Contrastive Learning and Representation Learning

Learning Useful Representations through Similarity Signals

Introduction to Representation Learning

Representation learning means automatically discovering the features or embeddings that best describe raw data. Imagine you have thousands of photos: rather than storing raw pixels, you want compact codes that capture essential concepts—like "beach" or "sunset." Good representations make downstream tasks (classification, retrieval) easier.

Key idea: Learn embeddings in a vector space where similar items are close together and dissimilar items are far apart.

Contrastive Learning Concept

Contrastive learning is a self-supervised approach where the model learns by comparing examples:

1. **Positive pairs**: Two different views (augmentations) of the same instance (e.g., two crops of the same image).
2. **Negative pairs**: Views from different instances.

Objective: Pull positive pairs closer in embedding space, push negative pairs apart.

Loss function (InfoNCE):

$$ext L = -\log \frac{\exp(\mathrm{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{N} \exp(\mathrm{sim}(z_i, z_k)/\tau)}$$

- z_i, z_j: embeddings of a positive pair
- $\mathrm{sim}(\cdot)$: similarity measure (e.g., cosine)
- τ: temperature hyperparameter
- N: total samples in the batch

This encourages the network to learn features that distinguish instances.

Short Example (PyTorch-style Pseudocode)

```
# X: batch of raw inputs
# aug(): random augmentation function
# encoder: neural network mapping input to embedding
# sim: cosine similarity function
# tau: temperature scalar

# 1. Create positive pairs
x1 = aug(X)
x2 = aug(X)
```

```
# 2. Encode
z1 = encoder(x1)  # shape: (batch, d)
z2 = encoder(x2)

# 3. Normalize embeddings
z1 = z1 / z1.norm(dim=1, keepdim=True)
z2 = z2 / z2.norm(dim=1, keepdim=True)

# 4. Compute similarity matrix
sim_matrix = z1 @ z2.T  # shape: (batch, batch)

# 5. Compute InfoNCE loss for each i
loss = 0
for i in range(batch):
    numerator = exp(sim_matrix[i,i] / tau)
    denominator = exp(sim_matrix[i] / tau).sum()
    loss += -log(numerator / denominator)
loss = loss / batch
```

This code shows how to build positive pairs, compute similarities, and apply the contrastive loss.

Discussing the Output

After training, the encoder maps inputs to an embedding space where:

- **Similar instances** (different augmentations of the same item) cluster together.
- **Different instances** are well-separated.

These embeddings can then be:

- **Fine-tuned** for classification with limited labels.
- **Used directly** for retrieval (nearest-neighbor search).

Reflection and Best Practices

Key Takeaways:

- Contrastive learning leverages self-supervision—no labels needed.
- Embeddings learned capture meaningful features for multiple tasks.
- Temperature τ controls the tightness of clustering.

Common Pitfalls:

- **Batch size matters**: Too small batches limit negative pairs.
- **Augmentation choices**: Poor augmentations can hinder learning.
- **Compute cost**: Large batch similarities are expensive.

Real-World Applications:

- **Computer vision**: SimCLR, MoCo for image embeddings.

- **Text**: Contrastive pre-training for sentence embeddings (SimCSE).
- **Multimodal**: CLIP uses image-text pairs.

This guide provides a thorough, beginner-friendly overview of contrastive and representation learning, complete with key equations and code examples. Download the PDF for an instantly publishable chapter.