Deeply Integrated Deep Learning

A Work-in-Progress Report

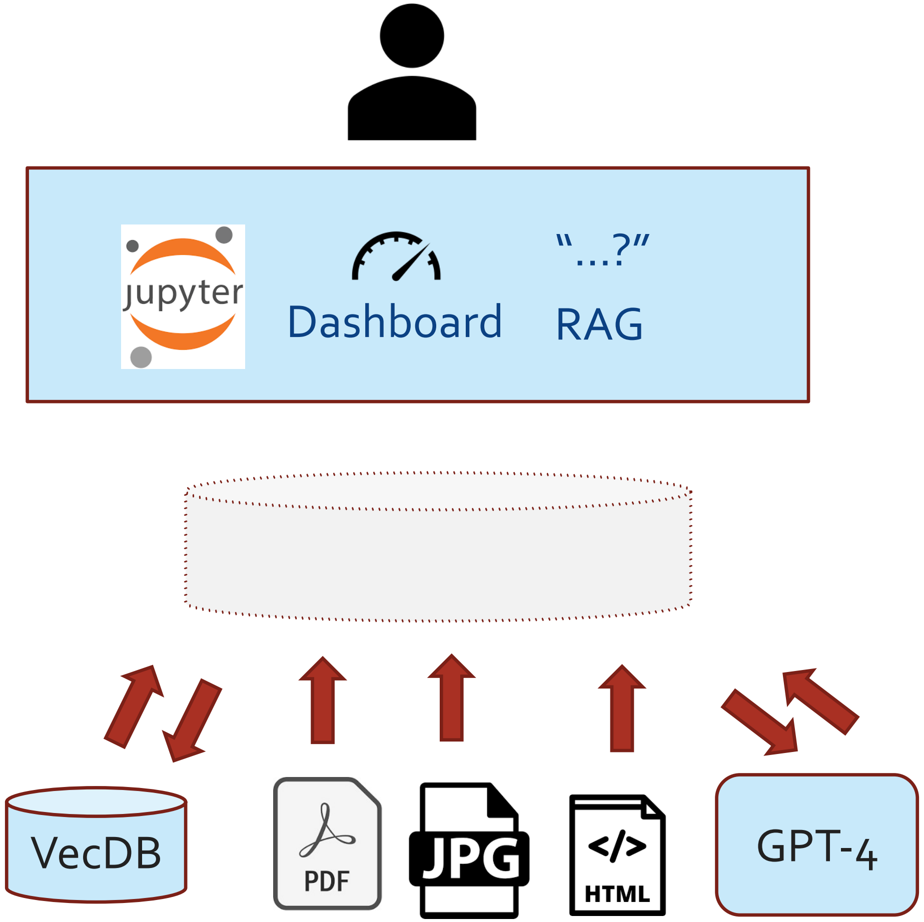
Zachary G. Ives

with Peter Baile Chen (MIT), Yi Zhang (AWS), Phillip Hilliard, Rajeev Alur, Erik Waingarten

Simons Workshop on Logic and Algebra for Query Evaluation

November 15, 2023

A New Kind of "Database"?



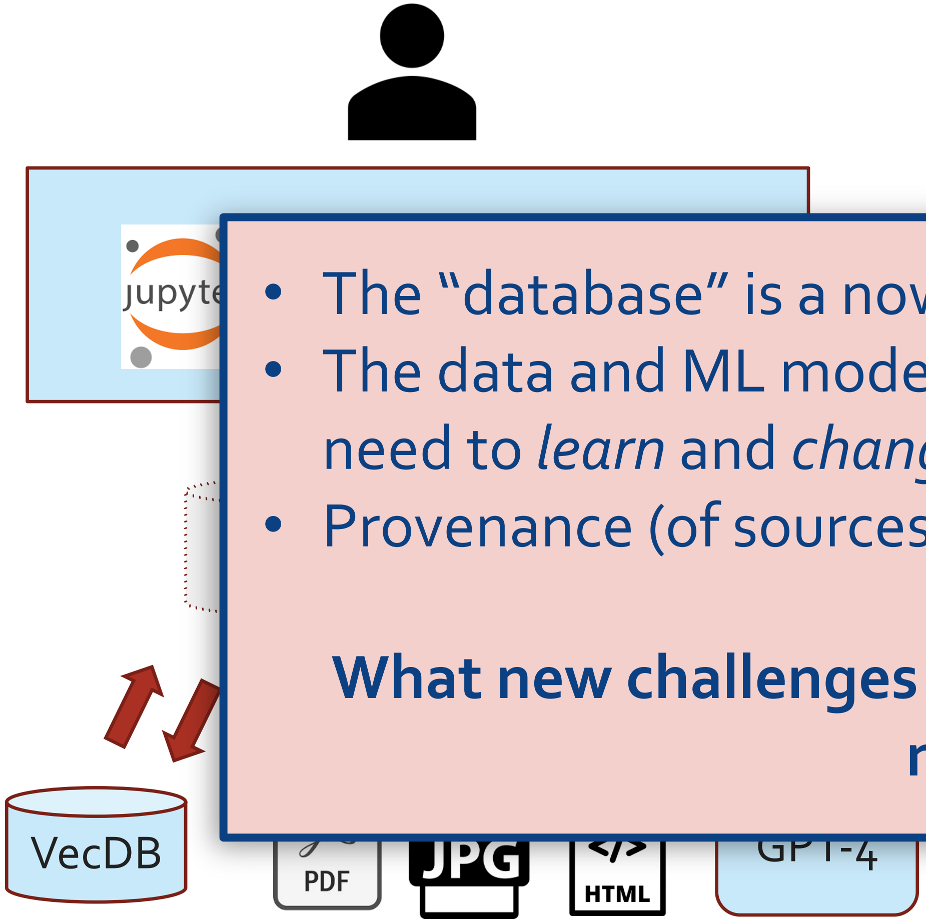
Transformers (and LLMs) enable use of **unstructured**, **multimodal**, **open-world** data

- Entity and relationship extraction into relations, triples, etc.
- General-domain entity resolution / record linking is possible – even over text, images, etc. through *embeddings*

Opportunities in enabling *discovery*, *monitoring and forecasting*, and *open Q&A*

- Cross-enterprise or cross-field data management
- Google Scholar and equivalent
- Exploratory data analysis, model building
- Real-time surveillance *with* privacy

A New Kind of "Database"?



Transformers (and LLMs) enable use of **unstructured**, **multimodal**, **open-world** data

- Entity and relationship extraction into
- on / record
- ext, images,
- *ry,*
- *pen Q&A*
- data
- Google Scholar and equivalent
- Exploratory data analysis, model building
- Real-time surveillance *with* privacy

- The "database" is a now a *view* (with ML components)
- The data and ML models are imperfect – thus, we need to *learn* and *change*!
- Provenance (of sources, extractors, evidence) matters

What new challenges and opportunities lie in this model?

Some Example (Prediction) Use Cases

Information extraction – document to triples

PDF → extract_triples → $\sigma_{\text{conf} > \tau}$

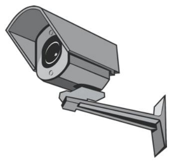
Event detection – with multiple models + provenance

detect_mention

detect_mention

→ $\cup$ → $\sigma_{\text{conf} > \tau}$

Surveillance with privacy – fuzzify people granted access



→ detect_faces → face2embed → $\bowtie_{\text{argmin (L1(embed, profile))}}$ → χ_{img} → fuzzify

Allowed(id, profile)

A Baseline Algebra with Tensor Ops (Inspired by Pandas/MLlib)

Nested relational model: attributes are scalars, lists, or *tensors* (vectors, matrices)

Tuples p_i: <A, B, C> where p_i is a provenance semiring expression

Attribute ←→ tensor:

- **fields2vector**$_{\text{fields,vec\_name}}$
- **vector2fields**$_{\text{vector,field\_names}}$
- **onehot**$_{\text{fields,field\_name\_prefix}}$

Aggregate:

- **nested_list**$_{\text{fields,field\_name}}$
- **stack_tensors**$_{\text{tensor\_field}}$
- **pivot**$_{\text{key\_field, value\_field}}$

Unnest:

- **explode**$_{\text{field,index\_field}}$
- **unstack**$_{\text{tensor\_field,index\_field}}$
- **melt**$_{\text{fields,key\_field\_name,val\_field\_name}}$

Do We Get Benefits from Looking *Inside* the NN UDFs?

- Structured queries are based on **constraints** and **equivalences** over tuples, nested relations, etc.
- ML stages are largely based on tensors, dot products, activation functions

If we go beyond "black box" ML ops, can we combine the two to benefit?

- Neurosymbolic systems like Scallop [Naik], DeepProbLog [Manhaeve+] show more effective **learning** by integrating logical constraints
- We think logical constraints can open new opportunities on the **prediction** side, for studying ***efficiency, updates and maintenance*** **and more**!

A Baseline Algebra with Tensor Ops (Inspired by Pandas/MLlib)

Nested relational model: attributes are scalars, lists, or *tensors* (vectors, matrices)

Tuples p_i: <A, B, C> where p_i is a provenance semiring expression

Attribute ←→ tensor:

fields2vector$_{\text{fields,vec\_name}}$

vector2fields$_{\text{vector,field\_names}}$

onehot$_{\text{fields,field\_name\_prefix}}$

Aggregate:

nested_list$_{\text{fields,field\_name}}$

stack_tensors$_{\text{tensor\_field}}$

pivot$_{\text{key\_field, value\_field}}$

Unnest:

explode$_{\text{field,index\_field}}$

unstack$_{\text{tensor\_field,index\_field}}$

melt$_{\text{fields,key\_field\_name,val\_field\_name}}$

Neural network:

Connectivity/dot products: **linear, conv1D, conv2D, conv3D**

Activation: **relu, sigmoid, tanh**

batchnorm

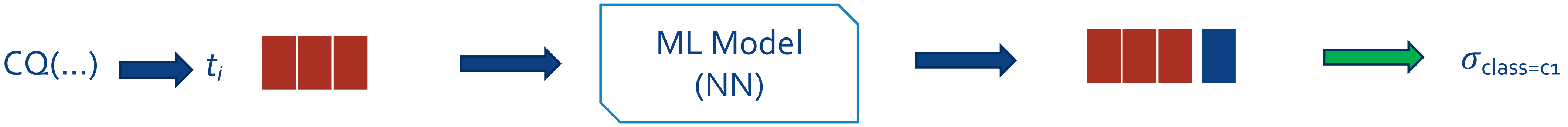
maxpool

softmax

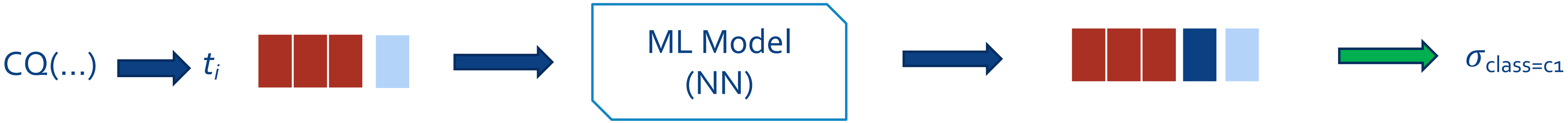
…

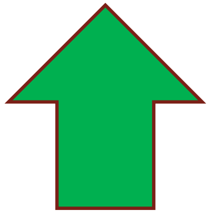
A “Detection” Query in More Detail

Prediction is based on **features** within CQ tuples followed by **selection**…



But *provenance* provides important features too*!

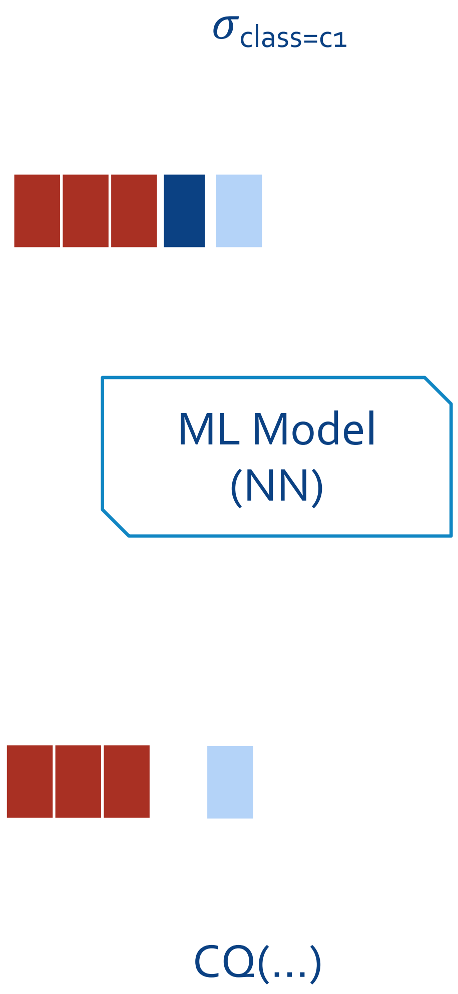




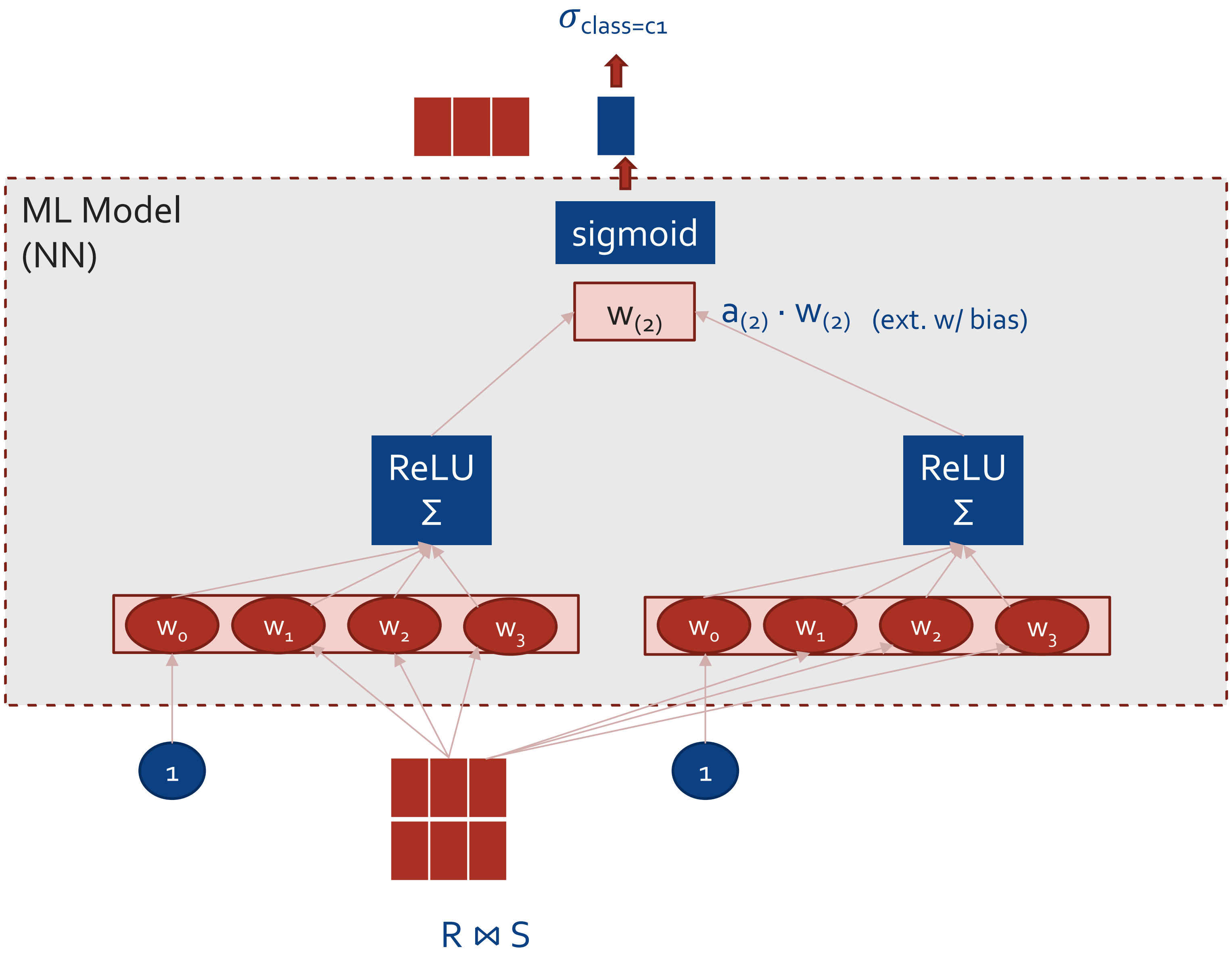
Often:

Select for class = c1
Or score(c1) > $\boldsymbol{\tau}$

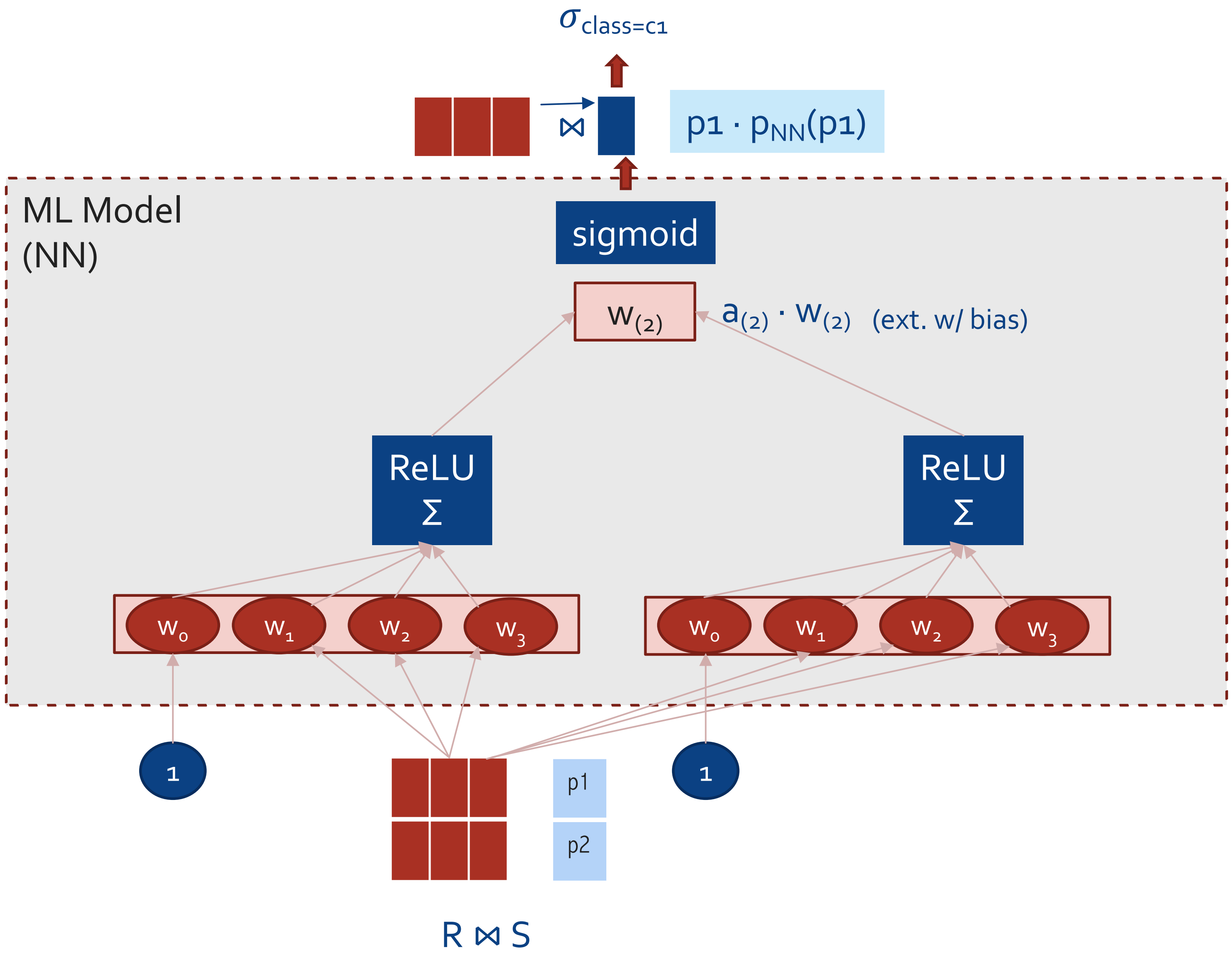
A "Detection" Query in More Detail



A “Detection” Query in More Detail



A Neural Network in a Query … As a Dependent Join



Zheng+21: general extensions to provenance semiring model for multimodal data extraction, substring matching, etc.

Treat NN() as a dependent join *with input bindings!*

Learning and Provenance: View Update Redux → View Maintenance

Suppose a view output is wrong! $\hat{y} \rightarrow y$

Compute loss$(y, \hat{y})$ and *back-propagate* through NN, scaling weights based on gradients

Tuples annotated with *gradient semirings* (Naik: Scallop) [Eisner, 2002] [Kimmig+ 2011] capture gradients of NN vs each parameter!

$p1 \cdot p_{NN}(p1)$

Annotation is pair (score, gradient_vec)

$$0 = (0, \vec{0}) \; and \; 1 = (1, \vec{0})$$

$$(p_1, \overrightarrow{\nabla p_1}) + (p_2, \overrightarrow{\nabla p_2}) = (p_1 + p_2, \overrightarrow{\nabla p_1} + \overrightarrow{\nabla p_2})$$

$$(p_1, \overrightarrow{\nabla p_1}) \cdot (p_2, \overrightarrow{\nabla p_2}) = (p_1 p_2, p_1 \overrightarrow{\nabla p_2} + p_2 \overrightarrow{\nabla p_1})$$

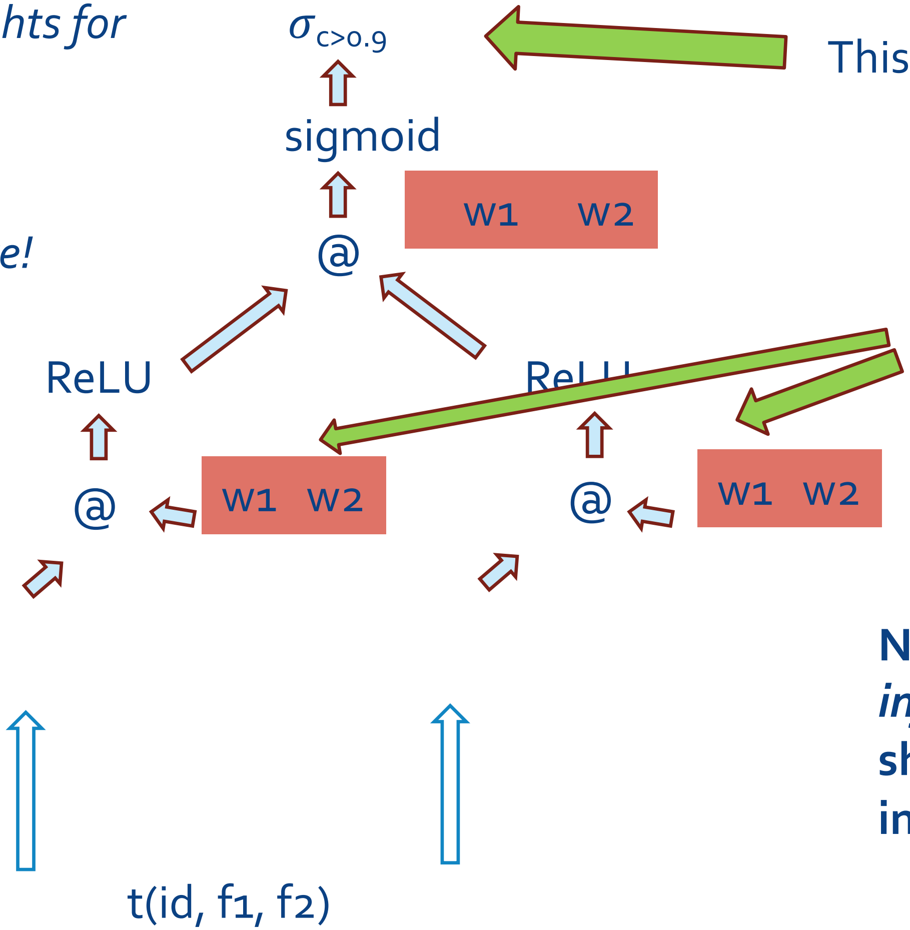
Let's consider two ways of making view recomputation fast!

1. Sideways information passing through NN layers
2. Exploiting functional equivalences with preferences

Making a Prediction: NN "Feed-Forward"

Omitting bias weights for simplicity!

And assuming all weights are positive!



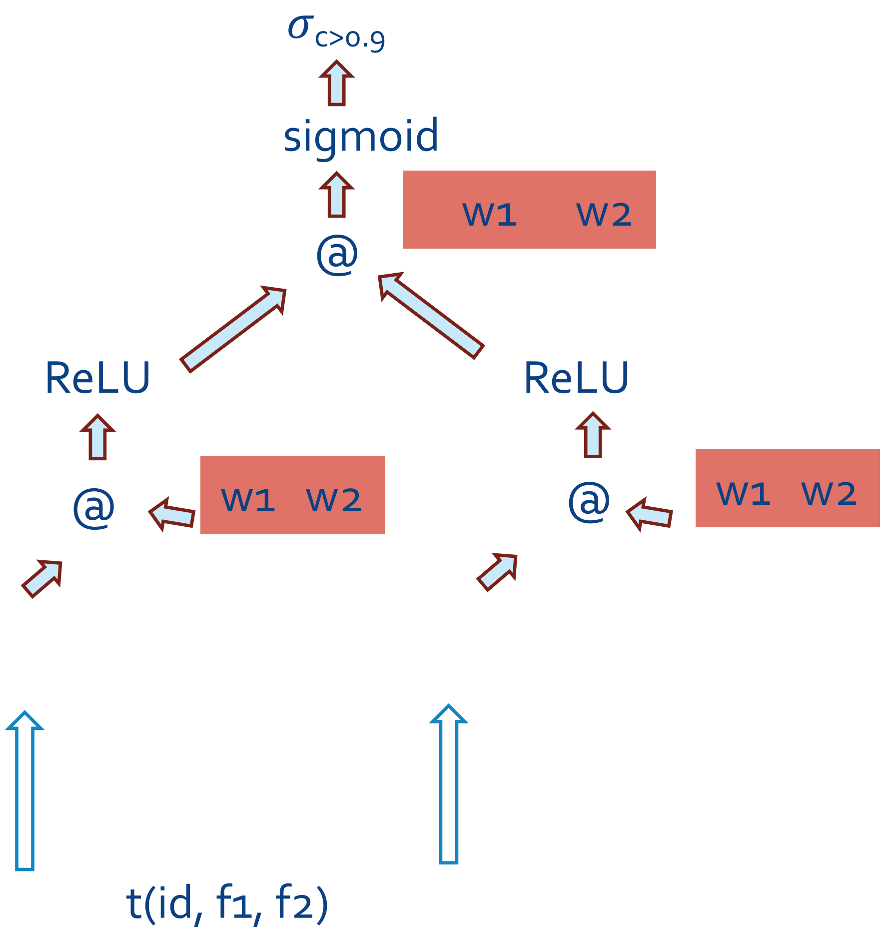
This prunes most of its inputs!

Could we skip processing most of these inputs, if their features are low?

Need a type of *sideways information passing*, where we share constraints on scores of inputs not yet seen.

Observations about Feed-Forward

Inspiration: Fagin's Algorithm (FA) [Fagin et al. 01] for thresholding



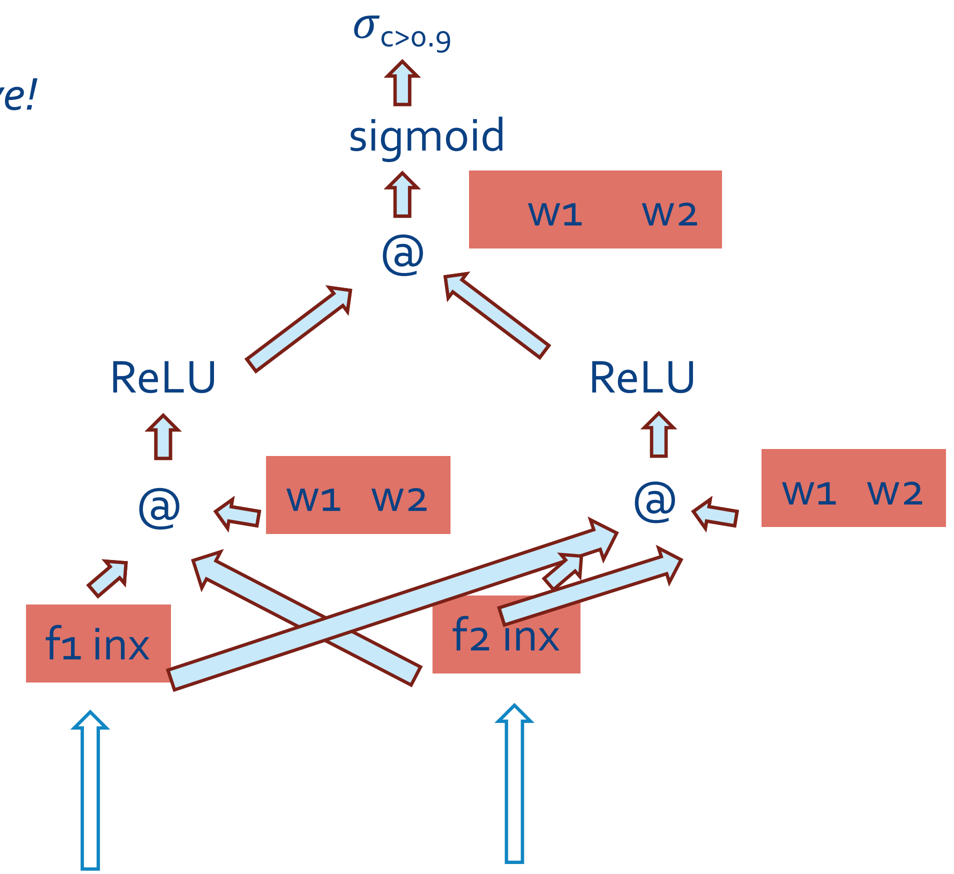
Abstracting from details: FA is a type of *sideways information passing*, where we share bounds over scores of inputs not yet seen.

If we have multiple computation stages we **prioritize** as appropriate.

Feed-Forward with SIPS

Inspiration: Fagin's Algorithm (FA) [Fagin et al. 01] for thresholding

Assuming all weights are positive!



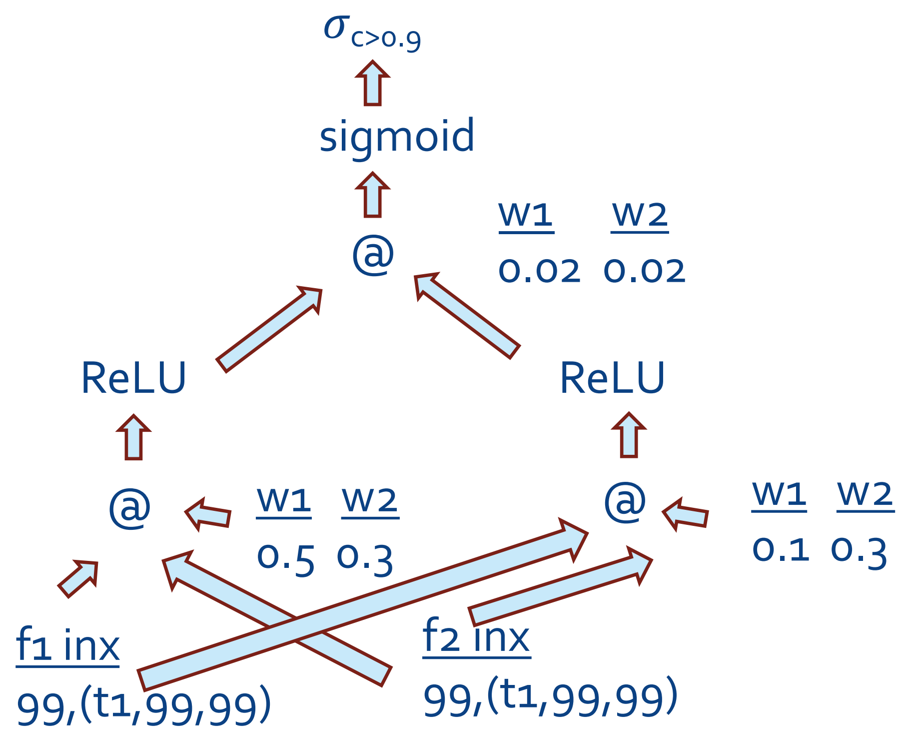
t(id, f1, f2) with unclustered **index by f1 and f2, in descending order**

Abstracting from details: FA is a type of *sideways information passing*, where we share **constraints on scores of inputs not yet seen.**

Can we bound our exploration across the activation units?

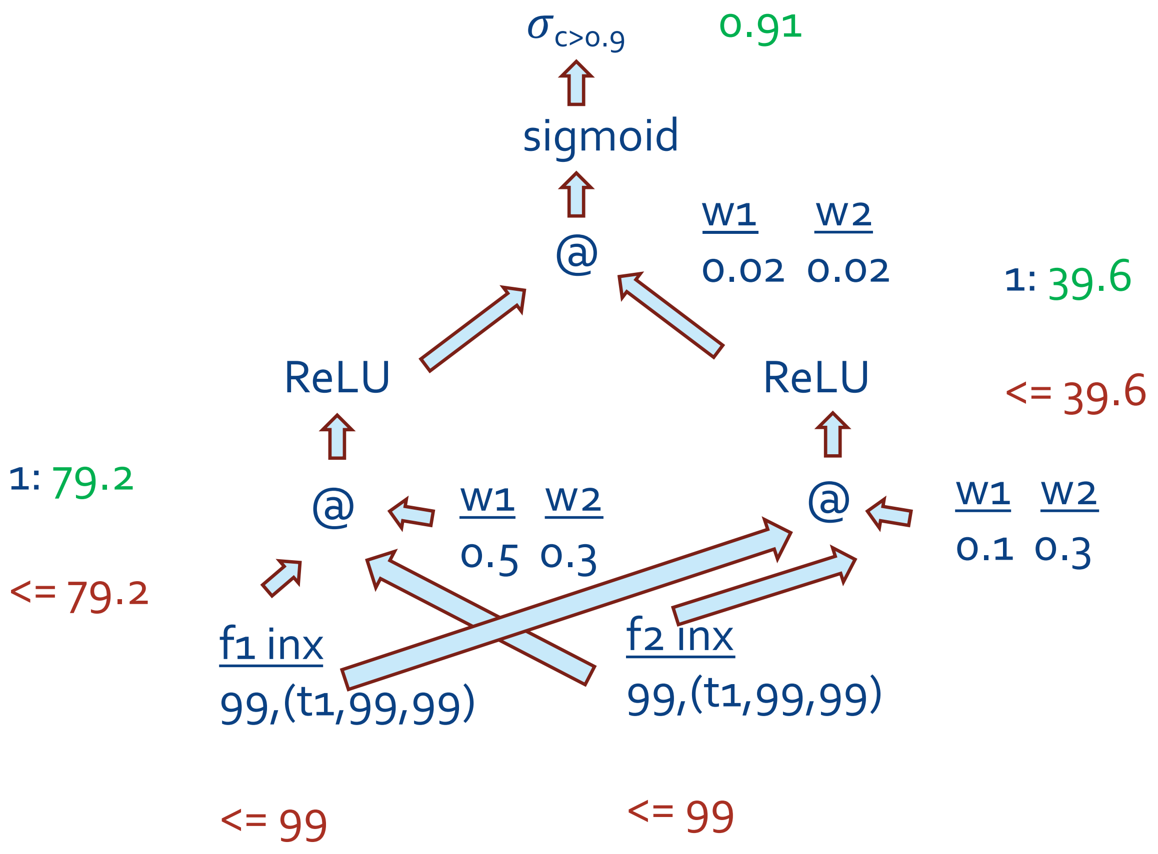
Feed-Forward with SIPS

Inspiration: Fagin's Algorithm (FA) [Fagin et al. 01] for thresholding



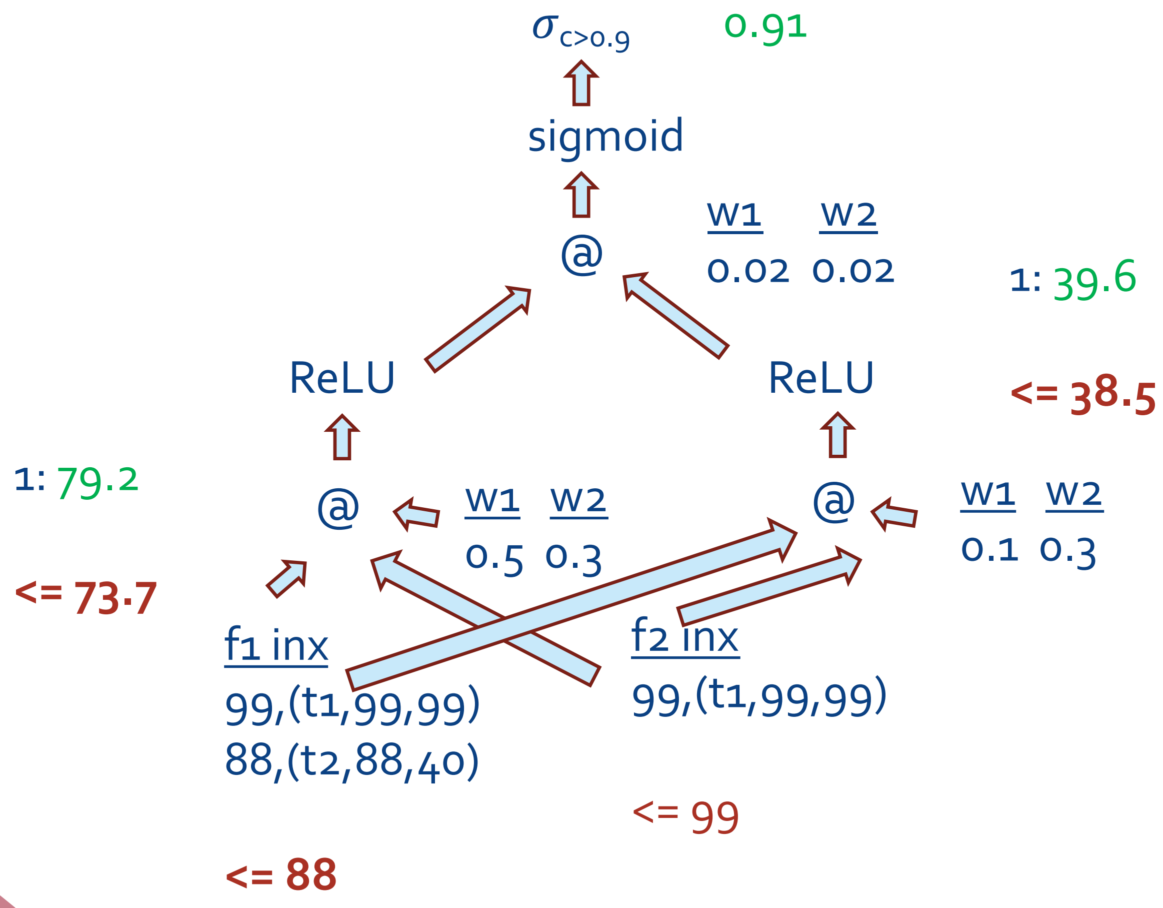
Feed-Forward with SIPS

Inspiration: Fagin's Algorithm (FA) [Fagin et al. 01] for thresholding



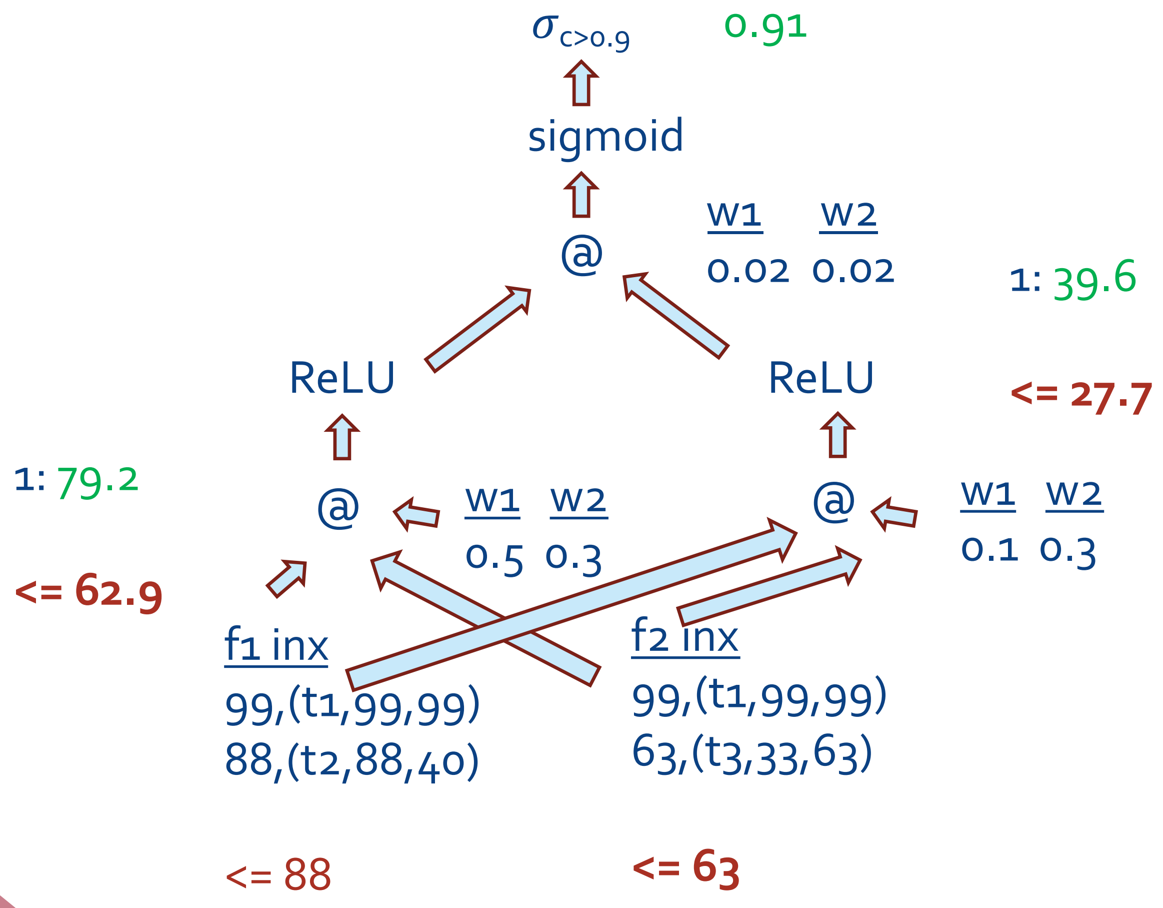
Feed-Forward with SIPS

Inspiration: Fagin's Algorithm (FA) [Fagin et al. 01] for thresholding



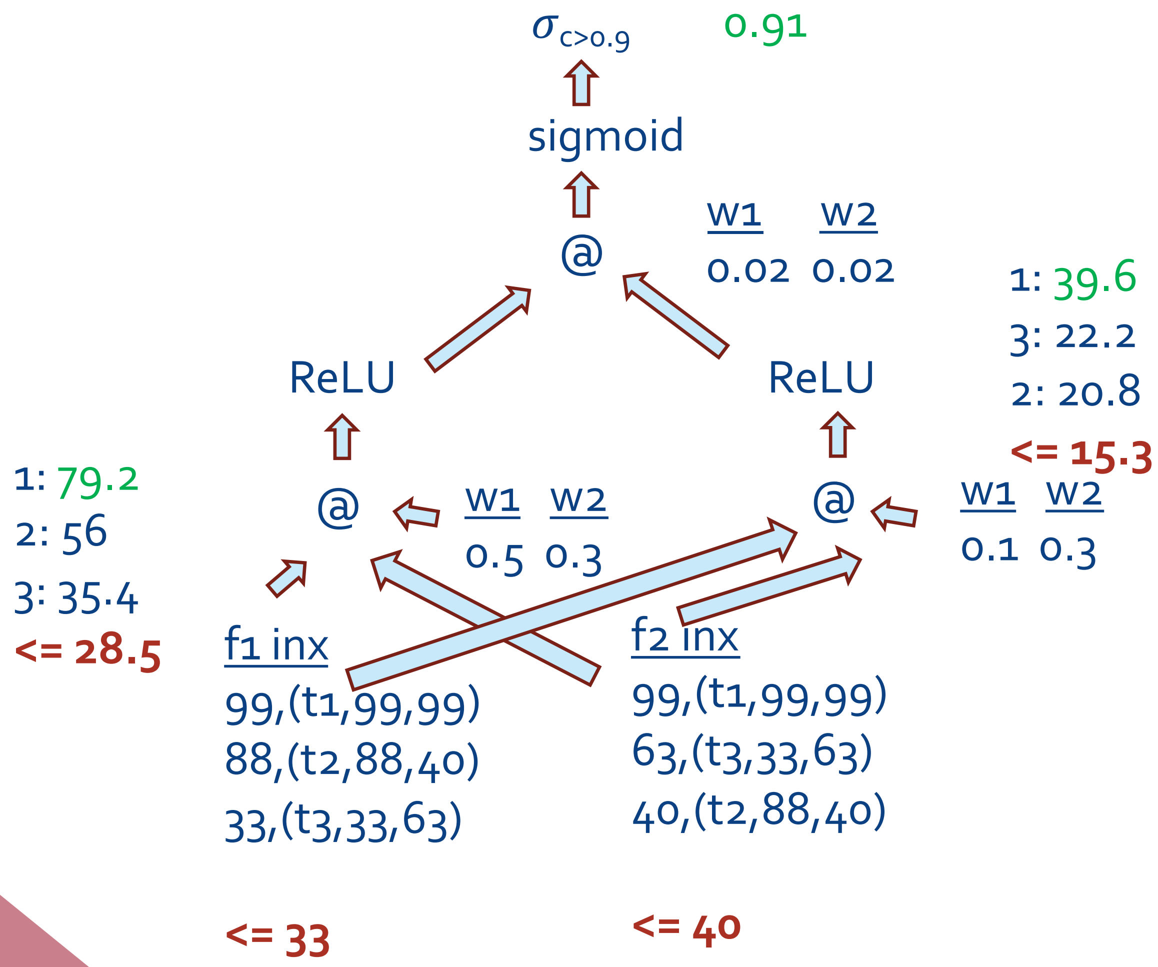
Feed-Forward with SIPS

Inspiration: Fagin's Algorithm (FA) [Fagin et al. 01] for thresholding



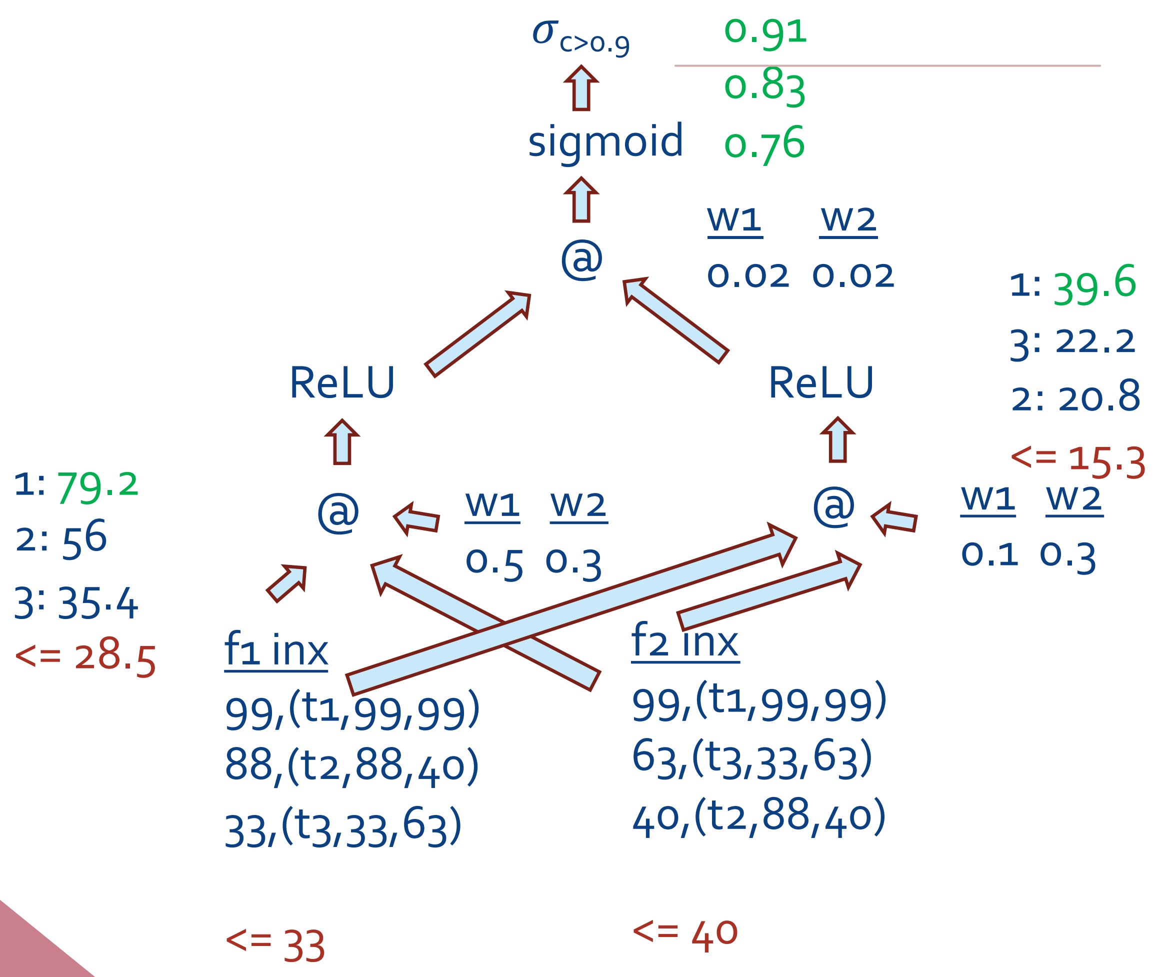
Feed-Forward with SIPS

Inspiration: Fagin's Algorithm (FA) [Fagin et al. 01] for thresholding



Feed-Forward with SIPS

Inspiration: Fagin's Algorithm (FA) [Fagin et al. 01] for thresholding



Propagate 3 tuples to
get all values > 0.9

Algorithm Sketch

[Chen et al. under review]

Pre-index subset of fields that are "important" to prediction (use Shapley values)

NN, on call to getNextTuple():

- Recursively (by layer) fetch from input streams, using **thresholding as a sideways information passing strategy** to minimize retrievals!
  - **Fetch** in *descending* ($w_i > 0$) or *ascending* ($w_i < 0$) order
  - **Update** thresholds on input sub-streams

Results in significant gains in amount of work if predictions are selective!

Caveat: modern hardware is optimized for regularized tensor computation!

For standard prediction, GPUs can be faster than "smart" processing with control flow!

BUT: what if we only make *small changes* to the state, i.e., *view maintenance*?

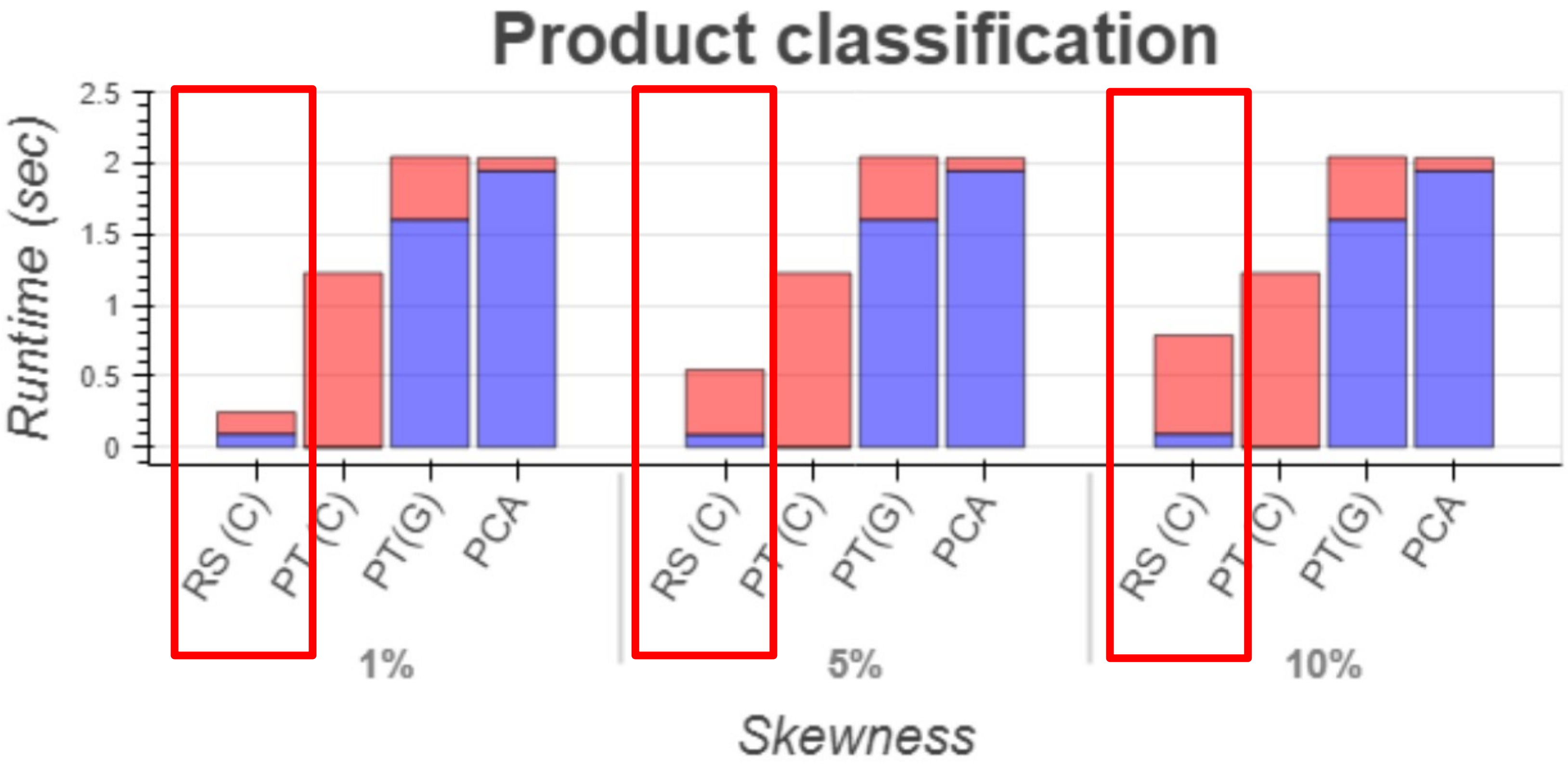
Incremental update upon change to model parameters:

- Recompute scores of existing outputs – threshold any that no longer pass
- Recursively fetch any new tuples above the threshold

We use thresholds at each layer from the prior iteration as a starting point!

Highlight: Roberta + Neural Network, 4 Classes

[Chen et al. under review]



RS (C) = ReSolution, PyTorch with thresholding, CPU-based
PT (C) = PyTorch, CPU-based
PT (G) = PyTorch, GPU-based
PCA = PyTorch Update on lower-dimensionality data

The Story So Far: Incremental Re-Classification

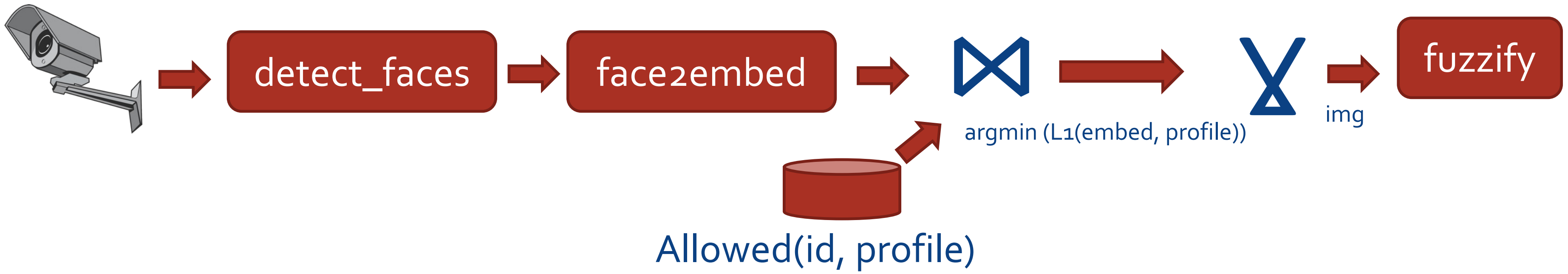
Across multiple datasets (image detection, products, info extraction)), 1.5 – 8x speedups (vs PyTorch in CPU or GPU mode) for incremental update, when we have up to 10% of the instances in the detected class

- **Naïve feed-forward** can be replaced by **smarter sideways information-passing**, early pruning!
  - i.e., applying ideas from query optimization to mixed selection-classification queries!

Sometimes, prediction is a bottleneck in the first place

Requires us to generalize **query (expression) equivalence** with ML ops!!!

Providing Consistent Performance



In a streaming setting: latencies of data processing + classification algorithms will depend on the data

- e.g. face detection +recognition on a crowd vs zero or one face

May have latency / throughput constraints

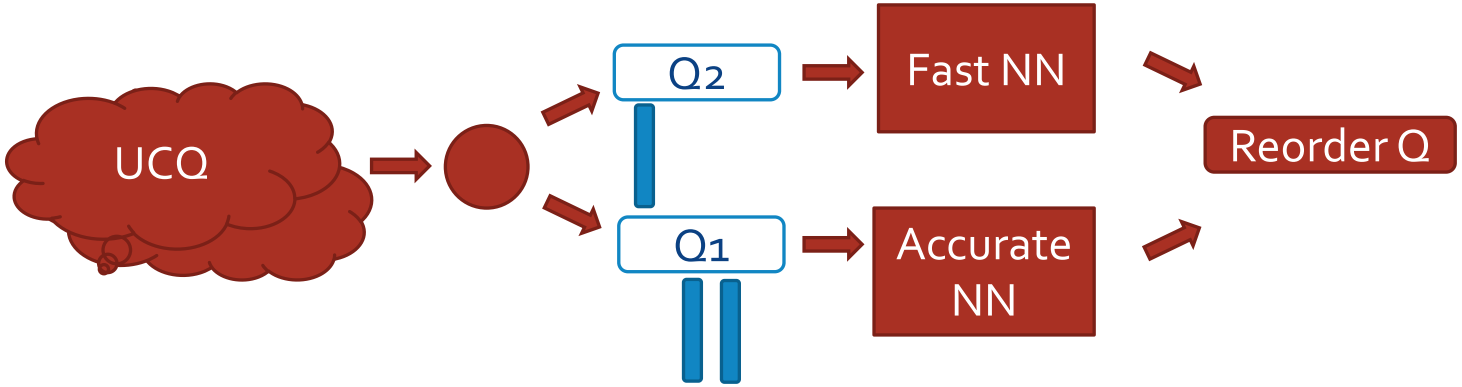
➢ Idea: consider alternative classifier implementations

- BUT: what does **equivalence** mean here???
- “Equivalent” in a functional sense, with a partial ordering based on “expected quality”
  - Precision, model size / complexity, CPU vs GPU, …
- Always want to push for **best quality** while **maintaining throughput**

Prioritized Eddies [Work-in-Progress]

With Phillip Hilliard, Rajeev Alur

Inspired by + adapted from **Eddies [Avnur & Hellerstein 2000]**



Assume *ordering* among preferences among multiple NN pipelines, e.g.:

Accurate face detector 95% F1 on validation data, 2 units running time >

Fast face detector 92% F1 on validation data, 1 units running time

Sketch of our approach:

1. When load on high-priority pipeline exceeds threshold, enable less-preferred pipeline
2. Then randomly choose queue, inversely proportional to queue length
3. Disable less-preferred pipeline once queues "drain"

Summary of Work-in-Progress on ML-Integrated Views

- ML-integrated views of data provide interesting new opportunities!
  - Connections between provenance and view update
  - Sideways information passing
  - Relaxed notions of equivalence
- Excited to get feedback, ideas, potential collaborations for exploring this rich space!

Some Foundational Questions with Practical Implications

1. Gradient semirings assign "blame" when we need to make an SGD update
   - Are there other generalizations of this idea?
2. Similarity joins between learned embeddings [with Erik Waingarten]
   - How do we compute top-k join results efficiently, with multiple sim. predicates?
   - Maintenance questions, esp. under fine-tuning?
3. Is there a clean provenance model for "functionally equivalent" expressions in the ML sense?
4. Can declarative techniques let us go beyond automated hyperparameter tuning for classifiers?
   - In a declarative system, where we are querying over "part" of a model's output – can we co-train a simplified model while using it, for efficiency?