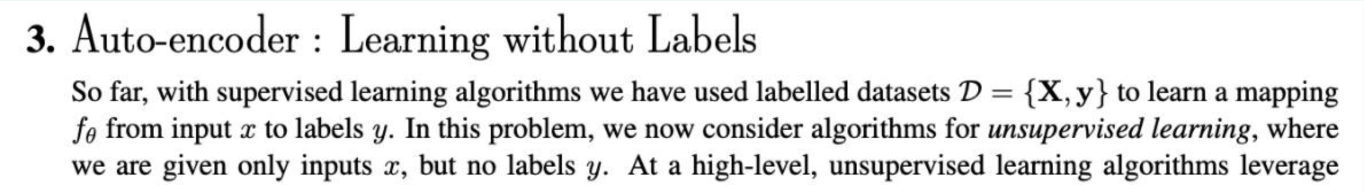
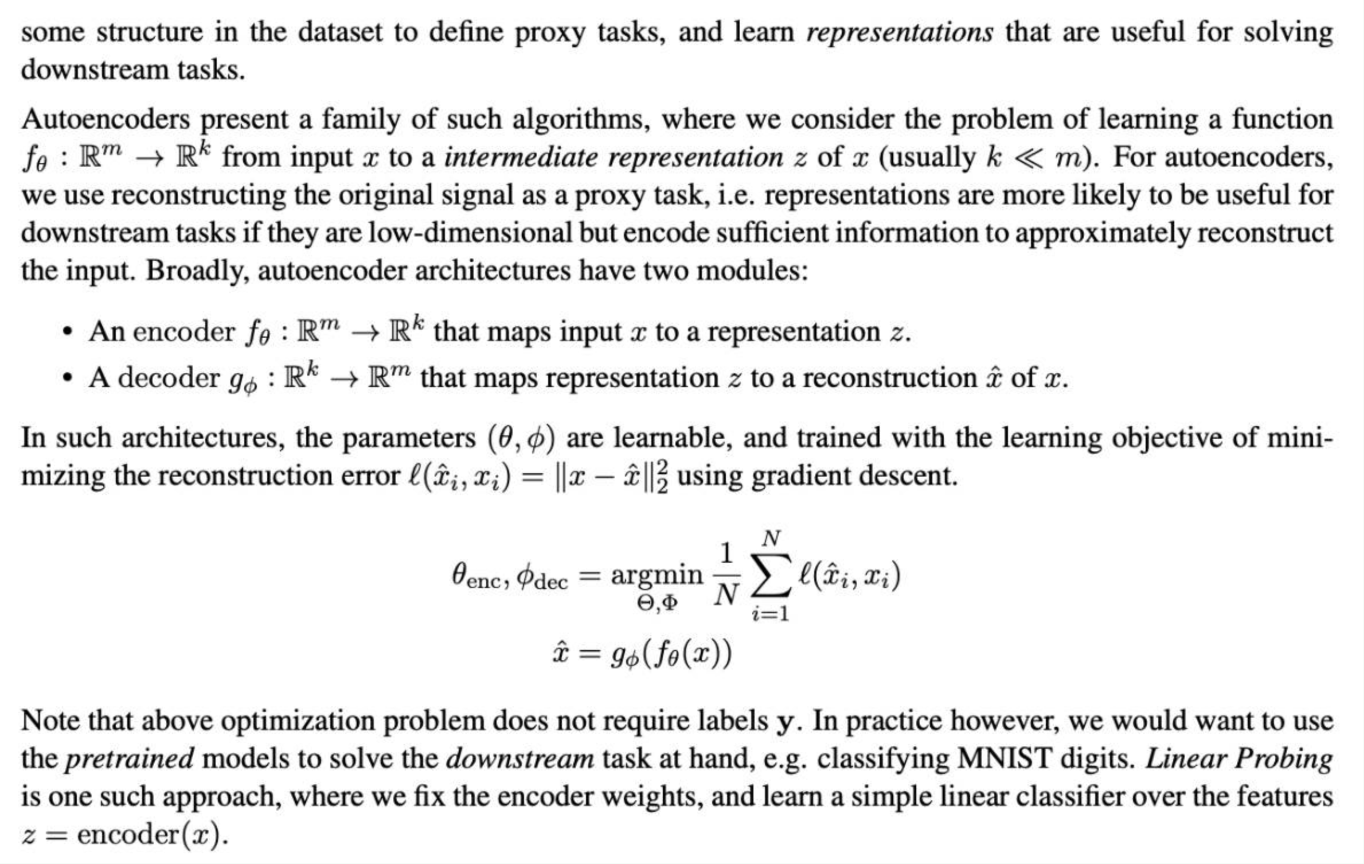
182 participation A



3. Auto-encoder : Learning without Labels

So far, with supervised learning algorithms we have used labelled datasets $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ to learn a mapping f_θ from input x to labels y. In this problem, we now consider algorithms for *unsupervised learning*, where we are given only inputs x, but no labels y. At a high-level, unsupervised learning algorithms leverage



some structure in the dataset to define proxy tasks, and learn *representations* that are useful for solving downstream tasks.

Autoencoders present a family of such algorithms, where we consider the problem of learning a function $f_\theta : \mathbb{R}^m \to \mathbb{R}^k$ from input x to a *intermediate representation* z of x (usually $k \ll m$). For autoencoders, we use reconstructing the original signal as a proxy task, i.e. representations are more likely to be useful for downstream tasks if they are low-dimensional but encode sufficient information to approximately reconstruct the input. Broadly, autoencoder architectures have two modules:

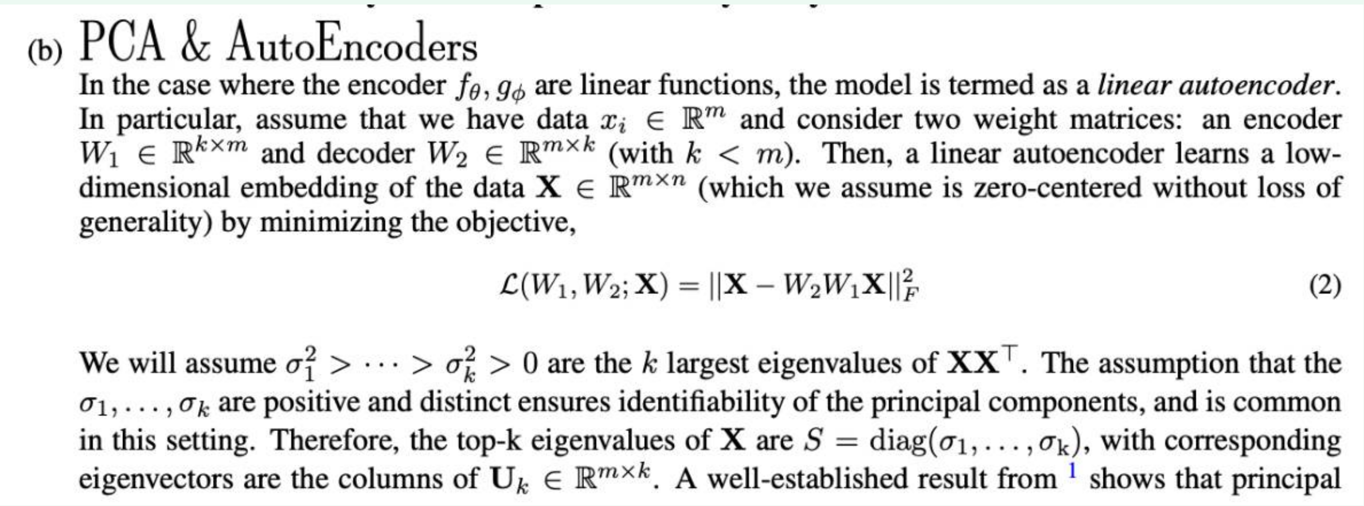
- An encoder $f_\theta : \mathbb{R}^m \to \mathbb{R}^k$ that maps input x to a representation z.
- A decoder $g_\phi : \mathbb{R}^k \to \mathbb{R}^m$ that maps representation z to a reconstruction $\hat{x}$ of x.

In such architectures, the parameters (θ, ϕ) are learnable, and trained with the learning objective of minimizing the reconstruction error $\ell(\hat{x}_i, x_i) = \|x - \hat{x}\|_2^2$ using gradient descent.

$$\theta_{\text{enc}}, \phi_{\text{dec}} = \underset{\Theta,\Phi}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{x}_i, x_i)$$

$$\hat{x} = g_\phi(f_\theta(x))$$

Note that above optimization problem does not require labels $\mathbf{y}$. In practice however, we would want to use the *pretrained* models to solve the *downstream* task at hand, e.g. classifying MNIST digits. *Linear Probing* is one such approach, where we fix the encoder weights, and learn a simple linear classifier over the features $z = \text{encoder}(x)$.

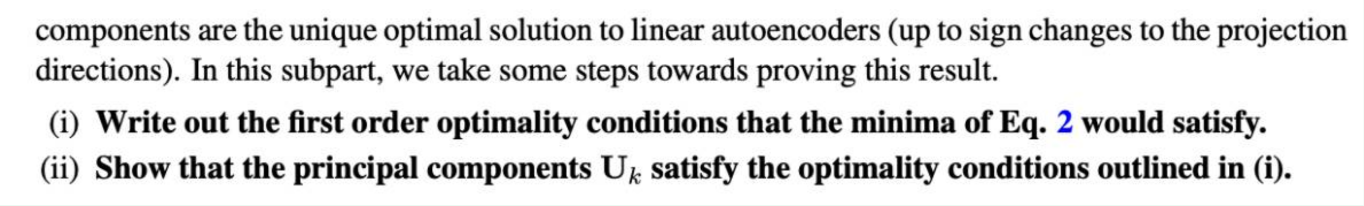


(b) PCA & AutoEncoders

In the case where the encoder f_θ, g_ϕ are linear functions, the model is termed as a *linear autoencoder*. In particular, assume that we have data $x_i \in \mathbb{R}^m$ and consider two weight matrices: an encoder $W_1 \in \mathbb{R}^{k \times m}$ and decoder $W_2 \in \mathbb{R}^{m \times k}$ (with $k < m$). Then, a linear autoencoder learns a low-dimensional embedding of the data $\mathbf{X} \in \mathbb{R}^{m \times n}$ (which we assume is zero-centered without loss of generality) by minimizing the objective,

$$\mathcal{L}(W_1, W_2; \mathbf{X}) = ||\mathbf{X} - W_2 W_1 \mathbf{X}||_F^2 \tag{2}$$

We will assume $\sigma_1^2 > \cdots > \sigma_k^2 > 0$ are the k largest eigenvalues of $\mathbf{X}\mathbf{X}^\top$. The assumption that the $\sigma_1, \ldots, \sigma_k$ are positive and distinct ensures identifiability of the principal components, and is common in this setting. Therefore, the top-k eigenvalues of $\mathbf{X}$ are $S = \text{diag}(\sigma_1, \ldots, \sigma_\text{k})$, with corresponding eigenvectors are the columns of $\mathbf{U}_k \in \mathbb{R}^{m \times k}$. A well-established result from [1] shows that principal



components are the unique optimal solution to linear autoencoders (up to sign changes to the projection directions). In this subpart, we take some steps towards proving this result.

(i) **Write out the first order optimality conditions that the minima of Eq. 2 would satisfy.**

(ii) **Show that the principal components $\mathbf{U}_k$ satisfy the optimality conditions outlined in (i).**

Answer part (i) by explicitely computing gradients step by step (no shortcuts), then for part (ii) explain whether the PCA solution using the top-k eigenvectors satisfies these optimality conditions and why.

We're given a **linear autoencoder** with

$$L(W_1, W_2; X) \;=\; \|X - W_2 W_1 X\|_F^2,$$

where $X \in \mathbb{R}^{m\times n}$, $W_1 \in \mathbb{R}^{k\times m}$ (encoder) and $W_2 \in \mathbb{R}^{m\times k}$ (decoder).

(i) First–order optimality conditions (gradients)

Write the loss as an element–wise sum:

$$L(W_1, W_2) = \sum_{i=1}^{m}\sum_{j=1}^{n}\Big(X_{ij} - (W_2 W_1 X)_{ij}\Big)^2.$$

Define

$$Y = W_2 W_1 X, \qquad E = X - Y.$$

So $L = \sum_{i,j} E_{ij}^2$.

Gradient w.r.t. W_2

Take a single element $W_{2,ab}$. Using the chain rule:

$$\frac{\partial L}{\partial W_{2,ab}} = \sum_{i,j} 2E_{ij}\,\frac{\partial E_{ij}}{\partial W_{2,ab}} = \sum_{i,j} 2E_{ij}\left(-\frac{\partial Y_{ij}}{\partial W_{2,ab}}\right).$$

Now expand Y:

$$Y_{ij} = (W_2 W_1 X)_{ij} = \sum_{r=1}^{k} W_{2,ir}(W_1 X)_{rj} = \sum_{r=1}^{k}\sum_{s=1}^{m} W_{2,ir} W_{1,rs} X_{sj}.$$

Differentiate w.r.t. $W_{2,ab}$; only terms with $i = a$ and $r = b$ survive:

$$\frac{\partial Y_{ij}}{\partial W_{2,ab}} = \delta_{ia}\sum_{s=1}^{m} W_{1,bs} X_{sj}.$$

Thus

$$\frac{\partial L}{\partial W_{2,ab}} = -2\sum_{j} E_{aj}\sum_{s} W_{1,bs} X_{sj}.$$

Recognize matrix products. First

$$(EX^T)_{as} = \sum_{j} E_{aj} X_{sj},$$

so

$$\sum_j E_{aj} \sum_s W_{1,bs} X_{sj} = \sum_s W_{1,bs} (EX^T)_{as} = (EX^T W_1^T)_{ab}.$$

Therefore, in matrix form

$$\nabla_{W_2} L = \frac{\partial L}{\partial W_2} = -2EX^T W_1^T.$$

Substitute $E = X - W_2 W_1 X$ and define $S := XX^T$:

$$\nabla_{W_2} L = -2(X - W_2 W_1 X) X^T W_1^T = -2SXW_?(\text{clean up}) = -2SX?\text{wait}$$

Let's do it cleanly:

$$\nabla_{W_2} L = -2XX^T W_1^T + 2W_2 W_1 XX^T W_1^T = -2SW_1^T + 2W_2 W_1 S W_1^T.$$

So the **first-order optimality condition** from this gradient is

$$\nabla_{W_2} L = 0 \quad \iff \quad (X - W_2 W_1 X) X^T W_1^T = 0 \quad \text{or} \quad W_2 W_1 S W_1^T = S W_1^T.$$

Gradient w.r.t. W_1

Now take an element $W_{1,cd}$. Again:

$$\frac{\partial L}{\partial W_{1,cd}} = \sum_{i,j} 2E_{ij} \left(-\frac{\partial Y_{ij}}{\partial W_{1,cd}} \right).$$

From the expression for Y_{ij},

$$\frac{\partial Y_{ij}}{\partial W_{1,cd}} = \sum_{r,s} W_{2,ir} X_{sj}\, \delta_{rc} \delta_{sd} = W_{2,ic} X_{dj}.$$

So

$$\frac{\partial L}{\partial W_{1,cd}} = -2 \sum_{i,j} E_{ij} W_{2,ic} X_{dj}.$$

First sum over i:

$$\sum_i E_{ij} W_{2,ic} = (W_2^T E)_{cj}.$$

Then multiply by X_{dj} and sum j:

$$\frac{\partial L}{\partial W_{1,cd}} = -2 \sum_j (W_2^T E)_{cj} X_{dj} = -2(W_2^T E X^T)_{cd}.$$

Thus, in matrix form,

$$\nabla_{W_1} L = -2W_2^T E X^T.$$

Substitute $E = X - W_2W_1X$ and again $S = XX^T$:

$$\nabla_{W_1} L = -2W_2^T XX^T + 2W_2^T W_2W_1XX^T = -2W_2^T S + 2W_2^T W_2W_1S.$$

So the **first-order optimality condition** from this gradient is

$$\nabla_{W_1} L = 0 \quad \iff \quad W_2^T(X - W_2W_1X)X^T = 0 \quad \text{or} \quad W_2^T W_2W_1S = W_2^T S.$$

Summary of first-order conditions

Let $S = XX^T$. A stationary point (W_1, W_2) must satisfy

$$\boxed{\begin{array}{lllll} (1) & (X - W_2W_1X)X^T W_1^T = 0 & \iff & W_2W_1SW_1^T = SW_1^T, \\ (2) & W_2^T(X - W_2W_1X)X^T = 0 & \iff & W_2^T W_2W_1S = W_2^T S. \end{array}}$$

These are the first-order optimality conditions requested in (i).

(ii) Does the PCA solution satisfy these conditions?

PCA on X gives us the eigendecomposition of

$$S = XX^T = U\Sigma^2U^T,$$

with eigenvalues $\sigma_1^2 > \sigma_2^2 > \cdots > \sigma_k^2 > 0$ and orthonormal eigenvectors $U = [u_1, \ldots, u_m]$. Let $U_k \in \mathbb{R}^{m\times k}$ be the matrix of the top-k eigenvectors.

The standard linear-PCA autoencoder chooses

$$W_1^{\text{PCA}} = U_k^T \in \mathbb{R}^{k\times m}, \qquad W_2^{\text{PCA}} = U_k \in \mathbb{R}^{m\times k}.$$

Then $W_2W_1 = U_kU_k^T$ is the orthogonal projector onto the top-k principal subspace.

We now check the two optimality conditions.

Condition (1)

Compute the left-hand side:

$$W_2W_1SW_1^T = U_kU_k^T SU_k.$$

Since $SU_k = U_k\Sigma_k^2$ (eigenvector relation), we get

$$U_kU_k^T SU_k = U_kU_k^T U_k\Sigma_k^2 = U_k\Sigma_k^2,$$

because $U_k^T U_k = I_k$.

The right-hand side of condition (1) is

$$SW_1^T = SU_k = U_k\Sigma_k^2.$$

Thus

$$W_2 W_1 S W_1^T = S W_1^T,$$

so the PCA choice satisfies condition (1).

Equivalently, in the original form

$$(X - W_2 W_1 X) X^T W_1^T = (X - U_k U_k^T X) X^T U_k = S U_k - U_k U_k^T S U_k = U_k \Sigma_k^2 - U_k \Sigma_k^2 = 0.$$

Condition (2)

First, note that

$$W_2^T W_2 = U_k^T U_k = I_k.$$

So the left-hand side becomes

$$W_2^T W_2 W_1 S = I_k W_1 S = W_1 S = U_k^T S.$$

For a symmetric matrix S, the left and right eigenvectors coincide. Since $S u_i = \sigma_i^2 u_i$, we also have $u_i^T S = \sigma_i^2 u_i^T$. Stacking the top-$k$ eigenvectors,

$$U_k^T S = \Sigma_k^2 U_k^T.$$

Now the right-hand side of condition (2) is

$$W_2^T S = U_k^T S = \Sigma_k^2 U_k^T.$$

Thus

$$W_2^T W_2 W_1 S = W_2^T S,$$

so condition (2) is also satisfied.

In the original error form,

$$W_2^T (X - W_2 W_1 X) X^T = U_k^T (X - U_k U_k^T X) X^T = U_k^T S - U_k^T U_k U_k^T S = U_k^T S - U_k^T S = 0.$$

Conclusion

- The gradients (derived elementwise) are

  $$\nabla_{W_2} L = -2(X - W_2 W_1 X) X^T W_1^T, \qquad \nabla_{W_1} L = -2 W_2^T (X - W_2 W_1 X) X^T,$$

  leading to first-order optimality conditions $(X - W_2 W_1 X) X^T W_1^T = 0$ and $W_2^T (X - W_2 W_1 X) X^T = 0$.
- The PCA solution $W_1 = U_k^T$, $W_2 = U_k$ (projection onto the span of the top-k eigenvectors of $X X^T$) satisfies both conditions, because the residual $X - U_k U_k^T X$ is orthogonal to the principal subspace, and the eigenvector relations $S U_k = U_k \Sigma_k^2$ and $U_k^T S = \Sigma_k^2 U_k^T$ make the two equations hold exactly.

Can you explain why U_k^T U_k = I_k implies SU_k = U_k Sigma_k^2? I'm thinking it's because once we substitute S into the equation, U_k^T U_k results in a rectangular identity matrix, meaning we're left with just the top k elements (hence U_k and Sigma_k^2 instead of just U * Sigma^2)?

Good question — the implication **doesn't** come from $U_k^T U_k = I_k$ *by itself*; it comes from combining:

1. the eigendecomposition $S = U\Sigma^2 U^T$, and
2. the orthonormality of U (which gives you the “rectangular identity” you're thinking of).

Let me walk it slowly in the block-matrix way you're hinting at.

Setup

We have the eigendecomposition of $S \in \mathbb{R}^{m\times m}$:

$$S = U\Sigma^2 U^T,$$

where

- $U \in \mathbb{R}^{m\times m}$ is orthogonal, so $U^T U = UU^T = I_m$,
- $\Sigma^2 = \operatorname{diag}(\sigma_1^2, \dots, \sigma_m^2)$.

Partition U into the top-k eigenvectors and the rest:

$$U = [\,U_k \;\; U_\perp\,], \quad U_k \in \mathbb{R}^{m\times k},\; U_\perp \in \mathbb{R}^{m\times(m-k)}.$$

Similarly, split Σ^2 into blocks:

$$\Sigma^2 = \begin{bmatrix} \Sigma_k^2 & 0 \\ 0 & \Sigma_\perp^2 \end{bmatrix}, \quad \Sigma_k^2 = \operatorname{diag}(\sigma_1^2, \dots, \sigma_k^2).$$

Where the “rectangular identity” appears

From orthonormality:

$$U^T U = I_m \quad \Rightarrow \quad \begin{bmatrix} U_k^T \\ U_\perp^T \end{bmatrix} [\,U_k \;\; U_\perp\,] = \begin{bmatrix} U_k^T U_k & U_k^T U_\perp \\ U_\perp^T U_k & U_\perp^T U_\perp \end{bmatrix} = \begin{bmatrix} I_k & 0 \\ 0 & I_{m-k} \end{bmatrix}.$$

From the top-left and bottom-left blocks we get:

- $U_k^T U_k = I_k$,
- $U_\perp^T U_k = 0$.

Now multiply U^T **by** U_k:

$$U^T U_k = \begin{bmatrix} U_k^T \\ U_\perp^T \end{bmatrix} U_k = \begin{bmatrix} U_k^T U_k \\ U_\perp^T U_k \end{bmatrix} = \begin{bmatrix} I_k \\ 0 \end{bmatrix}.$$

This is exactly the “rectangular identity” you were intuitively thinking about: a tall $m \times k$ matrix whose top block is I_k and bottom block is zeros.

So, **the key matrix is $U^T U_k = \begin{bmatrix} I_k \\ 0 \end{bmatrix}$**, not just $U_k^T U_k = I_k$ in isolation.

Deriving $SU_k = U_k \Sigma_k^2$

Now compute SU_k using the eigendecomposition:

$$\begin{aligned} SU_k &= (U\Sigma^2 U^T)U_k \\ &= U\Sigma^2(U^T U_k). \end{aligned}$$

We just showed $U^T U_k = \begin{bmatrix} I_k \\ 0 \end{bmatrix}$, so:

$$\Sigma^2(U^T U_k) = \begin{bmatrix} \Sigma_k^2 & 0 \\ 0 & \Sigma_\perp^2 \end{bmatrix} \begin{bmatrix} I_k \\ 0 \end{bmatrix} = \begin{bmatrix} \Sigma_k^2 \\ 0 \end{bmatrix}.$$

Therefore

$$SU_k = U \begin{bmatrix} \Sigma_k^2 \\ 0 \end{bmatrix} = [\, U_k \;\; U_\perp \,] \begin{bmatrix} \Sigma_k^2 \\ 0 \end{bmatrix} = U_k \Sigma_k^2.$$

That’s the desired relation.

Intuition in words

- U is orthogonal, so multiplying by U^T “expresses” a vector in the eigenbasis.
- U_k consists of the first k eigenvectors, so when you project them into the full eigenbasis by U^T, you just get the first k standard basis vectors stacked over zeros — that’s your “rectangular identity.”
- Σ^2 then scales those basis vectors by their eigenvalues (only the top k are nonzero in that block), producing $[\Sigma_k^2; 0]$.
- Multiplying back by U maps this back into the original space, giving exactly $U_k \Sigma_k^2$.

So your intuition about “we’re left with just the top-k elements (hence U_k and Σ_k^2)” is right — the precise math mechanism is $U^T U_k = [I_k; 0]$ coming from orthonormality, plus the block-diagonal structure of Σ^2.

can you try showing that the principal components satisfy the optimality conditions by utilizing transformations V_1 = W_1 U and V_2 = U^T W_2? Then, try re-writing the optimality conditions with these and argue why it must hold. Additionally, try moving all terms to the left side and equating it to 0 (while also doing some factoring).

Nice, this is exactly the right way to think about it. Let's do the change of variables carefully and then plug in the PCA weights.

---

1. Original optimality conditions

From the gradients we had:

$$\nabla_{W_2} L = -2(X - W_2W_1X)X^TW_1^T, \qquad \nabla_{W_1} L = -2W_2^T(X - W_2W_1X)X^T.$$

Setting these to zero and using $S = XX^T$ gives the first-order conditions

$$\begin{aligned} &(\mathrm{A}) \quad (X - W_2W_1X)X^TW_1^T = 0 \iff (I - W_2W_1)SW_1^T = 0, \\ &(\mathrm{B}) \quad W_2^T(X - W_2W_1X)X^T = 0 \iff W_2^T(I - W_2W_1)S = 0. \end{aligned}$$

We also have the eigendecomposition

$$S = XX^T = U\Sigma^2U^T, \quad U \in \mathbb{R}^{m\times m} \text{ orthogonal}, \ \Sigma^2 = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_m^2).$$

---

2. Change of variables with V_1, V_2

Define

$$V_1 := W_1U \quad (k \times m), \qquad V_2 := U^TW_2 \quad (m \times k).$$

Equivalently,

$$W_1 = V_1U^T, \qquad W_2 = UV_2.$$

Also note:

$$W_2W_1 = UV_2V_1U^T.$$

And since U is orthogonal, we can write

$$I = UIU^T,$$

so

$$I - W_2W_1 = UIU^T - UV_2V_1U^T = U(I - V_2V_1)U^T.$$

We'll now rewrite the optimality conditions (A) and (B) in terms of V_1, V_2.

2.1 Condition (A) in the eigenbasis

Start from

$$(I - W_2W_1)SW_1^T = 0.$$

Substitute $S = U\Sigma^2U^T,\ W_1^T = UV_1^T,\ I - W_2W_1 = U(I - V_2V_1)U^T$:

$$\begin{aligned} 0 &= (I - W_2W_1)SW_1^T \\ &= U(I - V_2V_1)U^T \cdot U\Sigma^2U^T \cdot UV_1^T \\ &= U(I - V_2V_1)\Sigma^2V_1^T. \end{aligned}$$

Multiply on the left by U^T (invertible), which gives an equivalent condition:

$$\boxed{(I - V_2V_1)\Sigma^2V_1^T = 0.}$$

If you "move all terms to the left" and factor, this is

$$\Sigma^2V_1^T - V_2V_1\Sigma^2V_1^T = 0 \quad \Longleftrightarrow \quad (I - V_2V_1)\Sigma^2V_1^T = 0.$$

2.2 Condition (B) in the eigenbasis

Start from

$$W_2^T(I - W_2W_1)S = 0.$$

Substitute $W_2^T = V_2^TU^T$ and the same expressions as before:

$$\begin{aligned} 0 &= W_2^T(I - W_2W_1)S \\ &= V_2^TU^T \cdot U(I - V_2V_1)U^T \cdot U\Sigma^2U^T \\ &= V_2^T(I - V_2V_1)\Sigma^2U^T. \end{aligned}$$

Multiply on the right by U (invertible) to get an equivalent condition:

$$\boxed{V_2^T(I - V_2V_1)\Sigma^2 = 0.}$$

Again, moving everything to the left and factoring:

$$V_2^T\Sigma^2 - V_2^TV_2V_1\Sigma^2 = 0 \quad \Longleftrightarrow \quad V_2^T(I - V_2V_1)\Sigma^2 = 0.$$

So in the eigenbasis of S, the stationarity conditions are:

$$\begin{cases} (I - V_2V_1)\Sigma^2V_1^T = 0 \\ V_2^T(I - V_2V_1)\Sigma^2 = 0 \end{cases}$$

with all terms on the left-hand side.

---

3. Plugging in the PCA solution

For the PCA autoencoder, we choose

$$W_1 = U_k^T, \qquad W_2 = U_k,$$

where U_k contains the top k eigenvectors (columns of U).

Let's compute V_1 and V_2:

- $V_1 = W_1 U = U_k^T U$.
  If we write $U = [\,U_k \;\; U_\perp\,]$, then

  $$U_k^T U = [\,I_k \;\; 0\,] \quad \Rightarrow \quad V_1 = [\,I_k \;\; 0\,]$$

  (a $k \times m$ matrix that picks the first k coordinates).
- $V_2 = U^T W_2 = U^T U_k$:

  $$U^T U_k = \begin{bmatrix} I_k \\ 0 \end{bmatrix} \quad \Rightarrow \quad V_2 = \begin{bmatrix} I_k \\ 0 \end{bmatrix}$$

  (an $m \times k$ matrix embedding k-dim vectors into the first k coordinates).

Now compute

$$V_2 V_1 = \begin{bmatrix} I_k \\ 0 \end{bmatrix} [\,I_k \;\; 0\,] = \begin{bmatrix} I_k & 0 \\ 0 & 0 \end{bmatrix} =: P,$$

which is the projector onto the first k coordinates in this eigenbasis.

Also, in this basis

$$\Sigma^2 = \operatorname{diag}(\sigma_1^2, \dots, \sigma_k^2, \sigma_{k+1}^2, \dots, \sigma_m^2) = \begin{bmatrix} \Sigma_k^2 & 0 \\ 0 & \Sigma_\perp^2 \end{bmatrix}.$$

3.1 Check condition (A)

We need $(I - V_2 V_1)\Sigma^2 V_1^T = 0$.

First compute V_1^T:

$$V_1^T = \begin{bmatrix} I_k \\ 0 \end{bmatrix}.$$

Then

$$\Sigma^2 V_1^T = \begin{bmatrix} \Sigma_k^2 & 0 \\ 0 & \Sigma_\perp^2 \end{bmatrix} \begin{bmatrix} I_k \\ 0 \end{bmatrix} = \begin{bmatrix} \Sigma_k^2 \\ 0 \end{bmatrix}.$$

Next,

$$I - V_2 V_1 = I - P = \begin{bmatrix} 0 & 0 \\ 0 & I_{m-k} \end{bmatrix}.$$

Multiply:

$$(I - V_2V_1)\Sigma^2V_1^T = \begin{bmatrix} 0 & 0 \\ 0 & I_{m-k} \end{bmatrix} \begin{bmatrix} \Sigma_k^2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0.$$

So condition (A) holds.

3.2 Check condition (B)

We need $V_2^T(I - V_2V_1)\Sigma^2 = 0$.
First compute $(I - V_2V_1)\Sigma^2$:

$$(I - V_2V_1)\Sigma^2 = \begin{bmatrix} 0 & 0 \\ 0 & I_{m-k} \end{bmatrix} \begin{bmatrix} \Sigma_k^2 & 0 \\ 0 & \Sigma_\perp^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & \Sigma_\perp^2 \end{bmatrix}.$$

Now $V_2^T = [\, I_k \;\; 0 \,]$, so

$$V_2^T(I - V_2V_1)\Sigma^2 = [\, I_k \;\; 0 \,] \begin{bmatrix} 0 & 0 \\ 0 & \Sigma_\perp^2 \end{bmatrix} = [\, 0 \;\; 0 \,] = 0.$$

So condition (B) also holds.

---

4. Intuition in this transformed view

- In the eigenbasis of S, S becomes diagonal (Σ^2), and the PCA choice simply makes

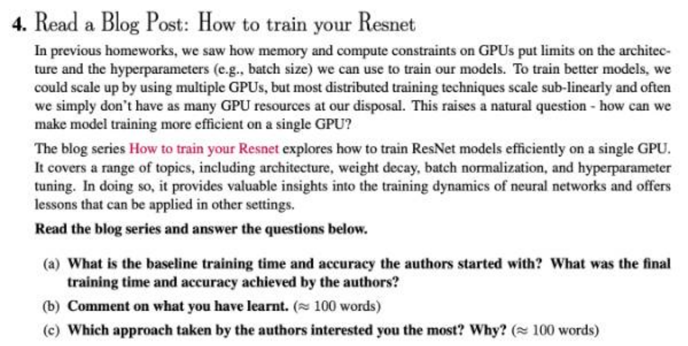
  $$V_1 = [I_k \; 0], \quad V_2 = [I_k; 0],$$

  so V_2V_1 is just “zero out all but the first k coordinates”.
- The optimality conditions say: after applying $(I - V_2V_1)$, the “energy” Σ^2 in the directions selected by V_1 must vanish (first condition), and similarly when viewed from the left by V_2^T (second condition).
- Because <PCA> chooses exactly the top k coordinates and V_2V_1 is the projector onto those coordinates, these conditions reduce to multiplying by a matrix that zeroes out exactly the block where $\Sigma^2V_1^T$ or Σ^2 has support, yielding zero.

So, using the V_1, V_2 transformation, we rewrote the first-order conditions as

$$(I - V_2V_1)\Sigma^2V_1^T = 0, \qquad V_2^T(I - V_2V_1)\Sigma^2 = 0,$$

and explicitly checked that the PCA choice (top-k eigenvectors) satisfies both.



4. Read a Blog Post: How to train your Resnet

In previous homeworks, we saw how memory and compute constraints on GPUs put limits on the architecture and the hyperparameters (e.g., batch size) we can use to train our models. To train better models, we could scale up by using multiple GPUs, but most distributed training techniques scale sub-linearly and often we simply don't have as many GPU resources at our disposal. This raises a natural question - how can we make model training more efficient on a single GPU?

The blog series How to train your Resnet explores how to train ResNet models efficiently on a single GPU. It covers a range of topics, including architecture, weight decay, batch normalization, and hyperparameter tuning. In doing so, it provides valuable insights into the training dynamics of neural networks and offers lessons that can be applied in other settings.

Read the blog series and answer the questions below.

(a) **What is the baseline training time and accuracy the authors started with? What was the final training time and accuracy achieved by the authors?**

(b) **Comment on what you have learnt.** (≈ 100 words)

(c) **Which approach taken by the authors interested you the most? Why?** (≈ 100 words)

resnet training.pdf
PDF

Can you answer these questions based on the blog series pdf I pasted? Specifically, what is the baseline training time and accuracy the authors started with? What was the final training time and accuracy achieved by the authors? Answer parts b and c as well

(a) **Baseline vs final time/accuracy**

From the reproduced DAWNBench setup, the authors' *starting baseline* on a single V100 was **94% test accuracy in 356 seconds** (3/5 runs reached 94% in 356 s). resnet training

After simplifying the network (removing the duplicated BatchNorm–ReLU), and then progressively optimizing data preprocessing and dataloader overhead (pre-doing common transforms, reducing worker processes, batching RNG calls, and finally doing augmentation on the main thread), they reached a **final training time of 297 seconds** while still achieving **94% test accuracy**. resnet training

---

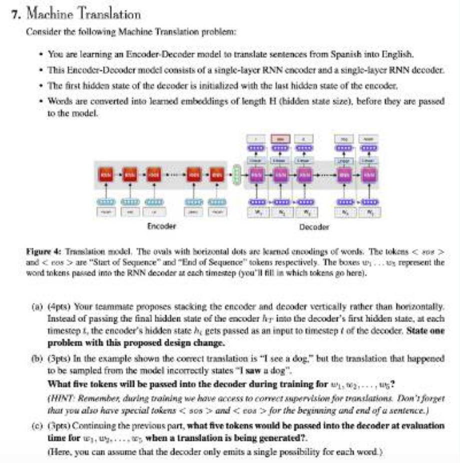
(b) **What I learned (~100 words)**

The post really drove home that training speed isn't only about model architecture or FLOPs—it's often bottlenecked by the *input pipeline*. I learned how things like redundant layers, slightly odd learning-rate schedules, and especially inefficient preprocessing can quietly waste a lot of time. The authors show that doing common transforms once, reducing dataloader workers, and batching random-number generation can shave tens of seconds off a relatively small CIFAR-10 run without hurting accuracy. More broadly, I came away with the lesson that profiling the entire training stack (CPU, data loading, augmentation) is essential before blaming the GPU or the model.

(c) **Most interesting approach (~100 words)**

The most interesting part to me was the systematic attack on preprocessing and randomness overhead. The idea of precomputing all "static" transforms once, then generating augmentation randomness in *bulk* at the start of each epoch, feels very elegant: it keeps the stochasticity you need for good generalization but drastically reduces per-batch Python and RNG overhead. I also liked the counterintuitive result that, after these changes, even a *single* dataloader worker became unnecessary—the GPU was finally the slow part again. It's a nice example of turning a fuzzy "data loading is slow" complaint into concrete, measurable optimizations.

Sources



7. Machine Translation

Consider the following Machine Translation problem:

- You are learning an Encoder-Decoder model to translate sentences from Spanish into English.
- This Encoder-Decoder model consists of a single-layer RNN encoder and a single-layer RNN decoder.
- The first hidden state of the decoder is initialized with the last hidden state of the encoder.
- Words are converted into learned embeddings of length H (hidden state size), before they are passed to the model.

Figure 4: Translation model. The ovals with horizontal dots are learned encodings of words. The tokens $< sos >$ and $< eos >$ are "Start of Sequence" and "End of Sequence" tokens respectively. The boxes $w_1 \ldots w_5$ represent the word tokens passed into the RNN decoder at each timestep (you'll fill in which tokens go here).

(a) (4pts) Your teammate proposes stacking the encoder and decoder vertically rather than horizontally. Instead of passing the final hidden state of the encoder h_T into the decoder's first hidden state, at each timestep t, the encoder's hidden state h_t gets passed as an input to timestep t of the decoder. **State one problem with this proposed design change.**

(b) (3pts) In the example shown the correct translation is "I see a dog," but the translation that happened to be sampled from the model incorrectly states "I **saw** a dog".
What five tokens will be passed into the decoder during training for $w_1, w_2, \ldots, w_5$**?**
(HINT: Remember, during training we have access to correct supervision for translations. Don't forget that you also have special tokens $< sos >$ *and* $< eos >$ *for the beginning and end of a sentence.)*

(c) (3pts) Continuing the previous part, **what five tokens would be passed into the decoder at evaluation time for** $w_1, w_2, \ldots, w_5$ **when a translation is being generated?**.
(Here, you can assume that the decoder only emits a single possibility for each word.)

Answer the questions on the screenshot. Here are the questions:

part a: Your teammate proposes stacking the encoder and decoder vertically rather than horizontally.
Instead of passing the final hidden state of the encoder h_T into the decoder's first hidden state, at each
timestep t, the encoder's hidden state h_t gets passed as an input to timestep t of the decoder. State one
problem with this proposed design change.

part b: In the example shown the correct translation is "I see a dog," but the translation that happened
to be sampled from the model incorrectly states "I saw a dog".
What five tokens will be passed into the decoder during training for w_1, w_2, . . . , w_5?
(HINT: Remember, during training we have access to correct supervision for translations. Don't forget
that you also have special tokens < sos > and < eos > for the beginning and end of a sentence.)

part c: Continuing the previous part, what five tokens would be passed into the decoder at evaluation
time for w_1, w_2, . . . , w_5 when a translation is being generated?.
(Here, you can assume that the decoder only emits a single possibility for each word.)

Here's how I'd answer each part:

(a) Problem with the proposed design

If at decoder timestep t you feed in encoder hidden state h_t (instead of a single summary h_T), then you're implicitly assuming a **one-to-one alignment between source position t and target position t** and that the source and target sequences have the same length.

But in machine translation, sentences often:

- have **different lengths**, and
- **reorder words/phrases** between languages.

With this design the decoder can't easily produce more (or fewer) words than the source sentence or reorder them, making it unsuitable for general translation.

(b) Tokens passed to the decoder during training

During training we use **teacher forcing**: at each timestep the decoder is fed the *correct previous token*, not what it predicted.

Correct translation:
`<sos>  I  see  a  dog  <eos>`

So the five input tokens $w_1, \ldots, w_5$ to the decoder are:

1. $w_1 = \texttt{<sos>}$
2. $w_2 = \texttt{I}$
3. $w_3 = \texttt{see}$
4. $w_4 = \texttt{a}$
5. $w_5 = \texttt{dog}$

(The model is trained to output $I, \text{see}, a, dog, \texttt{<eos>}$ at those steps.)

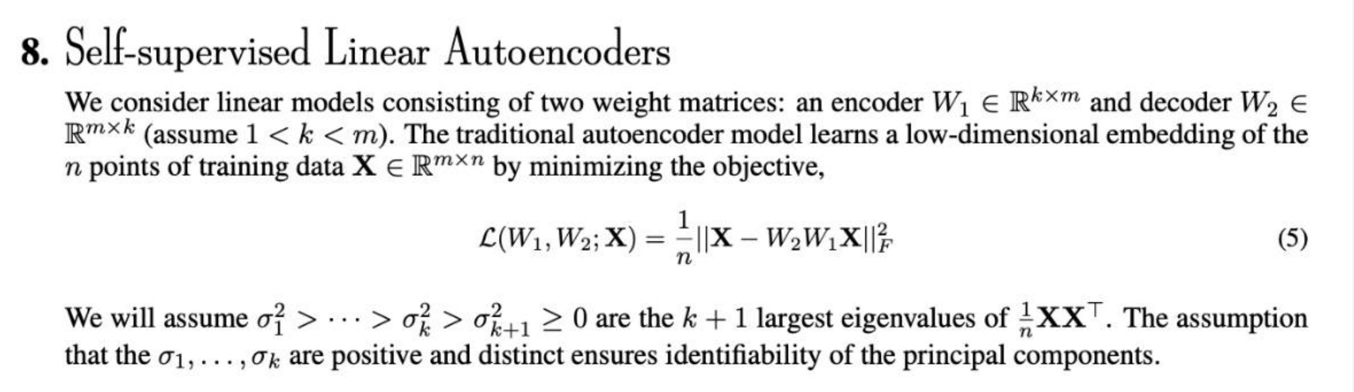
(c) Tokens passed to the decoder at evaluation time

At evaluation we don't have the ground-truth translation, so the decoder is fed its **own previous predictions** (greedy decoding, one possibility per word).

Sampled translation: "I saw a dog"
Sequence with special tokens:
`<sos>  I  saw  a  dog  <eos>`

So the five tokens $w_1, \ldots, w_5$ input to the decoder during generation are:

1. $w_1 = \texttt{<sos>}$
2. $w_2 = \texttt{I}$
3. $w_3 = \texttt{saw}$
4. $w_4 = \texttt{a}$
5. (w_5 = \texttt{dog}`

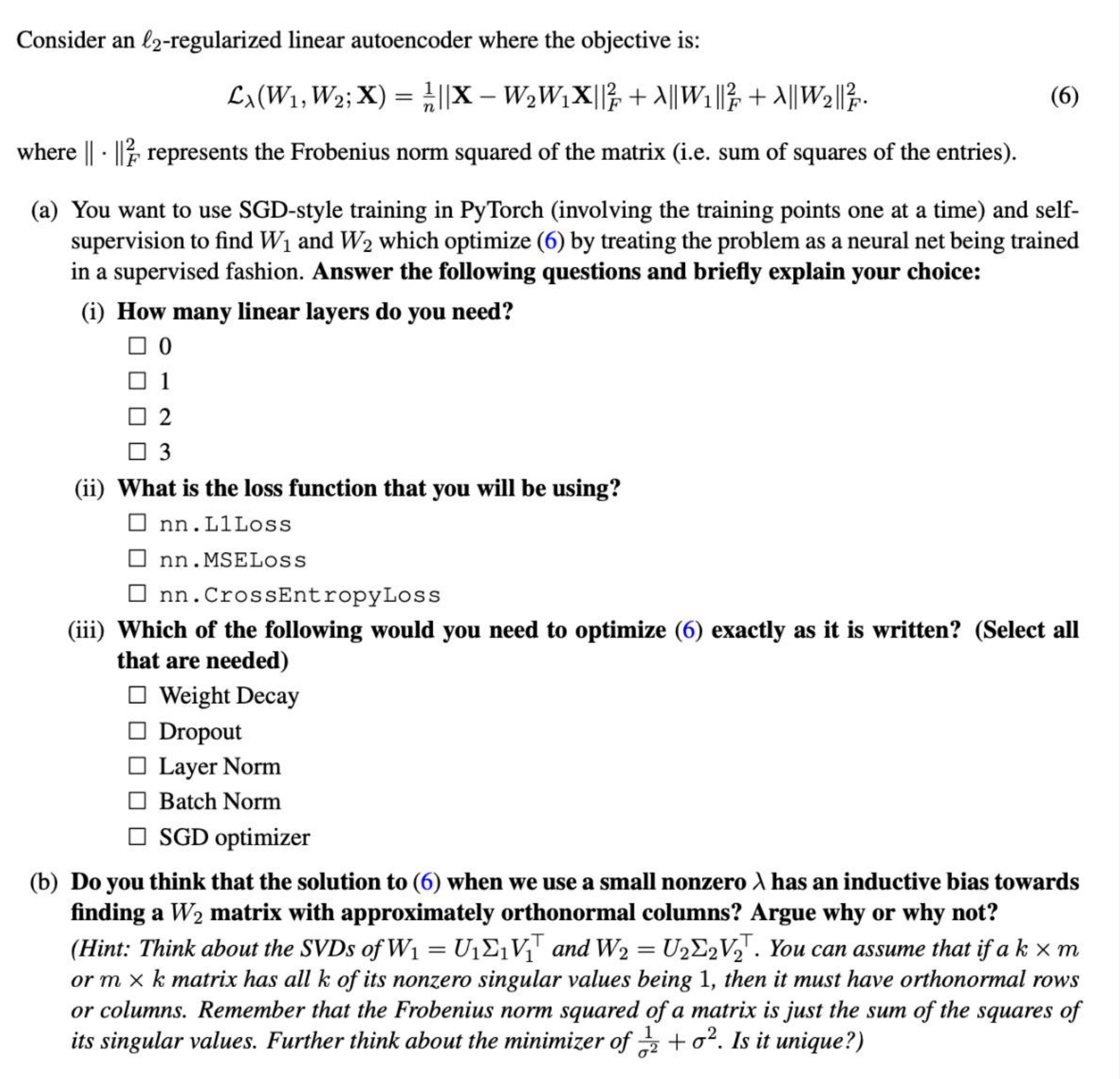


8. Self-supervised Linear Autoencoders

We consider linear models consisting of two weight matrices: an encoder $W_1 \in \mathbb{R}^{k\times m}$ and decoder $W_2 \in \mathbb{R}^{m\times k}$ (assume $1 < k < m$). The traditional autoencoder model learns a low-dimensional embedding of the n points of training data $\mathbf{X} \in \mathbb{R}^{m\times n}$ by minimizing the objective,

$$\mathcal{L}(W_1, W_2; \mathbf{X}) = \frac{1}{n}||\mathbf{X} - W_2 W_1 \mathbf{X}||_F^2 \tag{5}$$

We will assume $\sigma_1^2 > \cdots > \sigma_k^2 > \sigma_{k+1}^2 \geq 0$ are the $k+1$ largest eigenvalues of $\frac{1}{n}\mathbf{X}\mathbf{X}^\top$. The assumption that the $\sigma_1, \ldots, \sigma_k$ are positive and distinct ensures identifiability of the principal components.



Consider an ℓ_2-regularized linear autoencoder where the objective is:

$$\mathcal{L}_\lambda(W_1, W_2; \mathbf{X}) = \tfrac{1}{n}||\mathbf{X} - W_2 W_1 \mathbf{X}||_F^2 + \lambda\|W_1\|_F^2 + \lambda\|W_2\|_F^2. \tag{6}$$

where $\|\cdot\|_F^2$ represents the Frobenius norm squared of the matrix (i.e. sum of squares of the entries).

(a) You want to use SGD-style training in PyTorch (involving the training points one at a time) and self-supervision to find W_1 and W_2 which optimize (6) by treating the problem as a neural net being trained in a supervised fashion. **Answer the following questions and briefly explain your choice:**

(i) **How many linear layers do you need?**

☐ 0
☐ 1
☐ 2
☐ 3

(ii) **What is the loss function that you will be using?**

☐ `nn.L1Loss`
☐ `nn.MSELoss`
☐ `nn.CrossEntropyLoss`

(iii) **Which of the following would you need to optimize (6) exactly as it is written? (Select all that are needed)**

☐ Weight Decay
☐ Dropout
☐ Layer Norm
☐ Batch Norm
☐ SGD optimizer

(b) **Do you think that the solution to (6) when we use a small nonzero λ has an inductive bias towards finding a W_2 matrix with approximately orthonormal columns? Argue why or why not?**

(Hint: Think about the SVDs of $W_1 = U_1\Sigma_1 V_1^\top$ and $W_2 = U_2\Sigma_2 V_2^\top$. You can assume that if a $k \times m$ or $m \times k$ matrix has all k of its nonzero singular values being 1, then it must have orthonormal rows or columns. Remember that the Frobenius norm squared of a matrix is just the sum of the squares of its singular values. Further think about the minimizer of $\frac{1}{\sigma^2} + \sigma^2$. Is it unique?)

Solve these problems, which I have listed below for your reference:

You want to use SGD-style training in PyTorch (involving the training points one at a time) and selfsupervision to find W_1 and W_2 which optimize (6) by treating the problem as a neural net being trained
in a supervised fashion. Answer the following questions and briefly explain your choice:
(i) How many linear layers do you need?

(ii) What is the loss function that you will be using?
(iii) Which of the following would you need to optimize (6) exactly as it is written? (Select all that are needed)

Do you think that the solution to (6) when we use a small nonzero λ has an inductive bias towards finding a W2 matrix with approximately orthonormal columns? Argue why or why not?

Let

$$\mathcal{L}_\lambda(W_1, W_2; X) = \frac{1}{n}\|X - W_2 W_1 X\|_F^2 + \lambda\|W_1\|_F^2 + \lambda\|W_2\|_F^2.$$

We want to implement this as a neural net in PyTorch and train it with SGD on examples $x \in \mathbb{R}^m$ (input and target both x).

(a) Network + training choices

(i) How many linear layers?

Two linear layers.

- One **encoder** layer with weight $W_1 \in \mathbb{R}^{k\times m}$: $z = W_1 x$.
- One **decoder** layer with weight $W_2 \in \mathbb{R}^{m\times k}$: $\hat{x} = W_2 z = W_2 W_1 x$.

No nonlinearities or extra layers appear in (6), so exactly two `nn.Linear` layers (and no bias terms if we want to match it *exactly*).

(ii) Loss function

We want the reconstruction term

$$\frac{1}{n}\|X - \hat{X}\|_F^2 = \frac{1}{n}\sum_i \|x_i - \hat{x}_i\|_2^2,$$

so on a single example we want the squared L_2 reconstruction error. That is exactly **mean-squared error**, so we use

```
nn.MSELoss
```

(with input $\hat{x}$, target x).

(iii) What is needed to optimize (6) *exactly as written*?

Among the given options we need:

- ✅ **Weight Decay** – to implement the $\lambda\|W_1\|_F^2 + \lambda\|W_2\|_F^2$ penalty (either by adding it to the loss manually or by using the `weight_decay=λ` argument in the optimizer).
- ✅ **SGD optimizer** – some SGD-style optimizer is required to actually minimize the loss by gradient descent.

We **do not** need:

- ❌ Dropout – would randomly zero activations and change the objective.
- ❌ Layer Norm / Batch Norm – introduce extra parameters and change the forward mapping; they don't appear in (6).

So: **Weight Decay + SGD optimizer** only.

(b) Does a small nonzero λ bias W_2 toward (approximately) orthonormal columns?

Yes, it does – there is an inductive bias toward W_2 having (approximately) orthonormal columns.

Sketch of argument using the hint:

1. For a *good* autoencoder with small λ, the reconstruction term forces
   $W_2 W_1 \approx P$,
   where P is the rank-k projector onto the top-k principal subspace (eigenvalues ≈ 1, others ≈ 0).
2. Any factorization of P can be written as

$$W_2 = U_k A, \quad W_1 = A^{-1} U_k^\top,$$

   where U_k are the top-k eigenvectors of $\frac{1}{n} X X^\top$, and $A \in \mathbb{R}^{k\times k}$ is invertible. All such pairs give *identical* reconstruction error because $W_2 W_1 = U_k A A^{-1} U_k^\top = U_k U_k^\top = P$.
3. The ℓ_2 regularizer breaks this degeneracy.
   Let the singular values of A be $\sigma_1, \dots, \sigma_k > 0$. Then

$$\|W_2\|_F^2 = \sum_i \sigma_i^2, \qquad \|W_1\|_F^2 = \sum_i \frac{1}{\sigma_i^2},$$

   so the regularization term becomes

$$\lambda\left(\|W_2\|_F^2 + \|W_1\|_F^2\right) = \lambda \sum_{i=1}^{k} \left(\sigma_i^2 + \frac{1}{\sigma_i^2}\right).$$

4. For each i, the function

$$f(\sigma) = \sigma^2 + \frac{1}{\sigma^2}$$

   has derivative $f'(\sigma) = 2\sigma - 2/\sigma^3$.
   Setting $f'(\sigma) = 0$ gives $\sigma^4 = 1 \Rightarrow \sigma = 1$ (for $\sigma > 0$).
   Moreover $f''(\sigma) = 2 + 6/\sigma^4 > 0$, so $\sigma = 1$ is the unique global minimizer.

Therefore the regularizer is minimized when **all singular values $\sigma_i = 1$**.

5. When all singular values of A are 1, A is orthogonal (up to rotations), so

$$W_2 = U_k A$$

has singular values all equal to 1 and hence **columns of W_2 are orthonormal**. Any orthogonal A just rotates the basis within the principal subspace but keeps $W_2^T W_2 = I_k$, so the solution is not unique, but all solutions have orthonormal columns.

6. With a *small* nonzero λ, the reconstruction term forces $W_2 W_1$ very close to P, while the regularizer then prefers factorizations of that projector where the singular values of W_2 (and W_1) are close to 1. Thus, in practice, we get a **strong bias toward W_2 having approximately orthonormal columns** (up to rotations within the subspace).

So: **yes**, the ℓ_2-regularized linear autoencoder has an inductive bias toward W_2 with (approximately) orthonormal columns, though the particular orthonormal basis inside the principal subspace is not unique.

2/2