CI is the field of computing that draws from the successes of natural systems to develop alternate ways of solving computational problems in the real world.

CI = Computing + Nature: nature-inspired methods usually tolerate incomplete, imprecise and uncertain knowledge.

Fuzzy logic

Fuzzy logic → Generalization of ordinary logic. Allows other than absolute truth and utter falsity.

Fuzzy arithmetic → Formalism for making calculations with numerical quantities imprecisely known (fuzzy numbers defined as fuzzy sets on the real numbers).

Unlike Boolean logic (bivalent/Aristotelian), fuzzy logic is **multi-valued**:

- Fuzzy logic represents **degrees of membership** and degrees of truth
- Things can be part true and part false at the same time

Main characteristics

- **Knowledge** is interpreted as a collection of elastic/fuzzy constraints on a set of variables.
- **Inference** is viewed as a process of propagation of elastic constraints.
- Allows **decision making** with estimated values under incomplete or uncertain information.
- **Exact reasoning** → limiting case of approximate reasoning.

A **fuzzy set** A in X is characterized by its **membership function** $\mu A : X \rightarrow [0, 1]$ and $\mu A(x)$ is interpreted as the degree of membership of element x in the fuzzy set A for each $x \in X$.

Types of fuzzy sets

- **Ordinary fuzzy sets:** Their membership functions are often overly precise, i.e. they require that each element of the universal set be assigned a particular real number.
- **Type 2, 3, 4...:** Allowing their intervals to be fuzzy.

Terminology

- **Linguistic variable**: **fuzzy variable.**
- **Fuzzy rule**: **conditional statement** in the familiar form.
- **Fuzzy logic system:** system that uses fuzzy rules, fuzzy logic, and fuzzy sets

Fuzzy Inference

- **Mamdani**
    - Steps:
        - Fuzzify input variables
        - Evaluate rules
            - Membership function ← by applying compositional rule of inference.
            - T → fuzzy conjunctive operator (t-norm)
            - I → fuzzy implication operator (common choice minimum t-norm).
        - Aggregate rule outputs

        - Defuzzify
            - First Aggregation then Inference - FATI: (center of maximums, center of gravity, etc.)
            - First Inference then Aggregation - FITA: (maximum value)
    - Advantages:
        - Intuitive.
        - Widespread acceptance.
        - Well suited to human input.
- **Sugeno**
    - Zero-Order TSK FRBS: $\Sigma (hi*Yi) / \Sigma hi$
    - First-Order TSK FRBS:
        - $Z_1 = p1*x + q1*y + r1$
        - $Z_2 = p2*x + q2*y + r2$
        - $Z = h_1+h_2 / h_1*Z_1+h_2*Z_2$
    - Advantages:
        - Computationally efficient.
        - Works well with linear techniques (e.g., PID control).
        - Works well with optimization and adaptive techniques.
        - Guaranteed continuity of the output surface.
        - Well suited to mathematical analysis.

ANFIS: Adaptive Neuro-Fuzzy Inference System

- Using neural networks to adapt the parameters needed for Sugeno.
- Layer 1
    - Every node i is an adaptive node with a node function.
    - x (or y) is the input to node i and Ai (or Bi-2) is a linguistic label.
- Layer 2:
    - Every node in this layer is a fixed node labeled Π.
    - Output is the result of applying any other T-norm operator that performs fuzzy AND.
- Layer 3:
    - Every node in this layer is a fixed node labeled N.
    - Normalized firing strengths: Ratio of the i^{th} rule's firing strength to the sum of all rules' firing strengths.
- Layer 4:
    - Every node in this layer is an adaptive node.
    - Normalized firing strength X consequent parameters.
- Layer 5:
    - The single node in this layer is a fixed node labeled Σ.
    - Computes the overall output as the summation of all incoming signals.
- Two steps:
    - The premise part of a rule defines a fuzzy region, while the consequent part specifies the output within the region
    - S1 = Set of premise (nonlinear) parameters
    - S2 = Set of consequent (linear) parameters
- Advantages:
    - Can achieve a **highly nonlinear mapping**.

        - Superior to common linear methods in reproducing nonlinear time series.
    - The initial parameters are intuitively reasonable and all the input space is covered properly (**fast convergence**).
    - Fuzzy rules that are local mappings instead of global ones, then facilitate the **minimal disturbance** principle (the adaptation should not only reduce the output error for the current training pattern but also minimize disturbance to response already learned).

CART: Classification and Regression Trees

- Generates a tree partitioning of the input space, which relieves the "curse of dimensionality" problem (num. of rules increasing exponentially with num. of inputs)
- Grows a decision tree by determining a succession of splits (decision boundaries) that partition the training data into disjoint subsets.
- Starting from the root node (contains all the training data), an exhaustive search is performed to find the split that best reduces an error measure (cost function).
- Dataset is partitioned into two subsets. Same splitting method applied to both child nodes.
- Recursive procedure terminates
    - Error measure in a node < tolerance level
    - Error reduction < threshold

Neural Networks - Multilayer Perceptrons

Feed-forward Neural Networks

- Neural systems: neurons + synapses or weights without loops.
- Structure:
  - Input units + hidden units + output units → layers
  - Aggr. fct. → net-input (pre-activation) → act. fct. → net-output (activation)
- Aggregation functions (between the weights and the activation of the previous units):
  - Multilayer Perceptrons (MLPs) and Recurrent Neural Networks: **inner product**.
  - Radial-basis Function Networks: **squared Euclidean distance.**
  - Convolutional Neural Networks: **convolution operation.**
- Activation functions:
  - Identity, logistic, hyperbolic tangent, gaussian, rectifier linear, softplus, softmax.

Examples of machine learning problems

Regressions:

- Type of problem: **regression** (relationship between the scalar variable *y* (target) and the d-dimensional vector *x* of explanatory variables or features (input))
- Model representation: **linear.**
- **Linear regression:**
  - Cost function: **sum of squares**.
  - Optimization technique:
    - Normal equations:
      - $X'X\theta = X'Y$
      - Even when $(X'X)^{-1}$ is singular, they can be solved.
    - Gradient descent:
      - $\partial J(\theta t, D)/\partial \theta = X'X\theta t - X'Y = X'(h_{\theta t}(X) - Y)$
      - First-order iterative optimization algorithm for finding the minimum of a function.
      - Takes steps proportional to the negative gradient of the function (or an approximation) at the current point.
      - Usually obtains a local minimum of the function.
- **Ridge regression**:
  - Cost function: **(2-norm) regularized sum-of-squares error.**
  - Optimization technique: **modified normal equations**, **gradient descent.**
- **Lasso regression**:
  - Cost function: **(1-norm) regularized sum-of-squares error.**
  - Optimization technique: **quadratic programming techniques, least angle regression for Lasso.**

Support Vector Machines for Classification

- Type of problem: **classification**
- Model representation: $f_{SVM}(x) = b + \sum \alpha_i y^{(i)} K(x^{(i)}, x)$, where $\theta = (b, \{\alpha i\}^N_{i=1})$

- Cost function: Maximize α
- Optimization technique: coordinate descent, gradient projection, decomposition variants (SMO, SMV,...), interior point, primal-dual approaches, etc

Bayesian Networks

- Type of problem: **probability estimation**.
- Model representation: joint probability function P (x; θ), with different constraints.
- Cost function: log-likelihood of the data (maximize).
- Optimization technique: gradient ascent, expectation-maximization, etc

Regression Trees

- Type of problem: **regression.**
- Model representation: binary trees.
- Cost function: sum-of-squares error.
- Optimization technique: since finding the best partition in terms of the quadratic error is computationally unfeasible, greedy algorithms are performed, for example selecting a splitting variable j and a splitting point s such that minimize.

Neural networks

- Type of problem: **classification, regression, etc**
- Model representation: Single-layer Perceptron, Multilayer Perceptron, Radial-basis Function Network, Convolutional Neural Network, Recurrent Neural Network, etc
- Cost function: **sum-of-squares error, cross-entropy**, etc, with or without regularization
- Optimization technique: **gradient descent** (with many variations), etc

Single-layer perceptrons

- Rosenblatt's Perceptron: Supervised learning linear model consisting of a single neuron with adjustable synaptic weights and bias.
- Type of problem: **Classification.**
- Model representation: $h_{\omega}(x) = \text{sgn}(\omega \cdot x^{(i)})$
- Cost function: the number of mistakes (**classification accuracy**)
- Optimization technique: the Perceptron algorithm
- Applications:
    - **Linear regression (regression, sum of squares)** → normal equations, gradient descent
    - **Logistic regression (classification, cross-entropy) (equivalent to negative log-likelihood)** → gradient descent

Multilayer perceptrons

- Neural networks with several layers of weights, characterized by the fact that the aggregation function is the inner product between the weights and the activation of the previous units.
- Type of problem: **regression, classification.**
- Model representation: typical architecture of an MLP → units structured by layers
- Cost function:
    - **Sum-of-squares** error for **regression** problems

    - **Cross-entropy** for **classification** problems
        - For multiclass classification: **negative log-likelihood**
    - With or without **regularization** terms
- Optimization technique: gradient descent with backpropagation.
    - **Backpropagation (BP)** is an efficient algorithm to compute the derivative of the cost function of an MLP with respect to its weights (not optimization technique).
    - Output:
        - Regression: Activation = net-input
        - Classification: Activation = softmax
    - The derivative of the cost function (with respect to the weights) can be computed layer by layer. We assume that we have stored in memory the values of the pre-activations and activations of the units in the network.
        - In the last layer it is equal to the activation minus the result.
    - **Backpropagation**
        - Perform a forward pass with x to compute n and h.
        - Compute the local gradients of the last layer.
        - Compute the gradients of the last layer (local gradient x activation of l-1)
        - for l from L-1 to 1:
            - Compute the local gradient of the current layer
            - Compute the gradients of the current layer
        - Return $\partial J / \partial W$

Convolutional Neural Networks

NNs computing a convolution instead of a matrix multiplication in at least one of their layers.

Typically, convolution is combined with other ops., such as non-linear transformation and pooling.

Convolution

- Leads to
  - **Sparse interactions** (sparse connectivity, the same as having zero weights in a MLP).
  - **Weight sharing** with the aim of obtaining **translation invariance** (be able to find a pattern in any place of the input).
- Similar to
  - A local receptive field with shared weights.
  - Multiplication by a matrix with restrictions:
    - Equal weights in different units.
    - Zeros for several weights.
- Definition of $(F * W)(t)$
  - An expectation / weighted average (by W) of F around t.
  - The matching of two functions F and W at every t.
- Discrete convolution used in 2D CNNs (cross-correlation): $C(n, m) = (I * K)(n, m)$
  - I is the input, K is the feature detector or filter or kernel (the weights) and C is the feature map (the output).
  - The sum is defined over all valid values.
- Channels
  - Additional dimensions where we do not want to apply the convolution operation:
  - 2D: Color images (each pixel is a rgb vector).
    - Input image I ⟶ $N \times M \times 3$ **tensor**. Filter K_f ⟶ $A \times B \times 3$ tensor.
    - Convolution I x K ⟶ $N \times M \times 1$ tensor (matrix where channels are summed).
    - Each filter outputs one channel. F filters ⟶ $N \times M \times F$ tensor.
    - Outputs reshaped in channels ⟶ input of a new layer: Deep CNNs.
  - Bias terms ⟶ Typically, one bias per channel in the output and one bias per filter.
- Hyperparameters
  - **Size of the filter:**
    - $A \times B$. Output size $(N - A + 1) \times (M - B + 1)$.
  - **Zero-padding**
    - **Valid** ⟶ Convolution where entire filter is within image (reducing output size).
    - **Same** ⟶ Zeros added at borders so output size = input size (every pixel contributes to the same number of convolutions).
  - **Stride**
    - Sample only every *s* pixels in each direction ⟶ Downsampled convolution.
- Additional Operations and Layers in CNNs
  - **Non-linearities:** non-linear transformation in the convolution output (sigmoidal, ReLU...)
  - **Pooling layers**

        - Replaces output at a certain rectangle with a summary statistic of contents.
        - Reduces spatial size and parameters → reducing network computation.
        - Most common pooling functions:
            - **Max-pooling**
            - **Average-pooling**
        - Hyperparameters of the pooling operation:
            - Size of the rectangle (typically 2 × 2).
            - Stride (typically 2).
    - **Fully connected layers:** standard layers (typically on top of the convolutional layers)
- Architecture
    - Pooling layers inserted after several Convolutional + Non-linearity layers (deep feature learning/extraction)
    - Fully connected layers as output layers of the network (classification/discrimination)
- Training CNNs with Back-Propagation
    - For shared weights in convolutional layers → derivative = sum of back-propagated derivatives.
    - For pooling layers → derivative back-propagated according to the type of pooling and forward propagations (max-pooling only propagates the derivative of the maximum element).
    - For non-linearities and fully connected layers → derivative back-propagates as usual.
- What Makes CNN Work?
    - Good initialization of the weights.
    - Non-saturated activation functions as Rectified Linear Units (ReLUs) (or variants).
    - Adaptive learning rates (RMSProp, Adagrad,...).
    - Strong regularization techniques, such as dropout or other tricks, such as batch
    - normalization or local contrast normalization.
    - High performance resources (GPUs, supercomputers).
    - A large set of labeled examples.
    - Data where the convolution operation make sense.

The RBFNN

Gaussian Distribution

- Expectation: Mean.
- Examples from a class are noisy versions of an ideal class member (a prototype):
  - Prototype: modeled by the mean vector.
  - Noise: modeled by the covariance matrix.
- **Positive definiteness:**
  - Σ (covariance matrix) has to be real symmetric and positive definite (PD)
    - for all non-null vectors $x \in Rd$, $xT \Sigma x > 0$ must hold true.

Radial Basis Funtion (RBF) neural networks

- Output of a hidden neuron → distance between input and neuron's center (prototype).
  - Precise interpretation to the network output.
  - Allows to design de-coupled training algorithms.
- Roots at exact function interpolation.
  - The function *h* is expressed as a combination of basis functions Φ.
  - Basis functions
    - With radial contours of constant value centered at the data points $X_{n.}$
    - Use any norm in R^d (most often an Euclidean norm).
    - Non-singular for various choices of the basis functions (Micchelli's theorem).
    - $\Phi W = T$ can be solved by simple matrix inversion as $W = \Phi - 1T$
- Very often exact function interpolation setting is not attractive
  - High number (N) of interpolation points → complex and unstable solutions.
  - The outputs t_n depend stochastically on the inputs x_n → overfit solutions.
  - The interpolation matrix Φ can be singular or ill-conditioned.
  - The inversion of Φ grows as $O(N^3)$.
  - Control of complexity of the solution → **Regularization**
    - In one-hidden-layer NNs, regularization penalizes size of the weight matrix.
    - Transform to two-layer neural network:
      - Use a subset of the data points to center the basis functions.
      - First (hidden) layer as basis functions $\phi_i(x)$, centered at vectors c_i.
        - Constant basis function $\phi_0(x) = 1$ included to be associated with the bias weights wk_0 in the output layer.
      - Second (output) layer as linear combinations of basis functions.
      - If original matrix is non-singular → new matrix non-singular.
- Very popular choice for the ϕ_i → simple Gaussian.
- **Training**
  - 1st → Find activation function, center and covariance matrix → clustering algorithm
  - 2nd → Finds weights by any of the usual (linear) methods:
    - **Pseudo-inverse** (via the SVD), for **regression** (linear output activations).
    - **Logistic regression**, for **binary classification** (logistic output activations).
    - **Multinomial regression**, for **multiclass classification** (soft-max output activations).

MLP	RBFNNs
Global and distributed approximation of the underlying function.	Local and non-distributed*
Distributed representation → error surface with multiple local minima.	
Training times orders of magnitude larger	Not an iterative process as gradient descent
Generalize better outside of the local neighborhoods defined by the training set	
Require fewer neurons to approximate a non-linear function with the same accuracy.	
All the parameters trained simultaneously.	Hidden and output layers trained separately using very efficient hybrid algorithms.
Multiple hidden layers with complex connectivity.	One hidden layer and full connectivity.
Partition feature space with hyper-planes.	Constant activation boundaries of hidden units are hyper-ellipsoids (usually hyper-spheres).

* Only the hidden layers close to the output have more influence.

- If variance small → only a few centers influence (it is not smooth and have some peaks).
- If the variance large → result almost linear (much more smooth).

Deep Learning

Deep architectures

- **Discriminative**
  - Classical MLP: Computation starts in lowest layer (x), and propagates upwards
  - Output is h^L
- **Generative**
  - Computation starts in deepest layer (a Restricted Boltzmann Machine), that can sample h^L (symmetric connections).
  - Rest of the layers propagate activations downwards as standard MLPs.
  - The output is x
- Properties:
  - Deep → Many levels of non-linear operations
    - Neural networks with many hidden layers
    - Propositional formulae with many levels of sub-formulae
- Learning in AI:
  - Complex reality, lack of analytical understanding of the world to explain the huge variety of different images of the same object by varying position, orientation, lighting focus...
  - Useful representations of data are abstract
    - Abstract features → semantic content != low-level features of raw data.
    - Representations are invariant to most local changes in the input.
  - Machine Learning for AI → learning a representation preserving task-relevant features of data while being invariant to irrelevant factors of variation.
  - To make learning practical we need:
    - Learn from a very large sets of examples.
    - Learn from unlabeled data.
    - Learn with little human input.
  - State-of-the-art of Deep Learning → success in difficult tasks → good candidates to tackle these kind of problems.
- Deep Learning
  - Complex transformation of data using multiple layers of representation. Each layer constructed as a simple (but nonlinear) transformation of the previous.
    - Features from higher levels in the hierarchy are constructed on the basis of features in lower levels (deep).
      - Different levels of features → different levels of abstraction → human tasks that we do not know how to describe formally.
      - Deep hierarchy of features → system learns very complex functions
  - Learn without human interaction features that compose distributed representations.
  - Learning to transform one representation (in previous stage) into another.
  - Higher levels → more abstract and represent more complex features.
- 2006, Deep Belief Networks & stacked RBMs with pre-training to construct Deep Discriminative Neural Networks and Deep Auto-Encoders.
  - Greedy layer-wise pre-training (training of every layer).
  - Then fine-tuning of the whole network after the pre-training (back.

- Since then:
  - Labeled datasets much larger and computers much faster.
  - Found number of ways to avoid poor local minima:
    - Good initialization of the weights, such as pre-training.
    - Non-saturated activation functions, such as Rectified Linear Units (ReLUs).
    - Adaptive learning rates (RMSProp, Adagrad,...)
    - Strong regularization techniques, such as dropout or other tricks, such as batch normalization.
  - Results:
    - MNIST, Reuters, TIMIT, Transfer Learning, Large Scale Visual Recognition, IMAGENET, Google and Stanford Experiments.
  - Drawbacks:
    - DNNs have millions of parameters → We know very little of the whole (and huge) space of parameters.
    - They cannot explain what learn.
    - Adversarial examples.
    - Curse of dimensionality still an issue.
  - Practical information
    - More suitable data
      - High-dimensional data (hundreds, thousands, or even more)
      - Noise is not the main problem in the data
      - Structured data, with structure difficult to represent in simple model
      - Large amount of data (capture many factors of variation in real data)
    - Resources
      - Very good knowledge of the underlying concepts of Machine Learning and the models to construct.
      - Very efficient implementations of the software
      - High performance resources (GPUs, supercomputers)
      - A good team of scientists and engineers
      - Many hours (weeks, months,...) of trial-and-error runs

Experimental Issues

Machine learning → Main goal → Generalization

Optimality of the prediction → Measured by a loss function L, and averaged over the whole space to give the Risk functional.

- After selecting the model representation (NNs) → risk functional (generalization error) is a function of the parameters θ → main goal → selec best set of parameters.

Model selection

- **Training** → Obtaining a model from a particular set of parameters.
- **Validation** → Checking the model obtained in the training phase.
- **Test** → Check the model (test phase).
- Different configuration of parameters → trained and validated w. different datasets (shuffling).
- Test data cannot be used in the model selection phase.
- Large amount of data: **Holdout cross-validation** (or simply holdout)
    - Split whole data → model selection and test.
    - **Model selection** → for every configuration of the parameters, split model selection data set into training and validation data sets.
    - Model selected → train with the model selection data set and check with the test set.
- Small amount of data: **k-fold cross-validation**
    - Training set divided into *k* distinct folds
    - Train with *k* - 1 folds and validate with the remaining fold.
    - Repeat for each of the *k* possible folds and take the mean over all *k* folds.
    - Drawbacks → training repeated *k* times for every combination of parameters
    - Variations
        - **Leave-one-out** cross-validation
            - Data very scarce: k = N, N training phases, each with N - 1 examples.
        - **Double k-m-fold** cross-validation
            - Perform a k-fold cross-validation to obtain k folds ($k - 1$ folds for model selection and and 1 fold to test)
            - Perform a m-fold cross-validation on each of the k folds ($m - 1$ for training and 1 fold for validation)
            - Take the means over all folds

Problems

- **Underfitting:** models fit poorly the data → "relevant things to learn yet".
- **Overfitting:** models fit too much the data → "learned irrelevant or noisy things".
- Bias-Variance decomposition:
    - **Bias** → approximation capability (inverse)
    - **Variance** → complexity
        - Low complexity → high bias → underfit
        - Too high complexity → high variance → overfit

- **Model complexity**
    - Hidden units + layers + non-linearities + larger weights
    - Regularization
        - Weight constraints (limiting the size,...)
        - Weight sharing
        - L2 regularization
        - L1 regularization (not differentiable)
        - Early stopping
        - Adding noise (to the weights, activities,...)
        - Dropout
        - Batch normalization
- **Curse of Dimensionality**
    - Number of examples needed to "cover" the space grows exp. with input dimension.
        - High-dimensional spaces → impossible to manage such a huge data.
    - Models based on few examples overfit → input space is not properly represented.
    - Generalization improve with amount of data up to a certain point:
        - High-dimensional spaces → impossible to obtain enough data
        - Complexity of the model may be too high for any reasonable data size.
        - Data may be noisy, so that there will be errors that can not be avoided.
            - Types of **noise**
                - Noisy input values.
                - Missing values.
                - Mislabeled examples.
                - Irrelevant features.
            - May cause underfitting or overfitting depending on the type of noise, the complexity of the model and the amount of data
            - Difficult to remove the noise:
                - **Standard analysis** to detect very noisy input values.
                - Techniques of **outliers detection** to look for mislabeled or out-of-range examples.
                - **Feature Selection** to select the best subset of features (removing irrelevant features).
            - If few examples → + irrelevant features → + train & test error
            - If many examples → +irrelevant features → - train error & +test error

Pre-processing

- Important for NNs: in many practical applications it is one of the most significant factors.
- **Linear transformations**
    - Transform each variable to have zero mean and unit st dev (**z-scores**)
    - **Whitening:** similar to z-score but using cov. matrix to obtain uncorrelated features
    - **Linear scaling** to [0, 1]
- **Non-linear transformations (Feature extraction)**
    - Polynomial features: X
    - Non-linear filters, etc
- **Dimensionality reduction**

- **Principal Component Analysis**
- **Feature selection**

Error measure

- Regression problems → **sum-of-squares error** and variations (normalized, relative, etc)
- Classification problems → many error measures, most derived from the confusion matrix
  - Accuracy and Balanced Accuracy
  - Specificity (True Negative Rate)
  - Sensitivity (Recall, True Positive Rate)
  - Precision
  - F1 = 2 · Precision · Recall/(Precision + Recall)
  - False Negative Rate and False Positive Rate
  - AUC

Neural networks

- **Underfitting and Overfitting**
  - **Learning Curves**
    - Training error increases with examples (easier to approximate few examples)
    - Validation error decreases with examples (better models with more examples)
    - Training error usually smaller than validation error.
    - High bias (too low complexity)
      - Training and validation errors → high and similar → small gap between curves
      - The model cannot improve by having more examples
    - High variance (too high complexity)
      - Training error will be small and validation error large → large gap between curves
      - The model can improve by having more examples.
  - **Underfitting** → large training & validation error
    - Reduce underfitting → Increase complexity
      - Increasing the complexity of the architecture
      - Performing more training updates
  - **Overfitting** → small training errors and large validation errors
    - Reduce overfitting → Decrease complexity
      - Reducing the complexity of the architecture
      - Reducing the number of features (Feature Selection)
      - Using early stopping or some other form of regularization
- **Stochastic Gradient Descent**
  - Really important to use Stochastic Gradient in order to get reasonable convergence speeds for large data sets.
  - Best estimation of the gradient → averaging the gradient of all examples
    - Explore the parameter space by changing the parameters more frequently
    - **Mini-batch updates** → average of the gradient in a mini-batch of B examples

        - B = 1, we have **Online Gradient Descent**
        - B = training set size, we have **Batch (Standard) Gradient Descent**
        - Small values of B → frequent updates with less precision
    - SGD converges much faster than Batch Gradient Descent (very expensive) for large data sets, and it does not usually depend too much on the size of the batch.
    - Small values of B benefit from more exploration of the parameter space (kind of regularization due to the "noise" injected in the gradient estimator)
- **Parameters**
    - **Architecture**
        - Number of hidden layers and hidden units
            - Complexity of the network
            - Problem-dependent and not independent of other parameters
            - In the past → optimal → minimum number of layers and units.
            - Recent results → better to choose a large architecture (deep, wide or both) and apply strong regularization techniques.
        - Typical MLPs architectures are fully connected (not CNNs).
        - Non-linearity of the hidden units
            - Typical non-linear functions are **sigmoid** ones, such as the logistic function and the hyperbolic tangent function
            - Recently, rectifier non-linearities (**rectified linear** or **softplus)** → avoid the vanishing gradient problem in some cases
        - Output units activation → associated to cost function
            - Squared error → linear output units (regression)
            - Cross-entropy → softmax output units (classification)
            - Sigmoidal functions (together with target values normalized to their range) useful in some cases, optimization difficulties in other ones
            - The error propagation must be taken into account (rectified linear units can not propagate the gradient when the unit is saturated)
    - Regularization
        - **Early stopping** estimates the generalization error on a validation data set and stops the training when either overfitting or no progress is observed
        - **L_2 regularization** adds a term $\lambda \sum \theta^2_i$ to the cost function, penalizing large values of the parameters (depending on λ)
        - **L_1 regularization** (not differentiable) adds a term $\lambda \sum |\theta|$ to the cost function, encouraging sparsity of the parameters
        - The probability parameter for **Dropout**
            - For every training case, hidden units randomly omitted from the network with probability p (typically 0.5)
            - At test time, the "mean network" used but with their outgoing weights halved to compensate the fact that twice as many of them are active.
            - Very efficient way of performing model averaging with neural networks: Different network for each presentation of each training case but all share the same weights for the hidden units that are present.
        - Batch normalization does not have any parameter

- Each dimension is linearly transformed so as to have zero mean and unit standard deviation (z-score)
- Linear weights are added so that the original values x_j can be restored

- Parameters for Stochastic Gradient Descent
  - **Mini-batch size**
    - Typical values range between 16 and 128
    - The impact of this parameter is mostly computational
  - **Number of iterations**
    - Number of epochs (complete iteration through the training set)
    - Number of (mini-batch) updates
    - Strongly related to early stopping
    - Convenient to turn early stopping off when analyzing the effect of individual parameters
  - **Learning rate**
    - Usually the most important parameter to tune
    - Typical values → less than 1 and larger than 10^{-6}
    - Optimal learning rate usually close to (by a factor of 2) the largest learning rate that does not makes training diverge
    - It may change during training with a learning rate schedule: linear decaying, exponential decaying, etc
  - **Momentum method**
    - Technique for accelerating and smoothing out gradient descent.
    - Accumulates a "velocity vector" → next update as a linear combination of the actual gradient and this velocity vector
    - Parameter μ → weight given to accumulated updates
      - Value in [0, 1]
      - Similar to learning rate. May change during training
  - Biases and output weights can generally be initialized to 0, but the rest of the weights need to be carefully initialized
    - To break symmetries between hidden units of the same layer
    - To be small enough so as to not saturate neither the forward nor the backpropagation phase
    - **Xavier initialization** → initializes the weights randomly so as to have variance proportional to $1 / \sqrt{}$ fan-in + fan-out
    - **Unsupervised pre-training** to initialize the weights: an unsupervised model (such as a Restricted Boltzmann Machine or an Auto-encoder) is trained with the data, and the final weights of the unsupervised procedure are used as the initial weights of the supervised one
  - Variants
    - RMSProp, Adagrad, Adadelta, Adam, etc
- Alternatives to Gradient Descent
  - Conjugate Gradients
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS) and limited-memory BFGS (L-BFGS)
  - Davidon-Fletcher-Powell...

- Strategies for Searching the Parameters
    - Exponential search process
    - Searching for the value of a single parameter (being the rest fixed)
        - Set a starting interval and choose intermediate values in the log-domain
        - Expect a kind of a U-shape curve → There may be noisy variations
        - If best value is near the border → should explore beyond this border
        - No point in exploring the effect of small changes until a reasonable good setting has been found: start with a few values in the interval, and then refine the search around the best value found so far (multi-resolution search)
    - After the intervals for each parameter have been set, a grid search exhaustively explores all the combinations of these values
        - Advantages: it can be done in parallel
        - Disadvantages: it is exponential in the number of parameters
    - Serial?
        - **Coordinate descent:** change one parameter at a time, always making a change from the best configuration found so far
    - **Random sampling of parameters**
        - Uniform sampling in the interval of each parameter
        - Can include prior beliefs on the likely good values adding probabilities to the different values using Bayesian optimization with Gaussian processes
        - May be efficient
            - When there is a large subset of parameters close to the optimal one
            - When only a small subset of parameters matter

Evolutionary Computation

Biological terms:

- Chromosome (individual): a candidate solution to the problem
- Gene: single variable to be optimized
- Locus (plural loci): the specific location of a gene or position on a chromosome
- Allele: possible assignment of a value to a variable (a locus in a gene)

Evolutionary Algorithm: computer simulation in which a population of abstract representations of candidate solutions (individuals) to an optimization problem are stochastically selected, recombined, mutated, and then removed or kept, based on their relative fitness to the problem:

- Maintain population of individuals
- Select individuals according to fitness
- Breed them & mutate the offspring
- Form a new generation using the offspring and the old one

Main components

- **Coding function** to represent potential solutions as valid chromosomes
- **Fitness function** to evaluate the goodness of individuals
- **Initialization method** to create the initial population
- **Selection operator** to determine which individuals will undergo variation
- **Replacement strategy** to decide which individuals will be allowed in the next generation
    - Selection, replacement and fitness decrease diversity
    - $\mu >= 1$ individuals
    - $\lambda >= 1$ offspring
    - Strategies
        - $(\mu + \lambda)$ strategy → Mutation more important
        - (μ , λ) strategy, $\lambda >= \mu$ → Crossover more important
- **Genetic variation operators** (mutation, recombination) to perform the variation
    - Increase diversity
- **Termination criterion** to stop the process
    - Maximum number of generations
    - Maximum number of fitness function evaluations
    - Avg/max (relative) fitness function improvement
    - Below some tolerance (over a period of time)
    - Diversity below some tolerance (idem)
    - Closeness to optimum below some tolerance

Techniques

- **Genetic Algorithms** (GA) (bitstrings)
- **Genetic Programming** (GP) (computer programs)
- **Evolutionary Programming** (EP) (finite-state automata) → simulation of adaptive behavior in evolution (i.e. phenotypic evolution)
- **Evolution Strategies** (ES) (real-valued vectors) modeling the strategic parameters that control evolution itself ("evolution of evolution")

- **Classifier Systems** (CS) (if-then rules) → knowledge representation
- **Differential Evolution** (DE) → similar to GAs, differing in the reproduction mechanism used
- **Neuro Evolution** (NE) (neural networks)

EA vs derivative-based methods

- EAs can be much slower (but they are any-time algorithms)
- EAs are less dependent on initial conditions (still we need several runs)
- EAs can use alternative error functions:
    - Not continuous or differentiable
    - Including structural terms
- EAs do not get easily stuck in local optima
- EAs are better "scouters" (global searchers)

Hybrid algorithms (aka memetic)

- EAs + local tuner: the EA can be used to find a good set of initial solutions
- Hopefully a very good solution is on the basin of attraction of one of these points
- Iteration leads to Lamarckian evolution (and may be very slow)

Genetic Algorithms

Characteristics

- Individuals are bitstrings of length n
- Strategy is usually (μ , λ) with $\mu = \lambda$
- Selection is usually proportional to fitness
- Mutation is seen as "transcription error" (secondary discovery force)
- Recombination is called "crossover" (primary discovery force)

Bitstring → Loci → chromosome length n

Selection

- **Completely random**
  - No benefit for being better than others → little exploitation
- **Always select (only) the best**
  - Very high selective pressure → little exploration
- **Stochastic selection** (better fitness →higher chance of reproduction)
  - Fitness-proportional diversity → more balanced exploitation-exploration searches
  - **The Roulette Wheel**
    - Each solution gets a region on a roulette wheel according to its fitness
    - Spin the wheel, select solution marked by roulette pointer
    - May be very sensitive to the "details" of the fitness function
  - Variations
    - **Stochastic universal sampling**
      - Spin wheel once with as many equally-spaced pointers as individuals
    - **Remainder stochastic sampling**
      - Give a sure number of copies for the above-average individuals and perform standard SRW with remainder
    - **Rank selection**
      - Probability of selection is proportional to rank
      - Cope with
        - High selective pressure in the first few generations
        - High selective pressure due to a decrease in fitness variance as the search proceeds
- **Elitism**
  - Best A individuals from last B generations are kept unchanged for the next generation
  - Unrealistic but ensures best fitness of a generation never decreases
  - Entails a decrease in diversity (more exploitation)
  - Can be be subject to specific local improvement
    - Hill-climbing
    - Large(r) mutation steps
- **Tournament**
  - Randomly select k individuals (with replacement)
  - Select the individual with best fitness among these k individuals

- **Niching**
    - Split it into parts (niches) around promising parts of the search space
    - Reproduction within each niche, then merge back

Crossover

- **One-Point Crossover (1P)**
- **Two-Point Crossover (2P)**
- **Uniform Crossover (UX)**
    - UX evaluates each bit in the parent strings for exchange with a probability of 0.5
    - **HUX (Half Uniform Crossover)** is as UX, but half of non-matching bits are swapped

Mutation

- Binomial = independent Bernoulli "transcription errors"

Stopping techniques

- Limit the number of generations or fitness evaluations
- Detect convergence
    - No relative improvement in mean/max fitness
    - No change in best
    - Uniformity of fitness evaluations or individuals
    - Fitness evaluation beyond a predefined value
    - Closeness to optimum (if the optimal fitness is known)

Replacement

- **Generational genetic algortihms (GGA):**
    - Replace all parents with their offspring, (μ, λ) with $\mu = \lambda$
    - No overlap between populations of different generations
- **Steady-state genetic algorithms (SSGA):**
    - Offspring created, mutated, and used to replace some parents in the old generation
    - Overlap between populations of different generations
    - Replacement strategies:
        - Offspring replace the worst individuals of the current population (elitism)
        - Offspring replace random individuals of the current population
        - Individuals to be replaced are selected using «reverse» tournament selection (now for the worst one)
        - Offspring replace the oldest individuals of the current population

Binary representations

- Natural binary representation (boolean variables)
- Nominal-valued variables (e.g. type of product)
- Discrete structure (tree, graph, ...)
- Integer or ordinal-valued variables
- Continuous information (real numbers)
- Problems
    - Maximum attainable precision may be less than needed (for real numbers)
    - Introduction of «Hamming cliffs»

        - Two numerically adjacent values have bit representations far apart
        - **Gray coding:** Hamming distance between the representation of consecutive numerical values is 1

Schemata theory

- Schema S → template describing a subset of individuals
- What survives is not the individuals, but their genetic stuff
- Theorems:
    - Short, low-order (few fixed positions), above-average schemata receive exponentially increasing trials (individuals represented by the schema) in subsequent generations
    - ‘‘Genetic Algorithms work by discovering and exploiting building blocks --groups of closely interacting genes-- and then combining these blocks (via crossover) to produce successively larger blocks until the problem is solved.’’

Key ideas

- Encoding should respect the schemata: it must allow discovery of small building blocks from which larger, more complete solutions can be formed
- Encoding should reflect functional interactions, as proximity on the genome (linkage bias)
- Devise appropriate genetic operators:
    - All individuals correspond to feasible solutions and vice versa
    - Genetic operators preserve feasibility
- High flexibility and adaptability (many options).
- These decisions are highly problem-dependent
- Parameters are not independent: you cannot optimize them one by one
- Parameters can be adaptable, e.g. high in the beginning (more exploration), going down (more exploitation), or even be subject to evolution themselves!

Genetic Programming

Characteristics

- Individuals → computer programs represented as syntax trees (or as a set of syntax trees)
- Strategy is usually (μ , λ) with $\mu = \lambda$. Some of offspring may be copies of previous individuals
- Selection is usually proportional to fitness
- Crossover, mutation and cloning are alternative genetic operators with specified probabilities to create offspring
- The best-so-far individual is chosen as the result

Need to specify

- **Set of terminals** (independent variables, zero-argument functions and random constants) to be used as tree leaves
- **Set of primitive functions** to be used as internal nodes (typically including a conditional operator, similarly to LISP)
- **Fitness measure** to evaluate the computer programs
    - Amount of error between output and desired output over test cases
    - Accuracy of classifying objects into classes
    - Payoff that a game playing program produces
    - May be multi-objective in the sense that it combines two or more different elements
- Certain **parameters for controlling** the run (population size, operator probabilities, ...)
- **Termination criterion** (max number of generations, target value of the fitness measure)

Advanced features

- Grammar specifies functions or terminals that are permitted to appear as each argument of each function
- There are multiple function sets and multiple terminal sets → associated with types
- Initial random population → comply with the constrained syntactic structure
- Genetic operations → comply with the requirements of the constrained syntactic structure (e.g. crossover points must be of the same type)
- **Automatically Defined Functions (ADFs)**
    - GP implements parameterized reuse and hierarchical invocation of evolved code.
    - Each ADF in a separate function-defining branch within the program architecture
    - May possess zero, one, or more dummy variables (formal parameters) and may be called by the program's main branch, another ADFs or other branches
    - Constrained syntactic structure used to implement ADFs
    - Other types of branches, in addition to ADFs, to reuse code:
        - Automatically Defined Iterations (ADIs)
        - Automatically Defined Loops (ADLs)
        - Automatically Defined Recursions (ADRs)
        - Automatically Defined Stores (ADSs) provide means to reuse the result of executing code
- Program Architecture and Architecture-Altering Operations

- Each program is actually represented as a set of trees called branches instead of always being a single tree
- The architecture of a program consists of
    - the total number of branches
    - the type of each branch (result-producing branch, ADF, ADI, ADL, ADR or ADS)
    - the number of arguments (if any) possessed by each branch
    - if there is more than one branch, the nature of the hierarchical references (if any) allowed among the branches
- The architecture of a program may be
    - Prespecified by a human user
    - Automatically and dynamically created during a GP

Evolution Strategies

Characteristics

- Continuous search space R^n
- Various recombination operators
- Deterministic (μ, λ)-replacement
- Emphasis on mutation: n-dimensional Gaussian, zero expectation
- Self-adaptation of mutation parameters (first self-adaptive EA!)
- Generation of an offspring surplus $\lambda > \mu$

Representation

- objective space: $O := R^n$
- strategy space (stdevs and angles of mutation)
    - Individuals
        - object variables $x \in R^n$ to compute fitness F(x)
        - standard deviations $\sigma \in R^{n\sigma}_{+}$ to express variances
        - rotation angles $\alpha \in (-\pi, \pi]^{n\alpha}$ to express covariances

Evolution strategies

- **Mutation**
    - **Simple Self-Adaptive Mutation:** 1 mutation parameter per individual
    - **Diagonal self-adaptive Mutation:** 1 mutation parameter per individual and variable
    - **Correlated self-adaptive Mutation**: 1 covariance matrix per individual
- **Recombination**
    - Usually introduced as the first operator
    - Generates an intermediate population size of λ by generating one individual at a time out of ξ parents by looping $\lambda > \mu$ times (generation of a surplus)
    - Typically $\xi = 2$ (dual) or $\xi = \mu$ (global recombination):
        - **dual**: two random parents per individual
        - **global**: one parent held fixed and the other is chosen anew per each gene
    - Applied to both objective and strategy parameters (and often differently)
    - Two basic ways: choose randomly (discrete) and average (intermediate)
- Replacement
    - Strictly deterministic, rank-based
    - The μ best are treated equally
    - (μ, λ) selection:
        - offspring surplus $\lambda > \mu$
        - important for self-adaptation
        - useful for moving optima, noisy F, ...
    - Very strong selective pressure

Summary

- Self-adaptation of strategy parameters works without exogenous or centralized control
- needs mutation of all parameters
- needs generation of a surplus and (μ, λ) replacement
- needs recombination of all parameters