Lecture 2 of the

MLArchSys Seminar

Instructor: Thaleia Dimitra Doudali

Assistant Professor at IMDEA Software Institute

Universidad Politécnica de Madrid (UPM)

March 2023

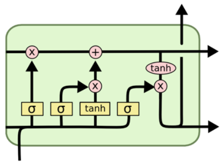
Outline of Today's Lecture

Systems Software ← ML ***for*** Systems — Machine Learning

Today's Paper:

Learning Memory Access Patterns

Milad Hashemi [1] Kevin Swersky [1] Jamie A. Smith [1] Grant Ayers [2*] Heiner Litz [3*] Jichuan Chang [1]
Christos Kozyrakis [2] Parthasarathy Ranganathan [1]



LSTMs

for Cache Prefetching

1. Prefetching Overview
2. LSTMs Overview
3. LSTMs for Prefetching
4. Evaluation
5. Lessons Learned

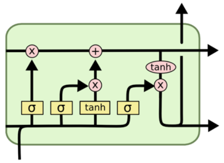
Outline of Today's Lecture

Systems Software ← ML ***for*** Systems ← Machine Learning

Today's Paper:

Learning Memory Access Patterns

Milad Hashemi [1] Kevin Swersky [1] Jamie A. Smith [1] Grant Ayers [2*] Heiner Litz [3*] Jichuan Chang [1]
Christos Kozyrakis [2] Parthasarathy Ranganathan [1]



LSTMs

for Cache Prefetching

1. Prefetching Overview

2. LSTMs Overview

3. LSTMs for Prefetching

4. Evaluation

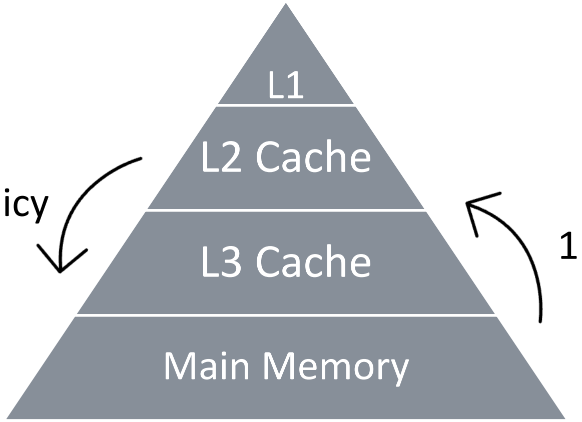
5. Lessons Learned

Cache Hierarchy

Data is allocated in memory. When accessed from memory, it gets cached.

2. Cache Eviction Policy

e.g., LRU, LFU



1. Cache Placement Policy

e.g., direct-mapped

Hardware

Memory Access Trace

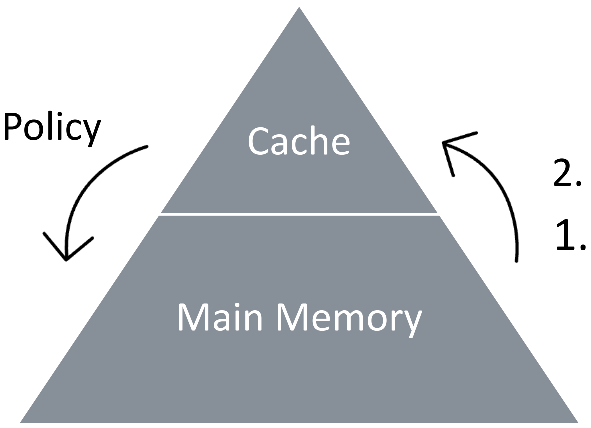
```
0x40001ee0: R 0xbfffe798
0x40001efd: W 0xbfffe7d4
0x40001f09: W 0xbfffe7d8
0x40001f20: W 0xbfffe864
0x40001f20: W 0xbfffe868
```

`0x40001f20:`	Program Counter (PC)
`W`	Access Type (Read or Write)
`0xbfffe868`	Memory Address of the data (hex number)

Data Prefetching

In addition to caching upon memory accesses, hardware prefetches data from memory into the cache, as well.

3. Cache Eviction Policy



2. Cache Placement Policy

1. Prefetch Data

Hardware

It's a prediction problem!
Predict *which* data will be accessed in the future and cache them.

What to Prefetch?

Learn various patterns.
- Sequential: A, A+1, A+2..
- Strided: A, A+4, A+8..
- Correlated: A, B, C, A, D, B, A
- More complex ones..

When to Prefetch?

- On every data access.
  - Overheads?
- On every cache miss.
  - Patterns filtered by cache.
- On prefetching hits.

Where to put the prefetched data?

- In the cache.
- In separate buffers.
  - Avoid cache pollution.

Next Line (1) + Stride Prefetchers (2)

Next Line Prefetcher

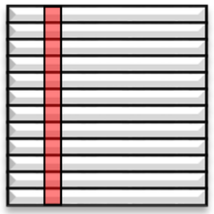
A	A+1	A+2	A+3

Data in Memory

Prefetch data one after the other A, A+1, A+2

- \+ Very easy to implement.
- − Works only on sequential patterns.

Stride Prefetcher

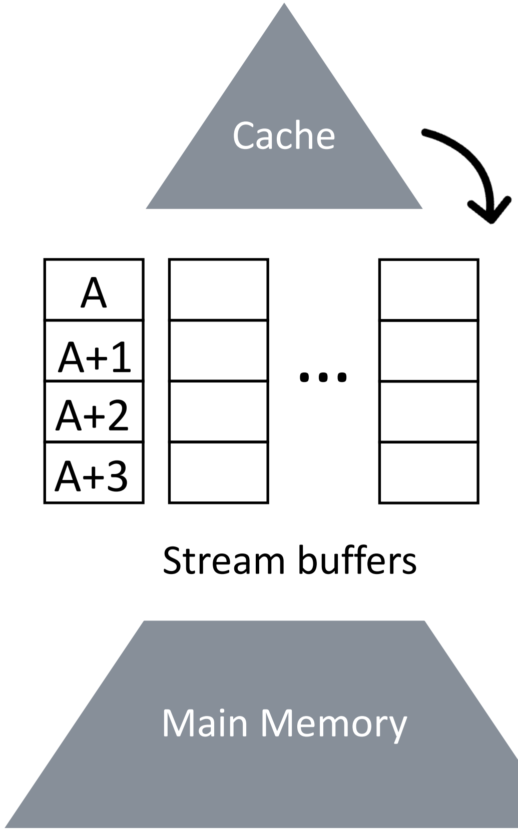


Column in matrix

Matrix allocated in memory.

- \+ Strided patterns are frequent.
- − Need a mechanism to detect length of the stride.

Stream Prefetcher (3)



1. Cache miss A

2. Check buffer “heads”.

3. **Found A**: put in cache, prefetch A+4 into buffer.

4. **Not Found A**: prefetch from memory a new stream buffer (A – A+3). LRU replacement of buffers.

+ Buffers do not “pollute” the cache.

− Extra hardware overhead (number and length of buffers).

Correlation Prefetcher (4)

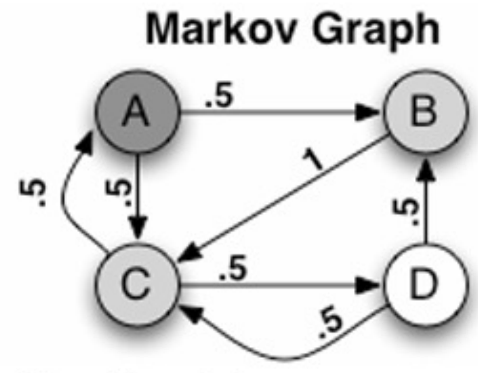
Prefetches data based on history of memory accesses.

Memory Accesses: A B C D C A C D B C A

Tag	1st time	2nd time
A	C	B
B	C	
C	A	D
D	B	C

⟵ Miss C - Prefetch A D

History Table



Edges: Probability

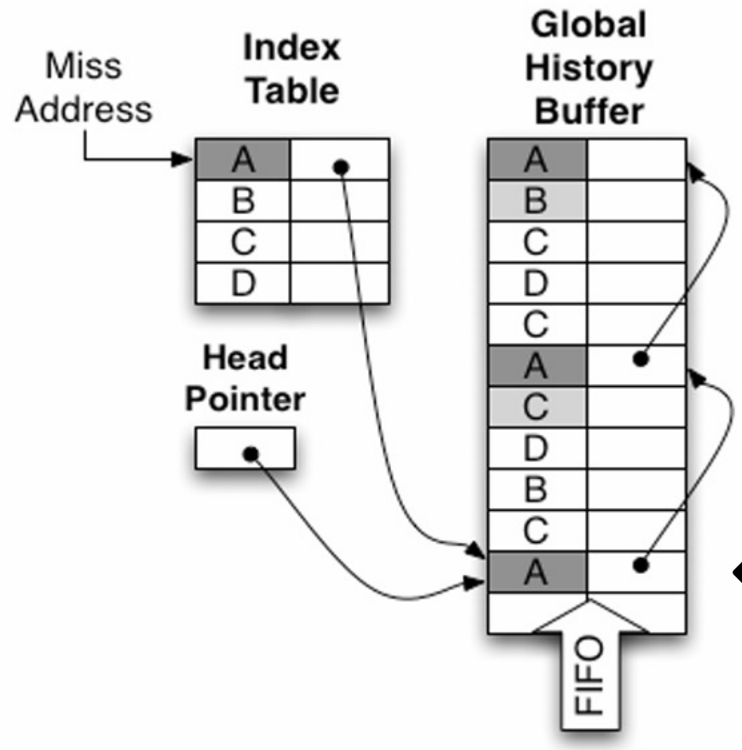
Records the address that was next to “tag” the past 2 times.

- ➕ Captures variety of patterns.
- ➖ Limited size, possible conflicts due to indexing.

Global History Buffer - GHB (5)

Decouples table indexes from the storage of prefetch-related history.

Memory Accesses: A B C D C A C D B C A



← Miss A - Prefetch C B

- Captures more complete history.
- Multiple table accesses.

Some version of GHB is mostly used in modern architectures!

Evaluation Metrics

True Positives: Cache hit due to prefetching.

		Real: Positive	Real: Negative
Prediction	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

$$\textbf{Accuracy} = \frac{\text{True Positives}}{(\text{True} + \text{False})\ (\text{Positives} + \text{Negatives})}$$

$$\textbf{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\textbf{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Timeliness = How early data is prefetched, versus when it is actually accessed, if at all.

Evaluation Example

		Cache	
		Hit = 40	Miss = 60
Prefetched	Yes = 80	True Positive	False Positive
	No = 20	False Negative	True Negative

$$\text{Accuracy} = \frac{40}{40 + 60 + 80 + 20} = 0.2$$

$$\text{Precision} = \frac{40}{40 + 60} = 0.4$$

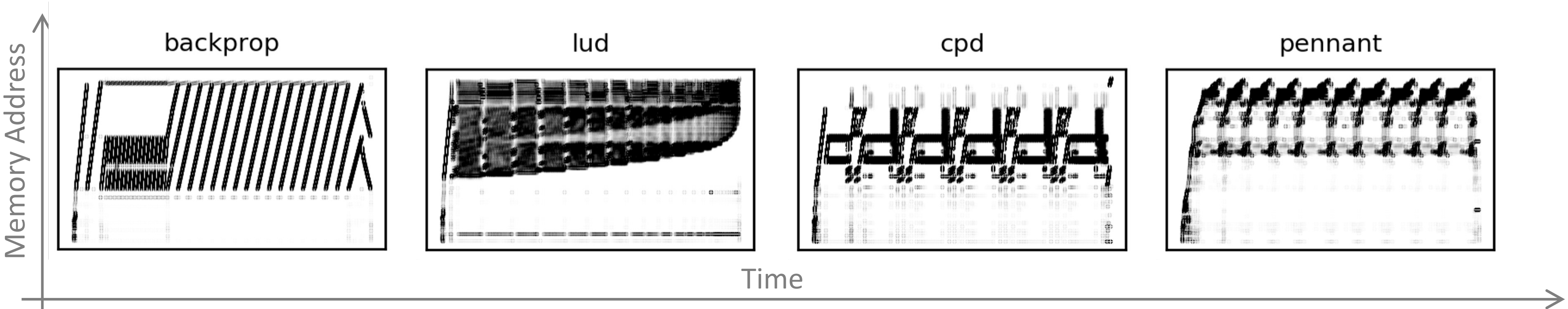
$$\text{Recall} = \frac{40}{40 + 20} = 0.67$$

Prefetching = Forecasting Time Series

Prefetching is a prediction problem = Forecasting future values of data that are ordered in time.

```
Timestamp1 0x40001ee0: R 0xbfffe798
Timestamp2 0x40001efd: W 0xbfffe7d4
Timestamp3 0x40001f09: W 0xbfffe7d8
Timestamp4 0x40001f20: W 0xbfffe864
Timestamp5 0x40001f20: W 0xbfffe868
```

= Time series of accesses to memory addresses.



Sequential strides | Triangular Traversal | Sparse Tensors | Cycles of Randomness

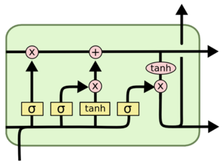
Outline of Today's Lecture

Systems Software ← ML ***for*** Systems — Machine Learning

Today's Paper:

Learning Memory Access Patterns

Milad Hashemi [1] Kevin Swersky [1] Jamie A. Smith [1] Grant Ayers [2*] Heiner Litz [3*] Jichuan Chang [1]
Christos Kozyrakis [2] Parthasarathy Ranganathan [1]



LSTMs ***for*** Cache Prefetching

1. Prefetching Overview

2. LSTMs Overview

3. LSTMs for Prefetching

4. Evaluation

5. Lessons Learned

ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of **Recurrent Neural Networks (RNNs)** used for forecasting.

In the context of this lecture's paper, LSTM is a solid box, no need to understand the internals.

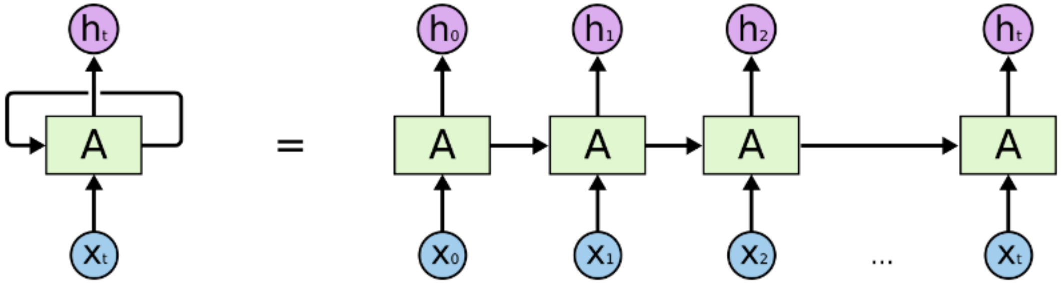
We'll focus on the inputs and outputs: what exactly it learns, what exactly it predicts.

input → LSTM-based Deep Neural Network → output

… but since you're curious let's see it's internal functionality.

ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of **Recurrent Neural Networks (RNNs)** used for forecasting.



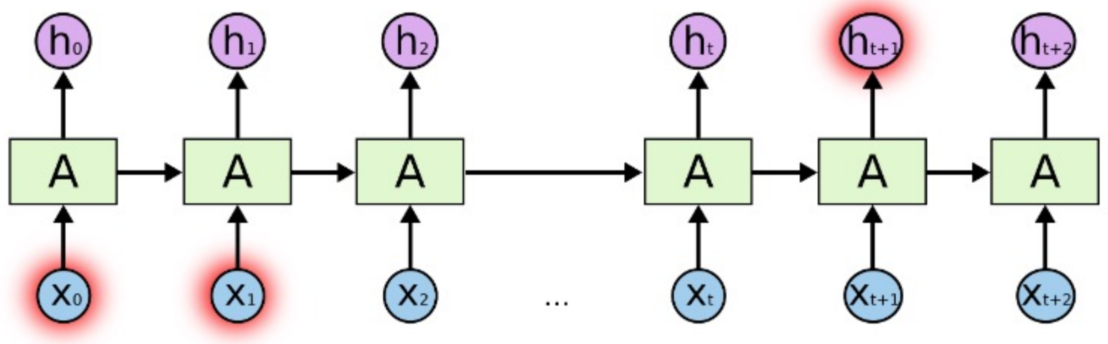
RNNs use information from many time steps $X_0, X_1 \,..\, X_t$ to make a prediction h_t

E.g., the clouds are in the .. sky.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of **Recurrent Neural Networks (RNNs)** used for forecasting.



RNNs struggle to capture long-term dependencies.

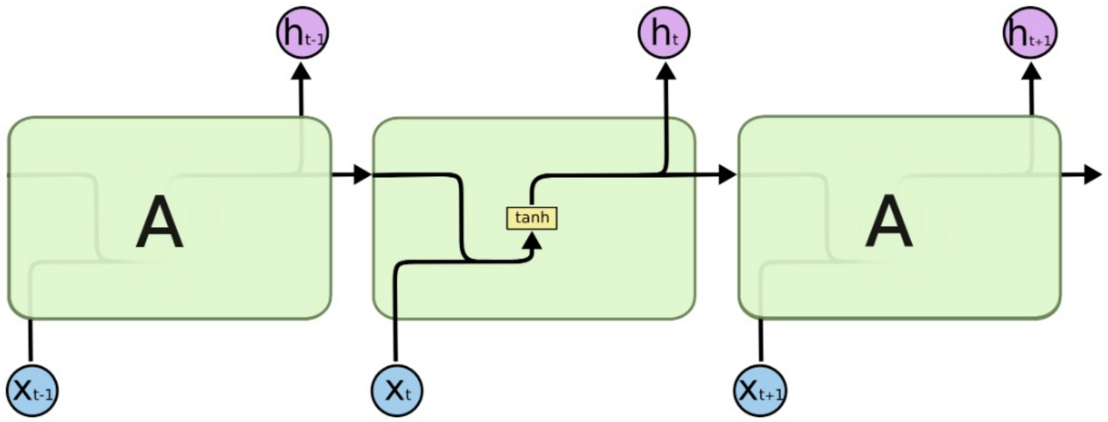
E.g., I grew up in France, I speak fluent .. French.

LSTMs to the rescue!

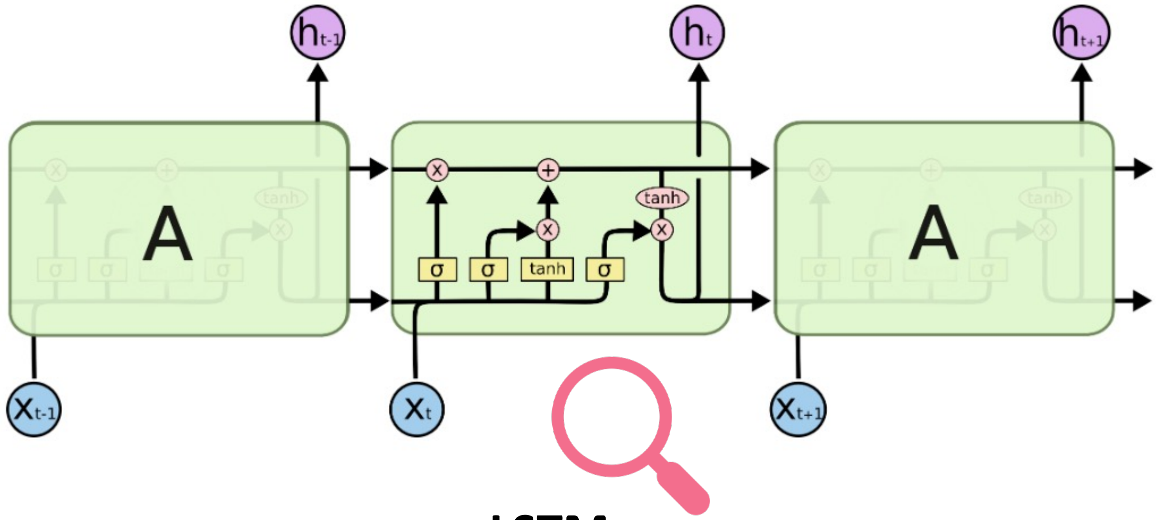
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of **Recurrent Neural Networks (RNNs)** used for forecasting.



vs



RNNs

1 internal layer

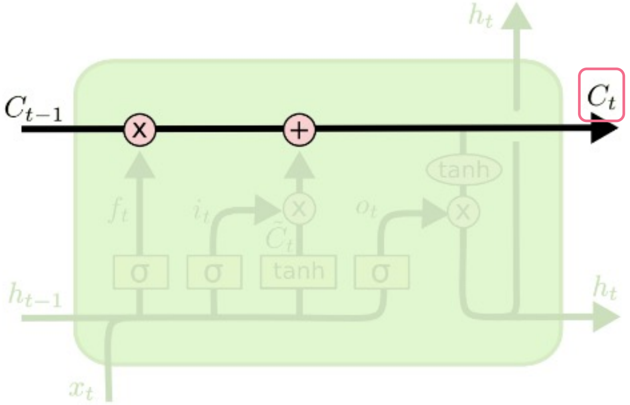
LSTMs

4 interacting internal layers

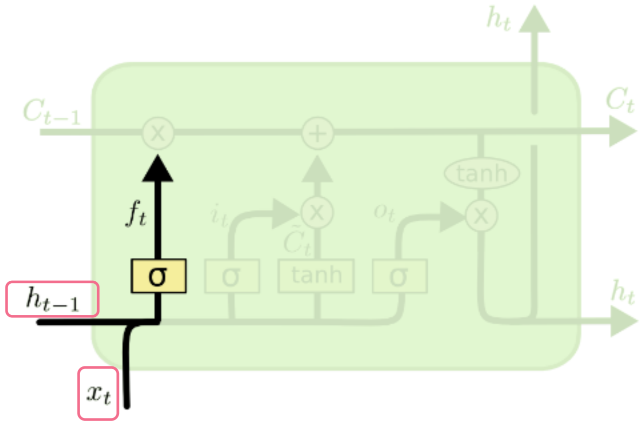
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of **Recurrent Neural Networks (RNNs)** used for forecasting.



Cell State C_t can change through pointwise operations.

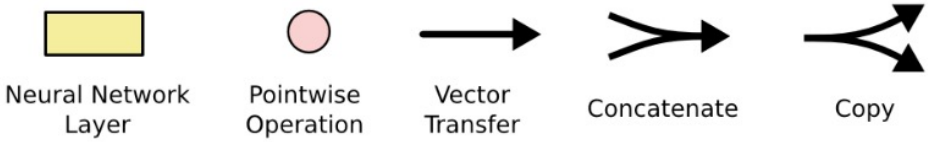


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

σ = sigmoid layer
f_t = “forget gate” layer

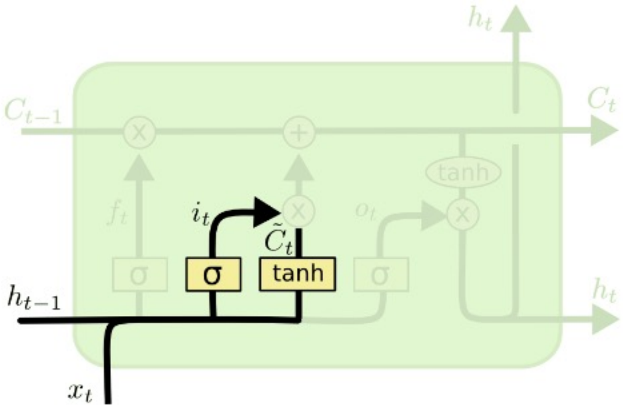
Take h_{t-1} and x_t and decide whether to keep (= 1), forget (= 0), or remember part of (< 1).

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/



ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of **Recurrent Neural Networks (RNNs)** used for forecasting.

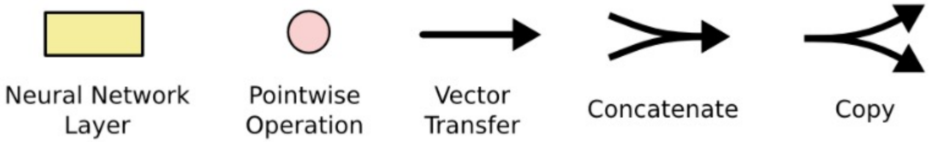


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$ = "input gate" layer = which values to update.

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$ = new candidate values to add to the cell state.

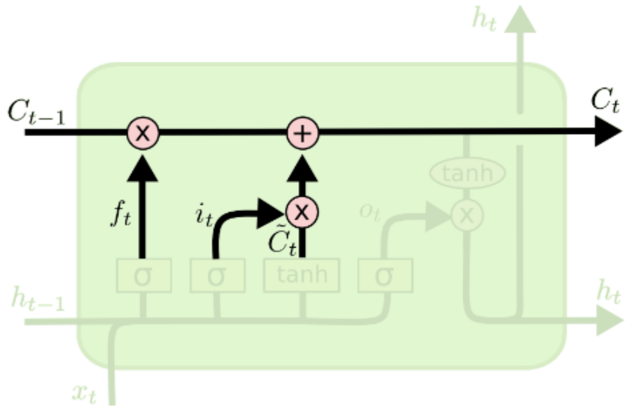
What new information are we storing in the cell state?

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/



ML for Forecasting Time Series

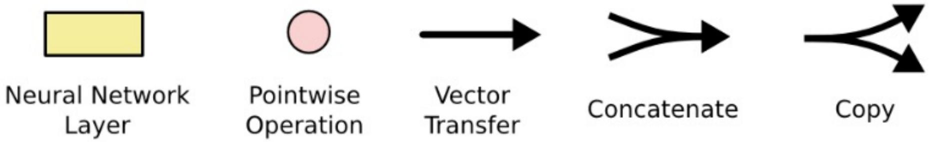
Long Short Term Memory (LSTM) networks are a type of **Recurrent Neural Networks (RNNs)** used for forecasting.



$$C_t = \underbrace{\boxed{f_t * C_{t-1}}}_{\text{forget}} + \underbrace{\boxed{i_t * \tilde{C}_t}}_{\text{add}}$$

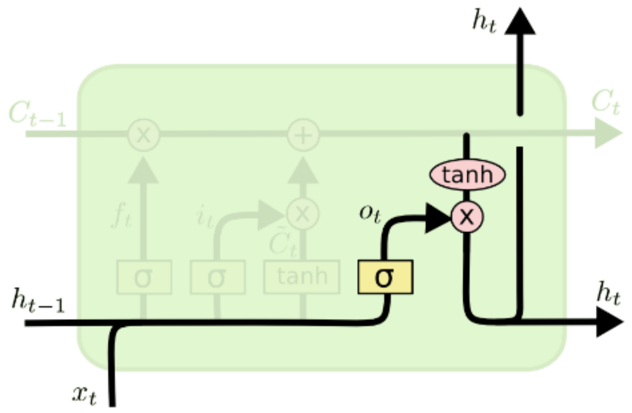
Update the old state C_{t-1} with the new one C_t.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/



ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of **Recurrent Neural Networks (RNNs)** used for forecasting.

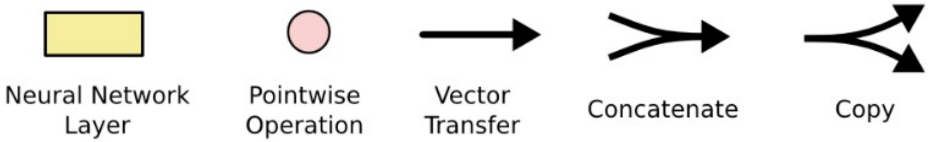


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$ = "output gate" layer.

$$h_t = o_t * \tanh\left(C_t\right)$$ = push between -1..1, to output part of the cell state.

Output h_t is a filtered version of C_t.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/



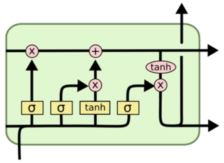
Outline of Today's Lecture

Systems Software ← ML ***for*** Systems — Machine Learning

Today's Paper:

Learning Memory Access Patterns

Milad Hashemi [1] Kevin Swersky [1] Jamie A. Smith [1] Grant Ayers [2*] Heiner Litz [3*] Jichuan Chang [1]
Christos Kozyrakis [2] Parthasarathy Ranganathan [1]



LSTMs ***for*** Cache Prefetching

1. Prefetching Overview

2. LSTMs Overview

3. LSTMs for Prefetching

4. Evaluation

5. Lessons Learned

Learning Memory Access Patterns

The data available for ML training is a *memory access trace*:

```
0x40001ee0: R 0xbfffe798
0x40001efd: W 0xbfffe7d4
0x40001f09: W 0xbfffe7d8
0x40001f20: W 0xbfffe864
0x40001f20: W 0xbfffe868
```

PC address

`0xbfffe868`

64-bit binary number

Possible values = 2^{64}

Normalizing that to [0, 1] leads to information loss.

input

`(0x40001f20, 0xbfffe864)`

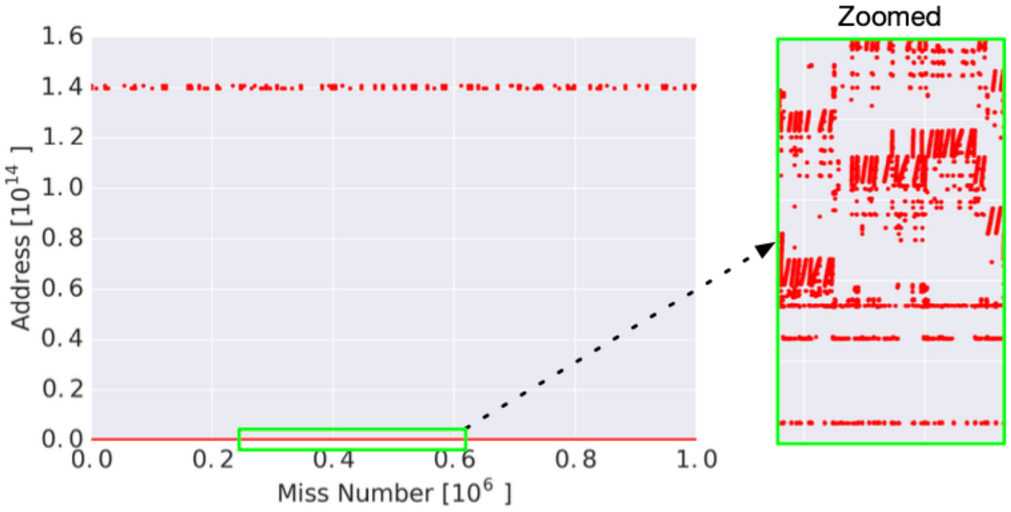
(PC, Adrress) at time t_N

→ LSTM-based Deep Neural Network →

output

`0xbfffe868`

Address at time t_{N+1}



Size of Trace: O(100M)

Huge.. and extremely sparce. Only O(10M) unique addresses.

Don't learn address numbers!

Prefetching as Classification

Memory footprint is sparse means that a relatively **small**, and **consistent** set of addresses is used.

Learn address deltas, not raw addresses!

The number of uniquely occurring deltas is *often* orders of magnitude **smaller** than uniquely occurring addresses.

input

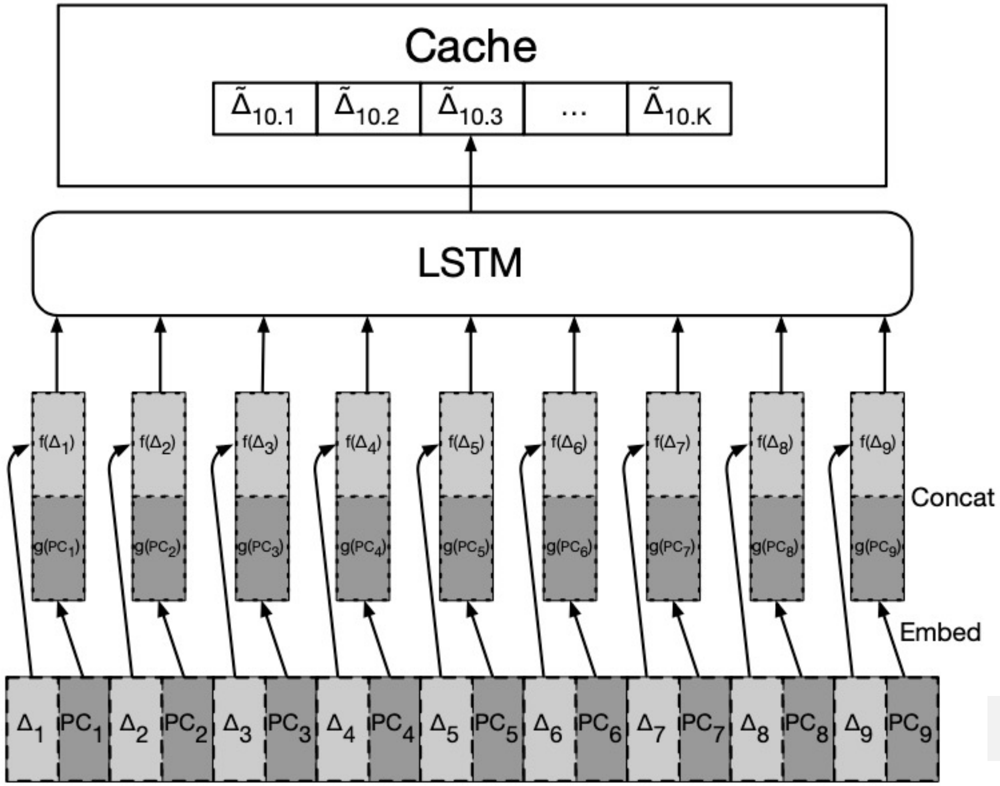
PC t_N, Address Delta(t_N, t_{N-1}) → LSTM-based Deep Neural Network → Address Delta(t_{N+1}, t_N)

output

1. Go through the memory access trace.
2. Compute address deltas for every (t_N, t_{N-1}).
3. Keep the deltas that appear at least 10 times.
4. Create a "vocabulary" of these unique deltas.

Prefetching as Classification = Prediction will be one of these deltas.

Approach 1: Embedding LSTM



3. Output

0	0	0	1	0	0	0

"one hot encoding"

Vector length = X
X = number of unique deltas
1 such vector per delta.

2. Concatenated embeddings

1. Input

```
PC tN, Address Delta(tN, tN-1)
```

($PC\ t_N,\ Address\ Delta(t_N,\ t_{N-1})$)

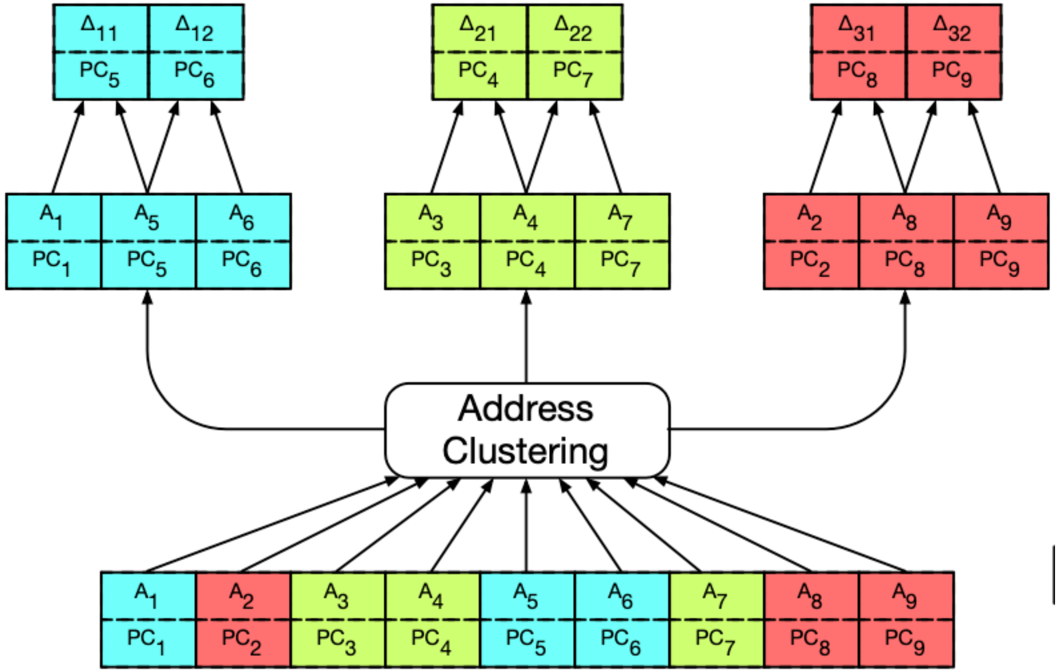
With Classification, the LSTM predicts probability for each of the X vectors.

4. Prefetching Action

Prefetch the top-10 predictions, at each timestep t_N.

Approach 2: Clustering + LSTM (1)

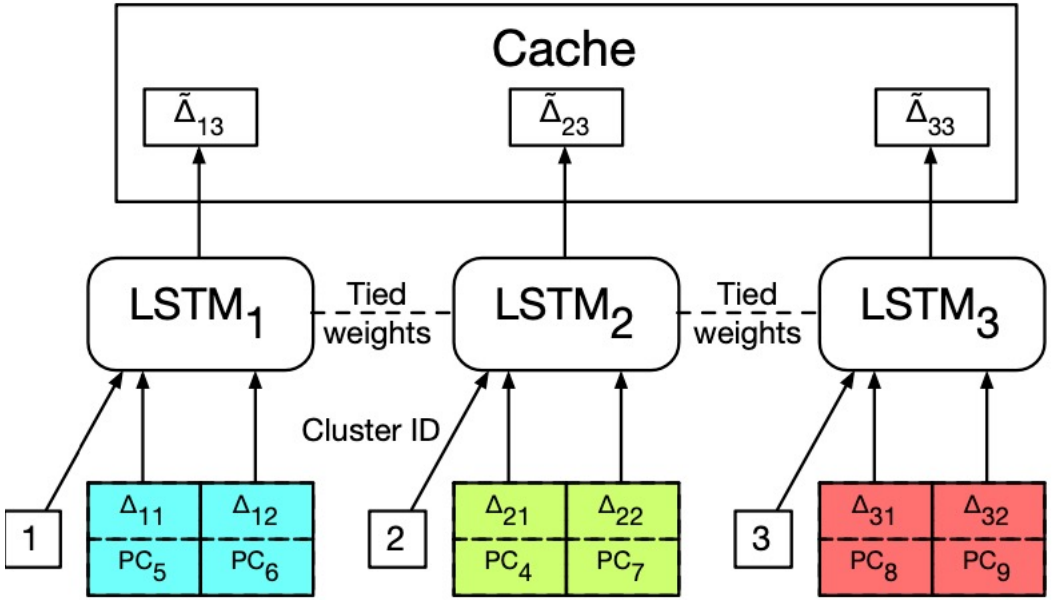
Focus on *local* context, e.g., data structures are stored in contiguous memory address and accessed repeatedly.



1. Run k-means to cluster the addresses.
2. Deltas are computed within each cluster.

+ Smaller "vocabulary" of unique deltas.
− Potentially missing the "global" context.

Approach 2: Clustering + LSTM (2)



1. Train an LSTM per cluster of deltas.
2. Add cluster ID as an extra feature.
3. Tie weights.

+ Reduced model size, faster training.

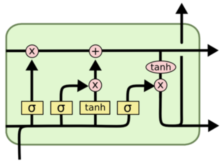
− 1 extra pre-processing step for clustering.

Outline of Today's Lecture

Today's Paper:

Learning Memory Access Patterns

Milad Hashemi [1] Kevin Swersky [1] Jamie A. Smith [1] Grant Ayers [2*] Heiner Litz [3*] Jichuan Chang [1]
Christos Kozyrakis [2] Parthasarathy Ranganathan [1]



LSTMs

for Cache Prefetching

1. Prefetching Overview

2. LSTMs Overview

3. LSTMs for Prefetching

4. Evaluation

5. Lessons Learned

Evaluation Metrics (1)

		Real: Positive	Real: Negative
Prediction	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

$$\textbf{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

$$\textbf{Precision-at-10} = \frac{\text{\# Correct Delta Predictions}}{\text{\# All Real Deltas}}$$

Correct Prediction, if the *real* delta is one of the 10 predictions.

Evaluation Metrics (2)

		Real	
		Positive	Negative
Prediction	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

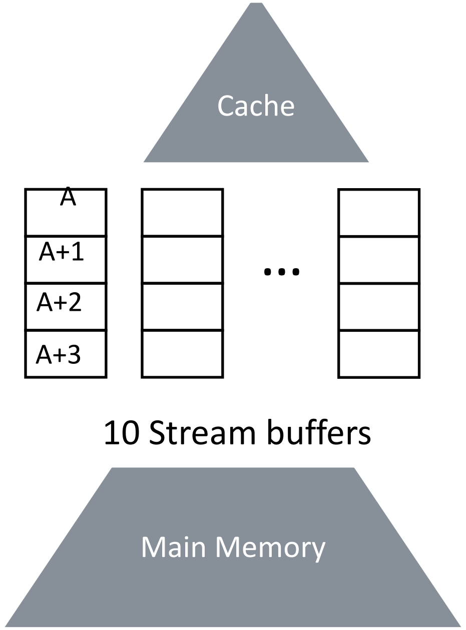
$$\textbf{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

$$\textbf{Recall-at-10} = \frac{\text{\# Unique Predicted Deltas}}{\text{\# All Predicted Deltas}}$$

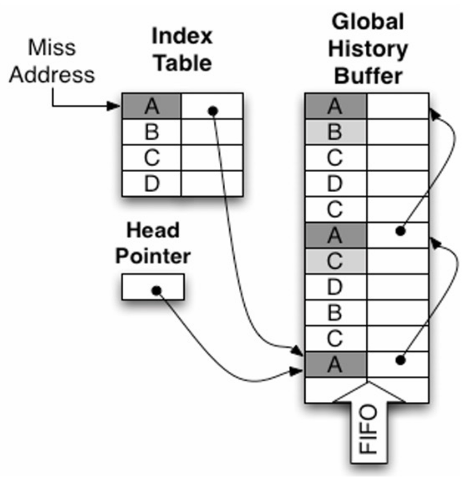
Records all 10 predicted deltas.
Quantifies the % of the “vocabulary” that could be predicted.

Evaluation Baselines

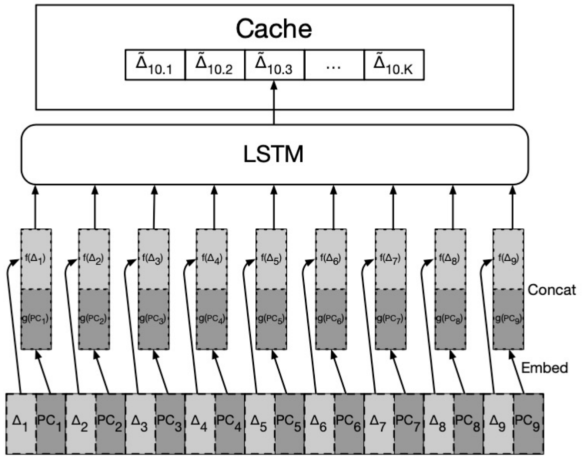
Stream   GHB   Embedding   Clustering



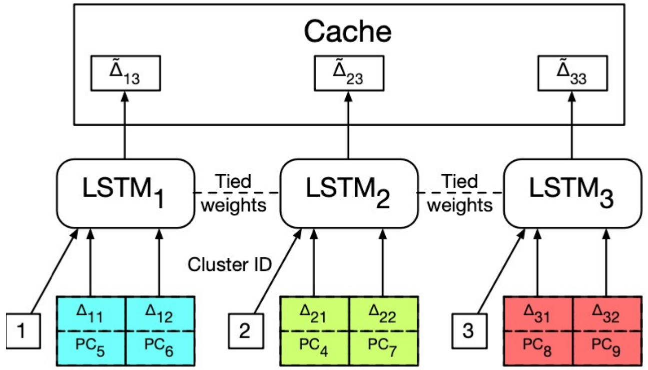
Stream Prefetcher



Global History Buffer (GHB)



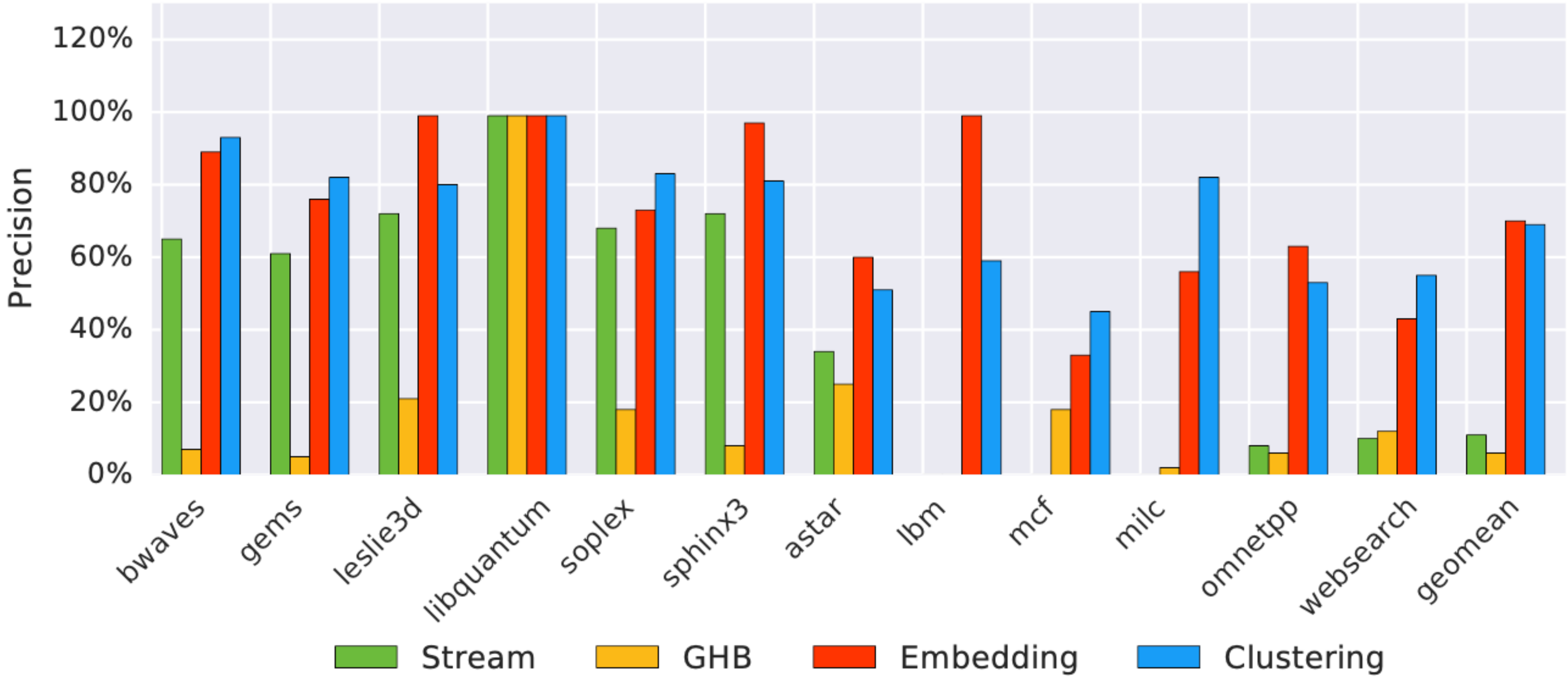
Embedding LSTM



Clustering LSTM

Evaluation (1)

The higher
the better

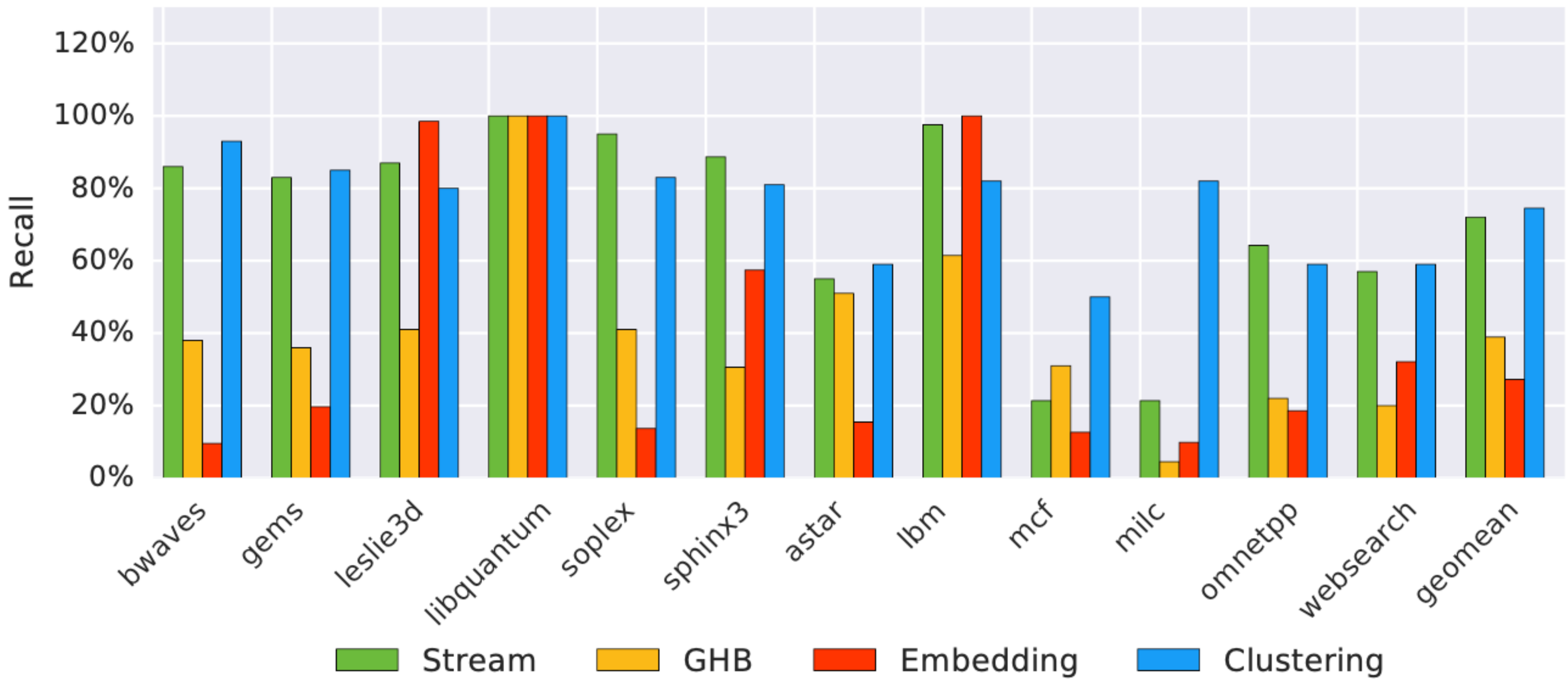


Remember..
They assume precision-at-10.

LSTM models achieve high precision, especially for complex patterns (e.g., websearch).
No great difference between the embedding and clustering LSTM.

Evaluation

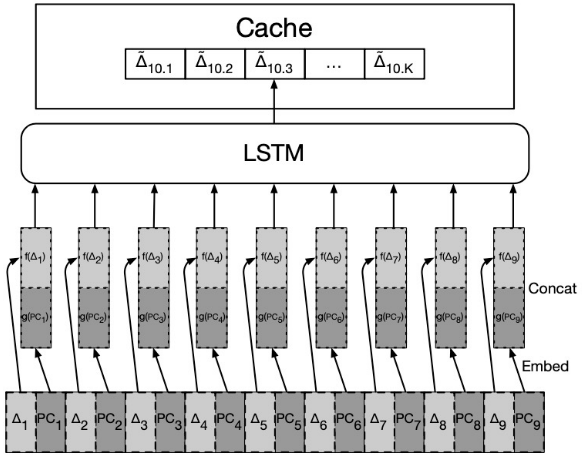
The higher
the better



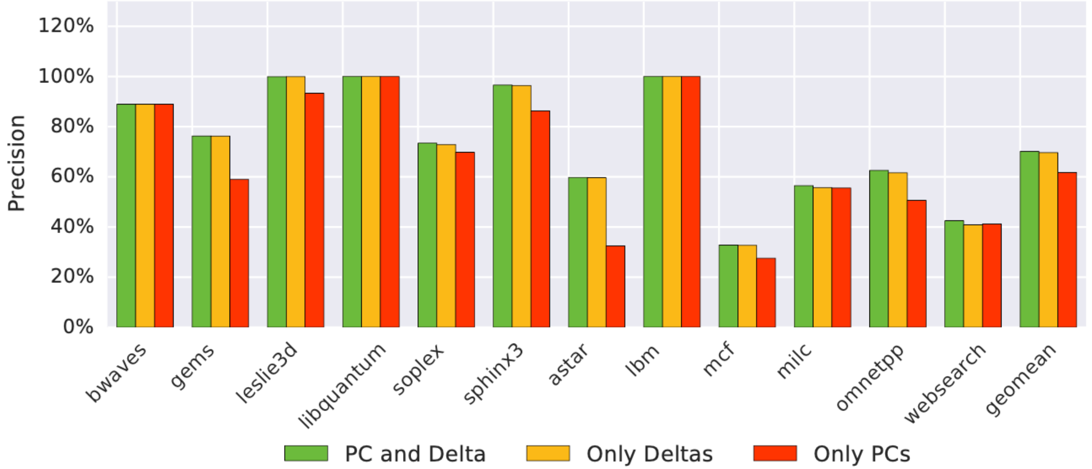
Stream prefetcher achieves highest recall, due to its dynamic vocabulary (set of deltas).
Clustering LSTM better than embedding, because creates better vocabulary (set of deltas).

Sensitivity to Feature Selection

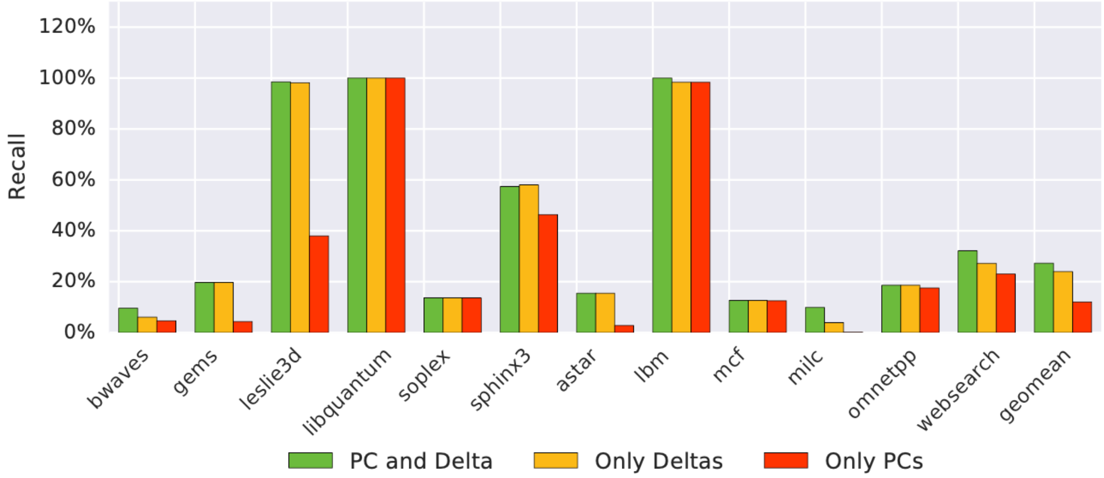
What happens when using
only PC or Deltas as input features.



Embedding LSTM



For precision, *only deltas* contributes the most.



For recall,
PC helps improve it.

Outline of Today's Lecture

Systems Software

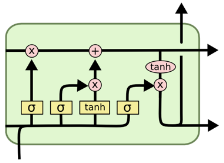
ML *for* Systems

Machine Learning

Today's Paper:

Learning Memory Access Patterns

Milad Hashemi [1] Kevin Swersky [1] Jamie A. Smith [1] Grant Ayers [2*] Heiner Litz [3*] Jichuan Chang [1]
Christos Kozyrakis [2] Parthasarathy Ranganathan [1]



LSTMs

for Cache Prefetching

1. Prefetching Overview

2. LSTMs Overview

3. LSTMs for Prefetching

4. Evaluation

5. Lessons Learned

Lessons Learned (1)

What to remember when using LSTMs for prefetching.

Don’t Learn the Address, learn Address Deltas instead.

```
0x40001ee0: R 0xbfffe798
0x40001efd: W 0xbfffe7d4
0x40001f09: W 0xbfffe7d8
0x40001f20: W 0xbfffe864
0x40001f20: W 0xbfffe868
```

Memory Access Trace

Size of Trace: O(100M)

− Huge.. and extremely sparce. O(10M) unique addresses.

$\underbrace{\texttt{0xbfffe868}}_{\text{64-bit binary number}}$

Possible values = 2^{64}

− Normalizing that to [0, 1] leads to information loss.

Record the most frequently seen

`Address Delta(`t_{N+1}`, `t_N`)`

Convert each unique delta to:

0	0	0	1	0	0	0

“one hot encoding”

+ “Small” set of deltas.
+ Classification: predict specific values.

Lessons Learned (2)

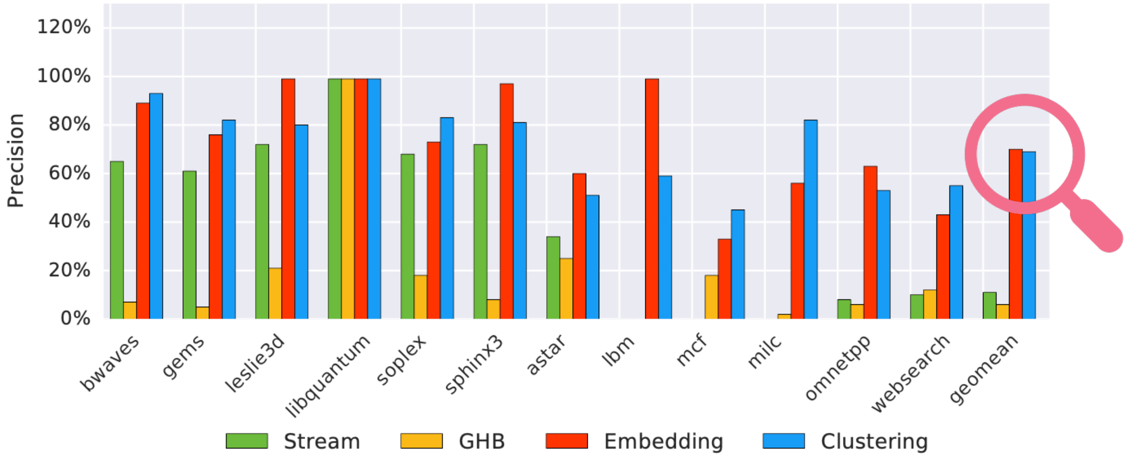
What to remember when using LSTMs for prefetching.

Prefetching allows for multiple predictions, thus higher *perceived* model accuracy.

Prefetch the top-10 predictions, at each timestep t_N.

$$\textbf{Precision-at-10} = \frac{\text{\# Correct Delta Predictions}}{\text{\# All Real Deltas}}$$

Correct Prediction, if the *real* delta is one of the 10 predictions.



LSTM models achieve much higher precision-at-10, not precision.

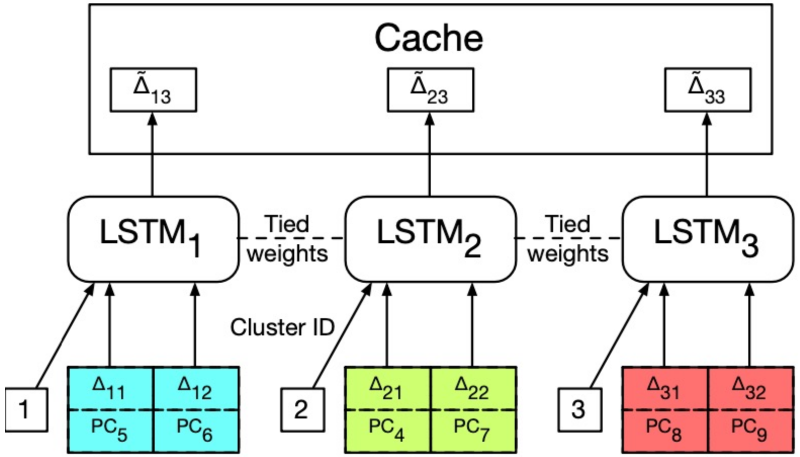
… and probably that's why observe similar performance between the Embedding and Clustering LSTMs.

Lessons Learned (3)

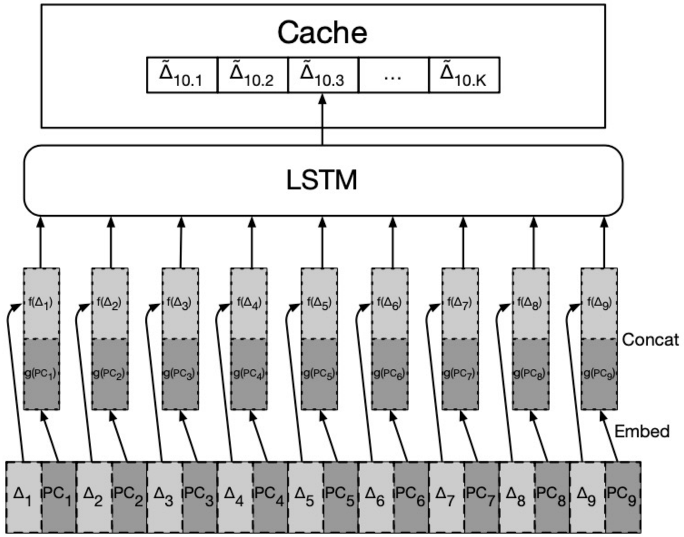
What to remember when using LSTMs for prefetching.

The Clustering LSTM delivers higher recall, but similar precision to the Embedding LSTM.

The Embedding of (PC, Delta) deliver high precision due to the Deltas and high recall due to the PCs.



Clustering LSTM



Embedding LSTM

Report Due March 28 at 18.00

Answer / expand upon these 4 questions:

1. What problem is the paper addressing and why is it important?
2. How do they approach to solve the problem?
3. What are the main evaluation results?
4. What are 2 things you will remember from this paper?

Contact

- Via email: `thaleia.doudali@imdea.org`

Website

https://thaleia-dimitradoudali.github.io/