Master's Thesis

Neurosymbolic AI for Social Cognition

Jarne Demoen
Vrije Universiteit Brussel

18/10/2025

1 Goal and Overview

We build a **Mixture-of-Experts (MoE)** model combining four emotion predictors:

- Three **face experts**: $e \in \{0, 1, 2\}$, each outputting 7 emotion logits $\boldsymbol{z}^{(e)} \in \mathbb{R}^7$,
- One **scene expert**: $\boldsymbol{z}^{(\text{scn})} \in \mathbb{R}^7$.

A learned **gating network** computes per-image weights $\boldsymbol{w}(\boldsymbol{x}) \in \Delta^3$ (the 4-dimensional simplex), and the experts are mixed on the probability level:

$$\boldsymbol{p}_{\text{neural}}(\boldsymbol{x}) = \sum_{e \in \{0,1,2,\text{scn}\}} w_e(\boldsymbol{x}) \underbrace{\text{softmax}\Big(\boldsymbol{z}^{(e)}(\boldsymbol{x})\Big)}_{\boldsymbol{p}^{(e)}(\boldsymbol{x})}.$$

This neural mixture is then combined symbolically with domain **prior knowledge** inside DeepProbLog:

$$p(E \mid \boldsymbol{x}, \text{Meta}) \;\propto\; p_{\text{neural}}(E \mid \boldsymbol{x}) \;\cdot\; p_{\text{prior}}(E \mid \text{Meta}),$$

which is exactly a *product-of-experts*. The final predicate `final_emo/4` corresponds to this product and is the one used for training and inference.

2 Notation and Shapes

For a mini-batch:

$$B = \text{batch size}, \quad C = 7 \text{ emotion classes}, \quad F = 3 \text{ face slots}.$$
$$\boldsymbol{z}_{\text{face}} \in \mathbb{R}^{B \times F \times C}, \quad \boldsymbol{z}_{\text{scn}} \in \mathbb{R}^{B \times C}.$$
$$\boldsymbol{p}_{\text{face}} = \text{softmax}(\boldsymbol{z}_{\text{face}}, -1) \in \mathbb{R}^{B \times F \times C}, \quad \boldsymbol{p}_{\text{scn}} = \text{softmax}(\boldsymbol{z}_{\text{scn}}, -1) \in \mathbb{R}^{B \times C}.$$

Gating weights: $\boldsymbol{w} \in \mathbb{R}^{B \times 4}$ with $\sum_e w_{b,e} = 1$ and $w_{b,e} \geq 0$.

3 Gating Inputs (Features)

The gate should learn which expert is reliable for each image. We feed it compact indicators of confidence and context.

Per expert (face or scene)

$$m^{(e)} = \max_c \; p_c^{(e)} \quad \text{(maximum probability, confidence)}$$
$$h^{(e)} = -\sum_c p_c^{(e)} \log p_c^{(e)} \quad \text{(entropy, uncertainty)}$$

Additional contextual signals

- `face_present`[e] $\in \{0, 1\}$ – whether face slot e exists,
- `num_faces` $\in \{0, 1, 2, 3\}$,
- Optionally, detection confidence, blur level, or similar.

The concatenated gate input per image may thus include:

$$\text{gate\_in} = [\boldsymbol{z}_{\text{face}} \text{ (flattened)}, \; \boldsymbol{z}_{\text{scn}}, \; \boldsymbol{m}, \; \boldsymbol{h}, \; \texttt{face\_present}, \; \texttt{num\_faces}],$$

but you can begin with only $(\boldsymbol{m}, \boldsymbol{h}, \text{flags})$ for simplicity.

4 Gating Network and Probability Mixture

The gate is a simple MLP producing 4 weights:

$$\boldsymbol{w}(\boldsymbol{x}) = \text{softmax}\Big(f_\theta(\text{gate\_features}(\boldsymbol{x}))\Big) \in \mathbb{R}^4.$$

The neural mixture:

$$\boldsymbol{p}_{\text{neural}}(\boldsymbol{x}) = \sum_{e \in \{0,1,2,\text{scn}\}} w_e(\boldsymbol{x}) \, \boldsymbol{p}^{(e)}(\boldsymbol{x}), \qquad \sum_e w_e(\boldsymbol{x}) = 1.$$

```
# probabilities per expert
p_face  = torch.softmax(z_face, dim=-1)    # (B,3,7)
p_scene = torch.softmax(z_scene, dim=-1)   # (B,7)

# gate features
max_face   = p_face.max(dim=-1).values
ent_face   = -(p_face * (p_face.clamp_min(1e-12)).log()).sum(dim=-1)
max_scene  = p_scene.max(dim=-1).values
ent_scene  = -(p_scene * (p_scene.clamp_min(1e-12)).log()).sum(dim=-1)

feat_list = [max_face, ent_face,
             max_scene.unsqueeze(-1), ent_scene.unsqueeze(-1),
             face_present.float(), num_faces]
gate_in = torch.cat([t.reshape(t.size(0), -1) for t in feat_list], dim=1)
# (B,D)

# MLP -> weights
w = torch.softmax(gate_mlp(gate_in), dim=-1)  # (B,4)
```

```
# probability mixture
p_mix = (
   (w[:,0].unsqueeze(-1).unsqueeze(-1) * p_face[:,0]) +
   (w[:,1].unsqueeze(-1).unsqueeze(-1) * p_face[:,1]) +
   (w[:,2].unsqueeze(-1).unsqueeze(-1) * p_face[:,2]) +
   (w[:,3].unsqueeze(-1)              * p_scene)
)
```

5 Symbolic Priors in DeepProbLog

Implementation sketch (PyTorch). Symbolic priors encode domain knowledge $p_{\text{prior}}(E \mid \text{Meta})$. In ProbLog, conjunction means multiplication of probabilities, so combining a neural predicate with a prior predicate yields a product-of-experts model.

Formally:

$$p(E \mid \boldsymbol{x}, \text{Meta}) \propto p_{\text{neural}}(E \mid \boldsymbol{x}) \cdot p_{\text{prior}}(E \mid \text{Meta}).$$

```
% Neural mixture (linked to your PyTorch MoE)
nn(moe_net, [Faces, Scene], E, [0,1,2,3,4,5,6]) :: neural_emo(Faces, Scene,

% Metadata facts (per example):
% num_faces(MetaId, N).    scene_tag(MetaId, Tag).
% Example:
%   num_faces(meta123, 0).
%   scene_tag(meta123, party).

% Soft baseline prior (every class gets some mass)
0.20::prior_emo(_, 0). 0.20::prior_emo(_, 1). 0.20::prior_emo(_, 2).
0.20::prior_emo(_, 3). 0.20::prior_emo(_, 4). 0.20::prior_emo(_, 5).
0.20::prior_emo(_, 6).

% Rules:
0.10::prior_emo(M, 0) :- num_faces(M, 0).  % damp anger if no faces
0.10::prior_emo(M, 1) :- num_faces(M, 0).  % damp disgust
0.10::prior_emo(M, 5) :- num_faces(M, 0).  % damp surprise

0.60::prior_emo(M, 3) :- scene_tag(M, party). % boost happy
0.15::prior_emo(M, 4) :- scene_tag(M, party). % damp sad

% Combine neural and prior via conjunction (product of probs):
final_emo(Faces, Scene, Meta, E) :-
    neural_emo(Faces, Scene, E),
    prior_emo(Meta, E).
```

Training: Train and query `final_emo/4`, not `neural_emo/3`.

6 Training Objective

DeepProbLog optimizes the negative log-likelihood of the final query:

$$\mathcal{L} = -\log p\Big(E^\star \,\Big|\, \boldsymbol{x}, \mathrm{Meta}\Big),$$

where p is the probability returned by the probabilistic reasoning engine for `final_emo`.

Implementation steps.

1. Build the PyTorch MoE (experts $\to$ logits, gate $\to \boldsymbol{w}$, mix $\to \boldsymbol{p}_{\mathrm{neural}}$).
2. Link `moe_net` to DeepProbLog via `nn(...)` predicate.
3. Add metadata facts: `num_faces/2`, `scene_tag/2`, etc.
4. Define symbolic `prior_emo/2`.
5. Train using `final_emo(Faces, Scene, Meta, E_gold)` queries.

7 Worked Example

Suppose (for one image):

- $\boldsymbol{p}^{(0)} = [.30, .05, .05, .30, .10, .10, .10]$,
- $\boldsymbol{p}^{(1)} = [.25, .05, .05, .35, .10, .10, .10]$,
- $\boldsymbol{p}^{(2)} = [.10, .10, .10, .50, .05, .10, .05]$,
- $\boldsymbol{p}^{(\mathrm{scn})} = [.05, .05, .10, .45, .15, .10, .10]$,
- $\boldsymbol{w} = [0.2, 0.2, 0.2, 0.4]$.

Then

$$\boldsymbol{p}_{\mathrm{neural}} = 0.2\boldsymbol{p}^{(0)} + 0.2\boldsymbol{p}^{(1)} + 0.2\boldsymbol{p}^{(2)} + 0.4\boldsymbol{p}^{(\mathrm{scn})}.$$

If metadata indicates `num_faces=0` and `scene_tag=party`, the symbolic priors amplify `happy` and damp `sad/anger/disgust/surprise`. The final DeepProbLog inference multiplies and renormalizes, shifting probability mass towards `happy` in an explainable way.

8 Practical Stability Tips

- Clamp in logs: use `clamp_min(1e-12)` for numerical stability.
- Normalize gating inputs: entropies $\in [0, \log C]$.
- If a face slot is empty: `face_present=0`, feed a flat or neutral distribution.
- Use small gate MLP (1–2 layers, 64 hidden) with weight decay to avoid overfitting.

9 Interface Summary

- **Neural side:** `nn(moe_net,[Faces,Scene],E,[0..6])` returns $\boldsymbol{p}_{\text{neural}}$.
- **Data:**
  - `Faces`: tensor source with 3 face crops or features.
  - `Scene`: scene image tensor.
  - `Meta`: symbolic facts per example.

10 Implement Now vs Later

Implement Now (core system)

1. Experts → logits → `softmax` → per-expert $\boldsymbol{p}^{(e)}$.
2. Gate MLP with **max-prob**, **entropy**, **face_present**, **num_faces**.
3. Probability mixture $\boldsymbol{p}_{\text{neural}} = \sum_e w_e \boldsymbol{p}^{(e)}$.
4. Symbolic priors (`prior_emo`) and training on `final_emo`.

Add Later (extensions)

- **Temperature calibration** per expert: learn τ_e so that $\boldsymbol{p}^{(e)} = \text{softmax}(\boldsymbol{z}^{(e)}/\tau_e)$.
- **Agreement features:** Jensen-Shannon or KL divergence between experts.
- **Logit mixture (log-sum-exp):**

$$\boldsymbol{z}_{\text{mix}} = \log\Big(\sum_e w_e\, e^{\boldsymbol{z}^{(e)}}\Big), \quad \boldsymbol{p}_{\text{neural}} = \text{softmax}(\boldsymbol{z}_{\text{mix}}).$$

- Richer priors (object tags, time, context) and symbolic features for the gate.

11 Checklist

1. Gate outputs 4 weights per sample (softmax-normalized).
2. Shape consistency: $(B, 3, 7)$, $(B, 7) \rightarrow (B, 7)$.
3. Priors are soft (no hard zeros).
4. Train and query `final_emo/4`.
5. Log which symbolic rules fired for explainability.