Data perturbation for Escaping Local Maxima in Learning

By

Gal Elidan and Matan Ninio and Nir Friedman

Dale Schuurmans

Outline

Background

Basic techniques for perturbing weights to escape local maxima:

- Random Reweighting
- Adversarial Reweighting

Performance Evaluation

Benefits of a basic approach

Background

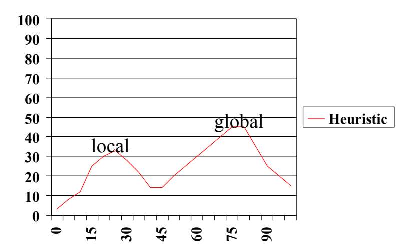
What is Data Perturbation?

What is Local Maxima?

Main idea to Escape Local Maxima?

- Change in training data to create useful ascent directions in hypothesis space rather than changing hypothesis data directly

Local Maxima & Global Maxima



Score of a Hypothesis

Optimization : search for the hypothesis that maximizes the score on the training data

Score of the hypothesis h on data D = {x[1],…, x[m]} is a sum of local scores on each individual example – additive

Score(h, D) = Σ_m score(h,x[m]) – penalty(h)

On reweighting the examples the score is augmented on considering probability distribution *w*

score(h, D, w) = Σ_m *M*. w_m score(h, x[m]) – penalty(h)

Greedy Hill climbing Search

Algorithm:

- expand the current state
- make the expanded state with the highest objective function value the next current state
- repeat

Reasons for finding search for escaping local maxima

Global maximum is intractable for decision trees, neural networks etc

Use local search techniques for finding the locally optimal hypotheses

Drawbacks:

Local maxima is common

Local search often yields poor results.

Previous Research

Variety of techniques developed to escape local maxima:

- Random restarts
- TABU search
- Simulated Annealing

But these all above techniques alter hypothesis in an oblivious fashion until it escape from local maxima

General Search Procedure

Procedure Perturbed Search(D, w^0, h^0, T^0, T_{final})

t $\longleftarrow$ 0

while $T^t > T_{final}$ do

$w^{t+1} \longleftarrow$ *reweight*(Score, w^t, T^t, h^t, D)

$h^{t+1} \longleftarrow$ *optimize*(Score, w^{t+1}, T^t, h^t, D)

$T^{t+1} \longleftarrow$ *reduce*(T^t, t)

$t \longleftarrow t + 1$

return h^t

Random Reweighting

Idea: Randomly samples weight profiles on training data

Motivated by Iterative local search methods in combinatorial optimization

Algorithm:

- Randomly reweighting the training example
- Evaluate the candidate hypotheses
- Standard optimization on perturbed score
- Repeat the process until weight perturbation reaches zero.

Random Reweighting

How the weights reach uniform distribution?

- probability distributed over M data instances
- Sample with a *Dirchelet* distribution with parameter β

$P(W = w) \alpha \Pi_m w_m^{\beta-1}$ where $\beta = 1/T^t$

As β grows larger the distribution reaches uniform distribution, since T^t decreases with number of iterations.

Adversarial Reweighting

Idea: Update the weights to directly challenge the current hypotheses

Motivated by exponential gradient search for constrained optimization problems

Weight update:

$w_m^{t+1} \longleftarrow w_m^t - \eta^*$ Score/ w_m

Here the score behaves like langrangian I.e. the local search attempts to maximize the score where as weight update attempts to minimize the score, in an adversarial fashion.

This approach is applicable whenever the original score is differentiable with respect to the weights.

Gradient of Langrangian

The Lagrangian used for the Adversarial Reweighting is of the form:

$L(h,w^{t+1}) = Score(h, D, w^{t+1}) + \beta\, KL(w^{t+1}||w^0) + \gamma\, KL(w^{t+1} || w^t)$

The derivative of $KL(w^{t+1}||w^0)$ and $KL(w^{t+1} || w^t)$ is simply

$$\frac{\Sigma_m w_m^{t+1} \log(w_m^{t+1} / w_m^0)}{w_m^{t+1}} = \log(w_m^{t+1} / w_m^0) + 1$$

$$\frac{\Sigma_m w_m^{t+1} \log(w_m^{t+1} / w_m^t)}{w_m^{t+1}} = \log(w_m^{t+1} / w_m^t) + 1$$

KL — Kullback-leilber measure

Relation to ensemble reweighting

- Boosting derives the weight update by differentiating the loss of an entire ensemble but here it is derived by taking only the derivative of the score of the most recent hypotheses
- Produce hypotheses that generalize well to unseen test data with out exploiting a large ensemble
- Hypotheses that obtain good scores is robust against perturbations of the training data, which confers generalization benefits

Learning Bayesian Networks from Data

Learning Bayesian network structure from complete data (structure search)

Optimizing Bayesian network parameters from incomplete data (Parametric EM)

Learning Bayesian network structure from incomplete data (Structural EM)

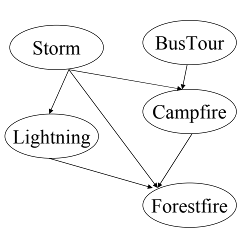
Idea: Learn the Bayesian network B that best matches D, for each of the above scenarios

Bayesian Network

Annotated directed acyclic graph that encodes a joint probability distribution over *x*

- *x* = { $x_1, x_2, \ldots, x_n$} finite set of random variables.
- nodes = $x_1, x_2, \ldots, x_n$ each annotated with *conditional probability distribution* $P(x_i | U_i)$.
- A Bayesian network B specifies a unique probability distribution over *x* : $P(x_1, x_2, \ldots, x_n) = \Pi_{i=1}^n P(x_i | U_i)$.

Bayesian Network



	S, B	S, ¬B	¬S, B	¬S, ¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

Campfire

P(Campfire = True|Storm = True, BusTour = True) = 0.4

Perturbing Structure Search

Idea: Search for a network structure B that best matches our training set D

BDe score is a function of simple *sufficient statistics* (S) of the data.

$S(D) = \Sigma_m s(x[m])$

Where s( ) is a function of a particular instance

S(D) counts the number of times an event occurred in the data

Perturb score: $S(D, w) = \Sigma_m M . W_m . s(x[m])$

Local search for finding the structure continues until convergence to a local maximum

Experimental Evaluation

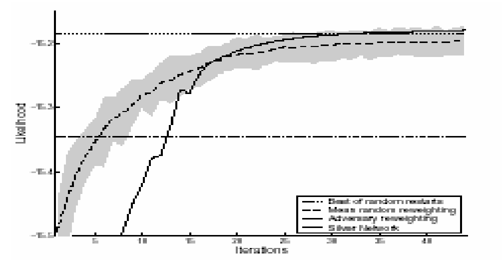
- Compare results to the golden model that has additional prior knowledge of structure
- Compare results with greedy hill climbing search augmented with TABU search and performs random restarts to improve the quality of results
- Results can be evaluated both in terms of scores on training data and generalization performance on test data

Outcome:

- Both measures correlate closely
- Both outperformed the random-starts method and golden model(synthetic *Alarm* network)

On overall *Random* method is best but on average *Adversary* method is best.

Test Set Performance



progress of the test set Likelihood (log-loss/instance) during iterations

Perturbing Parametric EM

Idea: Learning with incomplete data

Training instance in this case is not complete instance x[m] instead partial instance o[m]

Reason: Training Examples may have missing values or variables

Expectation-maximization is the common method used when sufficient statistics cannot be estimated due to incomplete instance

$P_0(X_1,X_2,\ldots X_n)$ is used to calculate expected sufficient statistics

$E[S(D) \mid P_0] = \Sigma_m \Sigma_{x[m]} s(x[m]) P_0(x[m] \mid o[m])$

$P_0(x[m] \mid o[m])$ is the probability of the complete instance given partial observation

Objective: Find the likelihood of the model on training data

Escaping Local Maxima

Reweighted Expected sufficient statistics using current weight vector:

$E[S(D) \mid P_0] = \Sigma_m M.w_m \Sigma_{x[m]} s(x[m]) P_0(x[m] \mid o[m])$

- Expected score is not the actual maximum point of true score
- Bias :

Models which are similar to the one with which expected sufficient statistics are calculated

Experimental Evaluation

Compared the methods with the alarm network

Results:

Adversary perturbation takes 15 times longer than single Parametric EM

Random perturbation takes 50 times longer than single Parametric EM

Perturbing Structural EM

Idea: Find an optimal structure for each iteration and then optimize the parameters with respect to that structure.

- Compute the *expected sufficient statistics*
- Search for the structure using the structure score
- Optimize the expected score

Experiments with real-life data

Two methods were applied for the real life data sets like soybean disease database

Have missing variables

Performed 5-fold cross validation and compared the log-loss performance on independent test data

Results of Test data set for several datasets for structure search and SEM

	Domain	Random	80%	Adv
Search	Stock	- 002	0.01	0.03
	Alarm	0.15	0.18	0.17
SEM	Rosetta	- 005	0.27	0.09
	Audio	0	0.39	0.23
	Soybean	0.19	0.32	0.19
	Alarm	0.254	0.31	0.33

Advantages

- Perturbation schemes are general and can be applied large variety of hypothesis spaces
- Use standard search procedure to find hypotheses

Future Work

- Combination of randomized element within the adversarial strategy
- Improve the implementation to reduce the number of iterations for realistic applications
- Explore improved ways to interleave the maximization and reweighting steps

Data Perturbation for Escaping Local Maxima in Learning

Gal Elidan, Matan Ninio, Nir Friedman
Hebrew University

Dale Schuurmans
University of Waterloo

Presented By
Kiran Vuppla

Comments By
Harsh Bapat

Weight Annealing Example

- TF = sum (w(i) * cos(PI * i/20 * x))
  - where i = { 1...20 }
  - w(i) = weight of each point
- Consider K points (20) equally spread out between 0 and 1
- Fit those point with a **cosine** function
- Look for x that maximizes TF

Weight Annealing implementation available at
http://www.cs.huji.ac.il/labs/learning/weightAnnealing/

No Annealing

- Initial weights 1/20 to all examples

```
Starting at -33.213
X=-33.672 Y=0.027
Finished with X=-33.672 Y=0.027
```


Random Annealing

- Initial weights = 1.0 for all examples
- Start Temp. = 10, End Temp. = 0.01, Cooling factor=0.99

```
X=-22.713 Y=-0.894 at temp=10.000
W[1] = 0.006W[2] = 0.017W[3] = 0.000W[4] = 5.755W[5] = 0.014W[6] =
   0.000W[7] = 0.001W[8] = 0.000W[9] = 2.998W[10] = 0.267W[11] =
   0.211W[12] = 3.339W[13] = 0.000W[14] = 0.000W[15] = 0.000W[16] =
   0.339W[17] = 0.000W[18] = 3.439W[19] = 3.612W[20] = 0.000
X=-22.212 Y=-0.161 at temp=9.900
W[1] = 0.000W[2] = 0.000W[3] = 0.120W[4] = 0.042W[5] = 0.101W[6] =
   0.235W[7] = 0.000W[8] = 0.000W[9] = 0.000W[10] = 2.920W[11] =
   0.108W[12] = 0.000W[13] = 0.000W[14] = 0.000W[15] = 1.217W[16] =
   0.000W[17] = 0.000W[18] = 15.248W[19] = 0.000W[20] = 0.009
                              .
                              .
                              .
                              .
X=-20.000 Y=0.000 at temp=8.345
X=-20.000 Y=0.000 at temp=8.262
Finished with X=-20.000 Y=0.000 after 20 loops
```


Adversarial Annealing

- Initial weight = 1.0 for all examples
- Start temp=5 End temp=0.01 Cooling rate= 0.99

```
Starting at -15.661
X=-14.324 Y=-0.034 at temp=5.000
X=-10.432 Y=0.062 at temp=4.950
X=-12.229 Y=0.107 at temp=4.901
X=-13.389 Y=-0.939 at temp=4.851
X=-17.003 Y=-1.001 at temp=4.803
X=-15.413 Y=-0.817 at temp=4.755
X=-16.525 Y=-0.400 at temp=4.707
X=-20.000 Y=0.000 at temp=4.660
X=-18.210 Y=-0.061 at temp=4.614
X=-1.638 Y=-3.795 at temp=4.568
X=-3.250 Y=-2.208 at temp=4.522
X=-2.119 Y=1.053 at temp=4.477
X=-0.000 Y=20.000 at temp=4.432
X=-0.000 Y=20.000 at temp=4.388
Finished with X=-0.000 Y=20.000 after 14 loops
```

Simulated Annealing

- Optimization Technique
- Also known as the Metropolis algorithm.
- Better Cooling and More Iterations lead to better results.
- Parallel Simulated Annealing: speed up SA.

Basic Idea:

- Starts with a high temperature T and any initial state.
- A neighborhood operator is applied to the current state *i* (having energy E_i) to yield state j (energy E_j).
- If $E_j < E_i$, j becomes the current state.
- Otherwise j becomes the current state with probability , $1+e^{(Ei - Ej)/T)}$. If j is rejected, then *i* remains the current state.
- All steps after the second one are repeated either a fixed number of times or until a quasi-equilibrium is reached.
- The entire above procedure is preformed repeatedly, each time starting from the current i and lower T.