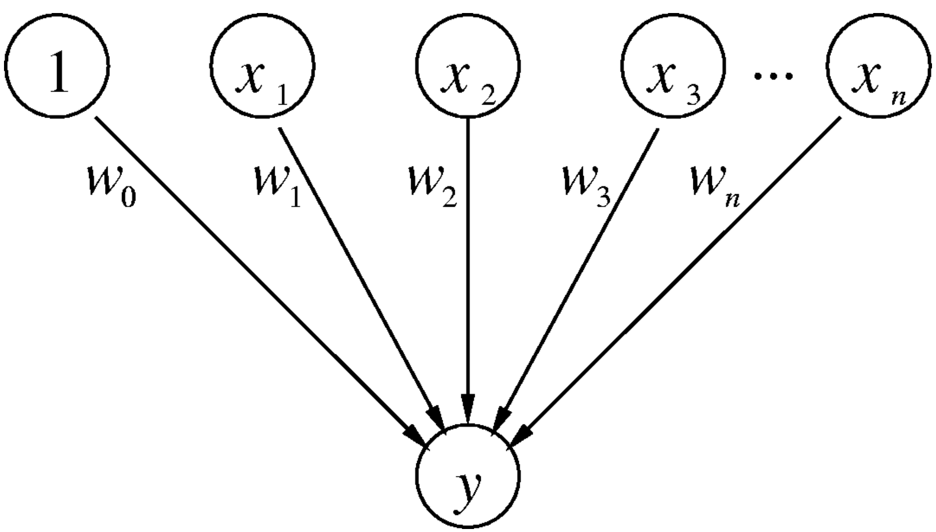
Strided Sampling Hashed Perceptron Predictor

Daniel A. Jiménez

Department of Computer Science & Engineering
Texas A&M University

Branch-Predicting Perceptron

- Inputs (x' s) are from branch history
- $n + 1$ small integer weights (w' s) learned by on-line training
- Output (y) is dot product of x' s and w' s; predict taken if $y \geq 0$
- Training finds correlations between history and outcome
- Keep a table of perceptron weights vectors selected by hash of PC



$$y = w_0 + \sum_{i=1}^{n} x_i w_i$$

Neural Prediction in Current Processors

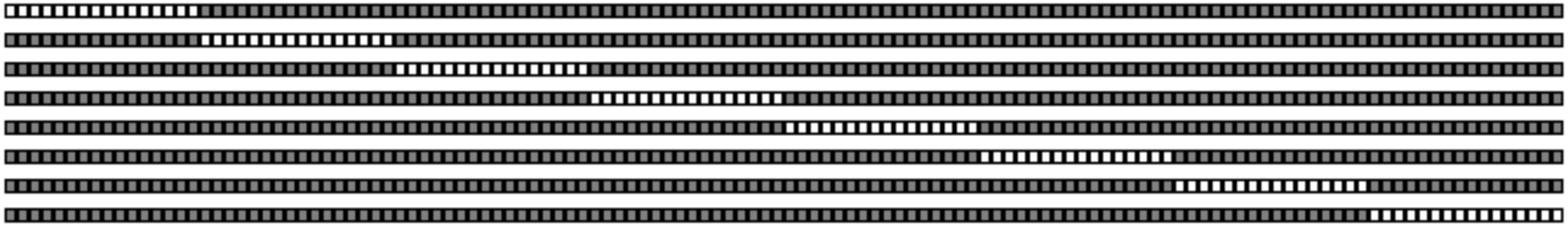
- We introduced the perceptron predictor [Jiménez & Lin 2001]
  - I and others improved it considerably through 2011
- Today, Oracle SPARC T4 contains S3 core with
  - “perceptron branch prediction”
  - “branch prediction using a simple neural net algorithm”
  - Their IEEE Micro paper cites our HPCA 2001 paper
  - You can buy one today
- Today, AMD “Bobcat” core used in C- and E-series APUs has
  - “neural net logic branch predictor”
  - You can buy one today

Hashed Perceptron

- Introduced by Tarjan and Skadron 2005
- Breaks the 1-1 correspondence between history bits and weights
- Basic idea:
  - Hash segments of branch history into different tables
  - Sum weights selected by hash functions, apply threshold to predict
  - Update the weights using perceptron learning

Choosing History Bits for Hashed Perceptron

- There is infinite flexibility in how we choose which bits to hash for which table
- For example, a naïve choice for history of 128 and 8 tables would be to choose non-overlapping equal-length chunks of history:



Choosing Bits cont.

- A better choice is geometric histories [Seznec 2005]
- This is the idea behind GEHL, and leads to TAGE if we use tagging instead of summing to find a prediction

Strided Sampling

- My idea: strided sampling
  - Choose "samples," i.e. variable length chunks starting at arbitrary history positions
  - The samples are chosen with arbitrary stride, i.e., the chunks are not all sequential but strided

- A few samples chosen per table
- Samples found with stochastic search (genetic algorithm)

Sample Parameters

- a – the starting history position of the sample [0..history-1]
- b – 1 past the ending history position of the sample [a..history-1]
- c – the source of the history [0..2]
  - global history, path history, or callstack history
- d – which table this sample pertains to [0..num_tables-1]
- e – the stride [1..8]

Learning the Samples

- On average, there are three samples per table
- The samples are learned by a genetic algorithm
  - Start with a population of random sets of samples
  - Mutate with low probability
  - Crossover to combine samples from different sets
  - Evaluate MPKI on the traces for each set (takes a long time)
  - Keep better sets of samples in the population
  - Continue with second step until convergence
- Computationally intensive – used custom parallel GA code
- Interestingly, GA often chose to use *a*=*b*, i.e. a Smith predictor or bias weight, for a few tables; basically it invented a skewed bimodal predictor [Michaud *et al*. 1997] or multiple bias tables [Jiménez 2004] as a component

Specialization

- Evolve a set of samples for each benchmark
- Keep a general set of samples in case of unknown benchmark
- In practice, program would be profiled to develop samples that would be communicated to the program on the next run, e.g. [Mahlke & Natarajan 1996] [Patil & Emer 2000][Jiménez *et al*. 2001] [Sherwood & Calder 2001][Jiménez 2005][Farooq & John 2013]
- Moin's little address shifting trick checkmates this approach
  - Wisdom of Sulayman
- I don't know what the results will be
  - Probably bad; I tuned with specialization as part of the mix and now the game has changed

Other Tricks

- Additional weights table indexed by local history
- Trivial branch filter – branches that are always or never taken don’t update the perceptron predictor
- Static predictor – always predict “not taken” first time
- Adaptive threshold training – as in Seznec’s O-GEHL
- Threshold fuzzing – compare to random value near θ
- Coefficient training – as in OH-SNAP [Jiménez 2011]
- Use history from other jumps (unconditional, indirect)
- Crazy hash functions from weirdos on the Internet

Predictor Parameters

- Some of the more important parameters:

Parameter	4KB	32KB
Number of perceptron tables	16	32
Bits per weight	6	6
Maximum history length	896	896
Entries per table	263	1109
Branch filter entries (2 bit)	1218	16384
Bits per coefficient logs	18	18
Local PHT size	512	2048
Local history length	5	7
Local histories	64	256

Results

- 4KB – 3.302 MPKI
- 32KB – 2.349 MPKI
- Unlimited – 1.860 MPKI

- These results are generated using the infrastructure originally distributed with CBP4. At the workshop the organizers presented alternate results using an infrastructure modified to defeat my specialization optimization.

Future Work

- What can we learn from samples to develop better predictors?
- Is there a way to quickly learn a set of samples for a program?
- Can we specialize dyncmially, e.g. set-dueling [Qureshi *et al*. 2006]?
- Can strided samples improve TAGE predictors?
  - I tried it; it didn't work but I only spent a couple of hours on it
- How close are we to the information-theoretic limit on accuracy?
  - Kolmogorov complexity would seem to imply that this is unknowable
- Can string matching help?
- Is it cruel to ask Ph.D. students to work on branch prediction?
  - Like throwing papers into a black hole