CS 282 Deep Neural Networks **Spring 2023**

Lecture 16: Autoencoding & Self-supervision

Lecturer: Anant Sahai *Scribe: Xin Chen, Yun Yeong Choi*

16.1 PCA & Self-supervision

In many contexts, we lack enough labeled data so we may need unsupervised learning. There are two kinds of unsupervised learning which are (1) Dimensionality reduction style(pre-regression) (2) Clustering style (classification). In this lecture, we will talk about PCA-like dimensionality reduction in deep learning, also known as self-supervision.

16.1.1 PCA

The standard PCA formulation for dimension reduction with full SVD is shown as Equation 16.1

$$X_{\text{train}} = \begin{bmatrix} \vec{x}_1 & \vec{x}_2 & \dots & \vec{x}_n \end{bmatrix} = U\Sigma V^\top \tag{16.1}$$

where $X_{\text{train}} \in \mathbb{R}^{d\times n}$, $U \in \mathbb{R}^{d\times d}$, $\Sigma \in \mathbb{R}^{d\times n}$ and $V \in \mathbb{R}^{n\times n}$. The mean of X_{train} is removed in this case.

Now we want to project X_{train} into a lower dimension subspace (rank k) S which is a span of the first k columns of U. Note $k < d$ and S is actually a subspace of the space $\mathbb{R}^d$. That is to say, $S = span(\vec{u}_1, \vec{u}_2, \dots, \vec{u}_k)$. And the reduction operation is $U_k^\top X_{\text{train}}$ that maps X_{train} into $\mathbb{R}^k$, where U_k is the concatenation of the first k columns in U. In a downstream task when we have a lot of unlabelled data with high dimensions, we can do PCA to reduce the dimension and hope that a good regression model can be fitted with a few data points of low dimensions.

16.1.2 Self-supervision

In self-supervision, we use a similar idea to project the X_{train} into a lower dimension but using a gradient descent method. Consider the optimization task in Equation 16.2.

$$\arg\min_{w_1,w_2} \frac{1}{n} \|X - w_2 w_1 X\|_F^2 \tag{16.2}$$

where $w_1 \in \mathbb{R}^{k\times d}$ and $w_2 \in \mathbb{R}^{d\times k}$. $w_2 w_1$ is a rank at most k linear operator that maps $\mathbb{R}^d$ to $\mathbb{R}^d$. Also by the Eckart-Young theorem, we know that at the optimality of minimization, $w_1^* w_2^*$ is a projection to subspace $S = span(\vec{u}_1, \vec{u}_2, \dots, \vec{u}_k)$.

From another point of view, we are projecting X to $S = C(w_2)$. The dimension reduction operation here is $w_1 X$ and at the optimal point, $w_1 X$ should be able to capture exactly the relevant degrees of the freedom of X. Note that here w_1 doesn't necessarily need to have orthonormal columns.

The architecture of this autoencoding is shown in Figure 16.1. Input X is fed into the encoder-decoder model and output $\hat{X}$. The loss $L(X, \hat{X})$ is then calculated.

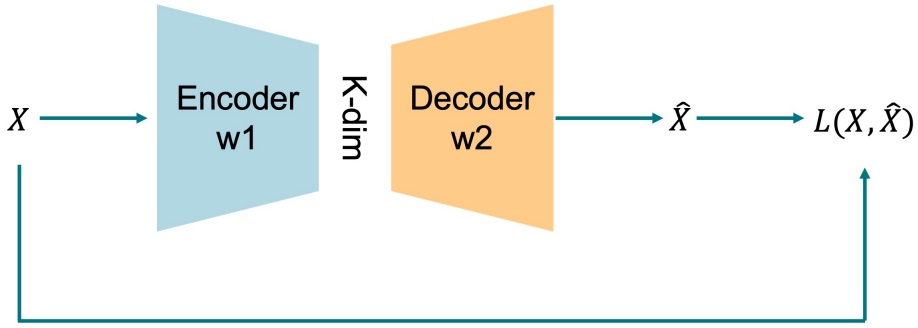

Figure 16.1: The encoder-decoder of a self-supervising model.

16.1.3 Autoencoding as Denoising/Purification

Now let's think about $w_1 X$ as a purification process. First, consider this problem from the aspect of PCA. We have a reality (a ground truth) of k-dimension. However, due to the interruption of d-dim noise, what we really measure is a d-dim signal from d different measurements. We can consider this as d linear operation plus measurement noises. Then the covariance of this signal is like Equation 16.3.

$$\sum = \begin{bmatrix} \sigma_1 + \sigma_1^{\text{noise}} & & & & \\ & \ddots & & & \\ & & \sigma_k + \sigma_k^{\text{noise}} & & \\ & & & \ddots & \\ & & & & \sigma_d^{\text{noise}} \end{bmatrix} \tag{16.3}$$

where σ_i is the singular value of the ground truth and σ_i^{noise} is the singular value of the noise. Now if we do PCA to only retain the dimension that corresponds to the first k singular values, and recover the dimension, we are denoising this measurement. The operation here can be considered as $U_k U_k^\top X$.

In the encoder-decoder architecture, $w_2 w_1 X$ is doing the same thing. Instead of using $w_2 w_1$ as the encoder-decoder, we can also use non-linear architecture as the encoder-decoder. Note that in this linear condition, the $w_2 w_1$ cannot be identity since an identity matrix cannot be computed by multiplying the encoder and decoder that has a "choke" point inside. However, when we use larger non-linear networks as the encoder-decoder, the learned weights may be just an "Identity" matrix. Therefore, we need to do regularization/data augmentation.

Questions:

- Can we enforce weight sharing between w_1 and w_2?
  $\rightarrow$ Yes. $U_k U_k^\top$ is an answer it self. However, in this case, we will have few parameters and confront with the issue of non-convexity. We may not solve the optimization task so we need to be careful when there are fewer weights. And to get U_k, we need to regularize the loss function.
- What will happen if we initialize w_1 and w_2 using zero matrix?
  $\rightarrow$ The gradient is zero and cannot update weights.

16.2 Data Augmentation in Self-supervision

16.2.1 Regularization with Noise

Instead of directly feeding the input X into the encoder-decoder, we can first add noises into the input. The modified architecture is shown in Figure 16.2. By adding noises to the input, we hope that the hidden representative can have some purification. Note that in autoencoding, the dimension of the hidden representative is not necessarily smaller than the input. We can also increase the dimension of the input matrix to get purification.

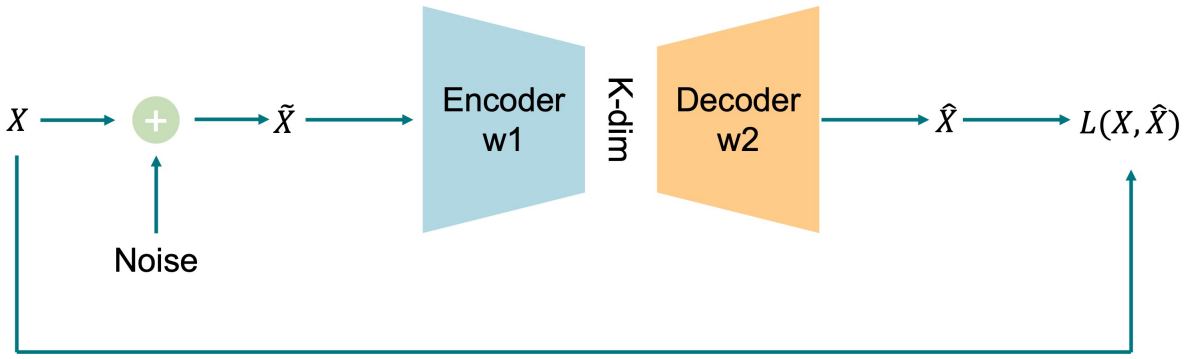

Figure 16.2: The encoder-decoder of a self-supervising model, with noises added.

16.2.2 Regularization with Masks

Another regularization method is to mask some of the elements in the input matrix, which is like a generalized noise method. For example, the input $\vec{X}$ is $\left[\ 1, 2, 3, 4, 5\ \right]$. We can mask two elements to get $\hat{X} = \left[\ 1, ?, 3, ?, 5\ \right]$. The "?" can be represented in several ways. We can use a "0" as the mask, so $\hat{X} = \left[\ 1, 0, 3, 0, 5\ \right]$. This looks like a data dropout process. Or we can assign a random value to "?" and record the positions of the mask using $\left[\ 1, 0, 1, 0, 1\ \right]$. The intuition behind using masking to reconstruct the input is that we have a matrix with "strong" dimensions and "weak" dimensions that are missing. This is similar to a lower-dimension matrix of the input. Now we want to reconstruct it by filling in the missing elements. Also, the architecture of the encoder-decoder is not necessarily like Figure 16.2. We can also do operations inside the encoder or decoder to block the flow of some directions. This is like adding a generalized "noise" inside the encoder/decoder, as shown in Figure 16.3.

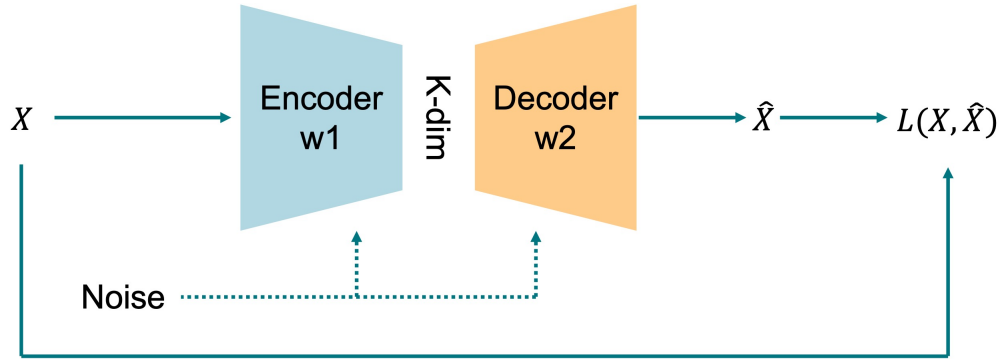

Figure 16.3: The encoder-decoder of a self-supervising model, with masks inside encoder/decoder.

16.3 What we wish this lecture also had to make things clearer?

1. We want to know more examples of using autoencoding in downstream tasks.