Enhancing Disentanglement through Learned Aggregation of Convolutional Feature Maps: A Study on the 2019 Disentanglement Challenge

Abstract

This paper details our submission for stage 2 of the 2019 disentanglement challenge. It introduces a straightforward image preprocessing technique for discovering disentangled latent factors. Our approach involves training a variational autoencoder using aggregated feature maps. These maps are obtained from networks that were pretrained on the ImageNet database, and we leverage the implicit inductive bias present in those features for disentanglement. This bias can be further strengthened by fine-tuning the feature maps with auxiliary tasks such as angle, position estimation, or color classification. Our method achieved second place in stage 2 of the competition. Code is publicly available.

1 Introduction

Methods that are fully unsupervised are unable to learn disentangled representations unless further assumptions are made through inductive biases on both the model and the data. In our submission, we utilize the implicit inductive bias included in models pretrained on the ImageNet database, and then improve it by fine-tuning such models on tasks that are relevant to the challenge such as angle, position estimation, or color classification. Our stage 2 submission builds upon our stage 1 submission, in which we used pretrained CNNs to extract convolutional feature maps as a preprocessing step before training a VAE. Although this approach provided adequate disentanglement scores, two weaknesses were identified with the feature vectors that were extracted. First, the feature extraction network is trained on ImageNet, which is dissimilar to the MPI3d dataset that was used in the challenge. Secondly, the mechanism for feature aggregation was chosen in an ad-hoc way, and likely did not retain all information needed for disentanglement. We address these issues by fine-tuning the feature extraction network as well as by learning how to aggregate feature maps from data by using the labels of the simulation datasets MPI3d-toy and MPI3d-realistic.

2 Method

Our method includes three steps: (1) a supervised fine-tuning of the feature extraction CNN, (2) extracting a feature vector from each image in the dataset using the fine-tuned network, and (3) training a VAE to reconstruct the feature vectors and disentangle the latent factors of variation.

2.1 Finetuning the Feature Extraction Network

In this step, we fine-tune the feature extraction network offline, before submitting to the evaluation server. The aim is to adapt the network so that it produces aggregated feature vectors that retain the necessary information for disentangling the latent factors of the MPI3d-real dataset. The network is fine-tuned by learning to predict the value of each latent factor using the aggregated feature vector of an image. To do so, we use the simulation datasets MPI3d-toy and MPI3d-realistic, specifically the images as inputs and the labels as supervised classification targets.

.

For the feature extraction network, we use the VGG19-BN architecture from the torchvision package. The input images are standardized using mean and variance across each channel as computed from the ImageNet dataset. We use the output feature maps from the last layer before the final average pooling (dimensionality 512 x 2 x 2) as the input to a feature aggregation module which reduces the feature map to a 512-dimensional vector. The aggregation module consists of three convolution layers using 1024, 2048, and 512 feature maps and kernel sizes of 1, 2, and 1 respectively. Each layer is followed by batch normalization and ReLU activation. We also utilize layerwise dropout with a rate of 0.1 before each convolution layer. Finally, the aggregated feature vector is L2-normalized. This was empirically found to be important for the resulting disentanglement performance. Then, for each latent factor, we add a linear classification layer that computes the logits of each class using the aggregated feature vector. These linear layers are discarded after this step.

We use both MPI3d-toy and MPI3d-realistic for training to push the network to learn features that identify latent factors in a robust way, regardless of details such as reflections or specific textures. We split each dataset randomly with 80

2.2 Feature Map Extraction and Aggregation

In this step, we use the fine-tuned feature extraction network to produce a set of aggregated feature vectors. We simply run the network on each image of the dataset and store the aggregated 512-dimensional vectors in memory. Again, inputs to the feature extractor are standardized such that mean and variance across each channel correspond to the respective values from the ImageNet dataset.

2.3 VAE Training

Finally, we train a standard β-VAE on the set of aggregated feature vectors. The encoder network consists of a single fully connected layer with 4096 neurons, followed by two fully-connected layers that parameterize the means and log variances of a normal distribution N used as the approximate posterior $q(z|x)$. The number of latent factors is determined experimentally. The decoder network has four fully-connected layers with 4096 neurons each, followed by a fully-connected layer parameterizing the means of a normal distribution N used as the conditional likelihood $p(x|z)$. The mean is constrained to the range [0, 1] using the sigmoid activation. All fully connected layers except for the final ones use batch normalization and are followed by ReLU activation functions. We use orthogonal initialization for all layers and assume a factorized standard normal distribution as the prior $p(z)$ on the latent variables.

For optimization, we use the RAdam optimizer with a learning rate of 0.001, $\beta_0 = 0.999$, $\beta_1 = 0.9$ and a batch size of 256. The VAE is trained for 120 epochs by maximizing the evidence lower bound, which is equivalent to minimizing

$$\frac{1}{B}\sum_{i=1}^{512}||\mu_i - x_i||^2 + 0.5\beta\sum_{j=1}^{C} 1 + log(\sigma_j^2) - \mu_j^2 - \sigma_j^2$$

where β is a hyperparameter to balance the MSE reconstruction and the KLD penalty term. Because the scale of the KLD term depends on the number of latent factors C, we normalize it by C such that β can be varied independently of C. It can be harmful to start training with too much weight on the KLD term. Therefore, we use the following cosine schedule to smoothly anneal β from $\beta_{start} = 0.005$ to $\beta_{end} = 0.4$ over the course of training:

$$\beta(t) = \{ \beta_{start} for t < t_{start}$$
$$\frac{1}{2}(\beta_{end} - \beta_{start})(1 + \cos(\pi \frac{t - t_{start}}{t_{end} - t_{start}})) + \beta_{start} for t_{start} \leq t \leq t_{end}$$
$$\beta_{end} for t > t_{end}$$

where $\beta(t)$ is the value for β in training episode $t \in 0, ..., N-1$, and annealing runs from epoch $t_{start} = 10$ to epoch $t_{end} = 79$. This schedule allows the model to initially learn to reconstruct the data well, and only then puts pressure on the latent variables to be factorized, which improved performance.

3 Discussion

Our method achieved second place in stage 2 of the competition. Compared to our stage 1 approach, our stage 2 approach resulted in large improvements on the FactorVAE and DCI metrics. On the public leaderboard, our best submission achieved first rank on these metrics. See appendix A for further discussion of the results.

Introducing prior knowledge makes the disentanglement task considerably easier, and this is reflected in the improved scores. However, our method uses task-specific supervision obtained from simulation, which restricts its applicability. Nevertheless, this demonstrates that such supervision can transfer to better disentanglement on real-world data, which was a goal of the challenge.