Chapt 7 Stochastic Methods

We've considered:

analytic methods ← computing derivatives / solving eqns ⟹ optimal model parameters

for complicated models — local derivatives → gradient methods — ML / NN

These approach don't scale well

Direct search, i.e, exhaustive search is impractical

⟹ stochastic search

idea: 1) bias search towards promising regions
2) let randomness drive exploration

Text discusses 2 classes of methods:

1) ~~Boltzmann~~ Simulated annealing / Boltzmann learning
2) evolutionary alg. i.e genetic alg. & genetic programming

⟹ we'll start with Genetic Algorithms

7.2 Stochastic Search

Assume variables S_i $i=1,\ldots,N$
S_i are discrete values i.e. ± 1

$E = -\frac{1}{2}\sum_{i,j=1}^{N} w_{ij} S_i S_j$ is the energy in this system

Goal: choose values of S_i so as to minimize E

note: $w_{ii}=0$ w_{ij} are symmetric bidirectional weighted edges i.e. interaction between S_i

Obs: 2^N possible configuration

$\Longrightarrow$ not practical to solve directly

7.2.1 Introduction of Simulated annealing (Need to refresh classes memory?)

7.2.2 Boltzmann factor

Each configuration (choice of S_i values) is indexed by γ

and has energy E_γ has the probability

$$P(\gamma) = \frac{e^{-E_\gamma/T}}{Z(T)}$$

where Z is a normalization constant and T is temperature.

The numerator: $e^{-E_\sigma/T}$ is the Boltzmann factor

The denominator is the partition function

$$Z(T) = \sum_{\sigma} e^{-E_\sigma/T}$$

i.e. the sum over all possible configurations

$\Rightarrow$ $P(\sigma)$ is a true probability

Note σ ranges from 1 to 2^N

$\Rightarrow$ not practical to compute $Z(T)$ in general

Simulated Annealing Alg

Start with: - High $T(1)$
- random states S_i

choose node i randomly suppose $S_i = +1$
- calculate E_a for S_i in state $+1$
- calculate E_b for S_i in state -1

if $E_b < E_a$ accept $S_i = -1$

Else accept $S_i := -1$ with probability

$e^{-\Delta E_{ab}/T}$ where $\Delta E_{ab} = E_b - E_a$

Note: with high T energetically less favorable state will be accepted more often

$\Rightarrow$ local energy minima possible to escape

Or with large enough T all configurations are ~ equal probs

repeat $\Rightarrow$ continue randomly selecting + testing nodes

$\Rightarrow$ gradually decrease T

Stop when T is very low

Show Algorithm 1 (Stochastic Simulated Annealing)

note 1 N_i denotes set of nodes connected with nonzero weights w_{ij} to node i

Note 2 when comparing E_a + E_b $\Rightarrow$ only consider nodes directly connected to S_i since S_i only affects its immediate neighbors in the network.

Issues: starting value of T
ending value
cooling schedule possibly $T(k+1) = cT(k)$ $0 < c < 1$
stopping criterion

$T(1)$ should be high enough that all configurations are roughly equally likely.

$\Longrightarrow T(1) >>$ any ΔE_{ab}

$\Longrightarrow$ cooling must be gradual $\Rightarrow$ in practice $0.8 < c < 0.99$
we need at least $N/2$ transitions to allow

why? $\Longrightarrow$ # transitions from any configuration to the global optimum is at most $N/2$ moves

- Show fig 7.3 explain

Show fig 7.4 focus on probability of state $p(\gamma)$ and expected energy $E[E]$ as a function of T

7.2.5

Deterministic Annealing

~~Satta~~ Idea: let S_i be continuous and dependent on T

i.e. $S_i = f(l_i, T) = \tanh[l_i/T)$

when $l_i = \sum_j w_{ij} s_j$ i.e. effect of node connected to S_i

Show fig 7.5 explain

$\Rightarrow$ with large T even large l_i does not force discrete S_i

$\Rightarrow$ small T $\Rightarrow$ discrete S_i with smaller l_i

Show Algorithm 2 (Deterministic Simulated Annealing)

Boltzmann Learning

note: recall concept of opposing magnets connected in a network

modification: separate magnets/nodes into two sets

α – input/output nodes $\Rightarrow \alpha^0 + \alpha^i$

β – internal, hidden nodes

Concept: clamp – hold input nodes fixed to the input value during classification

Probability of a visible configuration

$$\Rightarrow P(\alpha) = \sum_{\beta} P(\alpha, \beta)$$

$$= \frac{\sum_{\beta} e^{-E_{\alpha\beta}/T}}{Z}$$

where $E_{\alpha\beta}$ is the energy defined of the entire system

i.e $-\frac{1}{2} \sum_{i,j=1}^{N} w_{ij} s_i s_j$ (from last class)

and Z is again the old partition function $\sum_{\gamma} e^{-E_\gamma/T}$

Q: How can we measure the difference between the desired probability of visible state α & actual " " " α ?

A: Kullback-Leibler divergence i.e. relative entropy

$$D_{KL}(Q(\alpha), P(\alpha)) = \sum_{\alpha} Q(\alpha) \log \frac{Q(\alpha)}{P(\alpha)}$$

note: D_{KL} is zero only if $P(\alpha) = Q(\alpha)$ for all α

also - depends only on visible units α

Learning: idea: adjust weights based on difference between desired and actual visible states

$$\Rightarrow \Delta w_{ij} = \eta \frac{\partial D_{KL}}{\partial w_{ij}}$$

η ↑ learning rate

$$= \frac{\eta}{T} \left[\mathcal{E}_Q[s_i s_j]_{\alpha \text{ clamped}} - \mathcal{E}[s_i s_j]_{\text{free}} \right]$$

learning component aka teacher component

unlearning component aka student component

Stochastic Learning of Input-Output Association

idea: clamp $\alpha^i + \alpha^o$ in learning component
clamp only α^i in unlearning component

$$\Rightarrow \Delta w_{ij} = \frac{\eta}{T}\left[\varepsilon_Q[s_i s_j]_{\alpha^i \alpha^o \text{ clamped}} - \varepsilon[s_i s_j]_{\alpha^i \text{ clamped}}\right]$$

show fig 7.8.

Evolutionary Methods

Inspiration: biological evolution

note: biological evolution is massively parallel

→ these algorithm are naturally parallel

basic idea: population of individuals are instances a soln to an optimization problem

fitness function: score individual

⇒ fit individuals reproduce

⇒ offspring inherit traits from parents

Example: 3-D conformation of a molecule

good: ⇒ find most probable conformers

⇒ lowest energy state solutions

⇒ may be several stable states

reproduction would ~~combine~~ combine partial solns from best/fittest parents

Text focuses on population of classifiers 7.5.2

⟹ optimization goal: find a good classifier

notion: chromosome – binary string: encodes classifier

note: Two requirements for using G.A. approach!

1) binary string encoding of soln instance
2) fitness fcn for evaluating individual solns.

show Alg 4

Θ – fitness function
p_{co} – probability of crossover
p_{mut} – probability of site mutation

show fig 7.13

Genetic operators!

Replication: copy chromosome → no change

show fig 7.14

Crossover: 2 chromosomes are "mated"

1) randomly select a split point
2) cut both chromosomes at same point
3) recombine to get 2 new chromosomes

Mutation : bit flip

All bits in a chromosome are evaluated for mutation

Issue: Representation : How is the classifier encoded in a bit string?

Singlet approach: bits represent features

Example from text: bits specify feature in 2-layer perceptron (ie pixel features)

(1) ⇒ mapp out certain parts of string for certain features

Cross-over will mostly preserve segments

This allow good partial solns to be combined

or

(2) chromosome segments represent w in Multilayer NN.

⟹ need fixed topology (for mapping purpose)

or

(3) [show fig 7.15] chromosome segments may be parts of a decision tree.

Selection: 7.5.4

Fitness function: to choose ancestors of next generation

Pitfall: Q: What happens if we select only the most fit individuals

A: Approach a monoculture
⇒ inbreeding ☹ → premature convergence!!!

Note: progress depends on variance in a population

Soln 1. Always keep a few low scoring individuals

Soln 2. Use a probabilistic scoring function
→ occasionally low fitness individual selected
idea: probability of selection ∝ fitness

Example: show slide: $P(i) = \frac{e^{f_i/T}}{\mathcal{E}[e^{f_i/T}]}$

- probability of selecting chromosome i
- with fitness f_i
- and temperature T (control parameter)
- Note: expectation is over current generation

temp is high early ⇒ all chromosomes have good chance for selection
later → lower temperature

Genetic Programming

like GA, but different representation

GA → chromosomes are bit strings
GP → chromosomes are of computer programs

Show Fig 7.16

As with GA, we have:

<u>replication</u> - as before

<u>cross-over</u>: restrictions on valid snipping points

<u>mutation</u>: Element may mutate
i.e. number$_1$ → number$_2$
operator 1 → operator 2 (must have same arity)

<u>Insertion</u>: randomly insert code snippet from a set

Representation: must use some language

Syntactically Rich languages are difficult to use eg C++

⟹ LISP : syntactically very simple ⇒ already parsed by parentheses

LISP expressions are of the form

(<Operator> <operand$_1$> ...) typically 1 - 2 operands

show Fig 7.17

notion: wrapper — decides if code snippet is meaningful
→ cull ungramatical code snippets