Machine Learning in Computer Systems Research

Daniel A. Jiménez

Department of Computer Science & Engineering
Texas A&M University
Improving computer systems with machine learning

Computer systems:
- Architecture / microarchitecture
- Programming languages / compilers / run-time
- Operating systems

Machine learning:
- Using data to build a model of some aspect of a system
- Then using that model to improve the system
- Could be online or offline
Many Areas Have Been Explored

Cache partitioning
Memory controllers
Branch prediction
Prefetchers
Voltage scaling
Predicting path profiles
Improving GPU throughput
Resource management
Microprocessor design as a whole
Code scheduling
Code completion
Malware detection
Etc. etc. etc.
This talk

Other work in static branch prediction

Some of my and others’ work in dynamic branch prediction

Some of my work in cache management

Other work in cache management

Other work in other areas

Conclude
Branch Prediction (my favorite!)

Branch prediction is a natural problem for machine learning

Dynamic conditional branch prediction – based on binary inputs with a single output, billions of training pairs

Static branch prediction – big training corpus of existing programs with profile information, many ways to analyze features

Many examples in the literature
Static Branch Prediction

Calder et al., Corpus-based Static Branch Prediction, PLDI 1995
State-of-the-art heuristics (Ball and Larus, PLDI 1993) got ~25% misprediction rate
Calder et al. improved to ~20% misprediction rate
Used neural networks and a large corpus of programs
Features included control-flow idioms, opcodes, etc.
Their TOPLAS 1997 article also used decision trees
They used a few features and simple FFNNs.
What would be today’s approach?
Predicting Path Profiles


Uses deep learning to statically identify hot paths through a program

Output is a probable sequence of basic blocks

Problem maps well to a recurrent neural network

They show improvement over state of the art heuristics
Dynamic Branch Prediction


We propose using neural learning in the branch predictor

Simple perceptrons (individual neurons) have good accuracy

Latency was addressed in subsequent research:

- Jiménez in MICRO 2003, ISCA 2005
- Seznec’s O-GEHL in CBP, 2004
- Tarjan and Skadron’s hashed perceptron in TACO 2005
- Loh and Jiménez, WCED 2005
- Etc.

Now it’s in processors from AMD, SPARC, and Samsung
Branch-Predicting Neuron

Inputs ($x$’s) are from branch outcome history – taken or not taken
$n + 1$ 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

\[ y = w_0 + \sum_{i=1}^{n} x_i w_i \]
Accuracy Affected by Non-Linearity

Perceptrons can’t compute non-linear functions
Some branches have non-linear behavior

AND

XOR
Accuracy Improves With Path-Based Piecewise Linear Prediction

Maintains low latency, improves accuracy [ISCA 2005, TACO 2009]
Current approaches with hashing similarly overcome non-linearity

perceptron prediction

piecewise linear prediction
Cache Management

Cache placement/replacement/bypass

Prefetching
I thought this was a really nice result

The idea:

- LRU is boring. Place in MRU, promote to MRU
- Can we promote based on current position, and have a better placement heuristic?

Enormous search space. We applied genetic algorithms, leading to the first pub [Jiménez, MICRO 2013]. Practical design for PseudoLRU (less hardware, less read/modify/write)

Tried harder with multi-core workloads

Genetic algorithm found a simple recursive algorithm for placement and promotion! [Terán & Jiménez, HPCA 2016]
Minimal Disturbance Promotion [HPCA 2016]

To promote a block $B$

Find smallest unprotected region containing $B$

Move the first block in that region to MRU (i.e. do normal PLRU promotion on that block)

The rest of the blocks move with that block and are now protected

A minimal number of bits have been changed to protect $B$
Reuse Prediction

Dead block prediction – predicting whether a block will be used again before it’s evicted

Can be used for a variety of optimizations:
  - Placement/Replacement
  - Bypass
  - Prefetching
  - Etc.

We use perceptron learning to do dead block prediction

Reuse prediction sounds nicer, I’m trying to promote that term
Perceptron Learning for Reuse Prediction
[Terán and Jiménez, MICRO 2016]

Combine multiple features $F_{1..n}$
Each feature indexes different table $T_{1..n}$
$y_{out} = \text{sum of counters from tables}$
Predict dead if $y_{out} > \tau$
Sampler provides training data
Perceptron Learning Rule:
if mispredict or $|y_{out}| < \theta$ then
  for $i \in 1..n$
    $h = \text{hash}(F_i)$
    if block is dead $T[h]++$
    else $T[h]--$
Predictor Organization

6 tables
256 entries
6-bit weights
Per-core vectors

Features:
✓ $Pc_0$
✓ $Pc_1 >> 1$
✓ $Pc_2 >> 2$
✓ $Pc_3 >> 3$
✓ Tag of current block >> 4
✓ Tag of current block >> 7
Better Accuracy

Coverage rate:
- SDBP: 47.2%
- SHiP: 43.2%
- Perceptron: 52.4%

False positive rate:
- SDBP: 7.4%
- SHiP: 7.7%
- Perceptron: 3.2%

Here, false positive rate is false positives / all predictions
Multiperspectival Reuse Prediction

[Jiménez and Terán, MICRO 2017]

Take perceptron idea one step further

Use many different features to adapt to workload behavior

Huge search space; use genetic algorithm to select features

Significantly improved performance over (then) state of the art

One contribution was the set of parameterized feature

Another is the feature selection process
Configuring Hardware Prefetchers

Liao et al., *Machine Learning-Based Prefetch Optimization for Data Center Applications*, SC 2009

Authors evaluate several classifiers to predict the best configuration of the four Intel Core 2 hardware prefetchers

- Nearest neighbor
- Naïve Bayes
- C4.5 decision tree
- Ripper classifier
- Support vector machines
- Neural (multi-layer perceptron and radial basis function)

Performance within 1% of optimal configuration
Reinforcement Learning for Prefetching

Peled et al., Semantic Locality and Context-Based Prefetching Using Reinforcement Learning, ISCA 2015

Design a hardware prefetcher using reinforcement learning online

Use “contextual bandits” model (generalization of “multi-armed bandits”)

Online algorithm:
- Collects history data for learning, does feature selection
- Predicts using current context to generate prefetches
- Updates predictors based on observing results

Outperforms SMS
Many More Ideas!

Ipek et al., Self-Optimizing Memory Controllers: A Reinforcement Learning Approach, ISCA 2008

Wang & Ipek, Reducing Data Movement Energy via Online Data Clustering and Encoding, MICRO 2016

Won et al., Online Learning in Artificial Neural Networks for CMP Uncore Power Management, HPCA 2014

Rahman et al., Maximizing Hardware Prefetch Effectiveness with Machine Learning, HPCC 2015

Many More Ideas! continued


Qiu et al., *Phase-Change Memory Optimization for Green Cloud with Genetic Algorithm*, IEEE TOCS 2015


Many More Ideas! continued


I’m sure I’ve forgotten some people; feel free to shout out
Compiler etc. community too!

Moss et al., Learning to Schedule Straight-Line Code, NIPS 1997

Cavazos & Moss, Inducing Heuristics to Decide Whether to Schedule, PLDI 2004

Agakov et al., Using Machine Learning to Focus Iterative Optimization, CGO 2006

Raychev, Vechev & Yahav, Code Completion with Statistical Language Models, PLDI 2014

Yuan et al., Droid-Sec: Deep Learning in Android Malware Detection, SIGCOMM 2014
Next Steps

Problems in systems research often generate lots of data
Great for applying machine learning
Many students are interested in machine learning esp. neural
Good opportunity to convert them into architecture students!

How will you apply machine learning to improving systems?

Questions? Discussion?