition function, … However, choosing the right proposal distribution is essential for a correct execution of this methodology.

EXPLAIN THE MEANING OF THE ACRONYM ELBO, GIVING ALL TECHINICAL DETAILS RELATED TO IT. IN ADDITION, EXPLAIN WHY ELBO IS IMPORTANT;  
DESCRIBE IN DETAIL THE LOSS FUNCTION USED IN VARIATIONAL AUTOENCODERS

The acronym ELBO stands for Evidence Lower Bound, also known as variational lower bound, and it plays an important role in variational inference when training deep generative models such as VAEs. VAEs are probabilistic models, and as such they deal with latent variables, and they would ideally maximize the log-likelihood which is often intractable. ELBO’s goal is just that, provides a solution by offering a lower bound on this log-likelihood. To derive the ELBO, a model introduces an approximate posterior distribution which aims to approximate the true one. Moreover, as the name suggests, ELBO provides a lower bound to the true log-likelihood. In practice, ELBO has two main terms: reconstruction term, which encourage the decoder to generate data close to the input, and a regularization term, which ensures that the approximate posterior distribution stays close to the prior one. ELBO is important because it enables leaning when likelihood is intractable, indeed it provides an approximation, then is supports gradient-based learning because ELBO is typically differentiable, it allows for approximate inference, it encourages useful latent representations, and it helps to prevent overfitting.

DEFINE A VARIATIONAL AUTOENCODER. EXPLAIN HOW IT IS USED FOR AND HOW IT CAN BE TRAINED

A VAE is a deep generative model that extends the traditional autoencoder by incorporating probabilistic inference and latent variable modelling. It’s main goal is to learn both how to represent data in a latent space and how to generate new data points from this space. VAE consists of two main components: the encoder, which approximates the intractable true posterior distribution with a tractable distribution (using ELBO), and it implement a NN that learns to output the parameters of the approximate posterior distribution; and the decoder, which defines how latent variables generate data. This kind of autoencoders is used for several tasks: data generation, learning meaningful representation, manifold learning and hierarchical learning. The training process of such model relies on optimizing the ELBO, which is the tractable lower bound on the true intractable log-likelihood and it can be training using gradient-based optimization. Moreover, since VAE is a probabilistic model, the training phase uses MCMC method to estimate expectations.

DESCRIBE THE DIFFERENCES BETWEEN VAEs AND DENOISING AUTOENCODERS

Denoising autoencoders are designed to learn robust feature representations by training the model to reconstruct clean input data from a corrupted version. This denoising task forces the model to capture the underlying structure of the data distribution, helping it learns meaningful internal representations. In contrast, VAEs are probabilistic models, they learn a latent variable model where the encoder maps input into a distribution over latent variables, and the decoder generates outputs from these latent variables. The training procedure is done by minimizing the ELBO loss. Hence, DAEs focus on robustness and representation through denoising, without explicit generative modelling, while VAEs are designed to for probabilistic generation of new data and offer a principled approach to learning latent variables.

WHAT IS A DIFFERENTIABLE GENERATOR NETWORK? WHY IT IS USEFUL? GIVE A SIMPLE EXAMPLE OF A DGN

A DGN is a NN designed to map samples from a simple latent distribution, typically random noise, to sample that resemble a target data distribution. It acts as a transformation function, turning latent variables into outputs or into parameters of a distribution over the output. This kind of NN offers several advantages: DGNs allow to start from a simple latent space and learn a non-linear transformation that generates samples closely resembling real data; since DGNs are differentiable, they can be trained using gradient-based algorithms; DGNs serve as the decoder in VAEs, and as generator in GANs; finally DGNs help discover meaningful representations of data in the latent space. An example of DGN can be the generator in GANs or the decoder in VAEs.

DEFINE A GAN. EXPLAIN WHY IT IS USEFUL AND HOW IT CAN BE TRAINED;  
WHY A GAN IS USEFUL? DESCRIBE AND EXPLAIN THE MATHEMATICS OF ITS TRAINING ALGORITHM

A Generative Adversarial Network is a generative modelling approach built upon the concept of DGNs and that relies on a game between two NNs: a generator and a discriminator. The generator takes in input random noise (latent variables) and transforms it into data-like sample, while the discriminator tries to distinguish between real data and sample produced by the generator. This adversarial setup pushes the generator to create output that are increasingly realistic, aiming to fool the discriminator. This kind of NN is particularly valuable for generating high-quality, realistic samples, and it is also capable of learning the most salient features of the data. The training process is a sort of a game, indeed GANs are trained through a zero-sum game where the discriminator is optimized to correctly distinguish real from fake data, while the generator is trained to produce sample that can fool the discriminator. Indeed, the discriminator is trained to maximize its ability to distinguish, and the generator is trained to minimize the discriminator’s success. If both networks have enough capacity and are trained properly, the game converges when the generator reproduces the data distribution exactly.

WHAT ARE DIFFUSION MODELS?

Diffusion Models are a class of generative models that learn to produce complex data by simulating a gradual process of adding noise to the data and then learning how to reverse that process to recover the original structure, Their training phase is divided into two main phases: forward process (encoder) where the noise is added to the data over many small steps, and the reverse process (decoder) where a NN is trained to reverse the noise process step by step. The reverse process, since it’s done step by step, it may require a lot of time resulting to be computationally expensive.

EXPLAIN WHY DIFFUSION MODELS CANNOT BE SEEN AS A SPECIAL CASE OF NORMALIZING FLOWS

The difference between diffusion models and normalizing flows lie in their transformations and learning objectives. In diffusion models, the forward process is a fixed and stochastic process where data are progressively corrupted by adding gaussian noise in multiple steps. This process is not learned and is probabilistic, while the reverse process, the denoising phase, is learned. In contrast, normalizing flows are built around a sequence of invertible and deterministic transformations that directly map a simple distribution to the complex data distribution. These transformations are learned, and they are bijective. Moreover, another difference lies in the estimation of the likelihood, indeed flow are designed for exact likelihood estimation, while diffusion rely on approximations.

DESCRIBE IN DETAIL THE LOSS FUNCTION USED IN DIFFUSION MODELS

The loss function used to train diffusion models combines two terms: reconstruction term, which ensures the model can reconstruct the original image from an early noisy version, and a consistency term (KL divergences), which is the core of the loss and guides the model to revere the noise process. In particular, the consistency term checks if the reverse transition matches the true reverse transition. Minimizing the KL divergences aligns the model’s generative process with the correct denoising trajectory.

CAN U EXPLAIN THE DIFFERENCE BETWEEN DENOISING AUTOENCODERS AND DIFFUSION MODELS?

DAEs are trained to reconstruct clean data from a single-step corrupted input. They consist of a learned encoder-decoder pair: the encoder maps a noisy input to a latent representation, and the decoder attempts to reconstruct the original input from this latent code. The noise injection acts as a regularizer, encouraging the model to learn meaningful representations that capture the structure of the data. Diffusion Models, by contrast, rely on a multi-step noise process. The forward process, which adds Gaussian noise over many steps, is fixed and not learned. The core task is then to train a deep neural network (the "decoder") to reverse this process step-by-step, recovering the original data from pure noise. In short, DAEs are designed primarily for learning robust internal representations, while Diffusion Models are built for generating high-fidelity data through a principled, multi-step denoising process.