Self-Organizing Autoencoder

Anonymous Authors[1]

Abstract

We propose a deep version of SOM, and relate it to autoencoder. On one hand, it helps to explain why AE learns latent spaces that keeps topology, and why VAE tends to generate blurry results; on the other hand, It gives SOM a new perspective from probability theory.

1. Introduction

2. Related Work

3. Self-Organizing Autoencoder (SOAE)

First we recall what a SOM is. Then we extend the formulation to the deep learning context. At last, we compare the deep version of SOM with VAE.

3.1. Self-Organizing Map

Self-organizing maps often comes with two formulations: one is the original incremental algorithm; the other is a batch version. We use the original incremental version.

First define a set of latent nodes $\mathcal{Z} = \{z_1, \dots, z_M\}$ on a two-dimensional plane. Typically, points in $\mathcal{Z}$ are chosen to form a rectangular or hexagonal lattice. The distances between latent nodes are used to define neighborhoods through a *neighborhood function* $h_t(z', z)$. Each latent node z_i is related to data space by a *reference vector* $m_i \in \mathcal{X}$. Reference vectors are initialized first, and then refined by the Algorithm 1.

The neighborhood kernel can use the Gaussian function,

$$h_t(z', z) = e^{-\frac{\|z'-z\|^2}{2\sigma^2(t)}}, \tag{3}$$

where t is iteration time, and $\sigma(t)$ defines the width of the neighborhood kernel. Both the learning rate α_t and $\sigma(t)$ are monotonically decreasing functions of time.

Note that the update rule (2) is a step to optimize the loss

$$\min_{m_i} \frac{1}{2} h_t(z_c, z_i) \|x - m_i\|^2. \tag{4}$$

Algorithm 1 SOM (Kohonen, 1982; 1998)

Input: latent nodes $\mathcal{Z} \subset \mathbb{R}^2$, data space $\mathcal{X} \subset \mathbb{R}^n$, learning rate α_t, neighborhood function $h_t(z', z)$, max iteration time T

Initialize reference vectors $m_i \in \mathbb{R}^n$ for each $z_i \in \mathcal{Z}$
for $t = 1$ **to** T **do**
  Sample $x \in \mathcal{X}$
  Find *best matching node* m_c, where

$$c = \operatorname*{argmin}_i \{\|x - m_i\|\} \tag{1}$$

  Update reference vectors

$$m_i \leftarrow m_i + \alpha_t h_t(z_c, z_i)\big(x - m_i\big), \quad \forall i \tag{2}$$

end for

3.2. Deep Version

The original SOM is a shallow neural network. We propose a deep version. Changes are,

- Discrete two-dimensional latent nodes become a continuous latent space $\mathcal{Z} \subset \mathbb{R}^m$ with dimension m.
- Difference between two points in data space $\mathcal{X}$ can be measured by any given distance function $d(\cdot, \cdot)$, *e.g.*, the distance induced by the L1/L2 norm.
- Reference vector of a latent vector $z \in \mathcal{Z}$ is represented by a deep network G, *i.e.*, $G(z) \in \mathbb{R}^n$.
- The best matching node is represented by another network C. Given a sample $x \in \mathcal{X}$, the best matching node $C(x)$ is found via minimizing the loss,

$$\min_C d(x, G(C(x))). \tag{5}$$

  Here the network G is fixed.

- For any given $z' \in \mathcal{Z}$ and iteration t, the integration of the neighborhood function $h_t(z', z)$ over the latent space $\mathcal{Z}$ is required to be 1, and we call it the *neighborhood kernel*.

$$\int_{\mathcal{Z}} h_t(z', z) dz = 1. \tag{6}$$

Algorithm 2 SOAE

Input: latent space $\mathcal{Z} \subset \mathbb{R}^m$, data space $\mathcal{X} \subset \mathbb{R}^n$, neighborhood kernel $h_t(z', z)$, max iteration time T, C iteration time T_1, G iteration time T_2

Initialize network G, C
for $t = 1$ **to** T **do**
  for $t_1 = 1$ **to** T_1 **do**
    Sample $x \in \mathcal{X}$
    Update C with loss (5).
  end for
  for $t_2 = 1$ **to** T_2 **do**
    Sample $z \sim h_t(C(x), \cdot)$
    Update G with loss (9).
  end for
end for

- At step t, the update rule (2) now becomes a single optimization step of G for the objective,

$$\min_G \int_{\mathcal{Z}} h_t(C(x), z) d(x, G(z)) dz. \tag{7}$$

The update step in (7) contains an integration over the latent space. For given C, x, and t, treat $h_t(C(x), \cdot)$ as a probability distribution over $\mathcal{Z}$. Rewrite the integration as an expectation, we have

$$\min_G \mathbb{E}_{z \sim h_t(C(x), \cdot)} \big[d(x, G(z)) \big]. \tag{8}$$

Then we can optimize this approximately by sampling some finite number of latent vectors. For example, in the case of sampling only one point z, the objective is,

$$\min_G d(x, G(z)). \tag{9}$$

We summarize the procedure in Algorithm 2.

3.3. Relation to Autoencoder

In the context of autoencoder, G is the decoder, and C is the encoder. Equation (5) is like the autoencoder loss. When training G, the sampling of $z \sim h_t(C(x), \cdot)$ is like the variational autoencoder (Kingma & Welling, 2013). However, we no not need the reparametrization trick because the gradient of z need not to propagate to network C. The kl-divergence loss counterpart for regularizing conditional distribution $P(z|x)$ is absent from loss here, because the neighborhood kernel took its place. It also resembles denoising autoencoder (Vincent et al., 2008), albeit denoising the latent space rather than the data space. The training is like wake-sleep algorithm (Hinton et al., 1995).

In summary, from the algorithmic view, SOAE is like training an encoder C and decoder G using the wake-sleep algorithm. During the wake pass, *i.e.*, training of G, it resembles a VAE; while in the sleep pass, *i.e.*, training of C, it is like a normal autoencoder.

4. Experiment

4.1. Latent Space Reularization

When latent dimension is low, the norm of random latent vectors varies much. This might has side effect for numerical reasons. To workaround this, we propose a *torus-embedding mapping* and *sphere-embedding mapping* to make sure the embedded vectors share the same norm.

4.1.1. Torus-Embedding Mapping

Torus-embedding mapping $T : \mathbb{R}^m \to \mathbb{R}^{2m}$ brings latent vectors to a torus,

$$T(z) = (\cos z, \sin z). \tag{10}$$

Operations are done in element-wise.

4.1.2. Sphere-Embedding Mapping

Sphere-embedding mapping $S : \mathbb{R}^m \backslash \{0\} \to \mathbb{R}^m$ brings latent vectors to a unit sphere,

$$S(z) = \frac{z}{\|z\|}. \tag{11}$$

4.1.3. Importance Sampling

Let the gradients fix major problems.

4.1.4. Radius Based Neighborhood Sampling

The radius forms an Gaussian. The directions are uniform.

5. Conclusion

References

Hinton, Geoffrey E, Dayan, Peter, Frey, Brendan J, and Neal, Radford M. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158, 1995.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kohonen, Teuvo. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43 (1):59–69, 1982.

Kohonen, Teuvo. The self-organizing map. *Neurocomputing*, 21(1):1–6, 1998.

Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.