SQL4NN: Validation and Expressive Querying of Models as Data

Mark Gerarts

Juno Steegmans

Jan Van den Bussche

DSI DATA SCIENCE INSTITUTE

UHASSELT

WWW.UHASSELT.BE/DSI

Model Queries

Industry uses model registries that store models as binary objects (“black boxes”).

Our proposal: store models in a database as **intensional data**, allowing rich queries:

- Evaluate a model on some input(s) (inference)
- Compare output of multiple models
- Feature attribution
- Get number of layers and nodes per layer
- Identify prunable nodes
- Geometric representation
- Verify unseen inputs (e.g., monotonicity, boundedness)
- Does altering a few inputs slightly change the prediction?
- …

Evaluation

White-box

Verification

$\mathbf{FO(R)}$

First-order logic over the reals

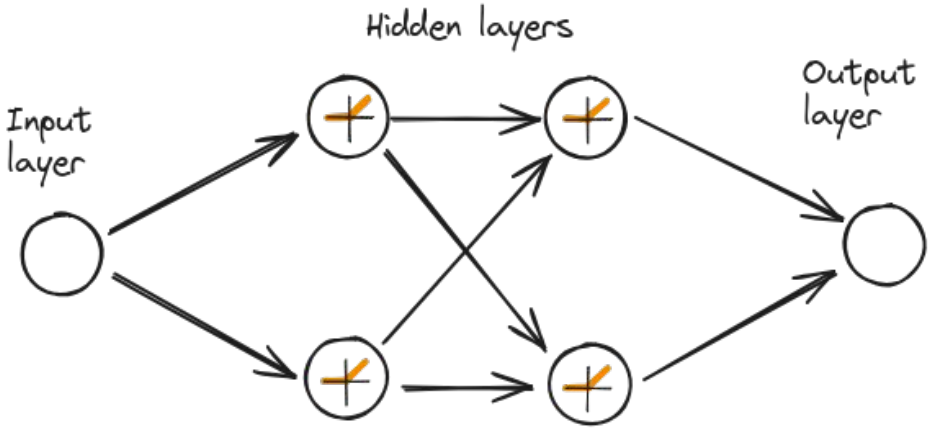
$$\forall b' \quad |F(a,b,c) - F(a,b',c)| < \epsilon$$

$\mathbf{FO(SUM)}$ = SQL

First-order logic over weighted structures

$$\sum_{x:E(in,x)} w(in,x) < \epsilon$$

Compilation



Node(id, bias)
Edge(src, dst, weight)
Input(vec_id, in_id, value)

$$t_1(u) := ReLU(b(u) + \sum_i w(in_i, u) \cdot val_i)$$

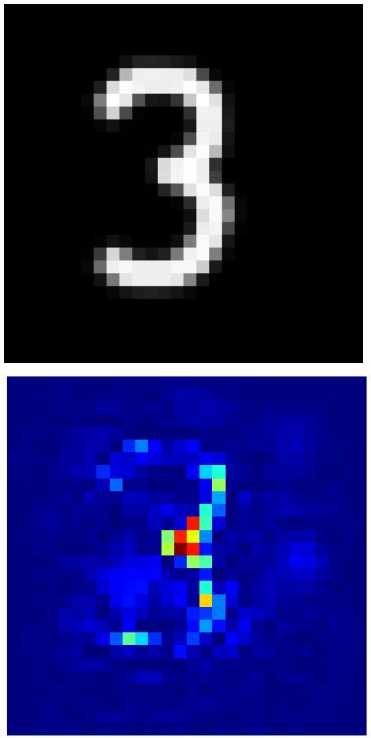
$$t_l(u) := ReLU(b(u) + \sum_{x:E(x,u)} w(x, u) \cdot t_{l-1}(x))$$

$$eval_j := b(out_j) + \sum_{x:E(x,out_j)} w(x, out_j) \cdot t_l(x)$$

```
WITH RECURSIVE
input_nodes AS (/**/), output_nodes AS (/**/),
tx AS (
   SELECT
       v.vec_id AS vec_id,
       GREATEST(0, n.bias + SUM(e.weight * v.val)) AS value,
       e.dst AS id
   FROM edge e
   JOIN input_nodes i ON i.id = e.src
   JOIN node n ON e.dst = n.id
   JOIN input v ON i.id = v.in_id
   GROUP BY e.dst, n.bias, v.input_set_id
   UNION ALL
   SELECT
       tx.vec_id AS vec_id,
       GREATEST(0, n.bias + SUM(e.weight * tx.value)) AS value,
       e.dst AS id
   FROM edge e
   JOIN tx ON tx.id = e.src
   JOIN node n ON e.dst = n.id
   GROUP BY e.dst, n.bias, tx.vec_id
),
t_out AS ( /* Join on last layer of tx and omit ReLU */)
SELECT vec_id, output_value, output_node_id
FROM t_out
ORDER BY vec_id, output_node_id;
```

Model(id, name, …)
Node(id, bias, **model_id**)
Edge(src, dst, weight, **model_id**)
Input(vec_id, in_id, value)

```
WITH RECURSIVE
input_nodes AS (/**/), output_nodes AS (/**/),
tx AS (
   SELECT
       i.model_id, v.vec_id AS vec_id,
       GREATEST(0, n.bias + SUM(e.weight * v.val)) AS value,
       e.dst AS id
   FROM edge e
   JOIN input_nodes i ON i.id = e.src
   JOIN node n ON e.dst = n.id
   JOIN input v ON i.id = v.in_id
     AND v.model_id = i.model_id
   GROUP BY i.model_id, e.dst, n.bias, v.vec_id
   UNION ALL
   SELECT
       tx.model_id, tx.vec_id AS vec_id,
       GREATEST(0, n.bias + SUM(e.weight * tx.value)) AS value,
       e.dst AS id
   FROM edge e
   JOIN tx ON tx.id = e.src AND tx.model_id = e.model_id
   JOIN node n ON e.dst = n.id
   GROUP BY tx.model_id, e.dst, n.bias, tx.vec_id
),
t_out AS ( /* Join on last layer of tx and omit ReLU */
SELECT model_id, vec_id, output_value, output_node_id
FROM t_out
ORDER BY model_id, vec_id, output_node_id;
```

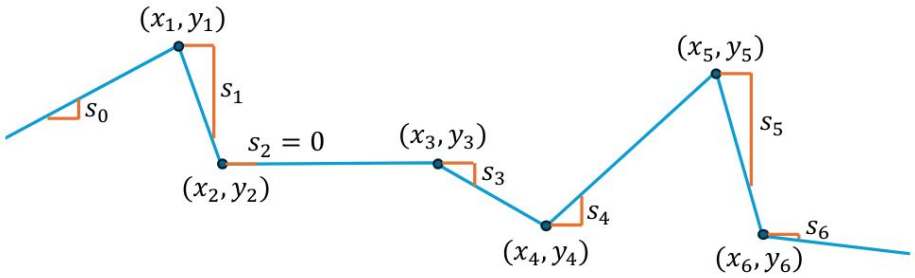


```
WITH input AS (
    SELECT
        i2.in_id AS vec_id,
        i1.in_id,
        CASE WHEN i1.in_id = i2.in_id
            THEN 0
            ELSE i1.val
        END AS val
    FROM input i1, input i2
),
/* ... */
```

$$\sum_{x:E(in,x)} w(in,x) < \epsilon$$

```
SELECT dst
FROM edge
GROUP BY dst
HAVING MAX (ABS(weight)) <= epsilon
```

$$break_x := \frac{-b(u)}{w(in_1, u)}$$




```
WITH /* ... */
breakpoint_values AS (
    SELECT (-n.bias) / e.weight AS break_x,
    FROM node n
    JOIN edge e ON e.dst = n.id
    JOIN input_nodes i ON e.src = i.id
    JOIN hidden_nodes h ON h.id = n.id
    WHERE e.weight <> 0
    GROUP BY break_x, n.id
    ORDER BY break_x
),
/* ... */
points_and_slopes AS (
    SELECT
        u1_break_x AS x,
        u1_break_y AS y,
        (u2_break_y - u1_break_y) / (u2_break_x -
u1_break_x) AS slope
    FROM breakpoint_pairs
    ORDER BY x
)
SELECT * FROM points_and_slopes;
```

$$\forall x \forall x' (x \geq x') \implies (F(x) \geq F(x'))$$

```
SELECT NOT EXISTS(
    -- Search for counterexample
    SELECT 1
    FROM points_and_slopes
    -- Closest breakpoint left of lower
bound
    WHERE x >= (SELECT MAX(x)
                FROM points_and_slopes
                WHERE x < -1)
    AND x < 1
    AND slope < 0
) AS 'Monotonicity property holds' ;
```

$$\forall x(x \geq -2\pi \wedge x \leq 2\pi)$$
$$\implies (F(x) \geq -1 \wedge F(x) \leq 1)$$

```
WITH all_points_in_range AS (
   SELECT x, y FROM points_and_slopes
   WHERE x >= -6.28 AND x <= 6.28
   UNION ALL
   -- Lower bound of the input range.
   SELECT -6.28, (-6.28 - x) * slope + y AS y
   FROM points_and_slopes
   WHERE x = (SELECT MAX(x) FROM points_and_slopes WHERE x <
-6.28)
   UNION ALL
   -- Upper bound of the input range.
   SELECT 6.28, (6.28 - x) * slope + y AS y
   FROM points_and_slopes
   WHERE x = (SELECT MIN(x) FROM points_and_slopes WHERE x > 6.28)
)
SELECT NOT EXISTS(
   SELECT 1
   FROM all_points_in_range
   WHERE (y < -1 OR y > 1)
) AS 'Boundedness property holds' ;
```

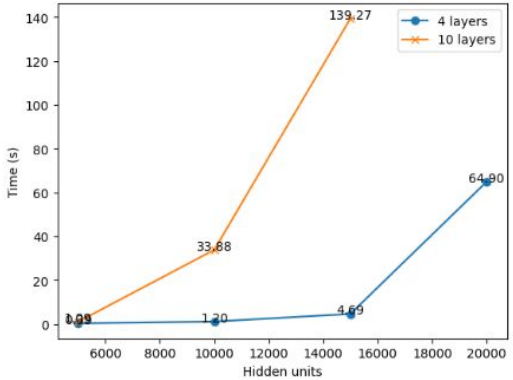
```
CREATE MACRO prune_and_verify(number_of_nodes_to_prune)
AS TABLE
/* ... */

WITH RECURSIVE pruned_models AS (
   SELECT
       0 AS number_of_pruned_nodes,
       (SELECT property_holds
        FROM prune_and_verify(number_of_pruned_nodes)
       ) AS property_holds
   UNION ALL
   SELECT
       number_of_pruned_nodes + 100,
       (SELECT property_holds
        FROM prune_and_verify(number_of_pruned_nodes)
       ) AS property_holds
   FROM pruned_models
   WHERE monotonic
)
SELECT * FROM pruned_models;
```

Conclusion

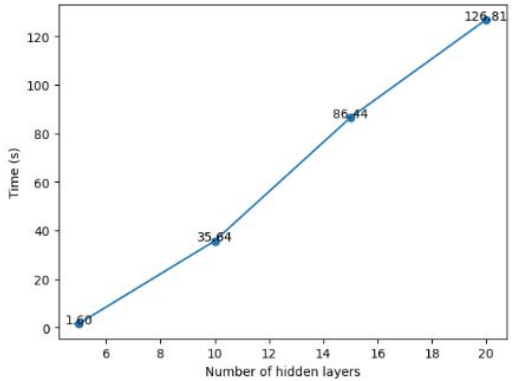
SQL can query neural networks, with applications in explainability and **verification**.

Main challenges:

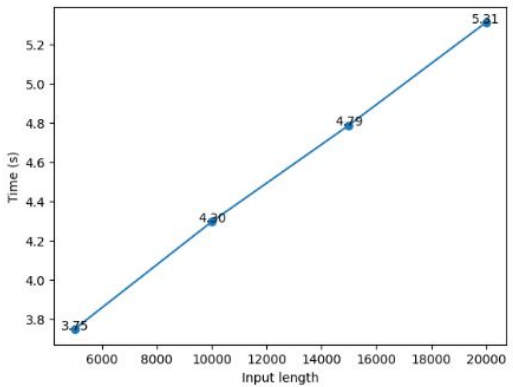
- Network restrictions (especially for verification)
- Scaling to large networks (thousands of nodes works reasonably on commodity hardware already)



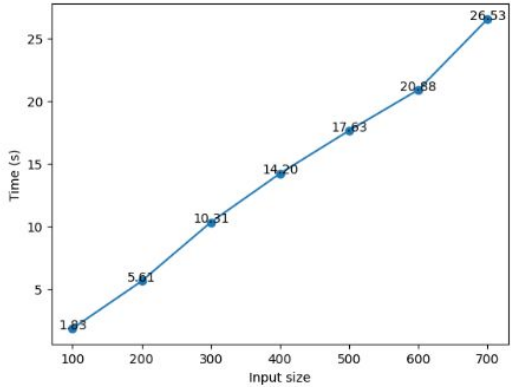
(a) Scaling the number of hidden units for a network with 4 hidden layers, and one with 10 hidden layers.



(b) Scaling the number of hidden layers.



(c) Scaling the input length.



(d) Scaling the input size.