Ordered binary speaker embedding

Jiaying Wang

2023/02/10

Content

- Abstract
- Related work
- MAE and MAE-R/S/RS
- Experiments

Abstract

- Background
    - Storage
    - Computational efficiency

- Contribution

  ordered & binary speaker embedding
    - Storage
        - data type: float32-->int
        - vector length: 256-->m ($m \in (0,256)$)
    - Computational efficiency
        - distance: cosine distance-->hamming distance
        - use binary tree to store enroll embeddings: O(Nm)-->O(m) (To be verified)

Related work

LSH

$$h_r(x) = \begin{cases} 1 & if\ r^T x \geq 0 \\ 0 & otherwise \end{cases}$$

$$\begin{aligned} \mathbf{Pr}[h(A) = h(B)] &= 1 - \frac{\theta}{\pi}, \text{ where} \\ \theta &= \cos^{-1}\left(\frac{|A \cap B|}{\sqrt{|A| \cdot |B|}}\right) \end{aligned}$$

- Core contribution of LSH： map x-vector to 0 or 1 by the product of a hyperplane randomly sampled from a zero-mean multivariate Gaussian distribution by *b* hash functions
- Advantage: simple calculation method brings high computational efficiency
- Disadvantage: the randomly selected matrix leads to unstable experimental results

Related work

PCA

a. X: (m,n)-->(n,m)
b. demean X
c. find the covariance matrix
d. calculate eigen values and corresponding eigen vectors of the covariance matrix
e. the eigen vectors are arranged into a matrix according to the corresponding eigenvalues, and the first *k* rows are taken to form a matrix P, where k is the first *k* principal components

- Advantage
  - the embedding is ordered
- Disadvantage
  - still dense vectors
  - linear

Related work

PCA-like AE

Algorithm 1 PCA Autoencoder algorithm. Note, we have described the algorithm with a simple gradient descent, but any descent-based optimisation can be used (Adam, Adagrad etc)

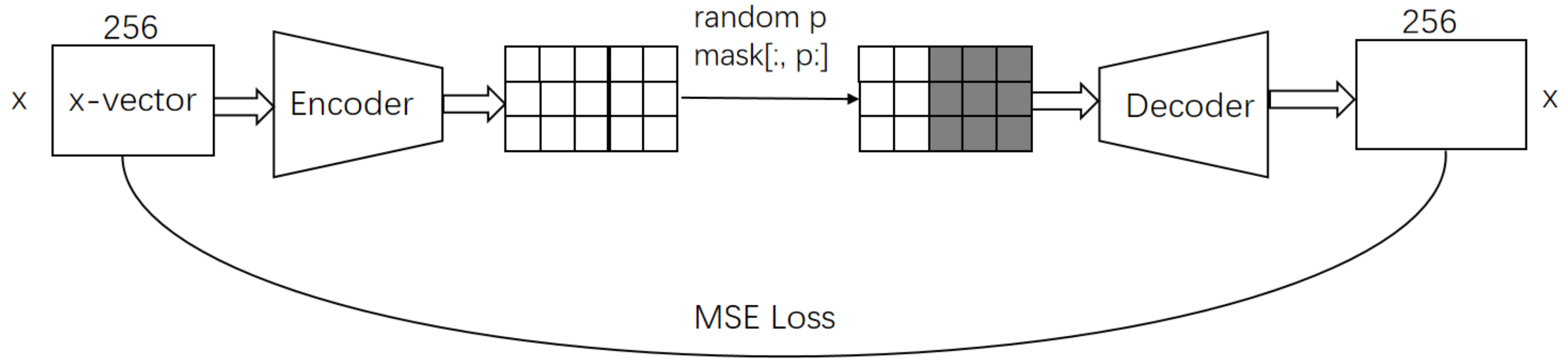
Data: X (dataset)
Parameters:
d_{max} : maximum latent space size
N : number of iterations to train each autoencoder
Result:
(E, D) : trained PCA Autoencoder

Train first latent dimension:
for $i = 1 \ldots N$ **do**
  $\theta^{(1)} = \theta^{(1)} - \alpha \nabla_{\theta^{(1)}} \mathcal{L}(X)$
Train rest of latent dimensions:
for $k = 2 \ldots d$ **do**
  Train next latent dimensions, keeping the codes $j = 1 \ldots k-1$ *fixed at each iteration* :
  for $i = 2 \ldots N$ **do**
    $\theta^{(i)} = \theta^{(i)} - \alpha \nabla_{\theta^{(i)}} \mathcal{L}(X)$

- Core contribution: train by step
- Advantage
  - ordered (the paper claims)
  - independent
- Disadvantage
  - large time cost to train the model
  - the embeddings are not ordered for our data

MAE and MAE-R/S/RS

Masked AutoEncoder (MAE)



- Similar to PCA (differenct: the input X for pca can be reconstructed from embeddings)

- Advantage
  - the embedding is ordered
- Disadvantage
  - still dense vectors
  - linear (if only 1 layer without activation function)

MAE and MAE-R/S/RS

dense-->binary: How?

- LSH
- Sample
- Regularizer
- Sample+ Regularizer

MAE and MAE-R/S/RS

dense-->binary: How?

LSH

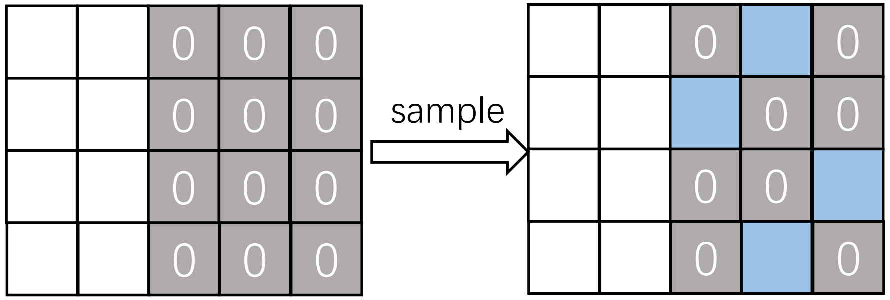
- use LSH on the latent code of MAE
- repeat the operation for 10 times and take the mean of topk results

MAE and MAE-R/S/RS

dense-->binary: How?

Sample

- The masked part of MAE is converted into binary encoding conforming to Bernoulli distribution



masked code   sampled code

```python
def forward(self, x):
    #encoder
    hidden_layer =self.encoder(x)

    #mask
    p = np.random.randint(hidden_layer.size(1))
    mask = torch.ones_like(hidden_layer, dtype=torch.float)
    mask[:,p:] = 0.0
    hidden_layer = hidden_layer * mask

    #sample
    hidden_layer[:,p:] = sample_z(hidden_layer[:,p:])

    #decoder
    decoded = self.decoder(hidden_layer)

    return hidden_layer, decoded
```

MAE and MAE-R/S/RS

dense-->binary: How?

Regularizer

- add a term to the loss function which calculates the distance to 1 for the unmasked part

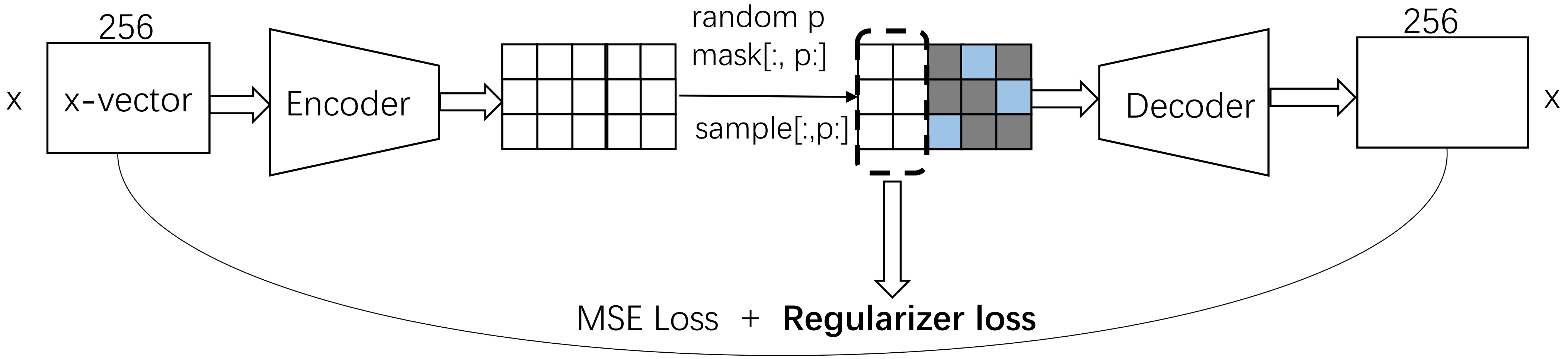
```
def hidden_loss(b):
    """
    compute hashing loss
    automatically consider all n^2 pairs
    """
    loss = (b.abs() - 1).abs().sum(dim=1).mean() * 2

    return loss

def AE_loss(mse_loss, input, hidden_layer, output, alpha,p):

    loss =  mse_loss(input,output)+ alpha* hidden_loss(hidden_layer[:,0:p])
    return loss
```

MAE and MAE-R/S/RS

dense-->binary: How?

Sample + Regularizer

- combine sample and regularizer

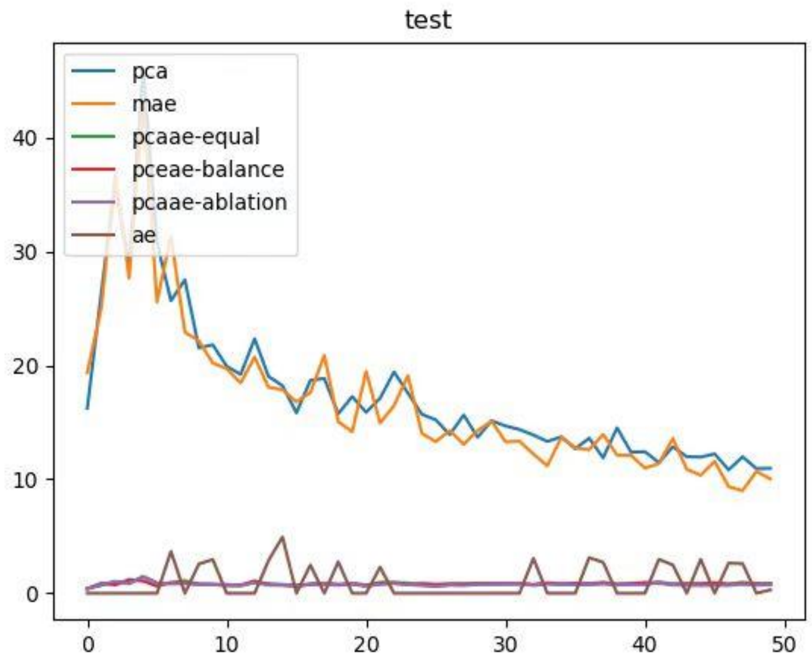


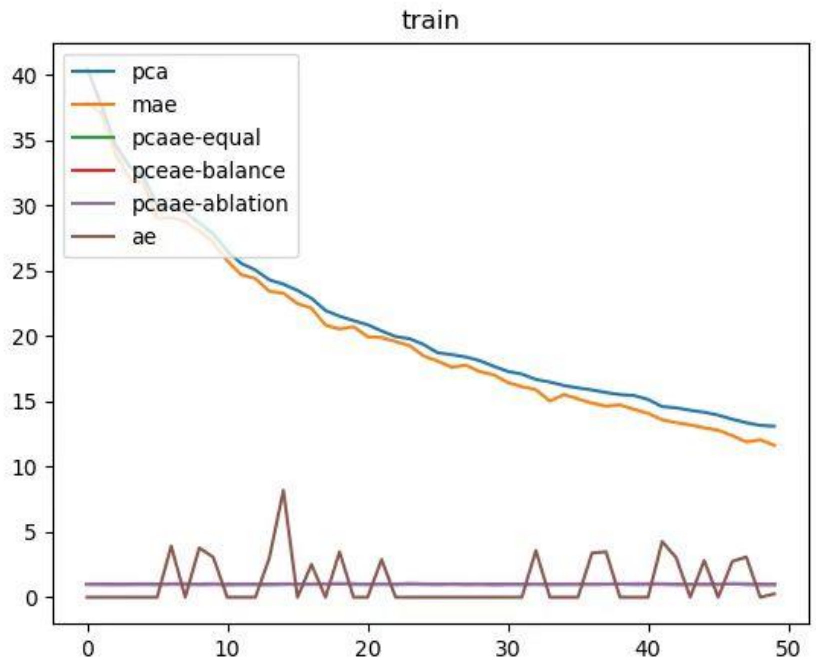
Experiments

I. Prove the order of latent space
II. Prove the improvement compared to baseline
    I. dense
        I. MAE
    II. binary
        I. MAE-R
        II. MAE-S
        III. MAE-RS

Experiments

Prove the order of latent space: variance of latent code





Experiments

Prove the improvement compared to baseline---dense

dataset =cnc	dense													
	Baseline	PCA	MAE	PCA	MAE	PCA	MAE	PCA	MAE	PCA	MAE	PCA	MAE
	256dims	20dims	20dims	32dims	32dims	40dims	40dims	64dims	64dims	80dims	80dims	96dims	96dims
Top1	0.706	0.529	0.544	0.619	0.624	0.644	0.649	0.679	0.683	0.691	0.691	0.697	0.695
Top3	0.800	0.678	0.687	0.743	0.751	0.762	0.767	0.781	0.785	0.792	0.789	0.796	0.794
Top5	0.844	0.736	0.748	0.796	0.802	0.806	0.811	0.827	0.829	0.837	0.833	0.837	0.836

conclusion

- the shorter the code length, the better mae is (than pca)

Experiments

Prove the improvement compared to baseline---binary

dataset =cnc	dense	binary					
	Baseline	LSH	PCA-LSH	MAE-LSH	MAE-S	MAE-R	MAE-RS
	256dims	20bits	20bits	20bits	20bits	20bits	20bits
Top1	0.706	0.157	0.183	0.185	0.232	0.227	0.218
Top3	0.800	0.266	0.323	0.324	0.381	0.386	0.368
Top5	0.844	0.326	0.403	0.404	0.462	0.470	0.447

dataset=cnc	dense	binary					
	Baseline	LSH	PCA-LSH	MAE-LSH	MAE-S	MAE-R	MAE-RS
	256dims	32bits	32bits	32bits	32bits	32bits	32bits
Top1	0.706	0.241	0.276	0.278	0.326	0.339	0.327
Top3	0.800	0.362	0.427	0.430	0.487	0.491	0.476
Top5	0.844	0.424	0.505	0.509	0.562	0.569	0.553

conclusion

- all of our method(MAE-LSH, MAE-R/S/RS) are better than baseline
- R/S/RS is better than LSH

Work to be done

- full/fix test on MAE-R/S/RS
- speed test