Mobile Robot Self-Driving Through Image Classification Using Sum-Product Networks

Student: Renato Lui Geh
Advisor: Prof. Dr. Denis Deratani Mauá

Institute of Mathematics and Statistics — University of São Paulo

Sum-Product Networks

Definition 1 (Sum-product network).

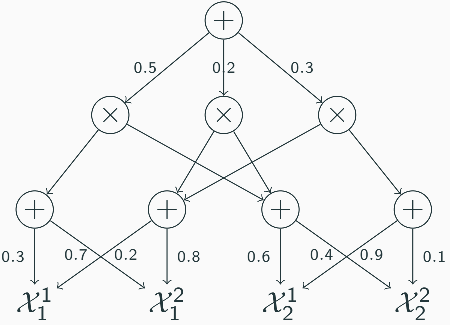
A sum-product network (SPN) is a DAG where each node n is either:

1. A tractable univariate probability distribution;
2. A product of SPNs: $v_n = \prod_{j \in \mathrm{Ch}(n)} v_j$; or
3. A weighted sum of SPNs: $v_n = \sum_{j \in \mathrm{Ch}(n)} w_{n,j} v_j$.

Where v_n is the value of node n, $\mathrm{Ch}(n)$ its set of children and $w_{n,j}$ the weight of edge $n \to j$.

Example

Sums can be interpreted as *mixture models* and **products** as their *components*. **Leaves** can take the form of categorical or continuous *distributions*.

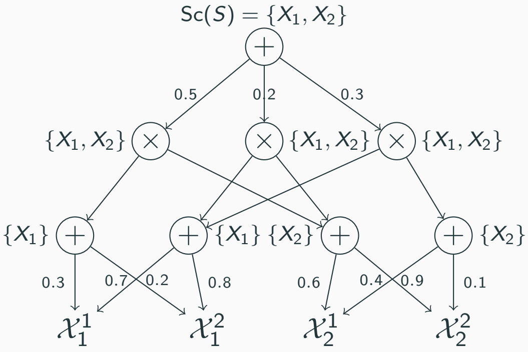


Where each leaf $\mathcal{X}_i^j$ is a probability distribution over RV X_i.

The scope Sc(n) of node n is the union of the scope of its children.

The scope of a leaf is the set of all variables in the distribution.

Let S be the root of the SPN below:



Validity

Definition 2 (Validity).

Let S be an SPN. If S correctly computes and marginalizes an unnormalized probability $\phi(\mathbf{X})$, then it is said to be *valid*.

If for every sum node n

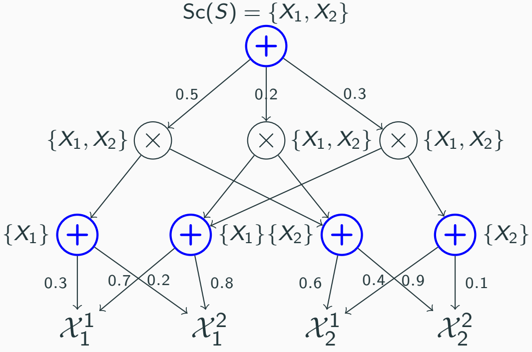
$$\forall j \in \mathrm{Ch}(n), w_{n,j} \geq 0 \text{ and } \sum_{j \in \mathrm{Ch}(n)} w_{n,j} = 1$$

then S represents the probability distribution itself.

A sufficient, yet not necessary, condition for validity is *completeness* and *consistency* (Poon and Domingos 2011).

Definition 3 (Completeness).

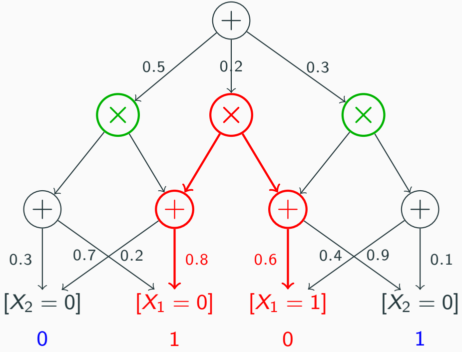
An SPN S is said to be complete, iff for each sum node $s \in S$, all children of s have same scope.



Consistency

Definition 4 (Consistency).

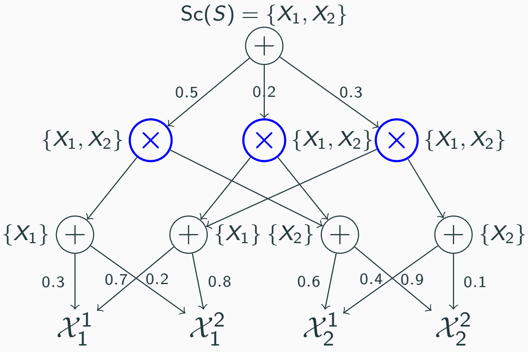
An SPN S is said to be consistent, iff no variable appears with a value v in one child of a product node, and valued u, with $u \neq v$, in another.



$$X = \{X_1 = 0, X_2 = 1\}$$

Definition 5 (Decomposability).

An SPN is decomposable iff no variable appears in more than one child of a product node (i.e. scopes are disjoint).



Decomposability vs Consistency

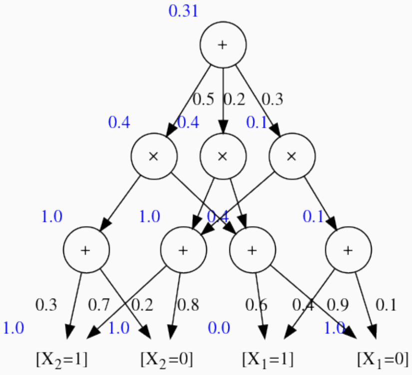
Decomposability implies **consistency**.

But **decomposability** is much easier for learning, and allows for an interpretation of product nodes as *independencies* between variables.

Robert Peharz et al. 2015 shows **decomposable** SPNs are as representable as solely **consistent** ones.

Probability of evidence

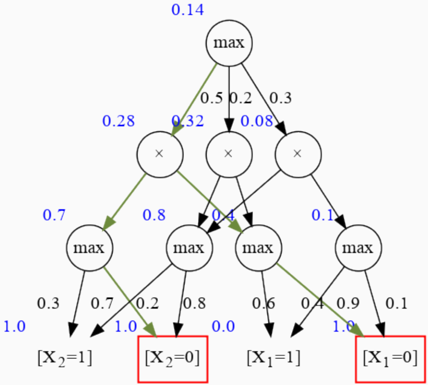
Single backward pass computes $S(X = \{X_1 = 0\}) = 0.31$. Linear on the number of edges.



Marginalizing variables means taking the mode of the corresponding distributions.

Maximum a posteriori probability

Replace sums with max nodes. Backward pass followed by forward pass computes **approximate** most probable explanation, i.e. find $M(E) = \arg\max_{x \in X} P(X = x, E)$.



Learning

Structure

- PD-Dense architecture (Poon and Domingos 2011)
- **Clustering on Variables** (Dennis and Ventura 2012)
- **Gens-Domingos LearnSPN** (Gens and Domingos 2013)
- Using deep learning techniques (R. Peharz et al. 2018)
- many others...

Weights

- **Generative and discriminative gradient descent**
- Generative Expectation-Maximization
- Extended Baum-Welch (Rashwan, Poupart, and Zhitang 2018)
- many others...

Dennis-Ventura architecture

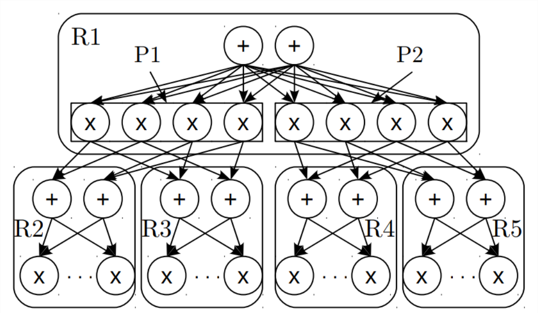
Idea:

- Sums are latent variables;
- Products are statistical independencies;
- A set of sums describes a *region*, and each sum is a potential semantical interpretation of its scope within a region;
- A set of products is a *partitioning* of regions.

In practice:

- Regions are clusters of similar pixels;
- Products partition regions into subregions;
- Use k-clustering for both tasks.

Dennis-Ventura architecture



Dennis and Ventura 2012

- 2-clustering on **instances** to create regions.
- 2-clustering on **variables** to partition regions into two subregions.

Dennis-Ventura architecture for classification

Let k be the number of classification labels (in our case $k = 3$).

Original algorithm:

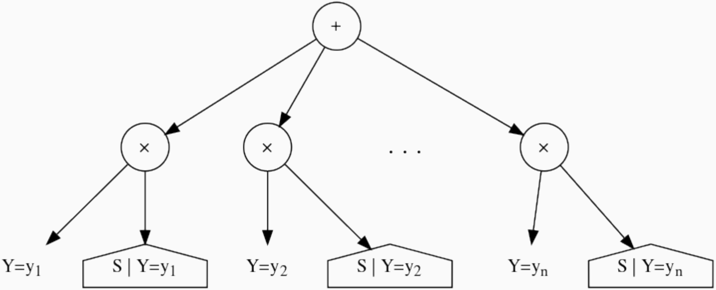
- Initial k-clustering to determine sub-SPNs for each label;
- Each sub-SPN should ideally model a single label;
- Does not scale when number of variables is high;
- Clustering might not guarantee sub-SPNs are split by classification labels.

Our version:

- Explicitly create sub-SPNs for each label;
- Each sub-SPN is trained with only data with assigned label;

Dennis-Ventura architecture for classification

Each product child of root acts as a switch. When the classification variable $Y = y$, all other sub-SPNs $S|Y = u$, $u \neq y$ are "switched off" and have value zero.



Result: much better performance compared to original.

Gens-Domingos architecture

Idea:

- Sums cluster similar instances/images;
- Products are independencies between variables;
- Recursively split dataset by instances or variables;
- Base case is a univariate distribution.

In practice:

- Clustering algorithm for sums (k-means, DBSCAN, ...);
- Statistical independence test for products (χ^2, G-test, ...);
- Multinomial or mixture of gaussians for leaves.

LearnSPN

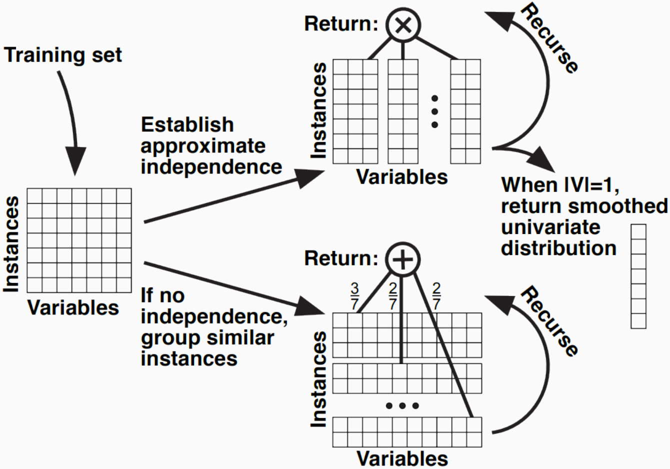
Algorithm 1 `LearnSPN`: Gens-Domingos structure learning schema

Input Set of instances D and scope X

Output SPN structure learned from D and X

```
if |X| = 1 then
    return univariate distribution over D_X
Partition X into P_1, P_2, ..., P_m st ∀i, j, i ≠ j, P_i ⊥ P_j
if m > 1 then
    return ∏_i LearnSPN(D, P_i)
Cluster D such that Q_1, Q_2, ..., Q_n are D's clusters
return ∑_i (|Q_i| / |D|) LearnSPN(Q_i, X)
```

Visualizing the Gens-Domingos architecture



Gens and Domingos 2013

Parameter learning

Inference	Weight updates
Soft	$\Delta w_{n,j} = \eta \dfrac{\partial S}{\partial w_{n,j}}(X, Y) - 2\lambda w_{n,j}$
Hard	$\Delta w_{n,j} = \eta \dfrac{c_{n,j}}{w_{n,j}} - 2\lambda w_{n,j}$

Generative gradient descent weight updates with L2 regularization.

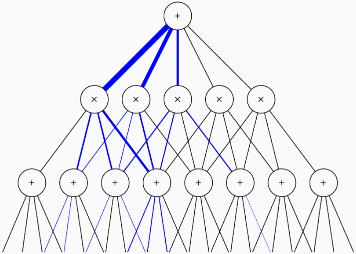
Inference	Weight updates
Soft	$\Delta w_{n,j} = \eta \left(\dfrac{1}{S(Y,X)} \dfrac{\partial S(Y,X)}{\partial w_{n,j}} - \dfrac{1}{S(X)} \dfrac{\partial S(X)}{\partial w_{n,j}} \right) - 2\lambda w_{n,j}$
Hard	$\Delta w_{n,j} = \eta \dfrac{\Delta c_{n,j}}{w_{n,j}} - 2\lambda w_{n,j}$

Discriminative gradient descent weight updates with L2 regularization.

Gradient diffusion

Soft gradient: the deeper the network, the faster the signal dwindles to zero.

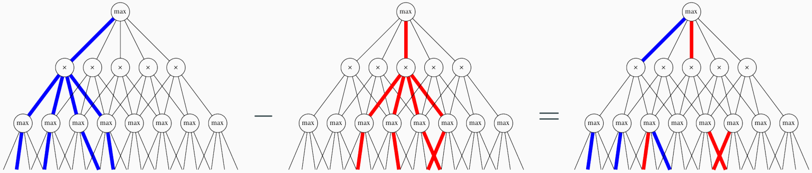
Hard gradient: derivatives are actually counts, so signal stays constant.



(a) Soft gradient **(b)** Hard gradient

Hard discriminative gradient

We want to optimize $P(Y|X)$, as ours is a classification task.
Using hard gradient helps with the gradient diffusion problem.



$\frac{\partial}{\partial W} \log M(Y, X)$ $\quad$ $\frac{\partial}{\partial W} \log M(X)$ $\quad$ $\nabla \log \tilde{P}(Y|X)$

Self-Driving

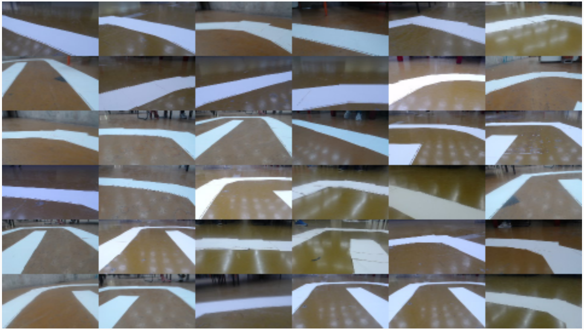
The task

Given a track, bot must be able to autonomously complete the whole course without going off road.

Inspired by Moraes and Salvatore 2018, which was itself inspired by Bojarski et al. 2016.

Dataset

Dataset used: Moraes and Salvatore 2018

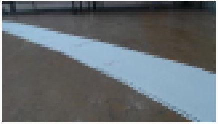


Lane tracking dataset with 80×45 RGB images. Each labeled with either UP, LEFT or RIGHT.

Self-driving as image classification

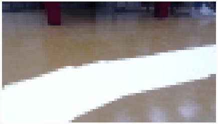
Let $X = \{X_0, X_1, \dots, X_{n-1}\}$ be an **image**. Every $X_i = x_i$ refers to the i-th pixel with a grayscale intensity of x_i.

Let $Y = \{\text{UP}, \text{LEFT}, \text{RIGHT}\}$ be the **classification variable**.



LEFT

UP



RIGHT

The entire scope of variables is $X \cup \{Y\}$.

Objective: $\arg\max_{y \in Y} P(Y = y|X)$

Pre-processing

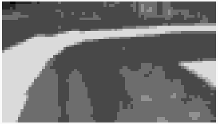
Pipeline:

original RGB image $\rightarrow$ grayscale $\rightarrow$ some T transformation.

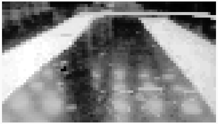
Three transformations tested:

1. Otsu binarization (Otsu 1979)
2. Quantization (resolution downscaling)
3. Histogram equalization

1



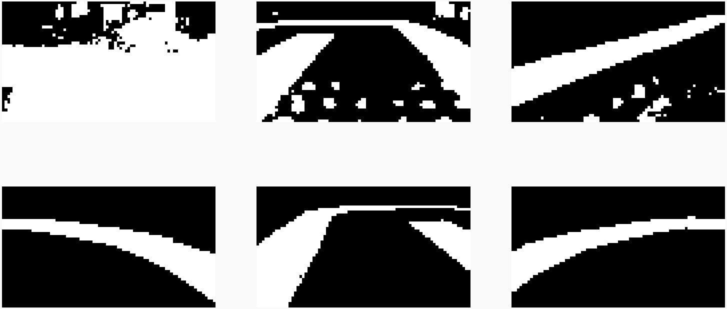
2



3

The problems with binarization

Hard threshold binarization produces lots of noise!



LEFT UP RIGHT

Otsu's works well, but takes too long!

Raspberry Pi 3 Model B — Berry

CPU: Quad Core 1.2GHz Broadcom BCM2837 64bit ARMv7

Memory: 1GB RAM

Storage: 16GB SSD

Brick

Lego Mindstorms NXT v2 — Brick

CPU: Atmel AT91SAM7S256 48MHz 32bit ARMv4

Memory: 64KB RAM

Storage: 256KB Flash

Robot

Berry handles inference, passing predicted label to **Brick**.

Brick handles motors according to label received from **Berry**.

Message passing through USB cable.

Evaluation approaches

Moraes and Salvatore 2018:

- Asynchronous;
- Actions are computed and passed to bot;
- Only moves when action is available, idles otherwise;
- Bot moves a *fixed* distance;
- *Accuracy* “matters more” than *inference speed*.

Our approach (real-time):

- Bot is always moving;
- Actions are computed in real-time;
- Action runs indefinitely until told otherwise;
- Bot movement depends on *prediction speed*;
- Balance between *accuracy* and *inference speed*;
- More “realistic”.

Driving with SPNs

Modelling

Every pixel X_i is a variable in the distribution represented by the SPN, i.e. no additional feature extraction, end-to-end.

Two architectures:

GD: LearnSPN with k-means and G-test

DV: Clustering on Variables

Three weight setups:

g: Generative gradient descent

d: Discriminative gradient descent

s: Proportional weights for GD, random weights for DV

Accuracy

Accuracy (%)	DV+g	DV+d	DV+s	GD+g	GD+d	GD+s
B	78.8	78.8	78.8	82.8	83.8	85.0
Q_2	78.6	78.0	78.0	78.6	80.4	79.4
$Q_2 + E$	76.6	76.6	76.8	79.6	82.8	81.8
Q_3	77.4	77.4	77.4	77.6	80.2	79.8
$Q_3 + E$	70.4	76.6	76.6	79.2	81.2	77.4
Q_4	78.2	78.4	78.2	76.0	**78.2**	76.4
$Q_4 + E$	76.6	76.6	76.8	76.0	74.6	80.6
Q_5	77.8	78.4	78.4	77.6	74.0	73.8
$Q_5 + E$	76.6	76.6	76.6	72.0	72.8	72.0
Q_6	77.4	78.4	78.4	75.2	**74.4**	72.0
$Q_6 + E$	76.0	76.4	76.4	73.0	75.0	73.6
Q_7	78.2	78.4	78.4	62.8	72.2	71.4
$Q_7 + E$	76.2	76.4	76.4	70.6	71.4	71.6
$\emptyset$	78.0	78.4	78.4	62.4	**62.4**	62.4
E	76.4	76.4	76.4	60.4	60.0	61.2

Inference time

Inference (secs)	DV+g	DV+d	DV+s	GD+g	GD+d	GD+s
B	0.23	0.25	0.25	0.38	0.37	0.31
Q_2	0.22	0.24	0.23	0.28	0.34	0.16
$Q_2 + E$	0.22	0.23	0.23	0.38	0.30	0.27
Q_3	0.22	0.23	0.22	0.22	0.32	0.17
$Q_3 + E$	0.22	0.23	0.22	0.34	0.32	0.31
Q_4	0.22	0.22	0.23	0.16	**0.17**	0.13
$Q_4 + E$	0.23	0.27	0.29	0.13	0.14	0.13
Q_5	0.22	0.26	0.28	0.07	0.05	0.02
$Q_5 + E$	0.22	0.29	0.25	0.05	0.05	0.02
Q_6	0.23	0.24	0.23	0.04	**0.05**	0.01
$Q_6 + E$	0.22	0.24	0.28	0.03	0.04	0.02
Q_7	0.23	0.23	0.26	0.03	0.01	0.01
$Q_7 + E$	0.22	0.26	0.24	0.01	0.01	0.01
$\emptyset$	0.22	0.26	0.23	0.02	**0.01**	0.01
E	0.23	0.23	0.22	0.01	0.01	0.02

Chosen models

Model 1: Q_4, **GD+d**

- **Accuracy:** 78.2%
- **Desktop time:** 170ms
- **Berry time:** 700ms

Model 2: Q_6, **GD+d**

- **Accuracy:** 74.4%
- **Desktop time:** 50ms
- **Berry time:** 150ms

Model 3: $\emptyset$, **GD+d**

- **Accuracy:** 62.4%
- **Desktop time:** $<$ 10ms
- **Berry time:** 75ms

The accuracy vs speed dilemma

Fundamental to find a balance between **accuracy** and **speed**.

$$
\begin{array}{lllll}
\uparrow \text{Network complexity} & \Rightarrow & \uparrow \text{Accuracy} & \Rightarrow & \downarrow \text{Inference speed} \\
\uparrow \text{Inference speed} & \Rightarrow & \downarrow \text{Accuracy} & \Rightarrow & \downarrow \text{Network complexity}
\end{array}
$$

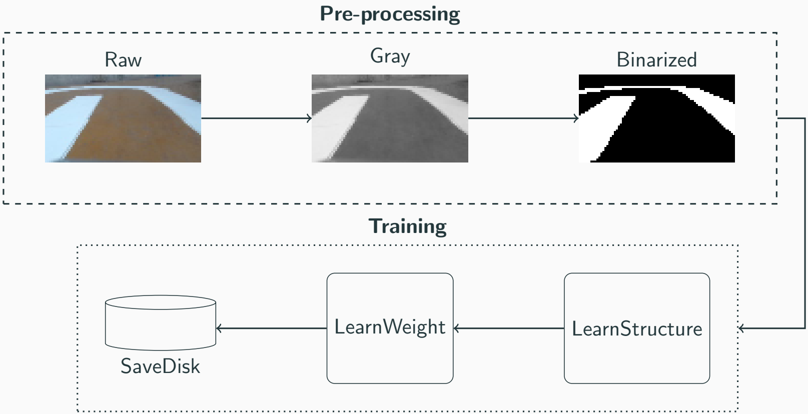
How much accuracy can we sacrifice for speed, and still have a reliable and safe model?

Some interesting data

- We used only 500 samples for training out of the total of 56172 images **(0.9% of the dataset)**.
- With 1000 samples we had much more accurate models, but inference takes much longer.
- Gens-Domingos LearnSPN with DBSCAN for clustering:
    - Only 500 training samples.
    - Network 32 times bigger than k-means variant.
    - Achieved **100% accuracy** on all tests!
    - Whopping **19.72 seconds** for inference on training computer.

Training

Training was done on an Intel i7-4500U CPU 1.80 Hz.



Saved SPN was then passed to the Raspberry.

Inference pipeline

Rasberry has to:

1. Capture raw camera data;
2. Convert data to grayscale;
3. Apply transformation to image;
4. Convert image into set of variable valuations, feeding SPN;
5. Compute probabilities for each label (LEFT, RIGHT, UP);
6. Send most probable label to Brick;
7. Record camera feed with probabilities overlay.

in **less than a second**!

Inference

Two possible options for computing $\arg\max_y P(Y = y|X)$:

1. Approximately:
   - Use MAP to compute most probable label in linear time.
   - This could be an option when the SPN is big.
2. Exactly:
   - Compute each $P(Y = y|X)$, $\forall y \in \mathrm{Val}(Y)$, get the max.
   - Since $|\mathrm{Val}(Y)|$ is small, feasible.

We chose to compute the exact probabilities.

Optimizations

The Raspberry has four cores. Let's make use of them!

CPU1: Capture and process image data.

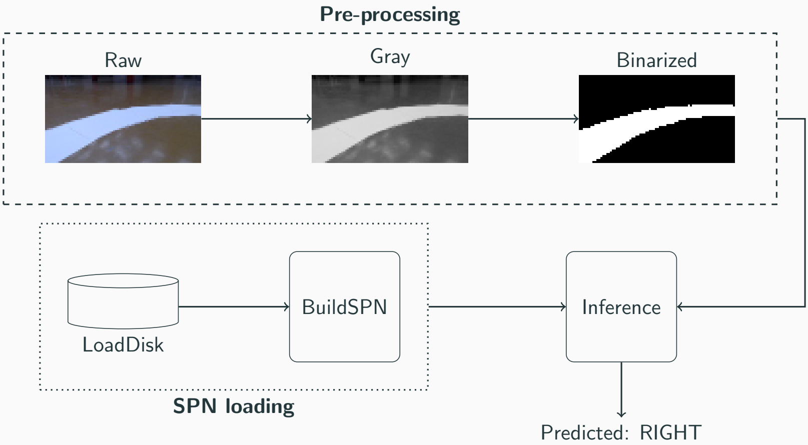
CPU2: Compute probability of label LEFT.

CPU3: Compute probability of label RIGHT.

CPU4: Compute probability of label UP.

Surprisingly, **CPU1** often takes the longest.

Prediction


Pre-processing
Raw
Gray
Binarized
BuildSPN
Inference
LoadDisk
SPN loading
Predicted: RIGHT

Comparison to neural networks

Comparison to Moraes and Salvatore 2018:

Model	Accuracy (%)	Speed (seconds)
DFN	81.3	$\approx$1.35
CNN	80.6	$\approx$1.35
Q_4, GD-SPN+d	78.2	$\approx$0.70
Q_6, GD-SPN+d	74.4	$\approx$0.15
GD-SPN+d	62.4	$\approx$0.07

Neural networks were more accurate, but real-time prediction with them is unfeasible.

Our implementation did not make use of the GPU, which could increase speed dramatically.

“Real world” scenario

Mobile Robot Self-Driving Through Image Classification Using Discriminative Learning of Sum-Product Networks — YouTube
(`https://youtu.be/vhpWQDX2cQU`)

Implementation

Inference and learning: GoSPN
(`https://github.com/RenatoGeh/gospn`)

Mobile robot implementation: GoDrive
(`https://github.com/RenatoGeh/godrive`)

Full thesis:
https://www.ime.usp.br/~renatolg/mac0499

Thank you.

Questions?

References i

- Bojarski, Mariusz et al. (Apr. 2016). "End to End Learning for Self-Driving Cars". In:
- Dennis, Aaron and Dan Ventura (2012). "Learning the Architecture of Sum-Product Networks Using Clustering on Variables". In: *Advances in Neural Information Processing Systems* 25.
- Gens, Robert and Pedro Domingos (2013). "Learning the Structure of Sum-Product Networks". In: *International Conference on Machine Learning* 30.
- Moraes, Paula and Felipe Salvatore (2018). *Self Driving Data*. URL: `https://github.com/felipessalvatore/self_driving_data`.

References ii

- Otsu, Nobuyuki (1979). "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66.
- Peharz, R. et al. (2018). "Probabilistic Deep Learning using Random Sum-Product Networks". In: *ArXiv e-prints*.
- Peharz, Robert et al. (2015). "On Theoretical Properties of Sum-Product Networks". In: *International Conference on Artificial Intelligence and Statistics 18 (AISTATS 2015)*.
- Poon, Hoifung and Pedro Domingos (2011). "Sum-Product Networks: A New Deep Architecture". In: *Uncertainty in Artificial Intelligence* 27.

Rashwan, Abdullah, Pascal Poupart, and Chen Zhitang (2018). "Discriminative Training of Sum-Product Networks by Extended Baum-Welch". In: *Proceedings of the Ninth International Conference on Probabilistic Graphical Models*. Vol. 72. Proceedings of Machine Learning Research, pp. 356–367.