École Polytechnique de Bruxelles

INFO-H-410

Techniques of Artificial Intelligence

Hugues Bersini

Summarized by
Charles Hamesse

June 4, 2016

Contents

Contents

Chapter 1

Introduction

First, let's review a brief history of the origins of AI, its objectives and challenges met throughout the years. It is common to distinguish the old AI from the new, expliciting the breakthrough that happened in response to some *failures* of the first type of AI.

1.1 Old-fashioned AI

Intelligence It is first considered as a set of mental inferences, allowing the ability to make deductions, planning, mental simulations, reasoning, logics, and so on. Also, we distinguish rational intelligence from *fake* intelligences, such as emotional intelligence, animal intelligence, embodied intelligence, collective intelligence. In other words, intelligence is described about IQ, chess, math, logical solving. All the rest is just skills.

The inferential engine At this time, we're able to determine a general canvas and state the requirements of such an AI problem. Those are the following:

1. Find the operators that can be applied: their pre-conditions need to match the current state of the world
2. Select one the control strategy: in depth or in width, with heuristics or not
3. Avoid looping
4. Be able to backtrack
5. Do that iteratively until you find the final state

The solution of a planning problem is the sequence of operators. If several solutions are elligible, it is often the one having the shortest sequence of operators that will be optimal.

The failures Man is embodied in his environment. He is a sophisticated sensori-motor process much before any cogitive process takes on. His perception is intrinsically and materially parallel. The sensori-motor processes essentially depends on their biological grounding: parallel and adaptable. The outside world is complex and to be able to cope with, a man requires an interface of similar complexity. But this complexity can be achieved by learning and experience rather than being handcrafted. Complex processes emerge from iterating simple mechanisms

1.2 Today's AI

Man possess 2 cognitive systems

1. Parallel, automatic, unconscious, reflex, adaptable and very efficient. Based on neuronal hardware, for playing tennis, piano, becoming an expert, etc.

2. Sequential, rigid, conscious and very laborious. Based on neuronal software, for playing chess, for testing IQ

Man goes from one to the other in the cases of breakdowns in his automatisms. Machine intelligence and human intelligence can be of different nature. For the machines today, recognizing a face is much more difficult than playing chess. But doesn't Kasparov in part play chess like indeed we recognize a face?

We can describe the following trends:
AI $\rightarrow$ ALife
Software $\rightarrow$ Hardware
Cognitive Science $\rightarrow$ Biology

Summary The animal hidden in each of us might be unavoidable on the road to intelligence. Our intellectual skills are embodied in our automatisms. They depart from there. In other words, don't ever try to fully understand what a chair is without having ever sat in it. Also, a turn back is needed towards our biological interface with the outside world.

Chapter 2

State space search

Search problems are described in terms of:
– An initial state. (e.g., initial chessboard, current positions of objects in world, current location)
– A target state.(e.g., winning chess position, target location)
– Some possible actions, that get you from one state to another. (e.g. chess move, robot action, simple change in location).

Search techniques systematically consider all possible action sequences to find a path from the initial to target state. The set of all possible states reachable from the initial state defines the search space. We can represent the search space as a tree.

2.1 Exhaustive search algorithms

We may use simple systematic search techniques, which try every possibility in systematic way. Those are referred to as brute force or blind techniques and include breadth first and depth first search among others.

The algorithms for breadth first and depth first search a very easy to implement. Both algorithms keep track of the list of nodes found. List is sometimes referred to as an agenda, but implemented using stack for depth first and a queue for breadth first.

Algorithm 1: Breadth-first search.

```
1  queue = {initial-state}
2  found = false
3  while queue not empty and not found
4      remove the first node N from queue
5      if N is a goal state
6          found = true
7      find all the successor nodes of N, and put them on the end of the queue
```

Algorithm 2: Depth-first search.

```
1  stack = {initial-state}
2  found = false
3  while stack not empty and not found
4      remove the first node N from stack
5      if N is a goal state
6          found = true
7      find all the successor nodes of N, and put them on the end of the stack.
```

Choice between algorithms When is one technique more appropriate than the other?
– Shortest path? BF
– Is memory a problem? DF
– Need to find the solution quickly? It depends on the structure of the search tree. To avoid long paths in DF search, define a depth limit. To find shortest path quickly, change DF search to iterative deepening.

Extensions to basic algorithm What if there are loops (i.e., we are search a graph)? How do you avoid (virtually) driving round and round in circles? Algorithm needs to keep track of which nodes have already been explored, and avoids redoing these nodes.

Algorithm 3: Variation of depth-first search.

```
1  stack = {initial-state}
2  visited = ∅
3  found = false
4  while stack not empty and not found
5      remove the first node N from stack
6      if N ∉ visited
7          visited = visited ∪ N
8          if N is a goal state
9              found = true
10         find all the successor nodes of N, and put them on the end of the stack.
```

2.2 Heuristic search algorithms

Depth first and breadth first search turn out to be too inefficient for really complex problems. Instead, we turn to "heuristic search" methods, which do not search the whole search space, but focus on promising areas.

To identify promising areas, we need an **evaluation function**. The evaluation function scores a node in the search tree on how close it is to the goal/target state.

Algorithm 1: Hill climbing.

```
1  current_state = initial_state
2  while current_state ≠ goal-state ∨ no change in current state
3      get the successors of current_state
4      evaluate the successors and assign them a score
5      if one of the successors is better than current_state
6          then set the new-current state to be the successor with the best score
```

This algorithm avoids loop, but may halt without success in local maximum.

Algorithm 2: Best first search.

```
1  agenda = {initial_state}
2  found = false
3  while agenda ≠ ∅ and not found
4      remove the best node N from agenda
5      if N is a goal state
6          found = true
7      find all successor nodes of N, assign them a score, and put them on the agenda organised as a pr
```

Best first search algorithm works almost the same way as depth/breadth search algorithms, but we use a priority queue where nodes with high scores are taken of the queue first. Hence still exhaustive search and performance depends on the quality of the evaluation function.

Algorithm 3: A*.

```
agenda = {initial_state}
found = false
while agenda ≠ ∅ and not found
    remove best node N from agenda
    if N is a goal state
        found = true
    for s in all successor nodes of n
        s.score  = cost(initial_state, s) + cost(s, goal_state)
    put all successor nodes on the agenda organised as a priority queue
```

This algorithm is basically an extension of the best-first search, taking the total path length into account. The score is based on the predicted total path "cost", that is, a sum of the cost/distance from initial to current node, and the predicted cost/distance to target node.

In BF search (if the cost of traversing a link is the same), the solution with the lowest cost will be found first, but it may take time. In DF search, a solution can be found quickly, yet it may not be a very good one. The A* algorithm finds a cheap solution quickly.

Summary General search methods can be used to solve complex problems, which are formulated in terms of initial and target state, and the primitive actions that take you from one state to next. Also, one may need to use heuristic search for complex problems, as search space can be too large.

Chapter 3

Concept learning

This section describes the functioning of algorithms able to learn from examples, determine version spaces and implement the candidate elimination algorithm.

3.1 The inductive learning hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

An example problem For the *EnjoySport* problem, the canvas is the following (refer to the slides for the full problem description).
Given:
- Instances X (e.g. possible days described by attributes Sky, AirTemp, Humidity, etc),
- Target function c: EnjoySport : $X \rightarrow 0, 1$,
- Hypotheses H: conjunctions of literals (e.g. $\langle$?, Cold, High, ?, ?, ? $\rangle$),
- Training examples D: positive and negative examples of the target function $\langle x_1, c(x_1)\rangle, ..., \langle x_m, c(x_m)\rangle$.

Determine:
A hypothesis h in H such that $h(x) = c(x) \quad \forall x \in D$.

Algorithm 1: Find-S.

```
1  h = most specific hypothesis in H
2  for each positive training instance x
3      for each attribute constraint a_i in h
4          if a_i satisfied by x
5              do nothing
6          else
7              replace a_i by the next more general constraint satisfied by x
8  return h
```

This algorithm seeks the most restrictive (i.e most *specific*) hypothesis that fits all the positive examples (negatives are ignored). This algorithm shows a few drawbacks:
- Can't tell whether it has learned concept
- Can't tell when training data inconsistent
- Picks a maximally specific h (why?)
- Depending on H, there might be several!

3.2 Version spaces

A hypothesis h is consistent with a set of training examples D of target concept c if and only if $h(x) = c(x)$ for each training example $\langle x, c(x)\rangle$ in D.

$$Consistent(h, D) \equiv (\forall \langle x, c(x)\rangle \in D)\ h(x) = c(x)$$

The version space, $VS_{H,D}$, with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with all training examples in D.

$$VS_{H,D} \equiv \{h \in H \mid Consistent(h, D)\}$$

Algorithm 1: List-then-eliminate.

```
1  VS ← list containing every hypothesis in H
2  for each training example ⟨h(x), c(x)⟩
3      remove from VS any hypothesis h for which h(x) ≠ c(x)
4  return VS
```

Representing version spaces The general boundary, G, of version space $VS_{H,D}$ is the set of its maximally **general** members. The specific boundary, S, of $VS_{H,D}$ is the set of its maximally **specific** members. Every member of the version space lies between these boundaries:

$$VS_{H,D} = \{h \in H \mid g \geq h \geq s,\ \forall s \in S,\ g \in G\}$$

where $x \geq y$ means x is more general or equal to y.

Algorithm 2: Candidate elimination.

```
1  G ← maximally general hypotheses in H
2  S ← maximally specific hypotheses in H
3  for each training example d
4      if d is a positive example
5          remove from G any hypothesis inconsistent with d
6          for each hypothesis s ∈ S inconsistent with d
7              S ← S \ s
8              S ← S ∪ {allminimalgeneralizationshofs|Consistent(h,d) ∧ somememberofGismoregeneralthanh}
9              S ← S \ {h | h > s ∀h ∈ H, s ∈ S}
10     if d is a negative example
11         remove from S any hypothesis inconsistent with d
12         for each hypothesis g in G that is not consistent with d
13             G ← G \ g
14             G ← G ∪ {allminimalspecializationshofg|Consistent(h,d) ∧ somememberofSismorespecificthanh}
15             G ← G \ {h | h < g ∀h ∈ H, g ∈ G}
```

To-do: notes on this algorithm

3.3 Bias

So far, we have three types of learners with different biases:

1. Rote learner: store examples, classify x if and only if it matches previously observed example.
2. Version space candidate elimination algorithm
3. Find-S algorithm

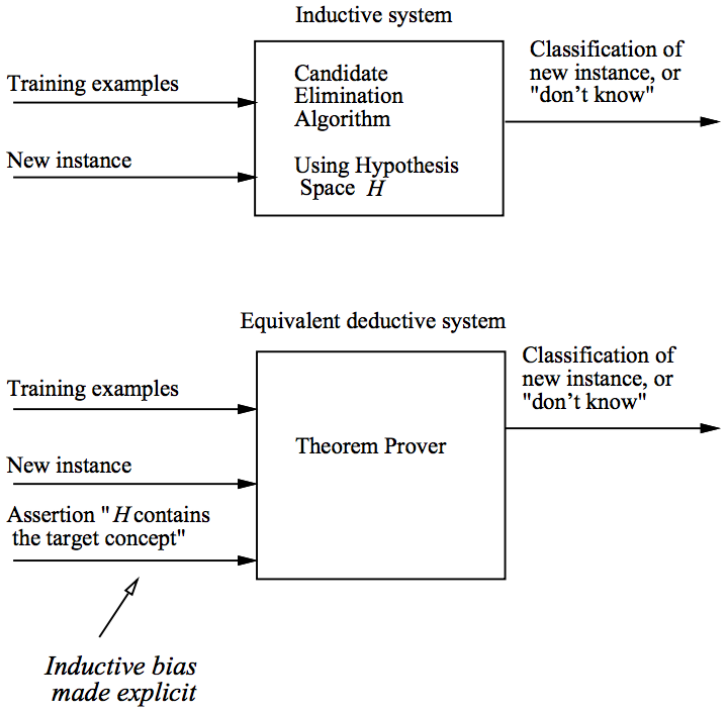
An unbiased learner The idea is to choose and H that expresses every teachable concept (i.e. H is the power set of X, 2^X).

Inductive bias Let's consider concept learning algorithm L, a set of instances X, a target concept c, a set of training examples $D_c = \{\langle x, c(x)\rangle\}$. Let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after training on data D_c.

The inductive bias of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c:

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$

where $A \vdash B$ means A logically entails B



Summary Points

1. Concept learning as search through H
2. General-to-specific ordering over H
3. Version space candidate elimination algorithm
4. S and G boundaries characterize learner's uncertainty
5. Learner can generate useful queries
6. Inductive leaps possible only if learner is biased
7. Inductive learners can be modelled by equivalent deductive systems

Chapter 4

Decisions trees

Decision trees are another formalism we can use to solve problems using learning algorithms. We may consider using decision trees when:

1. Instances are describable by (attribute,value) pairs
2. Target function is discrete valued
3. Disjunctive hypothesis may be required
4. Training date is possibly noisy

For example, such trees are used for equipment or medical diagnosis, credit risk analysis, modeling calendar scheduling preferences, etc.

4.1 Decision tree representation

The representation of decision trees include the following features:

1. Each internal node tests an attribute
2. Each branch corresponds to an attribute value
3. Each leaf node assigns a classification

Algorithm 1: Top-down induction of decision trees (main loop).

```
1 A ← the "best" decision attribute for next node
2 assign A as decision attribute for node
3 for each value of A
4     create new descendant of node
5 sort training examples to leaf nodes
6 if training examples perfectly classified
7     stop
8 else
9     iterate over new leaf nodes
```

4.2 Entropy, information gain

Entropy Let's consider a set of training examples S. We can denote $p_{\oplus}$, the proportion of positive examples in S and $p_{\ominus}$, the proportion of negative samples. The entropy is written:

$$Entropy(S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

It represents the expected number of bits needed to encode a class ($p_{\oplus}$ or $p_{\ominus}$) of randomly drawn members of S (under the optimal, shortest-length code), be cause an optimal length code assigns $-log_2 p$ bits to message having probability p.

Note: a more general way to write entropy is:

$$Entropy(S) \equiv -\sum_{x \in X} p(x) \log_2 p(x)$$

The information gain $Gain(S, A)$ is the expected reduction in entropy due to sorting the set S on the attribute A. It is computed with:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

4.3 ID3 learning algorithm

Algorithm 1: ID3.

```
1  ID3(Examples S, Target_Attribute t, Attributes A)
2      root ← new root node for the tree
3      if s_i positive ∀s_i ∈ S
4          return single-node tree root, labeled +.
5      else if s_i negative ∀s_i ∈ S
6          return single-node tree root, labeled −.
7      else if |A| = 0
8          return the single node tree root, labeled with the most common value of t in S
9      else
10         a ← arg max Gain(S, a_i) ∀a_i ∈ A
11         for each value v_i of a
12             b_i ← new tree branch below root, labeled v_i
13             Sv_i ← {s ∈ S | value(s, a) = v_i}
14             if Sv_i = ∅
15                 add new leaf node below b_i labeled with the most common target value in S
16             else
17                 add new subtree below b_i ID3 (Sv_i, a, A \ {a})
18     return root
```

ID3 (Iterative Dichotomiser 3) is an algorithm used to generate a decision tree from a dataset.

Properties of ID3

1. The hypothesis search space is complete. That is, the target function is surely there.
2. Its choices are statistically based, i.e the algorithm is robust to noisy data.
3. It does not guarantee an optimal solution; it can get stuck in local optima. It uses a greedy on each splitting iteration. We could use backtracking to find an optimal decision tree.

4. It can overfit to the training data. To avoid overfitting, smaller decision trees with high information gain attributes near the root should be preferred over larger ones (note that this algorithm usually produces small trees).

5. It is harder to use on continuous data. Searching for the best value to split by can be time consuming.

Inductive bias in ID3 Bias is a preference for some hypotheses, rather than a restriction of hypothesis space 2^X (the power set of instances X).

Attributes with many values If an attribute has many possible values, $Gain(S, A)$ might select it over others that might be more appropriate (e.g. attribute, pairs such as "date = June 3, 1996"). One approach is to use the $GainRatio(S, A)$ instead:

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

together with

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|}$$

where $S_i = \{s \in S \mid value(s, A) = v_i\}$

Attributes with Costs In some cases, attributes may come together with a cost. For example:
- Medical diagnosis, $BloodTest$ has cost EUR150
- Robotics, $Width\_from\_1ft$ has cost 23sec.

So how could we learn a consistent tree with low expected cost? There are two approaches based on the same idea: to replace gain by a function of itself and the cost.

1. Tan and Schlimmer (1990)

$$Gain(S, A) \rightarrow \frac{Gain^2(S, A)}{Cost(A)}$$

2. Nunez (1988)

$$Gain(S, A) \rightarrow \frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

where $w \in [0, 1]$ determines importance of cost.

Unknown Attribute Values What if some examples missing values of A? We may consider using training examples anyway, sort through the three considering the following aspects:

1. If node n tests A, assign most common value of A among other examples sorted to node n

2. Assign most common value of A among other examples with same target value

3. Assign probability p_i to each possible value v_i of A, and assign fraction p_i of example to each descendant in tree.

4. Classify new examples in the same fashion

4.4 Overfitting

Occam's razor It is a problem-solving principle: *"prefer the shortest hypothesis that fits the data"*. Note that there exist several formulations. Then comes the question, why prefer short hypotheses? Well, a short hypothesis that fits data is unlikely a coincidence, whereas a long hypothesis that fits data might be coincidence with higher probability. However, there are many ways to define small sets of hypotheses. Let's consider errors of hypothesis h over:

1. The training data: $error_t(h)$
2. The entire distribution D of data: $error_D(h)$

The hypothesis $h \in H$ overfits the training data if there is an alternative hypothesis $h' \in H$ such that:

$$error_t(h) < error_t(h')$$

and

$$error_D(h) > error_D(h')$$

To-do: define the error function

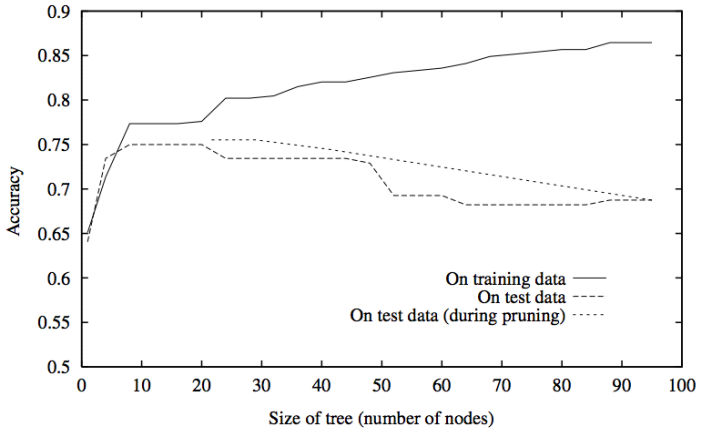
How to avoid overfitting We can think about stopping to grow when data split not statistically significant (in terms of information gain), or to grow the full tree, then post-prune. To select the "best" tree, we need to:

1. Measure performance over training data
2. Measure performance over separate validation data set
3. MDL: minimize $size(tree) + size(misclassifications(tree))$

Reduced-error pruning This is a technique that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting. It consists in doing the following operations until further pruning is harmful:

1. Evaluate impact on validation set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves validation set accuracy

It produces smallest version of most accurate subtree.


0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
Accuracy
0
10
20
30
40
50
60
70
80
90
100
Size of tree (number of nodes)
On training data
On test data
On test data (during pruning)

Chapter 5

Neural networks

5.1 Perceptron

The perceptron is an algorithm for supervised learning of binary classifiers: functions that can decide whether an input (represented by a vector of numbers) belongs to one class or another.

It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time. It is, in a way, akin to a linear discriminant.

Historically, it is also the first neural net inspired by human brain (which was then appearing as the best computer).

Associated learning In a perceptron, the learning is supervised. It is based on a couple of input patterns and desired outputs. If the activation of output neuron corresponds to the desired output, nothing happens. Otherwise, as inspired by neurophysiological data, if the neuron is activated: decrease the value of the connection and if it is unactivated: increase the value of the connection. This process is iterated until the output neurons reach the desired value. An approach to decreasing or increasing the value of connections is the *learning rule of Widrow-Hoff*. It considers the weight of a connection $w_{i,j}$, linking the output of neuron i to the input of neuron j:

$$w_{i,j}^{t+1} = w_{i,j}^{t} + \eta(t_j - o_j)x_i = w_{i,j}^{t} + \Delta w_{i,j}$$

where η is the learning rate, t_j the target output of neuron j. At first, the decision rule function f was chosen to be a step function with a certain threshold b, according to the data. That is:

$$f : x \mapsto y = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $w \cdot x$ is the dot product $\sum_{i=0}^{m} w_i x_i$, and m the number of inputs to the current neuron.

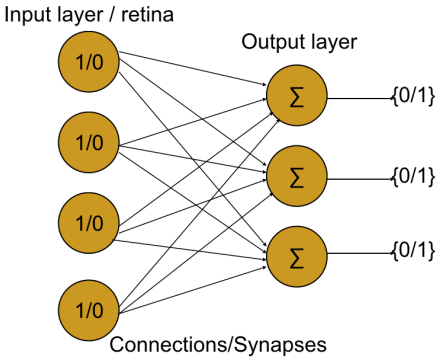


Figure 5.1: Constitution of a perceptron

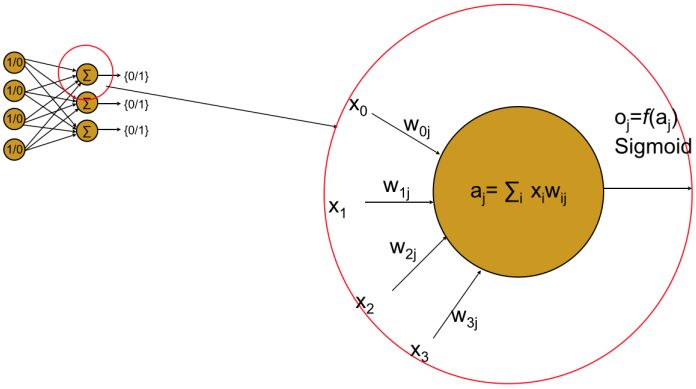


Figure 5.2: The neuron model. x_i is the activation of the input neuron i, a_j is the activation of the output neuron j, $w_{i,j}$ is the weight of the connection between neurons i and j, and o_j is the final output of the neuron, with respect to a_j and the decision rule f.

Gradient descent It is a first-order optimization algorithm, used with perceptrons. To find a local minimum of a function, one will take steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. In most cases, a sigmoid is used, rather than a Heaviside (or step) function for the statistical interpretation:

$$f : x \mapsto y = \frac{1}{1 + e^{-x}}$$

This function is interesting because its derivative is very easy to compute and therefore a good pick for the gradient descent technique:

$$f' = f(1 - f)$$

Limitations of the perceptron In some cases, the perceptron will not learn easily (if at all), as it cannot separate not linearly separable data. For example, the XOR gate, impossible to reproduce on a neural network, has killed all research on the subject for 20 years. Therefore, we had to wait for the magical hidden layer and for backpropagation for the perceptron to regain interest.

5.2 Associative memories

We will consider two types of associative memories:

- Hetero-associative
- Auto-associative

We will treat here only auto-associative memories, which are also known as auto-association memories or an autoassociation networks. It is a generic term that refers to all memories that enable one to retrieve a piece of data from only a tiny sample of itself.

Traditional memory stores data at a unique address and can recall the data upon presentation of the complete unique address.

Autoassociative memories are capable of retrieving a piece of data upon presentation of only partial information from that piece of data. Heteroassociative memories, on the other hand, can recall an associated piece of datum from one category upon presentation of data from another category.

Hopfield networks have been shown to act as autoassociative memory since they are capable of remembering data by observing a portion of that data.

As a matter of example, the fragments presented below should be all that's necessary to retrieve the appropriate memory:
- "To be or not to be, that is _____"
- "I came, I saw, _____"
Readers will be able to complete the phrases above, given only a portion. The conclusion to be drawn is that autoassociation networks can recall the whole by using some of its parts.

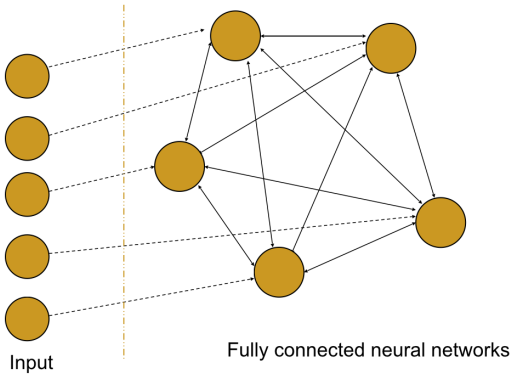
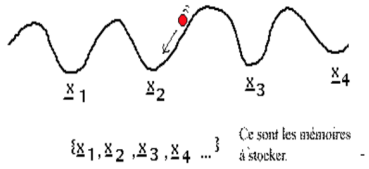


Figure 5.3: Constitution of an associative memory

Hopfield networks They are a form of recurrent artificial neural network that serve as content-addressable memory systems with binary threshold nodes. They are guaranteed to converge to a local minimum, but convergence to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum) can occur. Hopfield networks

also provide a model for understanding human memory.

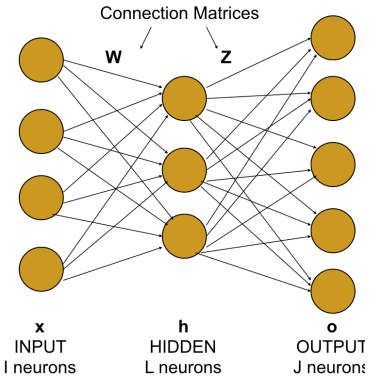
The newtork becomes a dynamical machine, it has been shown to converge into a fixed point, which is of minimal Lyapunov energy. These fixed point are used for storing patterns. We may then use a sort of gradient descent with:

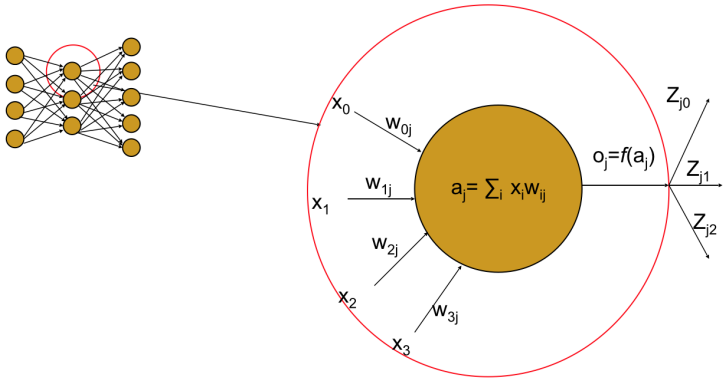


$$\Delta W_{i,j} = \sum_{\text{patterns}} X_i^P X_j^P$$

5.3 Multilayer perceptron

To increase accuracy and extend the original perceptron abilities, we add one or more layers of neurons between the input and the output layers. Together with the multilayer





perceptron come the backtracking technique and, more specifically, the backpropagation

algorithm. It is a supervised learning method for multilayer networks, its name refers to the backward propagation of error during the training of the network. The algorithm is roughly described by the following steps:

Algorithm 1: Neural network backtracking (main loop).

1 inject an ***input*** x
2 compute the intermediate h: $h = f(W \cdot x)$
3 compute the ***output*** o: $o = f(Z \cdot h)$
4 compute the error ***output*** δ_{out}: $\delta_{out} = f'(Z \cdot h) * (t - o)$
5 adjust z with respect to the error: $Z^{t+1} = Z^t + \eta * \delta_{out} * h = Z^t + \Delta^t Z$
6 compute the error on the hidden layer: $\delta_{hidden} = f'(W \cdot x) * (Z\delta_{out})$
7 adjust W with respect to this error: $W^{t+1} = W^t + \eta * \delta_{hidden} * h = W^t + \Delta^t W$

We can see it as a direct consequence of the chaining derivative of the gradient descent, where once again the sigmoid will come in handy. Also, we can show that in many cases, a neural network with more layers will produce more accurate results.

Chapter 6

Clustering

Clustering is the process of grouping a set of instances (data points or examples or vectors) into clusters (subsets or groups) so that instances within a cluster have high similarity in comparison to one another, but are very dissimilar to instances in other clusters. Clustering may be found under different names in different contexts, such as:

- Unsupervised learning
- Data segmentation
- Automatic classification
- Learning by observation

Similarities and dissimilarities of instances are based on the predefined features of the data. The most similar instances are grouped into a single cluster.

Clustering instances Let X be the unlabelled data set, that is,

$$X = x_1, x_2, ..., x_N$$

We may partition X into k clusters $C_1, ..., C_k$ so that the following conditions are met:

$$\begin{aligned} C_i &\neq \emptyset,\ \forall i \in [1,k] \\ \cup_{i=1}^{k} C_i &= X \\ C_i \cap C_j &= \emptyset,\ \forall i,j \in [1,k] \wedge i \neq j \end{aligned}$$

Requirements for clustering The goal of clustering is to group a set of unlabelled data. There are many typical requirements of clustering in machine learning and data mining, such as:

- Dealing with large data sets containing different types of attributes.
- Find the clusters with arbitrary shape.
- Ability to deal with noisy data in data streaming environment.
- Handling with high-dimensional data sets.
- Constraint-based clustering

Types of clustering methods The basic clustering methods are organised into the four categories:

1. Partitioning methods
   - The partitioning method constructs k clusters of the given set of N instances, where $k \leq N$. It finds mutually exclusive clusters of spherical shape using the traditional distance measures (Euclidean distances).
   - To find the cluster center, it may use mean or medoid (etc.) and apply iterative relocation technique to improve the clustering by moving instances from one cluster to another such as k-means clustering.
   - The partitioning algorithms are ineffective for clustering high-dimensional big data.
2. Hierarchical methods
   - The hierarchical methods create a hierarchical decomposition of N instances. It can be divided into two categories: the top-down (or divisive) approach, and the bottom-up (or agglomerative) approach.
   - The top-down approach starts with a single cluster having all the N instances and then split into smaller clusters in each successive iteration, until eventually each instance is in one cluster, or a termination condition holds.
   - The bottom-up approach starts with each instance forming a separate cluster and then successively merges the clusters close to one another, until all the clusters are merged into a single cluster, or a termination condition holds.
3. Density-based methods
   - The density-based methods cluster instances based on the distance between instances, which can find arbitrarily shaped clusters. It can cluster instances as dense regions in the data space, separated by sparse regions.
4. Grid-based methods
   - The grid-based methods use a multi-resolution grid data structure. It's fast processing time that typically independent of the number of instances, yet dependent on the grid size

Similarity measure A similarity measure (SM), $sim(x_i, x_l)$, can be defined between any two instances $x_i, x_l \in X$, so that with an integer value k, the clustering problem is to define a mapping $f : X \mapsto [1, k]$, where each instance, x_i is assigned to one cluster C_i, with $1 \leq i \leq k$.
Given a cluster C_i:

$$sim(x_{il}, x_{im}) > sim(x_{il}, x_j),\ \forall x_{il}, x_{im} \in C_i \wedge x_j \notin C_i$$

A good clustering is that instances in the same cluster are "close" or related to each other, whereas instances of different clusters are "far apart" or very different from one another, which together satisfy the following requirements:

- Each cluster must contain at least one instance.
- Each instance must belong to exactly one cluster.

Distance measure A distance measure (DM), $dis(x_i, x_l)$ where $x_i, x_l \in X$, is also often used in clustering. Let's consider the well-known Euclidean distance or Euclidean metric (i.e. straight-line) between two instances in Euclidean space:

$$dis(x_i, x_l) = \sqrt{\sum_{i=1}^{m}(x_i - x_l)^2}$$

where $x_i = (x_{i1}, x_{i2}, ..., x_{im})$ and $x_l = (x_{l1}, x_{l2}, ..., x_{lm})$ are two instances in Euclidean m-space.

6.1 K-Means clustering

It defines the centroid of a cluster, C_i as the mean value of the instances $\{x_{i1}, x_{i2}, ..., x_{iN}\} \in C_i$. It proceeds as follows:

1. First, it randomly selects k instances, $\{x_{k1}, x_{k2}, ..., x_{kN}\} \in X$ each of which initially represents a cluster mean or center.
2. Each of the remaining instances, $x_i \in X$ is assigned to the cluster to which it is most similar, based on the Euclidean distance between the instance and the cluster mean.
3. It then iteratively improves the within-cluster variation. For each cluster, C_i, it computes the new mean using the instances assigned to the cluster in the previous iteration. All the instances, $x_i \in X$ are then reassigned into clusters using the updated means as the new cluster centers.
4. The iterations continue until the assignment is stable, that is the clusters formed in the current round are the same as those formed in the previous round.

Cluster mean A high degree of similarity among instances in clusters is obtained, while a high degree of dissimilarity among instances in different clusters is achieved simultaneously. The cluster mean of $C_i = \{x_{i1}, x_{i2}, ..., x_{iN}\}$ is defined by:

$$Mean = C_i = \frac{\sum_{j=1}^{N} x_{ij}}{N}$$

Algorithm 1: k-Means clustering

```
1 X ← x1, x2, ..., xN, set of unlabelled instances.
2 C ← c1, c2, ...ck, set of empty clusters
3 do until no change
4     arbitrarily choose k instances xi ∈ X as the initial k clusters center
5     (re)assign each xi to the ck with the most similar mean.
6     update the k means, that is, calculate the mean value of the instances for each cluster
```

Drawbacks of k-means clustering The k-Means clustering is not guaranteed to converge to the global optimum and often terminates at a local optimum (as the initial cluster means are assigned randomly). It may not be used in some application such as when data with nominal features are involved. The k-Means method is not suitable for discovering clusters with non-convex shapes or clusters of very different size. The time complexity of the k-Means algorithm is $O(nkt)$, where n is the total number of instances, k is the number of clusters, and t is the number of iterations. Normally, $k \ll n$ and $t \ll n$.

6.2 Similarity-based clustering

A similarity-based clustering method (SCM) is an effective and robust clustering approach based on the similarity of instances, which is robust to initialise the cluster numbers and efficient to detect different volumes of clusters. SCM is a method for clustering a data set into most similar instances in the same cluster and most dissimilar instances in different clusters. The instances in SCM can self-organise local optimal cluster number and volumes without using cluster validity functions.

Similarity between instances Let's consider $sim(x_i, x_l)$ as the similarity measure between instances x_i and the lth cluster center x_l. The goal is to find x_l to maximise the total similarity measure:

$$J_s(C) = \sum_{l=1}^{k} \sum_{i=1}^{N} f(sim(x_i, x_l))$$

where, $f(sim(x_i, x_l))$ is a reasonable similarity measure and $C = \{C_1, ..., C_k\}$. In general, the similarity-based clustering method uses feature values to check the similarity between instances. However, any suitable distance measure can be used to check the similarity between the instances.

Algorithm 1: Similarity-based clustering.

$X \leftarrow \{x_1, x_2, ..., x_N\}$
$C \leftarrow \emptyset$
$t \leftarrow$ threshold value
$k \leftarrow 1$
$C_k \leftarrow x_1$
$C \leftarrow C \cup C_k$
for $i = 2$ to N do
    for $l = 1$ to k do
        find the lth cluster center $x_l \in C_l$ to maximize the similarity measure, $\arg\max_{x_l} sim(x_i, x_l)$
    if $sim(x_i, x_l) \geq t$ then
        $C_l \leftarrow C_l \cup x_i$
    else
        $k \leftarrow k + 1$
        $C_k \leftarrow \{x_i\}$
        $C \leftarrow C \cup C_k$
return C

6.3 Nearest neighbor clustering

Instances are iteratively merged into the existing clusters that are closest. In NN clustering a threshold, t, is used to determine if instances will be added to existing clusters or if a new cluster is created. The complexity of the NN clustering algorithm is depends on the number of instances in the dataset. For each loop, each instance must be compared to each instance already in a cluster. Thus, the time complexity of NN clustering algorithm is $O(n^2)$. We do need to calculate the distance between instances often, we assume that the space requirement is also $O(n^2)$.

Algorithm 1: Nearest-neighbor clustering.

$D \leftarrow \{x_1, x_2, ..., x_n\}$
$A \leftarrow$ adjacency matrix showing distance between instances
$t \leftarrow$ threshold value
$C_1 \leftarrow \{x_1\}$
$C \leftarrow \{C_1\}$

```
6   k ← 1
7   for i = 2 to n
8       x_m = arg min dis(x_i, x_m) ∀x_m ∈ C_j, ∀C_j ∈ C
9       if dis(x_i, x_m) ≤ t
10          C_m ← C_m ∪ x_i
11      else
12          k ← k + 1
13          C_k ← {x_i}
14          C ← C ∪ C_k
15  return C
```

Euclidean vs Manhattan distance The distance between the two points in the plane with coordinate (x, y) and (a, b) is given by:

$$\begin{aligned} EuclideanDistance((x,y),(a,b)) &= \sqrt{(x-a)^2+(y-b)^2} \\ ManhattanDistance((x,y),(a,b)) &= |x-a|+|y-b| \end{aligned}$$

6.4 Ensemble clustering

Ensemble clustering is a process of integrating multiple clustering algorithms to form a single strong clustering approach that usually provides better clustering results. It generates a set of clusters from a given unlabelled data set and then combines the clusters into final clusters to improve the quality of individual clustering.

- No single cluster analysis method is optimal.
- Different clustering methods may produce different clusters, because they impose different structure on the data set.
- Ensemble clustering performs more effectively in high dimensional complex data.
- It's a good alternative when facing cluster analysis problems.

Generally three strategies are applied in ensemble clustering:

1. Using different clustering algorithms on the same data set to create heterogeneous clusters.
2. Using different samples/ subsets of the data with different clustering algorithms to cluster them to produce component clusters.
3. Running the same clustering algorithm many times on same data set with different parameters or initialisations to create homogeneous clusters.

The main goal of the ensemble clustering is to integrate component clustering into one final clustering with a higher accuracy.

6.5 Subspace clustering

The subspace clustering finds subspace clusters in high-dimensional data. It can be classified into three groups:

1. Subspace search methods.

2. Correlation-based clustering methods

3. Biclustering methods.

A subspace search method searches various subspaces for clusters (set of instances that are similar to each other in a subspace) in the full space. It uses two kinds of strategies:

1. Bottom-up approach - start from low-dimensional subspace and search higher-dimensional subspaces.

2. Top-down approach - start with full space and search smaller subspaces recursively.

A correlation-based approach uses space transformation methods to derive a set of new, uncorrelated dimensions, and then mine clusters in the new space or its subspaces. It uses PCA-based approach (principal components analysis), the Hough transform, and fractal dimensions. Biclustering methods cluster both instances and features simultaneously, where cluster analysis involves searching data matrices for sub-matrices that show unique patterns as clusters.

Chapter 7

Evaluation of hypothesis

7.1 Two definitions of the error

The **true error** of hypothesis h with respect to target function f and distribution $\mathcal{D}$ is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)] \tag{7.1}$$

The **sample error** of hypothesis h with respect to target function f and data sample S is the proportion of examples h misclassifies.

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x)) \tag{7.2}$$

7.2 Estimators

Problems estimating error There are two problems to keep in mind:

1. Bias: if S is a training set, $error_S(h)$ is optimistically biased:

$$bias \equiv E[error_S(h)] - error_{\mathcal{D}}(h) \tag{7.3}$$

   To get an unbiased estimate, h and S must be chosen independently.

2. Variance: even with an unbiased training set S, $error_S(h)$ may still vary from $error_{\mathcal{D}}(h)$.

For example, if h misclassifies 12 of the 40 examples in S, $error_S(h) = 12/40 = .3$.

7.3 Confidence intervals for observed hypothesis error

Let's go through all this section with a quick example. If the training set S contains n examples, drawn independently of h and each other and $h \geq 30$, then $error_{\mathcal{D}}(h)$ lies with $N\%$ probability in the interval:

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}} \tag{7.4}$$

where the pairs (N, z_N) are the usual $(50, 0.67), (68, 1.00)$, $(80, 1.28)$, $(90, 1.64)$, $(95, 1.64)$, $(98, 2.33)$, $(99, 2.58)$.

7.4 Binomial distribution, normal distribution and central limit theorem

Same old same old.. In short:

1. Pick parameter p to estimate $error_{\mathcal{D}}(h)$
2. Choose an estimator $error_S(h)$
3. Determine probability distribution that governs estimator $error_S(h)$ governed by Binomial distribution, approximated by Normal when $n \geq 30$.
4. Find interval (L, U) such that $N\%$ of probability mass falls in the interval: use table of z_N values.

7.5 Paired t tests

1. Partition data into k disjoint test sets $T_1, T_2, ..., T_k$ of equal size, where this size is at least 30.
2. For i from 1 to k, do
$$\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B) \tag{7.5}$$
3. Return the value $\bar{\delta}$, where
$$\bar{\delta} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i \tag{7.6}$$

The confidence interval ($N\%$) estimate for d becomes

$$\bar{\delta} \quad pm \quad t_{N,k-1} s_{\bar{\delta}} \tag{7.7}$$

$$s_{\bar{\delta}} \quad \equiv \quad \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \bar{\delta})^2} \tag{7.8}$$

$\bar{\delta}$ is said to be approximately normally distributed.

7.6 Comparing learning methods

The idea is only to compute the errors of various learning algorithms and estimate their difference.

Chapter 8

Data, text and graph mining

Data mining is an interdisciplinary subfield of computer science. It is the computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use.

8.1 Data warehouses

Preparing the data It is a key step in the data mining process. Data has to be clean and homogeneous. Different variables must be expressed on the same scale, etc. You need to have regularities in the data.

The main techniques of data mining

- Clustering (unsupervised, only group the data)
- Outlier detection (check if some data is *unusual*, e.g. detect fraud - many ways to be dishonest and few ways to be honest)
- Association analysis (discovering interesting relationships hidden between various items of large data sets)
- Forecasting
- Classification

Note the difference between classification and clustering: in classification you have a set of predefined classes and want to know which class a new object belongs to. Clustering tries to group a set of objects and find whether there is some relationship between the objects.

8.2 Understanding and predicting data

Basically, we define a vector space, formatting our graph/text/image/whatever type of data it is as an $n-$dimensional, normalized (and why not some other interesting properties) vector.

8.3 Data mining techniques

Chapter 9

Metaheuristics: genetic algorithms

Genetic Algorithms and Ant Colony Optimisation

Chapter 10

Reinforcement Learning

(Chapter 13 Tom Mitchell) (slides used during the lecture)

Chapter 11

Recommender Systems