Master's Thesis

Neurosymbolic AI for Social Cognition - Literature Study

Jarne Demoen
Vrije Universiteit Brussel

29/10/2025

1 Neurosymbolic AI for Social Cognition

This thesis aims to combine neural and symbolic AI to improve how computers understand the term "social cognition". Social cognition is the way humans interpret others' emotions, intentions, and social situations.

- **Symbolic AI** uses explicit rules such as "people are happy at weddings"
- **Subsymbolic AI (deep learning)** uses pattern recognition from raw data (facial expression in images)
- **Neurosymbolic AI** merges both terms such that neural networks can extract features while logic modules can reason over structured knowledge

The research should determine whether symbolic knowledge improves recognition or reduces the need for more data when aiming for better performance. Additionally, it would also be interesting to find out whether the integration of symbolic knowledge enhances interpretability. I will mainly be using the FindingEmo dataset.

2 A Circumplex Model of Affect — James A. Russell

Russell's paper presents a psychological model of human emotions called the *Circumplex Model of Affect*, which proposes that emotions can be organized within a two-dimensional continuous space. Affect is the broadest and most general word psychologists use for how people feel emotionally or even physically. The continuous space consists of two axes:

- **Valence (Pleasure–Displeasure axis)**: how positive or negative a feeling is.
- **Arousal (Activation–Deactivation axis)**: how energized or calm the feeling is.

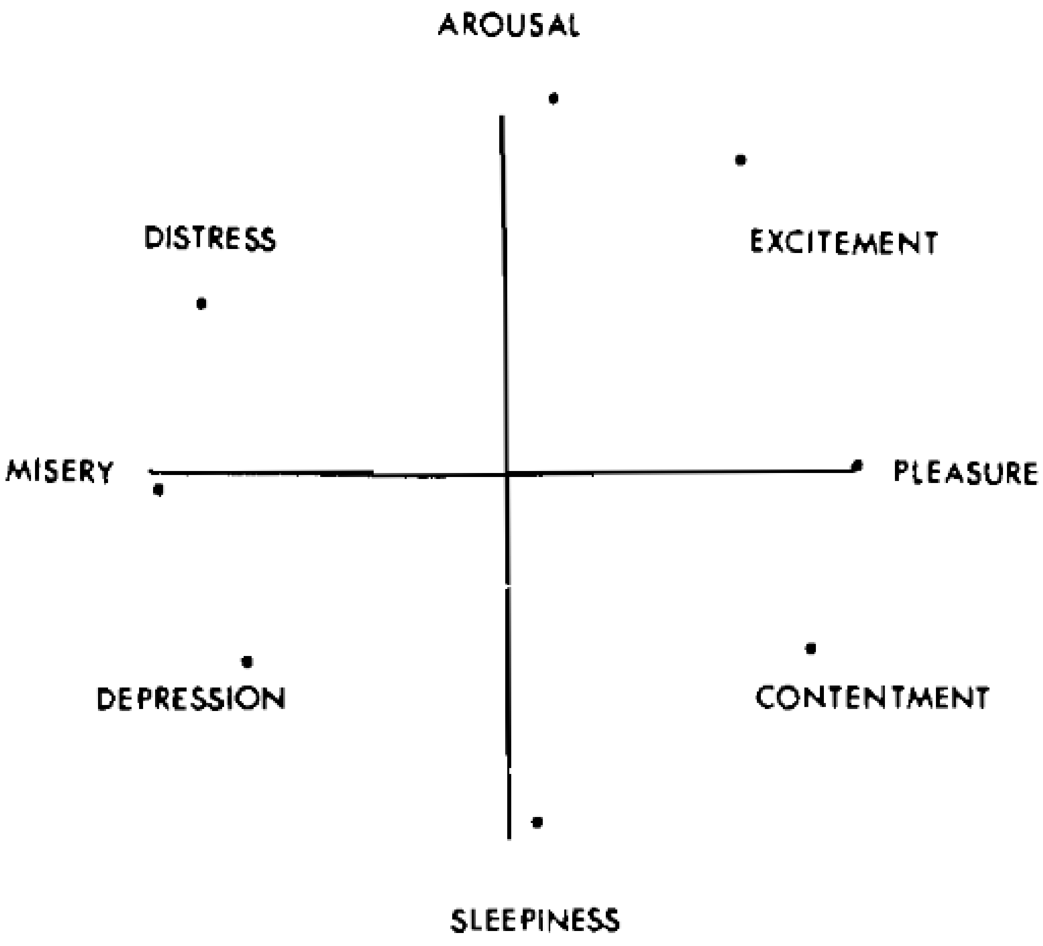
Together, these dimensions form a circular structure (*circumplex*) where emotions gradually blend into one another across valence and arousal, the same way as colors on a wheel. This circular arrangement provides a structured, interpretable framework for representing affect. In the context of neurosymbolic AI, it enables **symbolic encoding**

of affective knowledge (*anger* = unpleasant + high arousal) that can interact with neural perception modules, thus bridging psychological theory with AI reasoning.

In Schlosberg's early studies, participants categorized facial expressions of emotion. Errors in these categorizations revealed that similar emotions were frequently confused, indicating that they were **conceptually close**. From these confusion patterns, Schlosberg proposed a circular representation based on two bipolar dimensions:

- Pleasantness–Unpleasantness
- Attention–Rejection (or Activation–Deactivation)

Emotions, therefore, are better understood as positions along **continuous bipolar scales** rather than discrete, isolated categories. If one feels happy, one cannot simultaneously feel sad; if one feels tense, one cannot feel relaxed. Russell extended this reasoning, showing that any emotion word can be represented as a specific blend of valence and arousal, producing a full circle of affective experience.



Russell's experiment involved participants judging the relationships among the 28 emotion words. These served as the "points" later positioned within affective space using three techniques:

- **Ross' circular ordering:** Arranges variables along a circle, assuming circularity from the start.

- **Multidimensional scaling (MDS):** Derives spatial relationships purely from perceived similarity, without assuming circularity.
- **Unidimensional scaling:** Rates each term separately on pleasure–displeasure and arousal dimensions, producing two coordinates per emotion.

All three approaches yielded **remarkably consistent results**. Participants did not treat emotions as independent categories. Instead, emotion words such as *happy*, *excited*, and *content* had overlapping meanings, a property Russell called **fuzziness**. This fuzziness explains why affective terms distribute smoothly around a circle: emotions gradually blend into one another rather than having sharp boundaries. Participants' high consistency in word placement confirmed that people share a similar **mental map of emotions** structured around the valence and arousal dimensions.

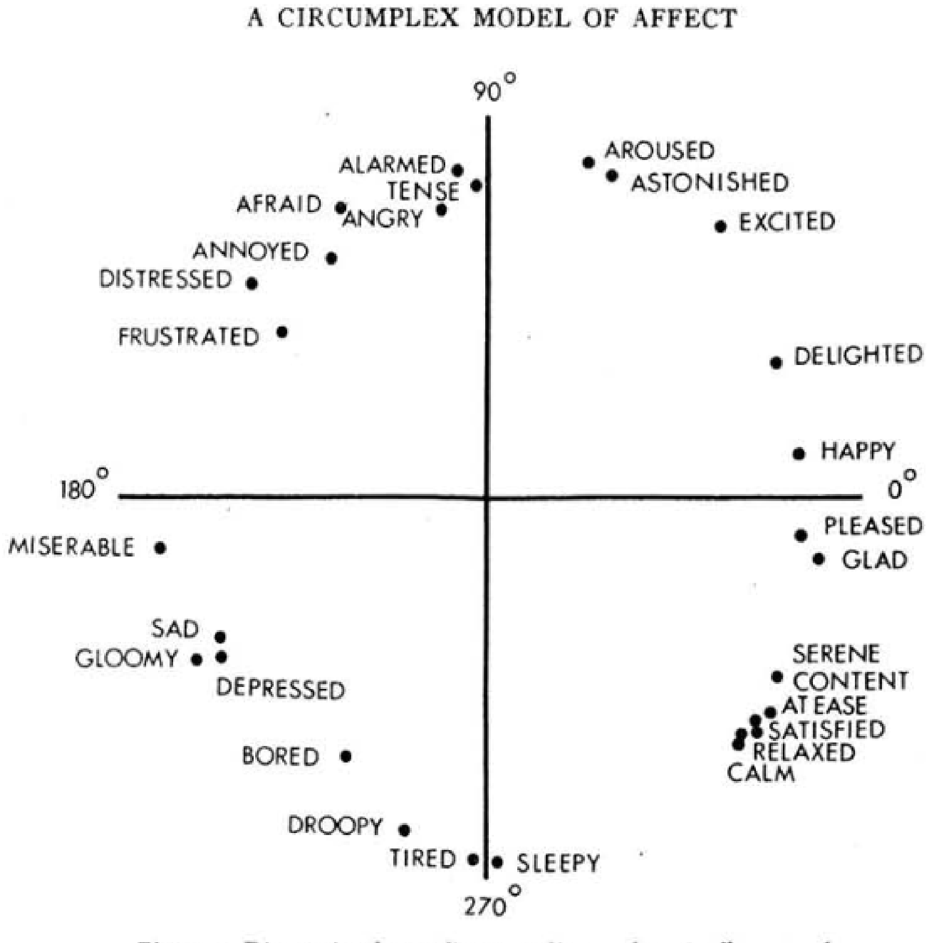


Figure 2. Direct circular scaling coordinates for 28 affect words.

Russell's results supported three major hypotheses about the cognitive structure of affect:

- **Bipolarity:** Opposite emotions (e.g., happiness–sadness) occur approximately 180° apart.
- **Two interpretable dimensions:** The horizontal and vertical axes correspond to pleasure–displeasure and arousal–sleepiness.
- **Continuity:** Rather than forming distinct clusters, emotion words spread continuously around the circle, creating a smooth emotional spectrum.

Earlier psychological models described emotions as 6–12 independent categories. Russell demonstrated instead that affective experience is **interconnected and continuous**, characterized by two bipolar dimensions that explain most of the variance in emotional life.

From feeling to cognition: Russell compared two types of emotion data: self-reports (how people feel) and judgment data (how people think about emotion terms). Traditionally, self-reports were believed to reveal genuine emotional experience, while judgment data were thought to capture only linguistic or semantic relationships. Yet, both data types yielded the same circular structure. This finding implies that **the way people experience emotions and the way they conceptualize them share the same underlying mental framework**.

Emotions are therefore not raw sensations; they are **interpreted experiences**. From a neurosymbolic perspective, this implies that the **symbolic layer of an AI system** can use this cognitive map as a structured knowledge base for affective reasoning. Neural networks may handle perception (detecting facial or scene context cues), while the symbolic layer interprets these signals within the valence–arousal framework, mirroring how humans conceptualize affect.

3 FindingEmo: An Image Dataset for Emotion Recognition in the Wild (Mertens et al., 2024)

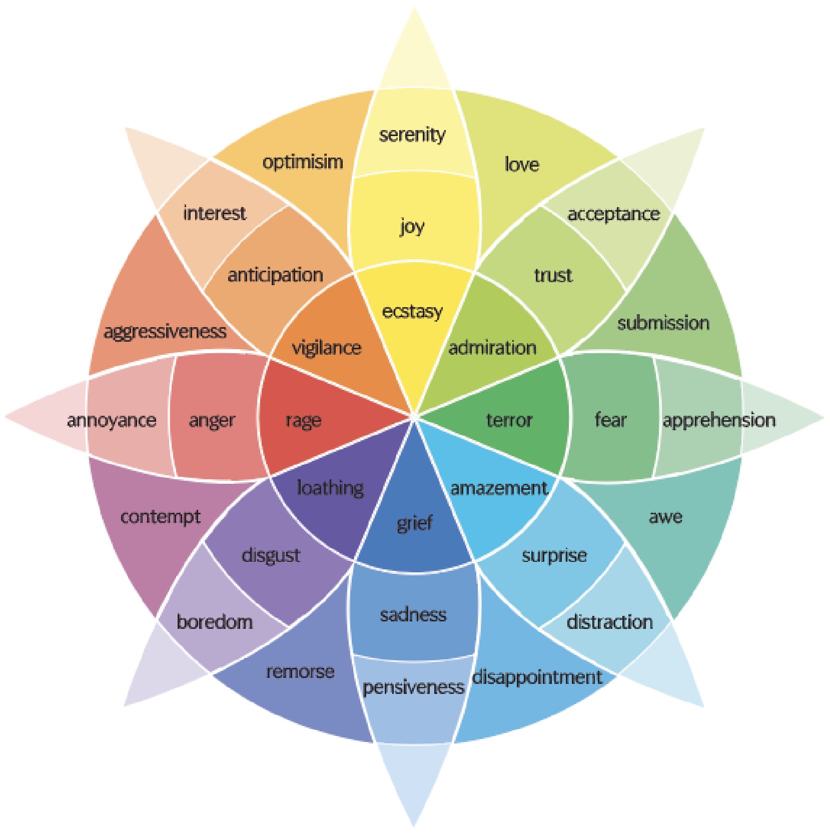
3.1 Overview and Motivation

The *FindingEmo* dataset introduces a new paradigm in emotion recognition by focusing on **complex social scenes** rather than isolated faces or individuals. Each image contains multiple people in naturalistic settings, such as weddings, protests, or family gatherings, and is annotated as a whole using three key dimensions: **Valence** (pleasantness), **Arousal** (intensity), and a discrete **Emotion** label.

Unlike traditional facial emotion datasets, which primarily identify emotions from close-up expressions, *FindingEmo* embraces the psychological insight that emotion understanding is inherently **contextual and relational**. A person's facial expression can be ambiguous when analyzed separately, but contextual cues such as objects, setting, body posture, and social interaction can clarify the overall meaning of the emotion. For instance, tears at a wedding imply joy rather than sadness. This makes *FindingEmo* particularly relevant for **social cognition research**, which studies how humans interpret emotions within social contexts.

3.2 Annotation Methodology

Each image is labeled with Valence and Arousal following **Russell's Circumplex Model of Affect**, with Valence in $[-3, 3]$ and Arousal in $[0, 6]$. The discrete emotion labels are based on **Plutchik's Wheel of Emotions (PWoE)**, a model that organizes human emotions into a circular structure. It contains 24 primary emotions grouped into eight families, each with three levels of intensity varying from mild to strong. Emotions that are psychological opposites, such as joy and sadness, are placed on opposite sides of the wheel. This structure makes it possible to label emotions at different levels of detail, using either the broader eight categories (Emo8) or all 24 specific emotions (Emo24).



The analysis of annotation reliability showed that people agreed more on **Valence** than on Arousal. This means that emotional intensity is more subjective; different people may perceive it differently, even in the same image. This finding reflects what emotion research has long suggested and offers useful guidance for building neurosymbolic models that can handle such uncertainty in human perception.

3.3 Dataset Properties and Challenges

The dataset exhibits **class imbalance**, with *joy* and *anticipation* overrepresented and *surprise* and *disgust* underrepresented, though the imbalance is milder than in prior datasets such as FER2013 or AffectNet. For neurosymbolic modeling, this imbalance is useful: symbolic reasoning components can integrate prior knowledge ("disgust" is rare but contextually specific) to improve robustness in low-frequency classes. For instance, disgust is likely in scenes involving unpleasant stimuli (spoiled food or injury), and fear often appears in threatening contexts (danger, darkness, aggressive postures). By integrating such rules into the reasoning process, the symbolic component could guide or adjust the neural predictions, improving recognition of these low-frequency emotions even when few training examples exist.

It is important to note that emotions like *surprise* and *anticipation* depend heavily on context. A model that relies only on visual cues, such as facial expressions or colors, might easily misinterpret whether these emotions are positive or negative. For example, being surprised at a birthday party is a joyful experience, while being surprised by bad news is not. A neurosymbolic model can consider this context to interpret the emotional meaning behind the scene correctly.

3.4 Baseline Model Performance

Baseline experiments applied transfer learning to several convolutional and transformer architectures, including AlexNet, VGG16, ResNet (18/50/101), DenseNet161, CLIP, and DINOv2. Each model's final layer was replaced and retrained on *FindingEmo* using the public dataset. Metrics included Weighted F1 and Average Precision (classification) and Mean Absolute Error and Spearman's ρ (regression).

Results indicate that **emotion recognition from social scenes is challenging**. Even state-of-the-art models like CLIP and DINOv2 achieved moderate performance. Among CNNs, **VGG16** emerged as the strongest baseline for both classification (Emo8) and regression (Valence, Arousal), followed closely by **ResNet50**. ImageNet-pretrained models outperformed Places365-pretrained ones, showing that object- and human-centric features are more relevant than scene semantics.

The authors observed that predicting **Valence** was easier and more consistent than predicting **Arousal**. This mirrors how people also tend to agree more on whether an emotion feels good or bad than on how strongly it is felt. When the models made mistakes, they often confused emotions that are close to each other on **Plutchik's Wheel**, such as mixing up joy with trust or anger with disgust (emotions that look and feel similar).

For this thesis, this pattern suggests that a symbolic reasoning layer could be designed to capture these adjacency relationships explicitly, helping the system distinguish between closely related emotions and make more consistent predictions.

Recommended training settings for future work include:

- **Loss functions:** UnbalancedCrossEntropyLoss (classification) and WeightedMSELoss (regression) to mitigate label imbalance.
- **Preprocessing:** Resize images to ≈800×600, preserve aspect ratio, and normalize with ImageNet statistics.
- **Implementation:** Fine-tune only the final layer of a pretrained VGG16 or ResNet50 model.

3.5 Beyond the Baseline: Multi-Stream Fusion

To explore whether combining information sources improves performance, the authors experimented with **late fusion**, concatenating features from multiple streams such as facial emotion predictions (OIToFER), EmoNet features, and Places365 features before passing them through a linear layer.

The results showed that performance improvements were **modest overall**. The only consistent gains came from incorporating **facial emotion features**, which means that the model was given additional input from a separate facial emotion recognition system. In practice, this involved detecting all visible faces in an image and analyzing each one with a pretrained facial emotion model (such as a ResNet trained on FER2013) to estimate expressions like happiness, anger, or sadness. These facial emotion predictions were then combined with the main model's scene-level features. This addition helped the model better understand subtle emotional cues in social scenes, confirming that faces still provide essential information about how people feel, even when emotions are influenced by context.

However, simply merging scene and facial features did not produce meaningful synergy. This finding is particularly relevant to neurosymbolic AI: it highlights that **true context understanding requires structured reasoning**, not just feature concatenation. Symbolic logic can instead integrate these perceptual cues into interpretable rules, such as:

$$\text{wedding dress} + \text{smiling crowd} \rightarrow \text{joy}.$$

The thesis implementation could use VGG16 as the main visual backbone, combined with a separate facial emotion recognition stream. Instead of simply merging multiple neural outputs, the system would rely on symbolic reasoning to interpret the broader scene context, using cues such as setting, actions, and relationships, to understand the emotional meaning behind the image.

3.6 Implications for Neurosymbolic AI

FindingEmo represents the first large-scale dataset explicitly designed for **social-level emotion recognition**. Its results demonstrate the difficulty of modeling collective emotions, even for powerful neural architectures. This supports the central motivation of the master's thesis: that **neurosymbolic AI**, combining deep perception with logic-based contextual reasoning, may be necessary to advance machine understanding of complex social emotions.

The *FindingEmo* dataset serves as an excellent foundation for developing and testing the neurosymbolic AI framework proposed in this thesis. In this setup, neural components such as VGG16 or ResNet50 act as perceptual modules that extract meaningful visual features from images, while the symbolic layer incorporates expert knowledge about social situations, roles, and typical emotional patterns. By combining these two perspectives, the system aims to go beyond simple pattern recognition and move toward a more human-like understanding of emotions, one that interprets feelings within their social and contextual setting, reflecting true **social cognition**.

4 DeepProbLog: Neural Probabilistic Logic Programming - Manhaeve et al

As far as we know, this is the first framework that truly brings together general-purpose neural networks and expressive probabilistic logic models, combining the strengths of both approaches into a single system that can be trained end-to-end from examples. Most recent approaches tried to make neural networks smarter by giving them reasoning abilities. They tried to rebuild logical reasoning inside a neural network so that everything can be trained like normal deep learning models. DeepProbLog flips that idea: it starts from an existing probabilistic logic programming language, ProbLog and extends it with the capability to process neural predicates.

The core idea behind DeepProbLog is to combine the strengths of probabilistic logic programming and neural networks within a single, trainable framework. In probabilistic logic programming, each statement (called an atom or predicate), for example, cat(image1), can have an associated probability expressing how likely it is to be true. DeepProbLog extends this concept by allowing some of these probabilities to come from neural networks instead of being predefined. This means that a neural network's output (for instance, a 0.9 probability that an image contains a cat) can be directly interpreted as a probabilistic fact within a logic program. Such facts are called neural predicates, logical statements whose truth probabilities are determined by neural models.

The key advantage of this design is that DeepProbLog retains all ProbLog's original logic-based reasoning capabilities, semantics, inference process, and implementation while integrating the perceptual power of neural networks. A key challenge in DeepProbLog is how to train the entire system end-to-end. The model takes raw inputs, such as images or text, that are processed by neural networks. Still, the supervision (the correct answer or loss signal) is only available at the final output of the logical reasoning process. In other words, the system must learn to perceive and reason, even though only the final result is evaluated.

To make this possible, DeepProbLog builds on an algebraic extension of ProbLog that supports automatic differentiation. This mathematical framework allows the model to trace how changes in the neural network outputs affect the overall reasoning outcome. As a result, the error signal (gradient) from the final prediction can be propagated backward through the logic program and into the neural networks. This makes it possible to train the entire hybrid model: logic and neural components together, using standard gradient descent, just like in regular deep learning systems.

4.0.1 Indirect Learning and Latent Representations in DeepProbLog

In *DeepProbLog*, the neural networks can learn *indirectly* through the logic-based reasoning process rather than from explicit labels. A clear example is the MNIST addition experiment, where the system is trained only on pairs of digit images and their total sum, for instance, an image of a 3 and an image of a 5 with the label "8." The model never sees the individual digit labels "3" or "5."

The logical part of the program defines the rule of addition:

```
addition(X, Y, Z) :- digit(X, DX), digit(Y, DY), Z is DX + DY.
```

When the system predicts the wrong sum, it calculates how far off it was (the loss). It uses that error to adjust both the probabilistic parameters and the neural network responsible for recognizing digits. Through this process, the neural network gradually learns to represent each image in a way that helps the logical rule "make sense" . In other words, to make the addition come out correctly.

This means that the network learns what each digit *must* look like for the reasoning to be correct, even though it is never told the answer directly. The digit-recognition network thus forms a **latent representation** of digits, an internal understanding of what distinguishes, for example, a 3 from an 8, purely because such knowledge is necessary for the logical rule to hold.

This form of *indirect* or *weak supervision* is one of the most powerful aspects of neurosymbolic systems such as DeepProbLog. It shows that logic can provide the structure and guidance for neural networks to learn meaningful concepts, even when no explicit training labels are available. In essence, the logic acts as the "teacher," defining the rules of the world, while the neural network learns how to perceive it in a way that makes those rules hold true.

4.0.2 Key Features of DeepProbLog

1. It is a **programming language** that directly supports neural networks and machine learning, while maintaining a well-defined formal semantics. As an extension of Prolog, it is *Turing equivalent*, meaning it has the full expressive power of a general-purpose programming language.

2. It **integrates logical reasoning with neural networks**, thereby unifying symbolic (logic-based) and subsymbolic (neural) representations and inference within a single model.

3. It **combines probabilistic modeling, programming, and reasoning with neural learning**. DeepProbLog extends the probabilistic logic programming language ProbLog, which itself can be viewed as a highly expressive directed graphical modeling language.

4. It can **learn a wide range of probabilistic logical neural models** directly from examples, including models for inductive programming and other forms of structured reasoning.

DeepProbLog is not restricted to fixed, predefined rules; it can also *induce* logical patterns directly from data. Instead of being explicitly told the reasoning structure, the system can infer which logical relationships best explain the examples it observes — a process known as *inductive logic programming.* For instance, given examples of family relationships, DeepProbLog could infer the general rule for a grandparent relationship (`grandparent(X,Z) :- parent(X,Y), parent(Y,Z)`). When combined with neural components for perception, this allows the framework to discover structured, rule-based reasoning patterns grounded in real-world data. Such an ability is especially valuable in social cognition, where the connections between visual cues, emotions, and context are often not explicitly stated but must be learned from examples.

4.1 Introducing DeepProbLog

4.1.1 Annotated Disjunctions (ADs)

An **annotated disjunction (AD)** in ProbLog provides a convenient way to model probabilistic choices among multiple, mutually exclusive outcomes. An AD has the general form:

$$p_1 :: h_1;\ p_2 :: h_2;\ \ldots;\ p_n :: h_n\ \ :-\ b_1, \ldots, b_m,$$

where each p_i is a probability such that $\sum_i p_i = 1$, and the h_i and b_j are logical atoms. The semantics of this construct are as follows: whenever all conditions $b_1, \ldots, b_m$ are true, exactly one of the outcomes h_i will hold, with probability p_i. All other outcomes are false, unless other parts of the program make them true.

```
0.4 :: earthquake(none); 0.4 :: earthquake(mild); 0.2 :: earthquake(severe).
```

This annotated disjunction expresses that exactly one of the three possible earthquake severities occurs in any given world, following the given probabilities. ProbLog programs with annotated disjunctions can always be transformed into equivalent programs without them; they are therefore considered *syntactic sugar* that improves readability and modeling convenience.

4.1.2 Neural Annotated Disjunctions (nADs) in DeepProbLog

A **DeepProbLog program** extends ProbLog by including **neural annotated disjunctions (nADs)**. An nAD connects the output of a neural network to a probabilistic logical predicate, allowing the neural predictions to participate in logical reasoning. Formally, an nAD has the following general form:

$$nn(m_q, \vec{t}, \vec{u}) :: q(\vec{t}, u_1); \ldots ; q(\vec{t}, u_n) : -b_1, \ldots, b_m,$$

where:

- m_q is the neural network that produces a probability distribution over the possible outcomes $u_1, \ldots, u_n$,
- $\vec{t} = (t_1, \ldots, t_k)$ is the vector of input terms to the predicate q,
- $b_1, \ldots, b_m$ are optional conditions (atoms) in the body of the rule.

From the perspective of ProbLog, this nAD behaves like a standard annotated disjunction: whenever all b_i hold, one of the outcomes $q(\vec{t}, u_i)$ becomes true with probability given by the neural network m_q. The output probabilities of the neural network are typically normalized by a softmax layer to ensure that $\sum_i p_i = 1$.

Example: MNIST digit classification. In the MNIST addition experiment, a neural network m_{digit} classifies handwritten digits from 0 to 9. The corresponding neural annotated disjunction is defined as:

```
nn(m_digit, Img, [0..9]) :: digit(Img, 0); ... ; digit(Img, 9).
```

Here, the predicate `digit(Img, D)` represents the belief that image `Img` contains digit `D`, and the probabilities of each `D` are provided by the neural network m_{digit}. DeepProbLog then uses these probabilistic outputs in combination with logical rules to perform reasoning tasks (e.g., addition) while maintaining full differentiability.

4.1.3 Proposed Handling of Neural Annotated Disjunctions in the Social Cognition Model

In the proposed neurosymbolic architecture, neural annotated disjunctions (nADs) serve as the interface between low-level perceptual modules and the symbolic reasoning layer. Each nAD corresponds to a neural network that outputs a probability distribution over a set of discrete categories, which is then integrated into the logical program as probabilistic facts. This design allows the model to combine the flexibility of deep learning for perception with the transparency and compositionality of logic-based reasoning.

Neural Perceptual Components. Two neural annotated disjunctions are defined: one for **scene classification** and one for **facial emotion recognition**. The scene network interprets the global social context of an image, while the face network predicts the emotional state of each detected individual. Formally, these are expressed as:

```
nn(scene_net, Img, [wedding, funeral, meeting]) ::
    scene(Img, wedding);
    scene(Img, funeral);
    scene(Img, meeting).

nn(emotion_net, Face, [happy, sad, neutral]) ::
    emotion(Face, happy);
    emotion(Face, sad);
    emotion(Face, neutral).
```

Each neural network produces a probability distribution through a softmax output layer, and these probabilities are interpreted by DeepProbLog as the annotated probabilities of the corresponding logical atoms. For example, if the emotion network predicts `[0.7, 0.2, 0.1]`, this is treated as probabilistic facts:

```
0.7::emotion(Face, happy);
0.2::emotion(Face, sad);
0.1::emotion(Face, neutral).
```

Symbolic Structure and Logical Reasoning. Each face is symbolically linked to its source image using the predicate `face_in(Face, Img)`. Logical rules then combine scene and emotion information to infer higher-level emotional states of the entire scene or social context. For instance:

```
positive_valence(Img) :-
    scene(Img, wedding),
    face_in(Face, Img),
    emotion(Face, happy).

negative_valence(Img) :-
    scene(Img, funeral),
    face_in(Face, Img),
    emotion(Face, sad).
```

This symbolic layer allows the model to reason about emotional coherence and to capture social conventions such as "people at weddings are typically happy" or "people at funerals are typically sad." Training is performed on higher-level predicates such as Training is performed on higher-level predicates such as `final_valence(Img, V)`, where `V in` {`positive, neutral, negative`}.

Gradients are propagated from the loss at this level through the reasoning chain into the neural predicates, enabling end-to-end training of the entire hybrid system.

Probabilistic Mixture of Sources. To model the relative contribution of different information sources (e.g., individual faces vs. scene context), a learnable probabilistic gate is introduced using annotated disjunctions with parameters $t(\cdot)$:

```
t(_)::use_source(face_0);
t(_)::use_source(face_1);
t(_)::use_source(face_2);
t(_)::use_source(scene).
```

These parameters are optimized during training, effectively allowing the system to learn how much to rely on facial or contextual information in different situations. The final emotion or valence of the scene is then derived through a mixture-of-experts formulation:

```
final_emo(Faces, Img, Emotion) :-
    use_source(face_0), face_emotion0(Faces, Emotion).
final_emo(Faces, Img, Emotion) :-
    use_source(face_1), face_emotion1(Faces, Emotion).
final_emo(Faces, Img, Emotion) :-
    use_source(face_2), face_emotion2(Faces, Emotion).
final_emo(Faces, Img, Emotion) :-
    use_source(scene),
    scene(Img, Context),
    context_to_emotion(Context, Emotion).
```

Proposed Extensions. To enhance robustness and realism, additional symbolic cues can be integrated:

- **Group emotion consistency:** enforce that most individuals in a single image share a similar valence, unless explicitly contradicted.
- **Contextual priors:** extend the `context_to_emotion/2` mapping to include probabilistic priors derived from empirical data or affective norms.
- **Relational cues:** introduce predicates for social relations (e.g., `interacting(F1, F2)`, `looking_at(F1, F2)`), allowing reasoning about mutual attention or empathy.

DeepProbLog Inference. Inference in DeepProbLog closely follows that of ProbLog, where the goal is to compute the probability of a query atom given the probabilistic facts and rules in the program. The process proceeds in four main steps: (i) the program is **grounded** with respect to the query, generating all relevant instances of probabilistic and logical clauses; (ii) the grounded program is then translated into a propositional formula over the truth values of the probabilistic facts; (iii) this formula is compiled into a **Sentential Decision Diagram (SDD)**, a compact data structure that supports efficient probabilistic inference; and (iv) the SDD is evaluated bottom-up to compute the probability of the query, using addition for logical disjunctions (OR-nodes) and multiplication for conjunctions (AND-nodes).

In DeepProbLog, inference proceeds identically, except that whenever a **neural predicate** is encountered during grounding, a forward pass through the corresponding neural network is performed to obtain probabilities for the ground atoms of the associated neural annotated disjunction (nAD). These probabilities are then used as the leaf values in the SDD, allowing seamless integration of neural predictions into logical inference.

4.2 Learning in DeepProbLog

Normally, a logic program is not differentiable. But DeepProbLog makes it differentiable by extending ProbLog into something called *Algebraic ProbLog (aProbLog)* In standard ProbLog, you define facts with probabilities. If you query something, it builds an SDD which is a logical circuit and evaluates it using addition for OR and multiplication for AND. To train a model we need to know how the output probability of the query changes if we change one of the input probabilities. So for example if an event has probability 0.7 instead of 0.6, how does the probability of the query change? This is basically the derivative of the query probability with respect to each fact's probability. This is where the **gradient semiring** comes in.

Algebraic ProbLog and the Gradient Semiring. To enable end-to-end learning, DeepProbLog extends ProbLog with the *gradient semiring* formalism introduced in Algebraic ProbLog (aProbLog). In standard ProbLog, each probabilistic fact is annotated with a probability p and inference is performed by evaluating the Sentential Decision Diagram (SDD) using probabilistic addition and multiplication for logical disjunctions and conjunctions. The gradient semiring generalizes this by associating each fact not only with its probability but also with its partial derivatives with respect to all learnable parameters. Hence, each element is represented as a pair $(p, \vec{g})$, where $\vec{g}$ stores the gradient of p with respect to the parameters. When the SDD is evaluated, both probabilities and gradients are propagated jointly using semiring addition and multiplication:

$$(a_1, \vec{a_1'}) \otimes (a_2, \vec{a_2'}) = (a_1 a_2, a_1 \vec{a_2'} + a_2 \vec{a_1'}), \qquad (a_1, \vec{a_1'}) \oplus (a_2, \vec{a_2'}) = (a_1 + a_2, \vec{a_1'} + \vec{a_2'}).$$

This is the chain rule in disguise! This allows DeepProbLog to compute both the success probability of a query and its gradient with respect to all learnable probabilistic parameters, thereby enabling differentiable probabilistic reasoning.

Gradient-Based Learning in DeepProbLog. DeepProbLog combines neural and probabilistic learning by computing gradients through the logical reasoning process. Each probabilistic fact $p :: f$ is extended with its gradient vector with respect to all learnable parameters, enabling differentiable inference using the gradient semiring. This allows DeepProbLog to propagate gradients from the loss on the final query back to both probabilistic and neural components.

To illustrate this process, the authors present a toy example involving two coins and an urn containing red and blue balls. The neural networks predict whether each coin shows heads or tails, while probabilistic facts describe the chance of drawing a red or blue ball. The rule for winning is defined as:

```
win :- heads.
win :- red.
```

The model is trained using only the observed outcome of `win` (win/loss), without direct supervision on the coin sides or ball color. During inference, DeepProbLog grounds the program and constructs a Sentential Decision Diagram (SDD), where each node carries both a probability and its derivative with respect to the parameters. For instance, the query probability may be $P(\texttt{win}) = 0.96$, with gradients of 0.4, 0.05, and 0.08 with respect to the probabilities of the first coin, second coin, and red ball respectively. These gradients indicate how sensitive the final outcome is to each component and are used to update the corresponding parameters through standard gradient-based optimization. This enables end-to-end training of neural networks and probabilistic logic parameters from supervision at the level of final logical outcomes.

4.3 DeepProbLog: Neural Probabilistic Logic Programming (Manhaeve et al.)

Overview. DeepProbLog integrates *probabilistic logic programming* (ProbLog) with *neural perception* in a single, end-to-end trainable framework. ProbLog attaches probabilities to facts and performs exact inference by grounding the program, translating it to a propositional formula, compiling it to an SDD, and evaluating it. DeepProbLog extends this by letting some probabilities come from neural networks (*neural predicates*), while keeping ProbLog's semantics and inference pipeline intact.

Core idea. If a neural model outputs a calibrated score that can be interpreted as a probability (e.g., an image contains a cat with probability 0.9), we can treat that output as a probabilistic fact inside the logic program, e.g., `cat(img1)` with weight 0.9. This preserves the logic's compositional reasoning while delegating perception to neural networks.

4.3.1 Indirect Learning and Latent Representations

DeepProbLog can train neural components *indirectly* via logical supervision. In the MNIST addition task, the program encodes

```
addition(X,Y,Z) :- digit(X,DX), digit(Y,DY), Z is DX + DY.
```

Training uses only pairs of images with their *sum* as a label; single-digit labels are never provided. The loss on the sum backpropagates through the reasoning steps into the neural predicate `digit/2`. Consequently, the network learns a *latent representation* of digits (what makes an image a 3 vs. an 8) because that knowledge is required for the rule to hold. This illustrates *weak supervision* via logical constraints.

4.3.2 Key Features

1. **Language with formal semantics.** DeepProbLog is a programming language that supports ML/NNs and inherits Prolog's expressivity (Turing-equivalent).
2. **Symbolic + subsymbolic integration.** Logical rules compose neural predictions; reasoning remains transparent and modular.
3. **Probabilistic modeling + learning.** It extends ProbLog (a highly expressive probabilistic logic language) with neural predicates.
4. **Learning rich models.** It supports end-to-end learning of probabilistic logical–neural models; in some settings, rules can be *induced* from data (inductive logic programming), useful when structure is implicit (just like in social contexts).

4.3.3 Annotated Disjunctions (ADs)

Definition

An **annotated disjunction (AD)** encodes a probabilistic choice among mutually exclusive outcomes:

$$p_1 :: h_1;\ \ldots;\ p_n :: h_n\ \ :-\ b_1, \ldots, b_m, \qquad \sum_i p_i = 1.$$

If the body $b_1, \ldots, b_m$ holds, exactly one head h_i is true with probability p_i.

Example.

```
0.4::earthquake(none); 0.4::earthquake(mild); 0.2::earthquake(severe).
```

ADs are syntactic sugar; they can be compiled to equivalent ProbLog without ADs.

4.3.4 Neural Annotated Disjunctions (nADs)

Definition

A **neural AD (nAD)** ties a neural network to a predicate by providing a probability distribution over its categorical outcomes:

$$nn(m_q, \vec{t}, \vec{u}) :: q(\vec{t}, u_1);\ \ldots;\ q(\vec{t}, u_n)\ \ :-\ b_1, \ldots, b_m.$$

The model m_q (typically with a softmax) yields the p_i used as the AD's head probabilities.

MNIST example.

```
nn(m_digit, Img, [0,1,2,3,4,5,6,7,8,9]) ::
    digit(Img,0); ... ; digit(Img,9).
```

4.3.5 Inference (Prediction Time)

- Ground the program w.r.t. the query, keeping only relevant clauses.
- Translate to a propositional formula over probabilistic facts.
- Compile to an SDD for efficient, exact inference.
- Evaluate bottom-up: OR-nodes use addition, AND-nodes use multiplication. For nADs, a forward pass computes leaf probabilities on demand.

4.4 Learning in DeepProbLog

Making logic differentiable. DeepProbLog uses *Algebraic ProbLog (aProbLog)* with the *gradient semiring*. Each literal carries a pair $(p, \vec{g})$: its probability and the gradient of that probability w.r.t. all learnable probabilistic parameters. Evaluation over the SDD propagates both values:

$$(a_1, \vec{g}_1) \otimes (a_2, \vec{g}_2) = (a_1 a_2,\ a_1\vec{g}_2 + a_2\vec{g}_1), \quad (a_1, \vec{g}_1) \oplus (a_2, \vec{g}_2) = (a_1 + a_2,\ \vec{g}_1 + \vec{g}_2).$$

Thus the root yields the query probability and its gradient; a standard loss (e.g., cross-entropy) drives gradient descent. For ADs, options are kept normalized (e.g., via softmax or renormalization).

4.5 Applying DeepProbLog to Social Cognition

Neural perception as nADs. We restrict neural components to two perceptual tasks:

```
nn(scene_net, Img, [wedding, funeral, meeting]) ::
    scene(Img,wedding); scene(Img,funeral); scene(Img,meeting).

nn(emotion_net, Face, [happy, sad, neutral]) ::
    emotion(Face,happy); emotion(Face,sad); emotion(Face,neutral).
```

Symbolic structure and reasoning. Faces are linked to the source image; rules capture social conventions and aggregate evidence:

```
face_in(Face, Img).

positive_valence(Img) :-
    scene(Img,wedding), face_in(Face,Img), emotion(Face,happy).

negative_valence(Img) :-
    scene(Img,funeral), face_in(Face,Img), emotion(Face,sad).
```

Training targets a high-level predicate such as `final_valence(Img,V)`, where `V` is one of {`positive, neutral, negative`}. Gradients flow from this loss through the reasoning chain into `scene_net` and `emotion_net`.

Mixture of sources (optional, learnable). To balance faces vs. context, introduce a probabilistic gate over sources:

```
t(_)::use_source(face_0); t(_)::use_source(face_1);
t(_)::use_source(face_2); t(_)::use_source(scene).

% Decompose list of faces if you pass them as [F0,F1,F2]
face_emotion0([F0,_,_],E) :- emotion(F0,E).
face_emotion1([_,F1,_],E) :- emotion(F1,E).
face_emotion2([_,_,F2],E) :- emotion(F2,E).

final_emo(Faces, Img, E) :- use_source(face_0), face_emotion0(Faces,E).
final_emo(Faces, Img, E) :- use_source(face_1), face_emotion1(Faces,E).
final_emo(Faces, Img, E) :- use_source(face_2), face_emotion2(Faces,E).
final_emo(_Faces, Img, E) :-
    use_source(scene), scene(Img,Ctx), context_to_emotion(Ctx,E).
```

Notes on modeling choices. ADs are ideal for mutually exclusive labels (scene type; per-face dominant emotion). For overlapping scene-level affect, use separate probabilistic predicates (e.g., `scene_is_formal/1`, `crowded/1`) that can co-occur, and reason about consistency symbolically.