Postgres and the Artificial Intelligence Landscape

Bruce Momjian

This presentation explains how to do machine learning inside the Postgres database.

https://momjian.us/presentations

Last updated: September, 2020

Outline

1. What is artificial intelligence?
2. Machine learning and deep learning
3. Demonstration using Postgres
4. Hardware/software efficiency
5. Tasks
6. Why use a database?

1. What is Artificial Intelligence?

Machines that mimic "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving".

https://en.wikipedia.org/wiki/Artificial_intelligence

What is Artificial about Artificial Intelligence?

If only the physical world exists, then human intelligence only differs from machine intelligence because it has not naturally developed. It hence differs only in how it is created. Human free will becomes an illusion.

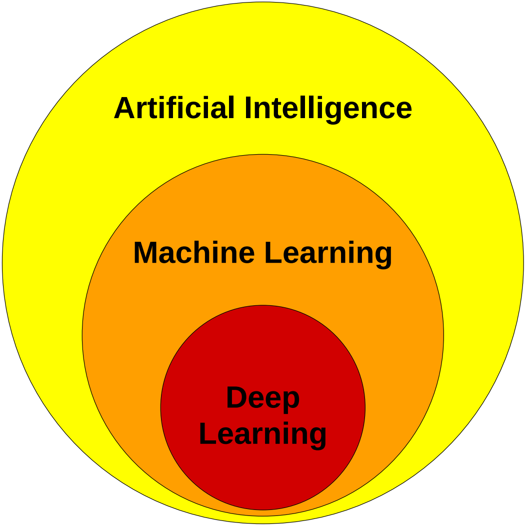
https://www.theatlantic.com/magazine/archive/2016/06/theres-no-such-thing-as-free-will/480750/

History of Artificial Intelligence (AI)

- Pre-computer philosophy
- Robotics
- Turing test
- Expert systems
- AI winter
- Like fusion energy, it is always ten years away

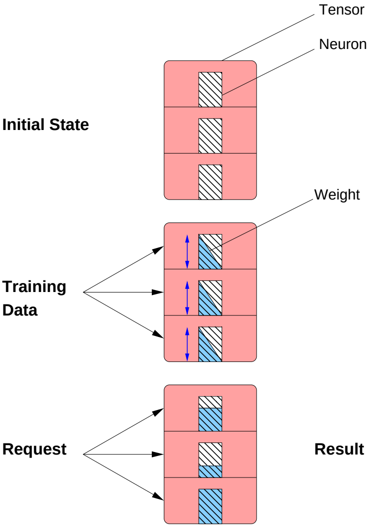
https://en.wikipedia.org/wiki/History_of_artificial_intelligence

2. Machine Learning and Deep Learning



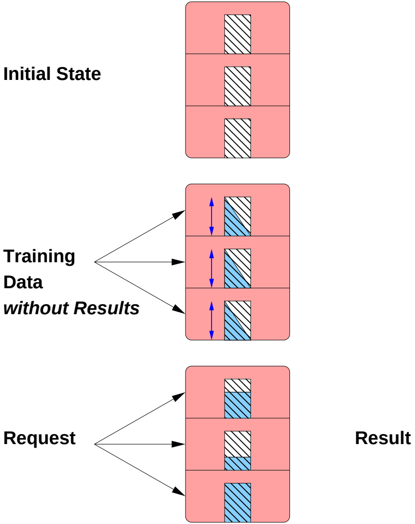
The most comprehensive video I have seen about machine learning is at https://www.youtube.com/watch?v=r0Ogt-q956I.

Machine Learning



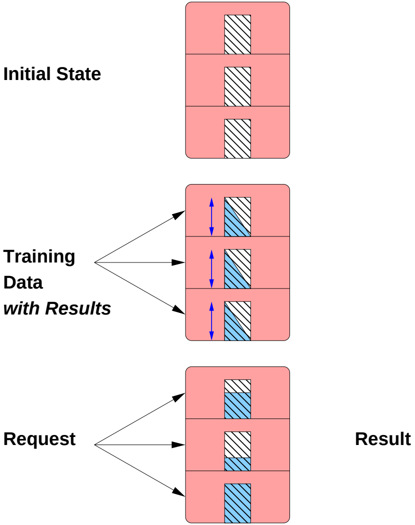
https://www.verypossible.com/insights/machine-learning-algorithms-what-is-a-neural-network

Unsupervised Machine Learning

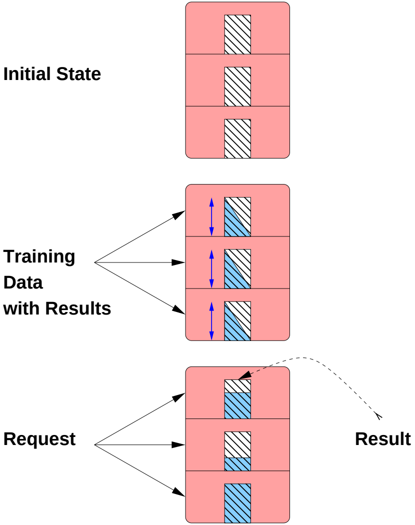


https://medium.com/@machadogj/ml-basics-supervised-unsupervised-and-reinforcement-learning-b18108487c5a

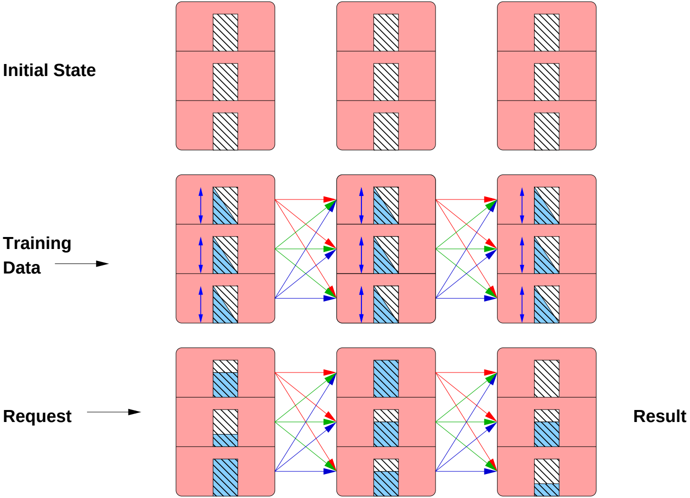
Supervised Machine Learning



Reinforcement Machine Learning



Deep Learning



https://www.zendesk.com/blog/machine-learning-and-deep-learning/

3. Demonstration Using Postgres: Does an Integer Have Non-Leading Zeros?

- 31903 is true
- 82392 is false

Install PL/Perl

```
CREATE EXTENSION IF NOT EXISTS plperl;
```

All queries in this presentation can be downloaded from https://momjian.us/main/writings/pgsql/AI.sql.

Generate Tensor

```
CREATE OR REPLACE FUNCTION generate_tensor(value INTEGER)
RETURNS BOOLEAN[] AS $$
        my $value = shift;
        my @tensor = (
                # this many digits or more?
                (map { length($value) >= $_ } 1..10),
                # divisible by zero?
                $value % 10 == 0,
        );
        # map to t/f
        grep { $_ = ($_ ? 't' : 'f') } @tensor;
        return encode_typed_literal(\@tensor, 'boolean[]');
$$ LANGUAGE plperl STRICT;
```

Create and Populate Input Layer

```
CREATE TABLE training_set(value INTEGER, training_output BOOLEAN,
                          tensor BOOLEAN[]);
WITH randint (value) AS
(
        SELECT (random() * (10 ^ (random() * 8 + 1)::integer))::integer
        FROM generate_series(1, 10000)
)
INSERT INTO training_set SELECT value, value::text LIKE '%0%',
                                generate_tensor(value)
FROM randint;
```

Input Layer

```
SELECT * FROM training_set LIMIT 10;
   value    | training_output |          tensor
------------+-----------------+-------------------------
  28762748 | f               | {t,t,t,t,t,t,t,t,f,f,f}
  44550313 | t               | {t,t,t,t,t,t,t,t,f,f,f}
        72 | f               | {t,t,f,f,f,f,f,f,f,f,f}
   4891026 | t               | {t,t,t,t,t,t,t,f,f,f,f}
      3413 | f               | {t,t,t,t,f,f,f,f,f,f,f}
        62 | f               | {t,t,f,f,f,f,f,f,f,f,f}
  86517976 | f               | {t,t,t,t,t,t,t,t,f,f,f}
       967 | f               | {t,t,t,f,f,f,f,f,f,f,f}
 636667644 | f               | {t,t,t,t,t,t,t,t,t,f,f}
     36419 | f               | {t,t,t,t,t,f,f,f,f,f,f}
```

Generate Weights for Tensor

```
CREATE OR REPLACE FUNCTION generate_weight(query TEXT, desired_output BOOLEAN)
RETURNS REAL[] AS $$
        my $rv = spi_exec_query(shift);
        my $status = $rv->{status};
        my $nrows = $rv->{processed};
        my $desired_output = shift;
        my @success_neurons = ();
        my @desired_neurons = ();
        my $desired_input = 0;
```

Generate Weights for Tensor

```
foreach my $rn (0 .. $nrows - 1) {
            my $row = $rv->{rows}[$rn];
            my $tensor = $row->{(sort keys %$row)[0]};
            my $training_output = $row->{(sort keys %$row)[1]};
            # only process training rows that match our desired output
            foreach my $neuron (0 .. $#$tensor)
            {
                    $success_neurons[$neuron] //= 0;
                    $desired_neurons[$neuron] //= 0;
                    # Neuron value matches desired output value;  does
                    # the value match the desired output?
                    if ($tensor->[$neuron] eq $desired_output)
                    {
                            # Prediction success/failures that match our
                            # desired output.
                            $success_neurons[$neuron]++
                                    if ($training_output eq $desired_output);
                            $desired_neurons[$neuron]++;
                    }
            }
            $desired_input++ if ($training_output eq $desired_output);
    }
```

Generate Weights for Tensor

```
        my @weight = ();
        my $sum = 0;

    # compute percentage of tests that matched requested outcome
        foreach my $neuron (0 .. $#success_neurons) {
                $weight[$neuron] = $desired_neurons[$neuron] != 0 ?
                        $success_neurons[$neuron] / $desired_neurons[$neuron] :
                        0;
                $sum += $weight[$neuron];
        }

        # balance weights so they total the observed probability;
        # this prevents an overly-predictive output value from skewing
        # the results.
        foreach my $neuron (0 .. $#weight) {
                $weight[$neuron] = ($weight[$neuron] / $sum) *
                                   ($desired_input / $nrows);
        }
        return encode_typed_literal(\@weight, 'real[]');
$$ LANGUAGE plperl STRICT;
```

Create Tensor_Mask

```
# Return weights where our neuron value matches the desired output
CREATE OR REPLACE FUNCTION tensor_mask(tensor BOOLEAN[], weight REAL[],
                                       desired_output BOOLEAN)
RETURNS REAL[] AS $$
        my $tensor = shift;
        my $weight = shift;
        my $desired_output = shift;
        my @result = ();

        elog(ERROR, 'tensor and weight lengths differ')
                if ($#$tensor != $#$weight);
        foreach my $i (0 .. $#$tensor) {
                push(@result,
                        ($tensor->[$i] eq $desired_output) ?
                        $weight->[$i] : 0);
        }
        return encode_typed_literal(\@result, 'real[]');
$$ LANGUAGE plperl STRICT;
```

Create Sum_Weight

```
CREATE OR REPLACE FUNCTION sum_weight(weight REAL[])
RETURNS REAL AS $$
        my $weight = shift;
        my $sum = 0;
        # sum weights
        foreach my $i (0 .. $#$weight) {
                $sum += $weight->[$i];
        }
        return encode_typed_literal($sum, 'real');
$$ LANGUAGE plperl STRICT;
```

Create Soft_Max

```
# Normalize the values so the probabilities total one
CREATE OR REPLACE FUNCTION softmax(val1 REAL, val2 REAL)
RETURNS REAL[] AS $$
        my $val1 = shift;
        my $val2 = shift;
        my $sum = $val1 + $val2;
        # What percentge is each of the total?
        my @result = (
                $val1 / $sum,
                $val2 / $sum,
        );
        return encode_typed_literal(\@result, 'real[]');
$$ LANGUAGE plperl STRICT;
```

Store Weights

```
CREATE TABLE tensor_weight_true AS
SELECT generate_weight('SELECT tensor AS x1, training_output AS x2
                        FROM training_set', true) AS weight;

CREATE TABLE tensor_weight_false AS
SELECT generate_weight('SELECT tensor AS x1, training_output AS x2
                        FROM training_set', false) AS weight;
```

Stored Weights

```
SELECT * FROM tensor_weight_true;
                            weight
---------------------------------------------------------------
 {0.020473005,0.021917565,0.024002228,0.026247077,0.028482921, \
  0.030471962,0.032726202,0.034238704,0.036621932,0,0.0641184}

SELECT * FROM tensor_weight_false;
                           weight
-------------------------------------------------------------
 {0,0.0820682,0.07662672,0.074060954,0.07129263,0.068018064, \
  0.06497674,0.061864104,0.059269458,0.058057636,0.06446551}
```

Test 100

```
WITH test_set (checkval) AS
(
        SELECT 100
)
SELECT softmax(
        sum_weight(
                tensor_mask(
                        generate_tensor(checkval),
                        tensor_weight_true.weight,
                        true)),
        sum_weight(
                tensor_mask(
                        generate_tensor(checkval),
                        tensor_weight_false.weight,
                        false))
)
FROM test_set, tensor_weight_true, tensor_weight_false;
         softmax
-------------------------
 {0.22193865,0.77806133}
```

Test 101

```
WITH test_set (checkval) AS
(
        SELECT 101
)
SELECT softmax(
        sum_weight(
                tensor_mask(
                        generate_tensor(checkval),
                        tensor_weight_true.weight,
                        true)),
        sum_weight(
                tensor_mask(
                        generate_tensor(checkval),
                        tensor_weight_false.weight,
                        false))
)
FROM test_set, tensor_weight_true, tensor_weight_false;
         softmax
-------------------------
 {0.11283657,0.88716346}
```

Test 487234987

```
WITH test_set (checkval) AS
(
        SELECT 487234987
)
SELECT softmax(
        sum_weight(
                tensor_mask(
                        generate_tensor(checkval),
                        tensor_weight_true.weight,
                        true)),
        sum_weight(
                tensor_mask(
                        generate_tensor(checkval),
                        tensor_weight_false.weight,
                        false))
)
FROM test_set, tensor_weight_true, tensor_weight_false;
         softmax
-------------------------
 {0.68860435,0.31139567}
```

Test One Thousand Values

```
WITH test_set (checkval) AS
(
        SELECT (random() * (10 ^ (random() * 8 + 1)::integer))::integer
        FROM generate_series(1, 1000)
),
```

Second Table Expression

```
ai (checkval, output_layer) AS
(
        SELECT checkval, softmax(
                sum_weight(tensor_mask(generate_tensor(checkval),
                           tensor_weight_true.weight, true)),
                sum_weight(tensor_mask(generate_tensor(checkval),
                           tensor_weight_false.weight, false))
        )
        FROM test_set, tensor_weight_true, tensor_weight_false
),
```

Third Table Expression

```
analysis (checkval, cmp_bool, output_layer, accuracy) AS
(
	SELECT checkval, checkval::text LIKE '%0%', output_layer,
		CASE checkval::text LIKE '%0%'
		-- higher/lower than random chance
		WHEN true THEN output_layer[1] - 0.5
		ELSE output_layer[2] - 0.5
		END
	FROM ai
)
```

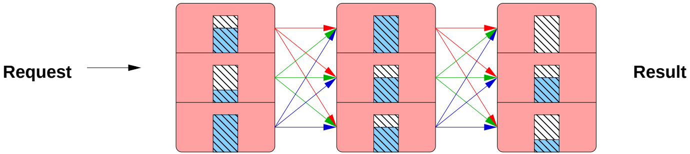
Final Table Expression

```
SELECT * FROM analysis
UNION ALL
SELECT NULL, NULL, NULL, AVG(accuracy)
FROM analysis
UNION ALL
SELECT NULL, NULL, NULL, SUM(CASE WHEN accuracy > 0 THEN 1 END)::REAL/COUNT(*)
FROM analysis;
 checkval  | cmp_bool |       output_layer        |       accuracy
-----------+----------+---------------------------+-----------------------
         6 | f        | {0.029198222,0.9708018}   |   0.47080177068710327
  61859931 | f        | {0.5459184,0.4540816}     | -0.045918405055999756
  53138008 | t        | {0.5459184,0.4540816}     |  0.045918405055999756
       727 | f        | {0.11283657,0.88716346}   |    0.3871634602546692
  33397006 | t        | {0.5459184,0.4540816}     |  0.045918405055999756
  38380069 | t        | {0.5459184,0.4540816}     |  0.045918405055999756
   8915576 | f        | {0.4306789,0.5693211}     |   0.06932109594345093
       446 | f        | {0.11283657,0.88716346}   |    0.3871634602546692
…
    (null) | (null)   | (null)                    |   0.15426481181383134
    (null) | (null)   | (null)                    |                 0.722
```

4. Hardware/Software Efficiency: Software

- Madlib
- Matlab
- Tensorflow
- Weka
- Scikit,
  - using PL/Python, https://www.cybertec-postgresql.com/en/machine-learning-in-postgresql-part-1-kmeans-clustering/
  - client-side, https://kb.objectrocket.com/postgresql/machine-learning-with-python-and-postgres-1114

GPU



- Tensors can have millions of neurons
- Deep learning can use thousands of tensor layers
- Every neuron passes its data to every neuron in the next layer
- This requires a lot of repetitive calculations
- GPUs are designed to efficiently perform simultaneous repetitive computations

5. Tasks

- Chess
- Jeopardy
- Voice recognition
- Search
- Recommendations
- Image detection
- Weather forecasting

Fraud Detection Example

Choose attributes:

- Charge amount
- Magnetic swipe, chip, pin, online charge
- Vendor distance from chargee billing address
- Distance from last chargee charge
- Vendor country
- Previous charges to this vendor for chargee
- Previous fraudulent charges by vendor

https://www.youtube.com/watch?v=ztVcNMg7TFs

Fraud Detection Steps

1. Choose attributes
2. Create machine learning neurons for each attribute
3. Create training data, with the required attributes of each transactions and its outcome, i.e., valid or fraudulent
4. Feed the training data into the machine to set the weights of each neuron, based on how much the neuron's attribute predicts the validity or fraudulence of transactions
5. Start feeding real data into the machine and get results
6. Feed correct and incorrect results back into the neurons to improve the accuracy of the weights, and to adjust for changes in the environment

6.Why Use a Database?

- Machine learning requires a lot of data
- Most of your data is in your database
- Why not do machine learning where your data is, in a database?

Advantages of doing Machine Learning in a Database?

- Use previous activity as training data
- Have seamless access to all your current data
- Take immediate action on AI results, e.g., commit transaction only if likely non-fraudulent
- AI can benefit from database transactions, concurrency, backup
- Other benefits include complex data types, full text search, GIS, indexing
- Postgres can do GPU-based computations inside the database (`https://momjian.us/main/blogs/pgblog/2020.html#June_29_2020`)

Conclusion

https://momjian.us/presentations

https://www.flickr.com/photos/corneveaux/