Report - Variational Sparse Coding

Table of contents

1 Introduction **3**
1.1 Datasets 3
1.2 Report Structure 3

2 Auto Encoder **5**
2.1 Architecture 5
2.2 Applications 5
2.3 Limitations 5

3 Variational Auto Encoder **6**
3.0.1 VAE's latent space 6
3.0.2 Evidence Lower Bound (ELBO) 6
3.0.3 β-VAE 7

4 Variational Sparse Coding **8**
4.1 Motivation 8
4.2 Recognition Model 9
4.3 Training Procedure 10
4.3.1 Prior Distribution & Objective 10
4.3.2 Warm-Up Strategy 11

5 Experiments **12**

6 Conclusion **13**

1 Introduction

Unsupervised learning in high-dimensional data poses significant challenges, particularly in discovering interpretable features and enabling controllable generation.

Variational Autoencoders (VAEs) provide a probabilistic framework for mapping complex data to lower-dimensional latent spaces, but they often fail to disentangle underlying factors of variation, especially when the number of true sources is unknown or when observations exhibit diverse attribute combinations.

The "Variational Sparse Coding" (VSC) paper by Francesco Tonolini et al. proposes a novel extension of VAEs, incorporating sparsity in the latent space via a Spike and Slab prior to address these issues.

This report for a statistical course explores the VSC model, detailing its theoretical foundations, implementation, and empirical validation.

1.1 Datasets

- **MNIST** : TODO…
- **Fashion-MNIST**: A collection of 28x28 grayscale images of clothing items (e.g., T-shirts, trousers), comprising 60,000 training and 10,000 test samples. It serves as a drop-in replacement for MNIST, offering richer variability for testing generative models.
- **Smiley** : TODO…
- **UCI HAR**: A dataset of accelerometer and gyroscope signals from smartphones, capturing six human activities (e.g., walking, sitting) across 30 subjects, with preprocessed segments of 128 time steps.

1.2 Report Structure

- **Autoencoders**: Introduces the basics of autoencoders and their role in high-dimensional data analysis.
- **Variational Autoencoders**: Explains VAEs, focusing on the Evidence Lower Bound (ELBO) and latent space regularization.
- **Variational Sparse Coding**: Details the VSC model, including the Spike and Slab prior and warm-up training strategy.

- **Experiments**: Summarizes empirical results, comparing VSC to -VAEs across multiple tasks.
- **Conclusion**: Highlights VSC's contributions and future directions.

2 Auto Encoder

An encoder is a neural network that compresses high-dimensional input data into a lower-dimensional latent representation.

2.1 Architecture

1. **Encoder**: Maps input $x \in \mathbb{R}^D$ to latent code $z \in \mathbb{R}^d$ (compression).
2. **Decoder**: Reconstructs $\hat{x}$ from z (reconstruction).

$$\mathcal{L}_{\text{AE}} = \text{Reconstruction\_term} = \|x - \text{Decoder}(\text{Encoder}(x))\|^2$$

Pytorch implementation: `logic/model/autoencoder.py`

2.2 Applications

- Feature extraction for clustering/visualization.
- Denoising (e.g., noisy Fashion-MNIST reconstruction).

2.3 Limitations

- Deterministic: No probabilistic latent space.
- No control over latent structure (e.g., disentanglement).

3 Variational Auto Encoder

Variational Autoencoders (*VAEs*) extend autoencoders into a probabilistic framework, enabling both representation learning and generative modeling.

Unlike traditional autoencoders, *VAEs* model the latent variables z as drawn from a distribution, typically a Gaussian, parameterized by the encoder.

3.0.1 VAE's latent space

- The encoder outputs parameters μ and σ^2
  - $q_\phi(z|x) = \mathcal{N}(z; \mu, \sigma^2)$.
- The decoder generates $p_\theta(x|z)$.
- The latent space prior is typically a standard Gaussian,
  - $p(z) = \mathcal{N}(0, I)$, which constrains the latent space to remain centered around the origin, promoting meaningful structure for visualization and generative tasks.
- The goal is to maximize the marginal likelihood $p(x)$, but this is intractable due to the integral:

$$p(x) = \int p_\theta(x|z)p(z)dz$$

3.0.2 Evidence Lower Bound (ELBO)

VAEs approximate the marginal likelihood $p(x)$ using the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\text{KL}}(q_\phi(z|x)\|p(z))$$

- **Reconstruction Term**: $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - encourages accurate data reconstruction.
- **KL Divergence Term**: $D_{\text{KL}}(q_\phi(z|x)\|p(z))$
  - regularizes the latent distribution to match the prior.

The ELBO is optimized with respect to encoder parameters ϕ and decoder parameters θ using the reparameterization trick for gradient computation.

```
def reparameterize(self, mu: torch.Tensor, logvar: torch.Tensor) -> torch.Tensor:
    std = torch.exp(0.5 * logvar)
    eps = torch.randn_like(std)
    return mu + eps * std
```

3.0.3 β-VAE

The -VAE introduces a hyperparameter β to control the weight of the KL divergence term in the ELBO:

$$\mathcal{L}_{\beta\text{-VAE}} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta D_{\mathrm{KL}}(q_\phi(z|x) \| p(z))$$

- When $\beta = 0$, the β-VAE reduces to a simple autoencoder model.
- When $\beta = 1$, the β-VAE reduces to the standard VAE.
- Increasing β encourages disentanglement in the latent space, making it more interpretable but potentially at the cost of reconstruction quality.

The β-VAE is particularly useful in scenarios where interpretability of the latent space is critical, such as in disentangled representation learning. However, careful tuning of β is required to balance reconstruction and disentanglement.

4 Variational Sparse Coding

4.1 Motivation

What is Posterior Collapse?

- **Problem**: In VAEs, some latent dimensions become “useless” – they encode no meaningful information.
- **Why it happens**:
  - The KL divergence term in ELBO forces latent variables to match the prior.
  - If the decoder is too powerful, it ignores latent variables, leading to **dimensions being permanently inactive**.
  - Result: Model uses only a few dimensions, losing sparsity and disentanglement.

How VSC Fixes It:

1. **Spike-and-Slab Warm-Up**
   - **Phase 1** ($\lambda = 0$):
     - Forces latent variables to behave like **binary codes** (spike dominates).
     - Model must “choose” which dimensions to activate (no continuous refinement).
   - **Phase 2** ($\lambda \to 1$):
     - Gradually introduces continuous slab parameters ($\mu_{i,j}, \sigma_{i,j}$).
     - Prevents early over-reliance on a few dimensions.
2. **Sparsity Enforcement**
   - **KL Sparsity Term**: Penalizes average activation rate $\bar{\gamma}_u$ if it deviates from α (e.g., $\alpha = 0.01$).
   - Forces the model to use **only essential dimensions**, avoiding redundancy.
3. **Dynamic Prior**

- Prior $p_s(z)$ adapts via pseudo-inputs x_u and classifier $C_\omega(x_i)$.
- Prevents trivial alignment with a fixed prior (e.g., $\mathcal{N}(0,1)$).

Result:

- Latent dimensions stay **sparse and interpretable**.
- No single dimension dominates; features are distributed across active variables. Variational Sparse Coding (VSC) extends VAEs by inducing sparsity in the latent space using a **Spike-and-Slab prior**, enabling feature disentanglement and controlled generation when the number of latent factors is unknown.

4.2 Recognition Model

Spike-and-Slab Encoder Distribution

$$q_\phi(z|x_i) = \prod_{j=1}^{J} \left[\gamma_{i,j}\mathcal{N}(z_{i,j}; \mu_{i,j}, \sigma_{i,j}^2) + (1-\gamma_{i,j})\delta(z_{i,j})\right]$$

Parameters: All outputs of a neural network (encoder).

- $\gamma_{i,j}$: Probability that latent dimension j is *active* for input x_i.
- $\mu_{i,j}, \sigma_{i,j}$: Mean and variance of the Gaussian (slab) for active dimensions.
- $\delta(z_{i,j})$: Dirac delta function (spike) forces inactive dimensions to exactly **0**.

Pytorch implementation of the reparameterization `logic/model/vsc.py`:

```
def reparameterize(self,
    mu: torch.Tensor,
    logvar: torch.Tensor,
    gamma: torch.Tensor
    ) -> torch.Tensor:

    lamb = self.lambda_val  # warm-up factor
    std = torch.exp(0.5 * logvar)
    eps = torch.randn_like(std)

    # Interpolate between a fixed (zero-mean, unit variance) slab
    # and the learned slab.
```

```
    slab = lam * mu + eps * (lam * std + (1 - lam))

    # Sample binary spike; note: torch.bernoulli is not differentiable.
    spike = torch.bernoulli(gamma)

    return spike * slab
```

4.3 Training Procedure

4.3.1 Prior Distribution & Objective

Prior

$$p_s(z) = q_\phi(z|x_{u^*}), \quad u^* = C_\omega(x_i)$$

- **Pseudo-inputs**: Learnable templates $\{x_u\}$ represent common feature combinations.
- **Classifier**: $C_\omega(x_i)$ selects the best-matching template x_{u^*} for input x_i.

Objective (ELBO with Sparsity)

$$\mathcal{L} = \sum_i \Big[-\mathrm{KL}(q_\phi \| p_s) + \mathbb{E}_{q_\phi}[\log p_\theta(x_i|z)]\Big] - J \cdot \mathrm{KL}(\bar{\gamma}_u \| \alpha)$$

- **KL Term**:
  - Aligns encoder $(\mu_{i,j}, \sigma_{i,j}, \gamma_{i,j})$ with prior $(\mu_{u^*,j}, \sigma_{u^*,j}, \gamma_{u^*,j})$.
  - Closed-form formula ensures fast computation.

- **Sparsity Term**:
  - Penalizes deviation from target sparsity α (e.g., 90% dimensions inactive).

Pytorch implementation in `logic/model/vsc.py`:

```
def compute_sparsity_loss(self, gamma: torch.Tensor) -> torch.Tensor:
    target = torch.full_like(gamma, self.prior_sparsity)
    return nn.functional.binary_cross_entropy(gamma, target)
```

4.3.2 Warm-Up Strategy

$$q_{\phi,\lambda}(z|x_i) = \prod_{j=1}^{J} \left[\gamma_{i,j}\mathcal{N}(z_{i,j}; \lambda\mu_{i,j}, \lambda\sigma_{i,j}^2 + (1-\lambda)) + (1-\gamma_{i,j})\delta(z_{i,j})\right]$$

- **Phase 1** ($\lambda = 0$):
  - Slab fixed to $\mathcal{N}(0,1)$ (binary-like latent codes).
  - Focus: *Which* features to activate.
- **Phase 2** ($\lambda \to 1$):
  - Gradually learn $\mu_{i,j}, \sigma_{i,j}$ (refine *how* to represent features).
- **Avoids collapse**: Prevents premature "freezing" of latent dimensions.

5 Experiments

6 Conclusion