PostgreSQL And Artificial Intelligence

Slides:
https://github.com/ibrarahmad/PostgreSQLTalks/tree/main/Conferences/

Ibrar Ahmed
Principal Engineer @ Pecona LLC
Software Career
Software industries since 1998.

PostgreSQL Career
• Working on PostgreSQL Since 2006.
• EnterpriseDB (Senior Software Architect) 10 Years
• Percona (Principal Engineer) 2018 – Present

Open-source
• PostgreSQL
• Google Chrome
• Google Chromium Project.

PostgreSQL Books
• PostgreSQL Developer's Guide
• PostgreSQL 9.6 High Performance

Ibrar Ahmed
Principal Engineer Percona LLC.
Artificial Intelligence

What is artificial intelligence?

PostgreSQL and AI

How to use AI in PostgreSQL

Demonstration

Demonstration Using Postgres
What is Artificial Intelligence?

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

Artificial Intelligence

• “Artificial Intelligence is no match for natural stupidity”

  Albert Einstein

• “A machine with strong A.I. is able to think and act just like a human. It is able to learn from experiences”

  Albert Einstein

• “I think the development of full artificial intelligence could spell the end of the human race”

  Stephen Hawking

• “The one who becomes the leader in this sphere will be the ruler of the world”

  Vladimir Putin
Types of Artificial Intelligence

1. **ANI** - Artificial Narrow Intelligence
   - ANI has a narrow range of ability to perform specific tasks.

2. **AGI** - Artificial General Intelligence
   - AGI can perform different tasks that humans can do.

3. **ASI** - Artificial Super-Intelligence
   - ASI is more capable than a human, learning from past experience and from new data to do a variety of tasks.
Sub-Fields of Artificial Intelligence

- Machine Learning
- Deep Learning
- Neural Network
- Cognitive Computing
- NLP and Computer Vision
Machine Learning
Supervised Learning (Regression, Classification)

Un-Supervised Learning (Association, Clustering)

Reinforcement Learning (Robotics)
Process Of Machine Learning

1. Get Data
2. Train Model
3. Clean, Prepare & Manipulate Data
4. Test Data
5. Improve
Deep Learning
Neural Networks

- Neural Networks, with more layers and modules
- It has a model which can take any input/output type and size
Machine Learning Vs Deep Learning

Machine Learning

Input → Feature extraction → Classification → Output

Deep Learning

Input → Feature extraction + Classification → Output

This Photo by Unknown Author is licensed under CC BY
Demonstration
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/Al.sql.
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;

<table>
<thead>
<tr>
<th>Value.</th>
<th>training_output</th>
<th>tensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>28762748</td>
<td>f</td>
<td>{t,t,t,t,t,t,t,t,f,f,f}</td>
</tr>
<tr>
<td>44550313</td>
<td>t</td>
<td>{t,t,t,t,t,t,t,t,f,f,f}</td>
</tr>
<tr>
<td>72</td>
<td>f</td>
<td>{t,t,f,f,f,f,f,f,f,f,f,f,f}</td>
</tr>
<tr>
<td>4891026</td>
<td>t</td>
<td>{t,t,t,t,t,t,t,t,f,f,f,f}</td>
</tr>
<tr>
<td>3413</td>
<td>f</td>
<td>{t,t,t,t,t,f,f,f,f,f,f,f}</td>
</tr>
<tr>
<td>62</td>
<td>f</td>
<td>{t,t,f,f,f,f,f,f,f,f,f,f,f}</td>
</tr>
<tr>
<td>86517976</td>
<td>f</td>
<td>{t,t,t,t,t,t,t,t,f,f,f}</td>
</tr>
<tr>
<td>967</td>
<td>f</td>
<td>{t,t,t,f,f,f,f,f,f,f,f,f}</td>
</tr>
<tr>
<td>636667644</td>
<td>f</td>
<td>{t,t,t,t,t,t,t,t,t,f,f,f}</td>
</tr>
<tr>
<td>36419</td>
<td>f</td>
<td>{t,t,t,t,t,t,t,t,t,f,f,f}</td>
</tr>
</tbody>
</table>
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 (Cont...)

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

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

# compute percentage of tests that matched requested outcome
foreach my $neuron (0 .. $#success_neurons) {
    $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[]');
```

```
-- 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 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 percentage is each of the total?

my @result = ( $val1 / $sum, $val2 / $sum, );

return encode_typed_literal(@(result, 'real[]'));

$$ LANGUAGE plperl STRICT;
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;
SELECT * FROM tensor_weight_true;

weight

{0.020473005, 0.021917565, 0.024002228, 0.026247077, 0.028482921, \n 0.030471962, 0.032726202, 0.034238704, 0.036621932, 0.0641184}

SELECT * FROM tensor_weight_false;

weight

{0, 0.0820682, 0.07662672, 0.074060954, 0.07129263, 0.068018064, \n 0.06497674, 0.061864104, 0.059269458, 0.058057636, 0.06446551}
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}
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}
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}
WITH test_set (checkval) AS ( 
    SELECT (random() * (10 ^ (random() * 8 + 1)::integer))::integer 
    FROM generate_series(1, 1000) 
),
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

\[
\text{analysis} \ (\text{checkval}, \ \text{cmp\_bool}, \ \text{output\_layer}, \ \text{accuracy}) \ \text{AS} \ ( \\
    \text{SELECT checkval, checkval::text LIKE } \%0\%', \ \text{output\_layer,} \\
    \text{CASE checkval::text LIKE } \%0\%' \\
    \quad \text{-- higher/lower than random chance WHEN true THEN output\_layer[1] - 0.5 ELSE output\_layer[2] = 0.5} \\
    \quad \text{END} \\
    \text{FROM ai} \\
) 
\]
**Final Table Expression**

```sql
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;
```

<table>
<thead>
<tr>
<th>Checkval</th>
<th>cmp_bool</th>
<th>output_layer</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>f</td>
<td>{0.029198222,0.9708018}</td>
<td>0.47080177068710327</td>
</tr>
<tr>
<td>61859931</td>
<td>f</td>
<td>{0.5459184,0.4540816}</td>
<td>0.045918405055999756</td>
</tr>
<tr>
<td>53138008</td>
<td>t</td>
<td>{0.5459184,0.4540816}</td>
<td>0.045918405055999756</td>
</tr>
<tr>
<td>727</td>
<td>f</td>
<td>{0.11283657,0.88716346}</td>
<td>0.3871634602546692</td>
</tr>
<tr>
<td>33397006</td>
<td>t</td>
<td>{0.5459184,0.4540816}</td>
<td>0.045918405055999756</td>
</tr>
<tr>
<td>38380069</td>
<td>t</td>
<td>{0.5459184,0.4540816}</td>
<td>0.045918405055999756</td>
</tr>
<tr>
<td>8915576</td>
<td>f</td>
<td>{0.4306789,0.5693211}</td>
<td>0.06932109594345093</td>
</tr>
<tr>
<td>446</td>
<td>f</td>
<td>{0.11283657,0.88716346}</td>
<td>0.3871634602546692</td>
</tr>
<tr>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>0.15426481181383134</td>
</tr>
<tr>
<td>(null)</td>
<td>(null)</td>
<td>(null)</td>
<td>0.722</td>
</tr>
</tbody>
</table>
Why to Use Database in AI?
Why Use 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 the 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.

General Artificial Intelligence Uses by Databases?

- User applications
  - Performance adjustments
  - Optimizer plans
  - Index creation/destruction
  - Database settings
  - Resource usage
- Alerting
  - Malicious activity
  - Resource exhaustion
Questions
Are you passionate about Open Source?!
We're looking for you!
Join us!
#RemoteWork
APPLY NOW: percona.com/careers