Chapter 3

Classical Optimization and Search Techniques

In this chapter we discuss a few popular optimization techniques in use in current day natural language processing algorithms. First we present the Hidden Markov Model (HMM) used for part-of-speech tagging (POS-tagging) among other tasks. Then we formulate the POS-tagging problem using HMM and present its classical solution which is due to the Viterbi algorithm.

Then we present the Maximum Entropy approach, which is a heuristic used in problems related to finding probability distributions. Next up is the Maximum Entropy Markov Model (MEMM), a discriminative model that extends a standard maximum entropy classifier by assuming that the unknown values to be learnt are connected in a Markov chain rather than being conditionally independent of each other. MEMMs find applications in information extraction, segmentation and in natural language processing, specifically in part-of-speech tagging.

This is followed by an overview of some methods namely Generalised Iterative Scaling and an improved iterative version of it, which find use in solving for the training objectives of many problems which use maximum likelihood estimation on the training data to get the parameters.

Then comes the concept of swarm intelligence, which is inspired by the action of insects such as ants. Finally we briefly discuss Boltzmann machines. They were one of the first examples of a neural network capable of learning internal representations, and are able to represent and (given sufficient time) solve difficult combinatoric problems.

3.1 Hidden Markov Model

The Hidden Markov model is a stochastic model in which the system being modelled is assumed to be a Markov process with unobserved (hidden) states. A key aspect of the HMM is its Markov property which is described in brief below along with other background definitions required.

3.1.1 Stochastic Process

Definition 1. ***Stochastic Process****: A stochastic process is a collection of random variables often used to represent the evolution of some random value over time.*

There is indeterminacy in a stochastic process. Even if we know the initial conditions, the system can evolve in possibly many different ways.

3.1.2 Markov Property and Markov Modelling

Definition 2. ***Markov Property****: A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present values) depends only upon the present state, not on the sequence of events that preceded it. That is, the process is memoryless.*

A Markov model is a stochastic model that follows the Markov property. Next we present the HMM through the urn problem which eases the exposition. In a further sub-section the formal description of the HMM is given.

3.1.3 The urn example

There are N urns, each containing balls of different colours mixed in known proportions. An urn is chosen and a ball is taken out of it. The colour of the ball is noted and the ball is replaced. The choice of the urn from which the n^{th} ball will be picked is determined by a random number and the urn from which the $(n-1)^{th}$ ball was picked. Hence, the process becomes a Markov process.

The problem to be solved is the following: Given the ball colour sequence find the underlying urn sequence. Here the urn sequence is unknown (hidden) from us and hence the name *Hidden* Markov Model. The diagram[1] below shows the architecture of an example HMM. The quantities marked on the transition arrows represent the transition probabilities.

[1]Source:`http://en.wikipedia.org/wiki/Hidden_Markov_model`

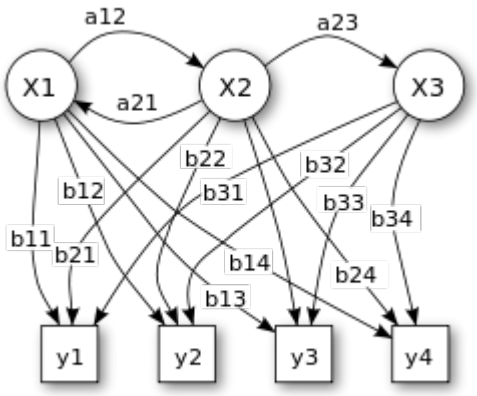

Figure 3.1: An example Hidden Markov Model with three urns

3.1.4 Formal Description of the Hidden Markov Model

The hidden Markov model can be mathematically described as follows:

N	$=$	number of states
T	$=$	number of observations
$\theta_{i=1...N}$	$=$	emission parameter for an observation associated with state i
$\phi_{i=1...N,j=1...N}$	$=$	probability of transition from state i to state j
$\boldsymbol{\phi}_{i=1...N}$	$=$	N-dimensional vector, composed of $\phi_{i,1...N}$; must sum to 1
$x_{t=1...T}$	$=$	state of observation at time t
$y_{t=1...T}$	$=$	observation at time t
$F(y\|\theta)$	$=$	probability distribution of an observation, parametrized on θ
$x_{t=2...T}$	$\sim$	$\text{Categorical}(\boldsymbol{\phi}_{x_{t-1}})$
$y_{t=1...T}$	$\sim$	$F(\theta_{x_t})$

3.1.5 The Trellis Diagram

Given the set of states in the HMM, we can draw a linear representation of the state transitions given an input sequence by repeating the set of states at every stage. This gives us the trellis diagram. A sample trellis is shown in Figure 3.2[2]. Each level of the trellis contains all the possible states and transitions from each state onto the states in the next level. Along with every transition, an observation is emitted simultaneously (in the figure a time unit is crossed and observations vary with time).

[2]Source: Prof. Pushpak Bhattacharyya's lecture slides on HMM from the course CS 344 - Artificial Intelligence at IIT Bombay, spring 2013

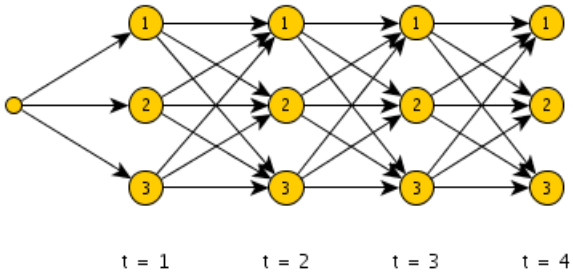

Figure 3.2: An Example Trellis

3.1.6 Formulating the Part-of-Speech tagging problem using HMM

The POS tagging problem can be described as follows. We are given a sentence which is a sequence of words. Each word has a POS tag which is unknown. The task is to find the POS tags of each word and return the POS tag sequence corresponding to the sentence. Here the POS tags constitute the hidden states. As in the urn problem, we again assume that words (balls) are emitted by POS tags (urns), a property called the lexical assumption. That is, the probability of seeing a particular word depends only on the POS tag previously seen. Also, as was the case in the urn problem, the probability of a word having a particular POS tag is dependent only on the POS tag of the previous word (urn to urn probability). Having modelled the problem as given above, we need to explain how the transition tables are constructed. The transition probabilities come from data. This is a data-driven approach to POS tagging, and using data on sentences which are already POS tagged we construct the transition tables. Given this formulation, we next present an algorithm which given an input sentence and the transition tables outputs the most probable POS tag sequence.

3.1.7 The Viterbi Algorithm

The Viterbi algorithm[13] is a dynamic programming algorithm for finding the most likely sequence of hidden states that result in the sequence of observed states. Here the hidden states are the POS tags (or urns in the example) and the observed sequence is the word sequence (ball colours).

The state transition probabilities are known (in practice these are estimated from labelled data) and so are the probabilities of emitting each word in the sentence given the POS tag of the previous word. We start at the start of the input sentence. We define two additional POS tags ^ and **$** to represent the tag for the start of the sentence and the terminal character at the end of the sentence (full stop, exclamation mark and question mark).

A straight-forward algorithm to find the most probable POS tag sequence (hidden sequence) would be to just try all possibilities starting from the beginning of

the sentence. Here, our problem has more structure. We will exploit the Markov assumption we made earlier to get a much more efficient algorithm which is precisely the Viterbi algorithm.

In the trellis for POS tagging problem the following are the major changes to be done.

- The observations (words) do not vary with time. Instead they vary with the position of the pointer in the input sentence.
- The states are the POS tags. The state transition probabilities are pre-computed using a POS-tagged corpus.

Next, we observe that due to the Markov assumption, once we have traversed a part of the sentence, the transition probabilities do not depend on the entire sentence seen so far. They depend only on the previous POS tag. This crucial observation gives rise to the Viterbi algorithm:

Suppose we are given a HMM with S possible POS tags (states), initial probabilities π_i of being in state i, the transition probabilities $P(s_j|s_i)$ of going from state i to j and the emission probabilities $P(x_t|s_i)$ of emitting x_t from the state s_i. If the input sentence is $x_1, x_2, \ldots, x_T$ then the most probable state sequence that produces the sentence $y_1, y_2, \ldots, y_T$is given by the recurrence relations

$$V_{1,k} = P(y_1|s_k)\pi_k \tag{3.1}$$
$$V_{t,k} = P(y_t|s_k)\text{max}_{s_x \in S}(P(s_k|s_x).V_{t-1,x}) \tag{3.2}$$

where $V_{t,k}$ is the probability of the most probable state sequence which emitted the first t words that has k as the final state. The Viterbi path (most likely state sequence) can be remembered by storing back pointers which contain the state s_x which was chosen in the second equation. The complexity of the algorithm is $O(|T||S^2|)$ where T is the set of words, the input sequence and S is the set of POS tags.

3.1.8 Pseudocode

Pseudocode for the Viterbi algorithm is given below:

```
# Given
# Set of states: Array S
# Start state: s0
# End state: se
# Symbol sequence: Array w
# State transition probabilities: Matrix a
# Symbol emission probabilities: Matrix b
# alpha: Matrix alpha

# All indices in arrays start on 1 in this pseudocode
```

```
# Returns
# Total probability: p

# Initialisation F1
foreach s in S do
   alpha [1][s] := a[s0][s]*b[s][w[1]]
done

# Induction F2
for i := 1 to length(w)-1 do
  foreach s in S do
    foreach s' in S do
      alpha[i+1][s] += alpha[i][s']*a[s'][s]
   done
    alpha[i+1][s] *= b[s][w[i+1]]
  done
done

# Termination F3
foreach s in S do
  p += alpha[length(w)][s]*a[s][se]
done

return p
```

In the next section, we present the concept of Maximum Entropy and see how it is applied to NLP tasks via an example for Statistical Machine Learning.

3.2 Maximum Entropy Approach

"Gain in entropy always means loss of information, and nothing more".

- G.N. Lewis (1930)

3.2.1 Entropy - Thermodynamic and Information

In statistical mechanics, entropy is of the form:

$$S = -k\sum_i p_i \log p_i,$$

where p_i is the probability of the microstate i taken from an equilibrium ensemble. The defining expression for entropy in Shannon's theory of information is of the form:

$$H = -\sum_i p_i \log p_i,$$

where p_i is the probability of the message m_i taken from the message space M. Mathematically H may also be seen as an average information, taken over the message space, because when a certain message occurs with probability p_i, the information $-\log p_i$ will be obtained.

A connection can be made between the two. If the probabilities in question are the thermodynamic probabilities p_i, the (reduced) Gibbs entropy σ can then be seen as simply the amount of Shannon information needed to define the detailed microscopic state of the system, given its macroscopic description. To be more concrete, in the discrete case using base two logarithms, the reduced Gibbs entropy is equal to the minimum number of yes/no questions needed to be answered in order to fully specify the microstate, given that we know the macrostate.

3.2.2 The Maximum Entropy Model

Language modelling is the attempt to characterize, capture and exploit regularities in natural language. In statistical language modelling, large amounts of text are used to automatically determine the models parameters, in a process known as training. While building models, we may use each knowledge source separately and then combine. Under the Maximum Entropy approach, one does not construct separate models. Instead, we build a single, combined model, which attempts to capture all the information provided by the various knowledge sources. Each such knowledge source gives rise to a set of constraints, to be imposed on the combined model. The intersection of all the constraints, if not empty, contains a (possibly infinite) set of probability functions, which are all consistent with the knowledge sources. **Once the desired knowledge sources have been incorporated, no other features of the data are assumed about the source. Instead, the worst (flattest) of the remaining possibilities is chosen.** Let us illustrate these ideas with a simple example.

3.2.3 Application to Statistical Machine Learning

Suppose we wish to predict the next word in a document[11], given the history, i.e., what has been read so far. Assume we wish to estimate $P(\text{BANK}|h)$, namely the probability of the word BANK given the documents history. One estimate may be provided by a conventional bigram. The bigram would partition the event space (h, w) based on the last word of the history. Consider one such equivalence class, say, the one where the history ends in THE. The bigram assigns the same probability estimate to all events in that class:

$$P_{\text{BIGRAM}}(\text{BANK}|\text{THE}) = K_{\{\text{THE,BANK}\}}$$

That estimate is derived from the distribution of the training data in that class. Specifically, it is derived as:

$$K_{\{\text{THE,BANK}\}} = \frac{C(\text{THE,BANK})}{C(\text{THE})}$$

Another estimate may be provided by a particular trigger pair, say (LOAN$\mapsto$ BANK). Assume we want to capture the dependency of BANK on whether or not LOAN occurred before it in the same document. Thus a different partition of the event space will be added. Similarly to the bigram case, consider now one such equivalence class, say, the one where LOAN did occur in the history. The trigger component assigns the same probability estimate to all events in that class:

$$P_{\text{LOAN}\mapsto\text{BANK}}(\text{BANK}|\text{LOAN}\in \text{h}) = K_{\{\text{BANK}|\text{LOAN}\in h\}}$$

That estimate is derived from the distribution of the training data in that class. Specifically, it is derived as:

$$K_{\{\text{BANK}|\text{LOAN}\in h\}} = \frac{C(\text{BANK,LOAN}\in h)}{C(\text{LOAN}\in h)}$$

These estimates are clearly mutually inconsistent. How can they be reconciled? Linear interpolation solves this problem by averaging the two answers. The backoff method solves it by choosing one of them. The Maximum Entropy approach, on the other hand, does away with the inconsistency by relaxing the conditions imposed by the component sources.

Consider the bigram. Under Maximum Entropy, we no longer insist that $P(\text{BANK}|h)$ always have the same value $K_{\{\text{THE,BANK}\}}$ whenever the history ends in THE. Instead, we acknowledge that the history may have other features that affect the probability of BANK. Rather, we only require that, in the combined estimate, $P(\text{BANK}|h)$ be equal to $K_{\{\text{THE,BANK}\}}$ on average in the training data.

$$\underset{\text{h ends in THE}}{E}[P_{\text{COMBINED}}(\text{BANK}|h)] = K_{\{\text{THE,BANK}\}}$$

where E stands for an expectation, or average. The constraint expressed by this equation is much weaker. There are many different functions P_{COMBINED} that would satisfy it. Similarly,

$$\underset{\text{LOAN}\in h}{E}[P_{\text{COMBINED}}(\text{BANK}|h)] = K_{\{\text{BANK}|\text{LOAN}\in h\}}$$

In general, we can define any subset S of the event space, and any desired expectation K, and impose the constraint:

$$\sum_{(h,w)\in S}[P(h,w)] = K$$

The subset S can be specified by an index function, also called selector function, f_S, an indicator for the belongingness of the pair (h,w) in S. So, we have

$$\sum_{(h,w)}[P(h,w)f_S(h,w)] = K$$

We need not restrict ourselves to index functions. Any real-valued function $f(h,w)$ can be used. We call $f(h,w)$ a constraint function, and the associated K the desired expectation. So, we have

$$\langle f, P\rangle = K$$

3.3 The ME Principle and a Solution

Now, we give a general description of the Maximum Entropy model and its solution. The Maximum Entropy (ME) Principle can be stated as follows[6]

1. Reformulate the different information sources as constraints to be satisfied by the target (combined) estimate.

2. Among all probability distributions that satisfy these constraints, choose the one that has the highest entropy.

Given a general event space $\{\mathbf{x}\}$, to derive a combined probability function $P(\mathbf{x})$, each constraint j is associated with a constraint function $f_j(x)$ and a desired expectation K_j. The constraint is then written as:

$$E_P f_j = \sum_{\mathbf{x}} P(\mathbf{x}) f_j(\mathbf{x}) = K_j$$

Given consistent constraints, a unique ME solution is guaranteed to exist, and to be of the form:

$$P(\mathbf{x}) = \prod_j \mu_j^{f_j(\mathbf{x})}$$

where the μ_js are some unknown constants, to be found.

3.3.1 Proof for the ME Formulation

Here, we give a proof for the unique ME solution that we proposed in the previous subsection. Suppose there are N different points in the event space, and we assign a probability p_i to each. Then, the objective to be maximised is the entropy, given by $H = -\sum_{i=1}^{N} p_i \ln p_i$. The constraints are:

$$\begin{aligned} \sum_i p_i &= 1 \\ \sum_i p_i f_j(x_i) &= K_j \quad \forall j \in \{1, 2, ..., m\} \end{aligned}$$

So, we introduce Lagrange multipliers and now maximise

$$\begin{aligned}
F &= -\sum_{i=1}^{N} p_i \ln p_i + \lambda(\sum_{i=1}^{N} p_i - 1) + \sum_{j=1}^{m} \lambda_j (\sum_{i=1}^{N} p_i f_j(x_i) - K_j) \\
\frac{\partial F}{\partial p_i} &= -\ln p_i - 1 + \lambda + \sum_{j=1}^{m} \lambda_j f_j(x_i) = 0 \\
\ln p_i &= \lambda - 1 + \sum_{j=1}^{m} \lambda_j f_j(x_i) \\
p_i &= e^{\lambda - 1} e^{\sum_{j=1}^{m} \lambda_j f_j(x_i)} \\
p_i &= e^{\lambda - 1} \prod_{j=1}^{m} e^{\lambda_j f_j(x_i)} \\
p_i &= a \prod_{j=1}^{m} \mu_j^{f_j(x_i)}
\end{aligned}$$

where $a = e^{\lambda - 1}$ is a normalization constant and $e^{\lambda_j} = \mu_j$

3.3.2 Generalized Iterative Scaling

To search the exponential family defined by $p_i = \prod_{j=1}^{m} \mu_j^{f_j(x_i)}$ for the μ_is that will make $P(x)$ satisfy all the constraints, an iterative algorithm exists, which is guaranteed to converge to the solution. GIS[5] starts with some arbitrary $\mu_i^{(0)}$ values, which define the initial probability estimate:

$$P^0(\mathbf{x}) = \prod_j {\mu_j^{(0)}}^{f_j(\mathbf{x})}$$

Each iteration creates a new estimate, which is improved in the sense that it matches the constraints better than its predecessor. Each iteration (say k) consists of the following steps:

1. Compute the expectations of all the f_j's under the current estimate function. Namely, compute $E_{P^{(k)}} f_j = \sum_{\mathbf{x}} P^{(k)}(\mathbf{x}) f_j(\mathbf{x})$

2. Compare the actual values $E_{P^{(k)}} f_j$'s to the desired values K_js, and update the μ_j's according to the following formula:

$$\mu_j^{(k+1)} = \mu_j^{(k)} \frac{K_j}{E_{P^{(k)}} f_j}$$

3. Define the next estimate function based on the new μ_js:

$$P^{(k+1)}(\mathbf{x}) = \prod_j \mu_j^{(k+1)^{f_j(\mathbf{x})}}$$

Iterating is continued until convergence or near-convergence.

3.4 Improved Iterative Scaling

Iterative Scaling and its variants are all based on the central idea of the Gradient Descent algorithm for optimizing convex training objectives. It is presented here using a model which occurs at many places in a maximum entropy approach to natural language processing.

3.4.1 The Model in parametric form

The problem we consider is a language modelling problem[9], which is to define the distribution $P(\mathbf{y}|\mathbf{x})$, where **y** and **x** are sequences. For eg, **y** can be the POS tag sequence and **x** the input sequence. Henceforth the boldface indicating that **x** is a sequence will be dropped unless the context demands further elucidation.

Given just the above information, the maximum entropy approach maximises the entropy of the model giving us a model of the following form.

$$P_\Lambda(y|x) = \frac{1}{Z_\Lambda(x)} exp\left(\sum_{i=1}^{n} \lambda_i f_i(x,y)\right). \tag{3.3}$$

where

- $f_i(x,y)$ is a binary-valued function, called a feature of (x,y), associated with the model. The model given above has n features.
- λ_i is a real-valued weight attached with f_i whose absolute value measures the 'importance' of the feature f_i. Λ is the vector of the weights: $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$.
- $Z_\Lambda(x)$ is the normalizing factor which ensures that P_Λ is a probability distribution.
$$Z_\Lambda(x) = \sum_y exp\left(\sum_{i=1}^{n} \lambda_i f_i(x,y)\right)$$

3.4.2 Maximum Likelihood

The next thing to do would be to train the model, i.e find the parameters λ_i so as to maximize some objective over the training data. Here, we choose to maximize the likelihood of the training data. The likelihood is computed by assuming that the

model is the correct underlying distribution and hence is a function of the parameters of the model. The likelihood of the training data is expressed as follows (N is the number of training instances):

$$\begin{aligned} M(\Lambda) &= \prod_{i=1}^{N} P(x_i, y_i) \\ &= \prod_{i=1}^{N} P_\Lambda(y_i|x_i) P(x_i) \end{aligned}$$

Now, we note that $log(x)$ is a one-to-one map for $x > 0$. Therefore the value of x which maximizes $f(x)$ is the same as that which maximizes $log(f(x))$. Henceforth we work with the logarithm of the likelihood expression as it is mathematically easier to work with. The log-likelihood expression denoted by $L(\Lambda)$ is given below:

$$\begin{aligned} L(\Lambda) &= log(M(\Lambda)) \\ &= \sum_{i=1}^{N} log\left(P_\Lambda(y_i|x_i)\right) + C \end{aligned}$$

where C is independent of Λ and is hence treated as a constant. It is dropped from the expression henceforth as it does not affect the maximization problem.
Now, we express the log-likelihood expression in terms of the empirical probability distribution $\tilde{p}(x, y)$ obtained from the training data as follows:

$$\tilde{p}(x, y) = \frac{c(x, y)}{\sum_{x,y} c(x, y)}$$

where $c(x, y)$ is the number of times the instance (x, y) occurs in the training data. The log-likelihood expression becomes the following:

$$\begin{aligned} L_{\tilde{p}}(\Lambda) &= \sum_{x,y} log\left(P_\Lambda(y|x)^{c(x,y)}\right) \\ &= \sum_{x,y} \tilde{p}(x, y) log\left(P_\Lambda(y|x)\right) \end{aligned}$$

We ignore $\sum_{x,y} c(x, y)$ as it is constant for a given training set ($= N$).

3.4.3 The objective to optimize

Hence we arrive the objective to be maximized. The maximum likelihood problem is to discover $\Lambda^* \equiv argmax_\Lambda L_{\tilde{p}}(\Lambda)$ where

$$\begin{aligned} L_{\tilde{p}}(\Lambda) &= \sum_{x,y} \tilde{p}(x,y) log\left(P_\Lambda(y|x)\right) \\ &= \sum_{x,y} \tilde{p}(x,y) \sum_i \lambda_i f_i(x,y) - \sum_{x,y} \tilde{p}(x,y) log\left(\sum_y exp\left(\sum_{i=1}^n \lambda_i f_i(x,y)\right)\right) \\ &= \sum_{x,y} \tilde{p}(x,y) \sum_i \lambda_i f_i(x,y) - \sum_x \tilde{p}(x) log(\sum_y exp\left(\sum_{i=1}^n \lambda_i f_i(x,y)\right)) \end{aligned}$$

3.4.4 Deriving the iterative step

Suppose we have a model with some arbitrary set of parameters $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. We would like to find a new set of parameters $\Lambda + \Delta = \{\lambda_1 + \delta_1, \lambda_2 + \delta_2, \ldots, \lambda_n + \delta_n\}$ which yield a model of higher log-likelihood. The change in log-likelihood is

$$\begin{aligned} L_{\tilde{p}}(\Lambda + \Delta) - L_{\tilde{p}}(\Lambda) &= \sum_{x,y} \tilde{p}(x,y) log P_{(\Lambda+\Delta)}(y|x) - \sum_{x,y} \tilde{p}(x,y) log P_\Lambda(y|x) \\ &= \sum_{x,y} \tilde{p}(x,y) \sum_i \delta_i f_i(x,y) - \sum_x \tilde{p}(x) log\left(\frac{Z_{(\Lambda+\Delta)}(x)}{Z_{(\Lambda)}(x)}\right) \end{aligned}$$

Now, we make use of the inequality $-log(\alpha) \geq 1 - \alpha$ to establish a lower bound on the above change in likelihood expression.

$$\begin{aligned} & L_{\tilde{p}}(\Lambda + \Delta) - L_{\tilde{p}}(\Lambda) \geq \sum_{x,y} \tilde{p}(x,y) \sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x) \frac{Z_{(\Lambda+\Delta)}(x)}{Z_{(\Lambda)}(x)} \\ = & \sum_{x,y} \tilde{p}(x,y) \sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x) \frac{\sum_y \exp\left(\sum_i (\lambda_i + \delta_i) f_i(x,y)\right)}{\sum_y \exp\left(\sum_i \lambda_i f_i(x,y)\right)} \\ = & \sum_{x,y} \tilde{p}(x,y) \sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x) \sum_y \left(\left(\frac{\exp(\sum_i \lambda_i f_i(x,y)}{Z_\Lambda(x)}\right) \exp\left(\sum_i \delta_i f_i(x,y)\right)\right) \\ = & \sum_{x,y} \tilde{p}(x,y) \sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x) \sum_y P_\Lambda(y|x) \exp\left(\sum_i \delta_i f_i(x,y)\right) \\ = & A(\Delta|\Lambda) \end{aligned}$$

Now we know that is we can find a Δ such that $A(\Delta|\Lambda) > 0$ then we have a improvement in the likelihood. Hence, we try to maximize $A(\Delta|\Lambda)$ with respect to each δ_i. Unfortunately the derivative of $A(\Delta|\Lambda)$ with respect to δ_i yields an equation containing all of $\{\delta_1, \delta_2 \ldots, \delta_n\}$ and hence the constraint equations for δ_i are coupled.

To get around this, we first observe that the coupling is due to the summation of the δ_is present inside the exponentiation function. We consider a counterpart expression with the summation placed outside the exponentiation and compare the two expressions. We find that we can indeed establish an inequality using an important property called the *Jensen's inequality*. First, we define the quantity,

$$f^{\#}(x,y) = \sum_i f_i(x,y)$$

If f_i are binary-valued then $f^{\#}(x,y)$ just gives the total number of features which are non-zero (applicable) at the point (x,y). We rewrite $A(\Delta|\Lambda)$ in terms of $f^{\#}(x,y)$ as follows:

$$A(\Delta|\Lambda) = \sum_{x,y} \tilde{p}(x,y) \sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x) \sum_y P_\Lambda(y|x) \text{exp}\left(f^{\#}(x,y) \sum_i \frac{\delta_i f_i(x,y)}{f^{\#}(x,y)} \right)$$

Now, we note that $\frac{f_i(x,y)}{f^{\#}(x,y)}$ is a p.d.f. Jensen's inequality states that for a p.d.f, $p(x)$,

$$\text{exp}\left(\sum_x p(x)q(x) \right) \leq \sum_x \text{exp}(p(x)q(x))$$

Now, using Jensen's inequality, we get,

$$\begin{aligned} A(\Delta|\Lambda) &\geq \sum_{x,y} \tilde{p}(x,y) \sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x) \sum_y P_\Lambda(y|x) \sum_i \left(\frac{f_i(x,y)}{f^{\#}(x,y)} \right) \text{exp}(\delta_i f_{\#}(x,y)) \\ &= B(\Delta|\Lambda) \end{aligned}$$

where $B(\Delta|\Lambda)$ is a new lower-bound on the change in likelihood. $B(\Delta|\Lambda)$ can be maximized easily because there is no coupling of variables in its derivative. The derivative of $B(\Delta|\Lambda)$ with respect to δ_i is,

$$\frac{\partial B(\Delta)}{\partial \delta_i} = \sum_{x,y} \tilde{p}(x,y) f_i(x,y) - \sum_x \tilde{p}(x) \sum_y P_\Lambda(y|x) f_i(x,y) \text{exp}(\delta_i f^{\#}(x,y))$$

Notice that in the expression for $\frac{\partial B(\Delta)}{\partial \delta_i}$ δ_i appears alone without the other parameters. Therefore, we can solve for each δ_i individually. The final IIS algorithm is as follows,

- Start with some arbitrary values for λ_is.
- Repeat until convergence
  - Solve for $\frac{\partial B(\Delta)}{\partial \delta_i} = 0$ for δ_i.
  - Set $\lambda_i = \lambda_i + \delta_i$

  for each i.

3.5 Swarm Intelligence

Swarm Intelligence (SI)[10] is a relatively new paradigm being applied in a host of research settings to improve the management and control of large numbers of interacting entities such as communication, computer and sensor networks, satellite constellations and more. Attempts to take advantage of this paradigm and mimic the behaviour of insect swarms however often lead to many different implementations of SI. Here, we provide a set of general principles for SI research and development. A precise definition of self-organized behaviour is described and provides the basis for a more axiomatic and logical approach to research and development as opposed to the more prevalent ad hoc approach in using SI concepts. The concept of Pareto optimality is utilized to capture the notions of efficiency and adaptability.

3.5.1 Foundations

The use of swarm intelligence principles makes it possible to control and manage complex systems of interacting entities even though the interactions between and among the entities is minimal.

As an example, consider how ants actually solve shortest path problems. Their motivation for solving these problems stems from their need to find sources of food. Many ants set out in search of a food source by apparently randomly choosing several different paths. Along the way they leave traces of pheromone. Once ants find a food source, they retrace their path back to their colony by following their scent back to their point of origin. Since many ants go out from their colony in search of food, the ants that return first are presumably those that have found the food source closest to the colony or at least have found a source that is in some sense more accessible. In this way, an ant colony can identify the shortest or *best* path to the food source.

The cleverness and simplicity of this scheme is highlighted when this process is examined from what one could conceive of as the ants perspective - they simply follow the path with the strongest scent (or so it seems). The shortest path will have the strongest scent because less time has elapsed between when the ants set out in search of food and when they arrive back at the colony, hence there is less time for the pheromone to evaporate. This leads more ants to go along this path further strengthening the pheromone trail and thereby reinforcing the shortest path to the food source and so exhibits a form of **reinforcement learning**.

But this simple method of reinforcement or positive feedback also exhibits important characteristics of efficient group behaviour. If, for instance, the shortest path is somehow obstructed, then the second best shortest path will, at some later point in time, have the strongest pheromone, hence will induce ants to traverse it thereby strengthening this alternate path. Thus, the decay in the pheromone level

leads to redundancy, robustness and adaptivity, i.e., what some describe as **emergent behaviour**.

Efficiency via Pareto Optimality

Optimization problems are ubiquitous and even social insects must face them. Certainly, the efficient allocation of resources present problems where some goal or objective must be maintained or achieved. Such goals or objectives are often mathematically modelled using objective functions, functions of decision variables or parameters that produce a scalar value that must be either minimized or maximized. The challenge presented in these often difficult problems is to find the values of those parameters that either minimize or maximize, i.e., optimize, the objective function value subject to some constraints on the decision variables.

In multi-objective optimization problems (MOPs) system efficiency in a mathematical sense is often based on the definition of Pareto optimality a well established way of characterizing a set of optimal solutions when several objective functions are involved. Each operating point or vector of decision variables (operational parameters) produces several objective function values corresponding to a single point in objective function space (this implies a vector of objective function values). A Pareto optimum corresponds to a point in objective function space with the property that when it is compared to any other feasible point in objective function space, at least one objective function value (vector component) is superior to the corresponding objective function value (vector component) of this other point. Pareto optima therefore constitute a special subset of points in objective function space that lie along what is referred to as the Pareto optimal frontier the set of points that together dominate (are superior to) all other points in objective function space.

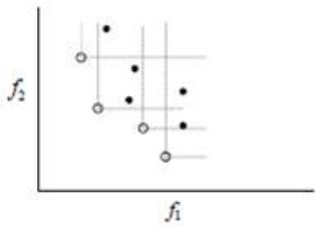

Figure 3.3: The Pareto Optimal frontier is the set of hollow points. Operational decisions must be restricted along this set if operational efficiency is to be maintained

Determining several Pareto optima can be quite valuable for enhancing the survival value of a species (or managing a complex system) because it enables adaptive behaviour. Thus, if in an ant colony a path to a food source becomes congested, then other routes must be utilized. Although the distances to food sources are generally minimized as is the level of congestion, these often conflicting objec-

tives can be efficiently traded off when the shortest distance is sacrificed to lessen the level of congestion.

The Measure of Pareto Optima: A rather intuitive yet surprisingly little known aspect of Pareto optima is its measure. This measure is based on the size of the set of points in objective function space that are dominated by the Pareto optimal frontier - in essence a Lebesgue measure or hypervolume.

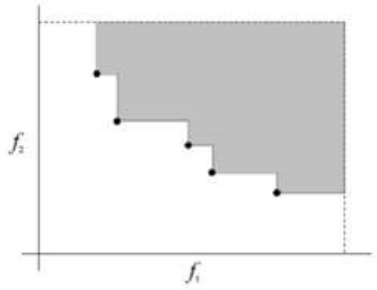

Figure 3.4: The Pareto hypervolume

3.5.2 Example Algorithms and Applications

- **Ant colony optimization**
  A class of optimization algorithms modelled on the actions of an ant colony, ACO is a probabilistic technique useful in problems that deal with finding better paths through graphs. Artificial 'ants' -simulation agents, locate optimal solutions by moving through a parameter space representing all possible solutions. Natural ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions.

- **Artificial bee colony algorithm**
  Artificial bee colony algorithm (ABC) is a meta-heuristic algorithm that simulates the foraging behaviour of honey bees. The algorithm has three phases: employed bee, onlooker bee and scout bee. In the employed bee and the onlooker bee phases, bees exploit the sources by local searches in the neighbourhood of the solutions selected based on deterministic selection in the employed bee phase and the probabilistic selection in the onlooker bee phase. In the scout bee phase which is an analogy of abandoning exhausted food sources in the foraging process, solutions that are not beneficial any more for search progress are abandoned, and new solutions are inserted instead of them to explore new regions in the search space. The algorithm has a well-balanced exploration and exploitation ability.

- **Particle swarm optimization**
  PSO is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some fitness criterion after each time-step. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large number of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.

3.5.3 Case Study: Ant Colony Optimization applied to the NP-hard Travelling Salesman Problem

Travelling salesman problem (TSP) consists of finding the shortest route in complete weighted graph G with n nodes and n(n-1) edges, so that the start node and the end node are identical and all other nodes in this tour are visited exactly once. We apply the Ant Colony[12] heuristic to obtain an approximate solution to the problem. We use virtual ants to traverse the graph and discover paths for us. Their movement depends on the amount of pheromone on the graph edges. We assume the existence of ant's internal memory. In symbols, what we have is:

- Complete weighted graph $G = (N, A)$
- N = set of nodes representing the cities
- A = set of arcs
- Each arc (i, j) in A is assigned a value (length) d_{ij} , which is the distance between cities i and j.

Tour Construction
τ_{ij} refers to the desirability of visiting city j directly after city i. Heuristic information is chosen as $\eta_{ij} = \frac{1}{d_{ij}}$.
We apply the following constructive procedure to each ant:

1. Choose, according to some criterion, a start city at which the ant is positioned;
2. Use pheromone and heuristic values to probabilistically construct a tour by iteratively adding cities that the ant has not visited yet, until all cities have been visited;
3. Go back to the initial city;

4. After all ants have completed their tour, they may deposit pheromone on the tours they have followed.

Continue for a fixed number of iterations or till the pheromone distribution becomes almost constant.

Ant System

The Ant System (proposed in 1991) uses the following heuristics and formulae for probability propagation

- Initialize the pheromone trails with a value slightly higher than the expected amount of pheromone deposited by the ants in one iteration; a rough estimate of this value can be obtained by setting

$$\tau_{ij} = \tau_0 = \frac{m}{C^{nn}}$$

  where m is the number of ants, and C^{nn} is the length of a tour generated by the nearest-neighbour heuristic.

- In AS, these m artificial ants concurrently build a tour of the TSP.

- Initially, put ants on randomly chosen cities. At each construction step, ant k applies a probabilistic action choice rule, called random proportional rule, to decide which city to visit next.

$$p_{ij}^k = \tau_{ij}^\alpha \eta_{ij}^\beta / \sum_{l \in N_i^k} \tau_{il}^\alpha \eta_{il}^\beta, \quad if\ j \in N_i^k$$

- Each ant k maintains a memory M_k which contains the cities already visited, in the order they were visited. This memory is used to define the feasible neighbourhood N_i^k in the construction rule.

- We can adopt any of the following two: *Parallel implementation:* at each construction step all ants move from current city to next one; *Sequential implementation*: ant builds complete tour before next one starts to build another

Update of Pheromone Trails

- Forget bad decisions:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \;\; \forall i,j, \text{where } \rho \in \{0,1\}$$

- So,if an arc is not chosen by the ants, its pheromone value decreases exponentially
- $\Delta\tau_{ij}^k$ is the amount of pheromone ant k deposits on the arcs it has visited and C^k is the length of tour T^k built by the k^{th} ant. Then, they are related as follows:

$$\Delta\tau_{ij}^k = 1/C^k, \text{ if arc } (i,j) \text{ belongs to tour } T^k; \text{ 0 otherwise}$$

- The update then happens as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k, \;\; \forall (i,j)$$

Computational Experiments For experiment, the problem of 32 cities in Slovakia has been solved using the ACO. The optimal solution to that problem has a length of route $1453km$. Parameters are $\alpha = 1, \beta = 5$. The number of iterations was set to 1000.

With $m = 1000$, the result was the tour with length $1621\ km$ in 34^{th} iteration (difference 11.56% from optimal route).

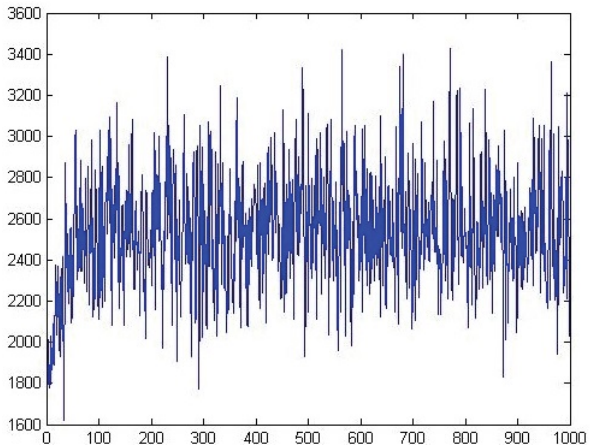

Figure 3.5: Search process for m=1000 ants

With $m = 5000$, algorithm ACO finds the tour with length $1532km$ in 21^{st} iteration (difference 5.44% from optimal route).

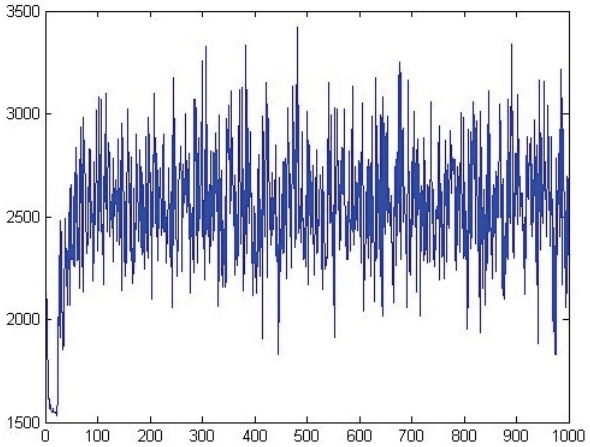

Figure 3.6: Search process for m=5000 ants

3.6 Boltzmann Machines

One of the first examples of a neural network capable of learning internal representations, Boltzmann machines[3] are able to represent and (given sufficient time) solve difficult combinatoric problems. They are named after the Boltzmann distribution in statistical mechanics, which is used in their sampling function.

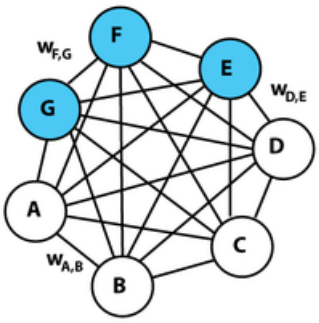

Figure 3.7: Graphical representation for a Boltzmann machine with a few labelled weights

3.6.1 Structure

A Boltzmann machine, is a network of stochastic units with an *energy* defined for the network. The global energy E, in a Boltzmann machine is:

$$E = -(\sum_{i<j} w_{ij}\, s_i\, s_j + \sum_i \theta_i\, s_i)$$

[3]Content and figure from `http://en.wikipedia.org/wiki/Boltzmann_machine`

where w_{ij} is the connection strength between unit j and unit i; $s_i \in \{0, 1\}$ is the state of unit i; θ_i is the bias of unit i in the global energy function.
The connections in a Boltzmann machine have two restrictions:

- $w_{ii} = 0 \qquad \forall i$. (No unit has a connection with itself.)
- $w_{ij} = w_{ji} \qquad \forall i, j$. (All connections are symmetric.)

3.6.2 Probability of a state

The difference in the global energy that results from a single unit i being 0(off) versus 1(on), written ΔE_i, is given by:

$$\Delta E_i = \textstyle\sum_j w_{ij}\, s_j + \theta_i$$

This can be expressed as the difference of energies of two states:

$$\Delta E_i = E_{\text{i=off}} - E_{\text{i=on}}$$

We then substitute the energy of each state with its relative probability according to the Boltzmann Factor (the property of a Boltzmann distribution that the energy of a state is proportional to the negative log probability of that state):

$$\Delta E_i = -k_B\, T \ln(p_{\text{i=off}}) - (-k_B\, T \ln(p_{\text{i=on}}))$$

where k_B is Boltzmann's constant and is absorbed into the artificial notion of temperature T. We then rearrange terms and consider that the probabilities of the unit being on and off must sum to one:

$$\begin{aligned}
\frac{\Delta E_i}{T} &= \ln(p_{\text{i=on}}) - \ln(p_{\text{i=off}}) \\
\frac{\Delta E_i}{T} &= \ln(p_{\text{i=on}}) - \ln(1 - p_{\text{i=on}}) \\
\frac{\Delta E_i}{T} &= \ln\left(\frac{p_{\text{i=on}}}{1 - p_{\text{i=on}}}\right) \\
-\frac{\Delta E_i}{T} &= \ln\left(\frac{1 - p_{\text{i=on}}}{p_{\text{i=on}}}\right) \\
-\frac{\Delta E_i}{T} &= \ln\left(\frac{1}{p_{\text{i=on}}} - 1\right) \\
\exp\left(-\frac{\Delta E_i}{T}\right) &= \frac{1}{p_{\text{i=on}}} - 1
\end{aligned}$$

We can now solve for $p_{\text{i=on}}$, the probability that the i^{th} unit is on.

$$p_{\text{i=on}} = \frac{1}{1+\exp(-\frac{\Delta E_i}{T})}$$

where the scalar T is referred to as the temperature of the system. This relation is the source of the logistic function found in probability expressions in variants of the Boltzmann machine.

3.6.3 Equilibrium State

The network is run by repeatedly choosing a unit and setting its state according to the above formula. After running for long enough at a certain temperature, the probability of a global state of the network will depend only upon that global state's energy, according to a Boltzmann distribution. This means that log-probabilities of global states become linear in their energies. This relationship is true when the machine is at *thermal equilibrium*, meaning that the probability distribution of global states has converged. If we start running the network from a high temperature, and gradually decrease it until we reach a thermal equilibrium at a low temperature, we may converge to a distribution where the energy level fluctuates around the global minimum. This process is called **simulated annealing**.