TATN-Tri-Modal Adversarial Transparency Network (TATN)

---

1 Introduction

TATN(Tri-modal Adversarial Transparency Network) combines:

1. **DSCD** — Dynamic Span–Sense Co–Detection: an online prototype-based module that jointly detects ambiguous spans (homographs) and predicts sense cluster IDs while computing multi-source uncertainty; and
2. **ASBN** — Adversarial Sense Balance Network: training-time discriminators that detect shortcut behaviors (frequency bias, context-sparsity guessing, and cross-lingual mismatch) and push the encoder to remove them using a Gradient Reversal Layer (GRL).

The following sections present the algorithmic details in an Overleaf-friendly format: short explanations followed by compact pseudocode suitable for inclusion in a paper.

2 Notation (quick reference)

S Source sentence (subwords) $S = [s_1, \ldots, s_L]$.

w_j Token type (subword string) at position j.

$h_j \in \mathbb{R}^d$ Contextual encoder vector for token j.

$C_w = \{c_{w,1}, \ldots, c_{w,K_w}\}$
Dynamic prototype set for type w.

p_j Soft distribution over prototypes for token j.

$\hat{y}_j$ Selected prototype / sense id (p_j).

$p_{\max}(j)$ Confidence $= \max_k p_{j,k}$.

U_j Combined uncertainty score (entropy + MC var + learned σ + novelty).

g_j Uncertainty gate (attention boost scalar in $(0, 1)$).

$z^{\mathbf{tgt}}_{a(j)}$ Aligned target embedding (parallel corpora).

$D_{\mathbf{freq}}, D_{\mathbf{ctx}}, D_{\mathbf{xl}}$ ASBN discriminators.

$\bar{\lambda}_k$ base GRL strength for discriminator k.

3 High-level pipeline (what/why)

1. **DSCD forward pass (inference training):** encode → compute multi-source uncertainty → dynamic prototype assignment/create/update → attention gating → joint span detection + sense assignment → sense-augmented token vectors $\{h'_j\}$ → decoder.

2. **ASBN (training only):** three lightweight discriminators monitor DSCD outputs and send reversed gradients (via GRL) to the encoder to eliminate shortcut signals. ASBN uses per-sample confidence weighting to avoid penalizing genuine uncertainty.

3. **TRG (optional):** rationale generator that consumes DSCD+ASBN diagnostics to create human-readable explanations for decisions.

4 Core DSCD forward-pass (compact pseudocode)

Below is a compact algorithm for the DSCD forward pass. The comments explain "what" is computed and "why."

Algorithm 1 DSCD forward pass (per sentence)

Require: Source sentence $S = [s_1, \ldots, s_L]$

for $j = 1$ to L **do** ▷ Encode and buffer
  $h_j \leftarrow \text{Encoder}(s_j)$ ▷ contextual embedding
  append h_j to buffer B_{w_j} ▷ maintain recent embeddings per type
end for
Compute per-type dispersion D_w using buffers B_w ▷ detect multi-sense candidates
for $j = 1$ to L **do** ▷ Per-token uncertainty and prototype logic
  if $C_{w_j} = \emptyset$ **then**
    create initial prototype (immediately or after $N_{\min}$ buffered examples)
  else
    compute cosine similarities $s_{j,i} = \cos(h_j, c_{w_j,i})$ for $i = 1..K_{w_j}$
    $p_j \leftarrow \text{softmax}(s_{j,\cdot}/T)$ ▷ soft assignment to prototypes
    $H_j \leftarrow -\sum_i p_{j,i} \log(p_{j,i} + \varepsilon)$ ▷ aleatoric entropy
    compute MC-dropout variance Var_j (run M stochastic passes)
    compute learned noise σ_j from a small σ-Net
    $d_{\min} \leftarrow \min_i(1 - s_{j,i})$ ▷ novelty to nearest centroid
    $U_j \leftarrow \alpha_1 H_j + \alpha_2 \sigma_j + \alpha_3 \text{Var}_j + \alpha_4 \text{norm}(d_{\min})$
  end if
  Compute adaptive threshold $\varepsilon^{new}_{w_j}$ from rolling mean/std of assignment distances
  if $d_{\min} > \varepsilon^{new}_{w_j}$ **and** $U_j > \delta_{\text{inst}}$ **then**
    create new prototype $c_{w_j,K+1} \leftarrow h_j$
  else
    assign token to nearest prototype $i^\star \leftarrow_i s_{j,i}$
    update centroid $c_{w_j,i^\star} \leftarrow (1-\eta)c_{w_j,i^\star} + \eta\, h_j$
  end if
end for
For flagged tokens (types with $D_w > \delta_{\text{type}}$ and $U_j > \delta_{\text{inst}}$) compute gate $g_j = \sigma(w_g(U_j - b_g))$
Boost base attention $a_j^{(0)} \leftarrow a_j^{(0)}(1 + \gamma g_j)$ and renormalize to $\tilde{a}_j$
Span prediction $\hat{b}_j \leftarrow \sigma(W_{\text{span}} h_j + b_{\text{span}})$ (binary)
Sense id $\hat{y}_j \leftarrow_i p_{j,i}$ (or clustering assignment)
Form sense-augmented vector:

$$h'_j \leftarrow \begin{cases} h_j + c_{w_j,\hat{y}_j} & \text{if } \hat{b}_j > 0.5 \\ h_j & \text{otherwise} \end{cases}$$

Send $\{h'_j\}, \{\tilde{a}_j\}$ to decoder

Explanation (short):

- The forward pass computes robust, multi-source uncertainty signals so the system knows which tokens are ambiguous.
- Prototypes are created/updated online using adaptive (per-type) thresholds, and EMA updates keep centroids stable.
- Flagged tokens receive larger attention via a learnable gate so the decoder sees more context about ambiguous positions.

5 ASBN — training-time adversarial module (plain description)

ASBN consists of three lightweight discriminators:

$D_{\mathbf{freq}}$ Frequency shortcut detector: checks whether the model's sense choice equals the most frequent sense for the token type.

$D_{\mathbf{ctx}}$ Context-sparsity detector: checks whether the encoder is ignoring weak contexts and defaulting to priors.

$D_{\mathbf{xl}}$ Cross-lingual detector: checks whether the source predicted sense aligns with the target token embedding (uses negatives created from wrong-sense prototypes).

Each discriminator receives the encoder representation *through a GRL*. The GRL forwards identity in the forward pass and multiplies gradients by $-\lambda$ in the backward pass, causing the encoder to learn representations that *hide* the shortcuts the discriminators exploit.

Per-sample GRL strengths are computed to avoid punishing honest uncertainty:

$$\lambda_{k,j} = \mathrm{clip}\big(\bar{\lambda}_k \cdot p_{\max}(j) \cdot (1 - U_j) \cdot g_j,\ 0,\ \lambda_{\max}\big)$$

where $k \in \{\mathrm{freq}, \mathrm{ctx}, \mathrm{xl}\}$.

6 ASBN pseudocode and integrated training step

The pseudocode below shows a single training-step (per batch) that integrates DSCD forward-pass (Algorithm 1) with ASBN discriminator updates and GRL-based adversarial training.

Algorithm 2 TATN v2 training step (per batch)

Require: batch of parallel sentences (source, target)

Run DSCD forward for each source sentence $\Rightarrow$ obtain $\{h_j, p_j, \hat{y}_j, p_{\max}(j), U_j, g_j\}$ and update prototypes (Alg. 1)
Compute decoder loss $\mathcal{L}_{\mathrm{MT}}$ (translation CE), span loss $\mathcal{L}_{\mathrm{span}}$ and sense loss $\mathcal{L}_{\mathrm{sense}}$
for each token j in batch **do**
    Compute per-discriminator GRL weights:

$$\lambda_{k,j} \leftarrow \mathrm{clip}\big(\bar{\lambda}_k \cdot p_{\max}(j) \cdot (1 - U_j) \cdot g_j,\ 0, \lambda_{\max}\big)$$

end for
$\triangleright$ — Discriminator forward passes (via per-sample GRL) —
$x_j^{\mathrm{freq}} \leftarrow \mathrm{GRL}(h_j; \lambda_{\mathrm{freq},j}) \oplus F_{w_j}$ and compute $\ell_j^{\mathrm{freq}} \leftarrow D_{\mathrm{freq}}(x_j^{\mathrm{freq}})$
$x_j^{\mathrm{ctx}} \leftarrow \mathrm{GRL}(h_j; \lambda_{\mathrm{ctx},j}) \oplus \mathrm{ctx\_stats}_j$ and compute $\ell_j^{\mathrm{ctx}} \leftarrow D_{\mathrm{ctx}}(x_j^{\mathrm{ctx}})$
$x_j^{\mathrm{xl}} \leftarrow \mathrm{GRL}(h_j; \lambda_{\mathrm{xl},j}) \oplus \phi_{\mathrm{tgt}}(z_{a(j)}^{\mathrm{tgt}})$ and compute contrastive loss $\ell_j^{\mathrm{xl}} \leftarrow D_{\mathrm{xl}}(x_j^{\mathrm{xl}})$
Aggregate discriminator losses:

$$\mathcal{L}_{\mathrm{ASBN}} \leftarrow \frac{1}{N} \sum_j \big(w_{\mathrm{freq}} \ell_j^{\mathrm{freq}} + w_{\mathrm{ctx}} \ell_j^{\mathrm{ctx}} + w_{\mathrm{xl}} \ell_j^{\mathrm{xl}}\big)$$

$\triangleright$ Update discriminator parameters ϕ (minimize $\mathcal{L}_{\mathrm{ASBN}}$)
`opt_phi.zero_grad();   L_ASBN.backward(retain_graph=True);   opt_phi.step();`
$\triangleright$ Update main model parameters θ (encoder, DSCD, decoder)
`opt_theta.zero_grad();`
$\mathcal{L}_{\mathrm{total}} \leftarrow \mathcal{L}_{\mathrm{MT}} + \lambda_{\mathrm{span}} \mathcal{L}_{\mathrm{span}} + \lambda_{\mathrm{sense}} \mathcal{L}_{\mathrm{sense}} + \lambda_{\mathrm{ASBN}} \mathcal{L}_{\mathrm{ASBN}} + \lambda_{\mathrm{reg}} \mathcal{R}$
`L_total.backward();   opt_theta.step();`
return updated parameters, logs (discriminator accuracies, homograph metrics, prototype stats)

Key remarks (practical):

- **GRL behavior:** forward pass identity; backward pass multiplies gradients by $-\lambda_{k,j}$ (frameworks like PyTorch can implement sample-weighted GRL or emulate it by weighting losses).
- **Confidence weighting:** we scale GRL so only *wrong-but-confident* shortcuts receive strong adversarial pressure.
- **Discriminator capacity:** keep discriminators small (1–2 layer MLPs) to avoid overpowering the encoder.
- **Warm-up:** train DSCD+NMT for 1–3 epochs before enabling full ASBN to let uncertainty estimates become meaningful.

7 Hyperparameters and monitoring

Suggested ranges

- Buffer size C: 100–1000; EMA η: 0.03–0.08.
- MC-dropout passes M: 3–8; temperature T: 0.6–1.0.
- $N_{\min}$ (prototype stabilization): 3–10; $K_{\max}$: 20.

- GRL base strengths: $\bar{\lambda}_{\text{freq}} = 1.0$, $\bar{\lambda}_{\text{ctx}} = 0.5$, $\bar{\lambda}_{\text{xl}} = 0.8$; $\lambda_{\max} = 2.0$.
- Learning rates: $lr_\theta = 3\text{e-}4$, $lr_\phi = 1\text{e-}4$.

What to log

- Discriminator accuracy and loss over time (should decrease if encoder hides shortcuts).
- Prototype creation rate, prototype counts per type, centroid drift.
- Homograph-slice translation metrics (Sense-F1, BLEU/COMET on homograph sentences).
- Distribution of U_j (uncertainty) and per-sample $\lambda_{k,j}$.

8 Practical tips and common debugging steps

- If discriminators dominate training: reduce $\bar{\lambda}_k$, reduce discriminator size, or decrease discriminator LR.
- If translation quality drops: lower λ_{ASBN} or only apply ASBN to a curated homograph-focused batch.
- If uncertainty U_j is noisy: increase warm-up period and verify MC-dropout/σ-net implementations.
- Use small, progressive changes: enable one discriminator at a time (first frequency, then context, then cross-lingual).