EECS 182 Deep Neural Networks
Fall 2026 Gireeja Ranade

Homework 8

This homework is due on Wednesday, October 28, 2026, at 10:59PM.

1. Setting up sequence prediction as RNN

The idea of recurrent information processing is quite basic and predates deep neural networks. Consider the following formulation:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t \tag{1}$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \mathbf{w}_t \tag{2}$$

This recurrent form has $\mathbf{x}_t$ denoting the (soon to be hidden) state at timestep t, $\mathbf{y}_t$ is the measurement (or label), $\mathbf{u}_t$ is some driving noise (assume it to be zero-mean iid Gaussian) at timestep t, and $\mathbf{w}_t$ is the similarly zero-mean iid Gaussian observation noise at each timestep. Here $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ are the weights that determine the state evolution and observations at **all** time steps.

(a) In the case where all weights $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are just matrices and we have access to full trajectories of $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{y}_t)$, **how can you setup the problem of learning $\mathbf{A}, \mathbf{B}, \mathbf{C}$ from this training data?**

Solution:

When the entire trajectories are given, the problem is solvable with least squares. Both the ordinary least squares and ridge regression could work depending on the data.

Particularly, we learn A,B by least squares:

$$\begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_T \end{bmatrix} = \begin{bmatrix} x_1 & u_1 \\ x_2 & u_2 \\ \vdots & \vdots \\ x_{T-1} & u_{T-1} \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix}.$$

Learn C similarly:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_T \end{bmatrix} C.$$

(b) Assume that the $\mathbf{u}_t = \mathbf{0}$ for simplicity. We know from Kalman filtering that if we know the $\mathbf{A}, \mathbf{B}, \mathbf{C}$ matrices, it is possible to create a steady-state filter which approximately tracks that state $\mathbf{x}_t$ from (1) given only the sequence of observations $\mathbf{y}_t$ as long as the (A, C) matrix is observable — note "observability" here is a technical condition that is dual to "controllability" and says that $[C, CA, CA^2, \ldots]$ is full rank. Such a filter can be given in recursive form as:

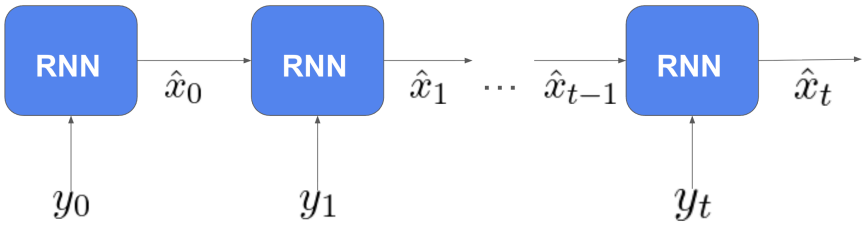
$$\widehat{\mathbf{x}}_t = \mathbf{A}'\widehat{\mathbf{x}}_{t-1} + \mathbf{B}'\mathbf{y}_t \tag{3}$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

Now, suppose we actually had training-time access to full trajectories of $(\mathbf{x}_t, \mathbf{y}_t)$ and want to learn $\mathbf{A}', \mathbf{B}'$ as in (3). **Comment on why simply using the approach of the previous part does not work and we can't use the ground-truth $\mathbf{x}_t$ to break the long chain of dependency in the state evolution of the filter across the sequence in the way that we were able to do for simple system identification with perfectly observed states.**

Solution: In the fully-observed system-ID problem of part (1), we simply used the known $\mathbf{x}_t$ values at each time step. However, for the filter we cannot simply put $\widehat{\mathbf{x}}_t$ in place of $\mathbf{x}_t$ in the training data. This is because the filter is a recursive model that depends on the previous state $\widehat{\mathbf{x}}_{t-1}$ to predict the next state $\widehat{\mathbf{x}}_t$.

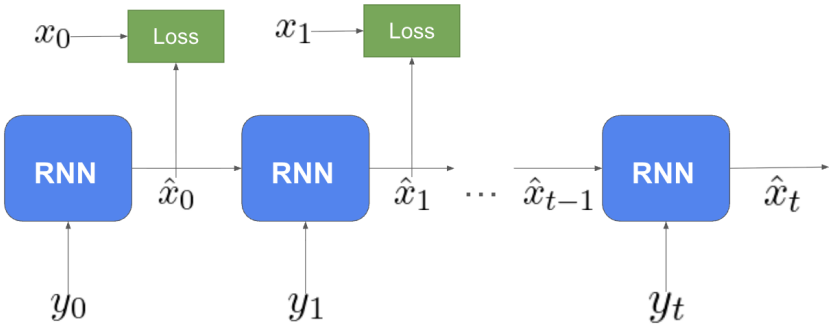
(c) Continuing the previous part (continue to assume that the $\mathbf{u}_t = \mathbf{0}$ for simplicity), **how can you train on full trajectories of $(\mathbf{x}_t, \mathbf{y}_t)$ in order to learn $\mathbf{A}', \mathbf{B}'$ from (3)?** Also, **modify the computation graph below to show what extra computations are required to compute the loss function.**



Note: in this case, for simplicity, do not attempt to setup any loss function involving trying to reconstruct the $\mathbf{y}_t$. Simply treat these as known inputs for this part.

While the filter doesn't see $\mathbf{x}_t$ at training time, we can define a loss function on how close $\widehat{\mathbf{x}}_t$ is to $\mathbf{x}_t$, and optimize the loss function to learn $\mathbf{A}'$ and $\mathbf{B}'$.

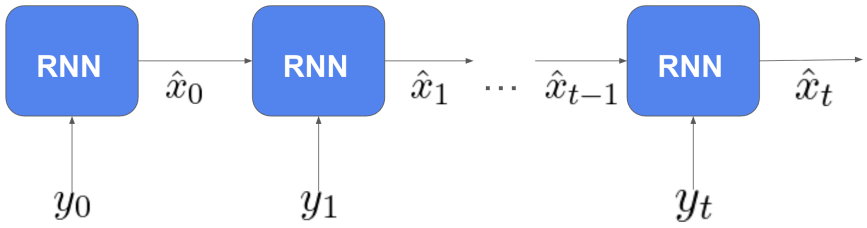
Solution: At every timestep, we can train the filter to minimize loss $||\mathbf{x}_t - \widehat{\mathbf{x}}_t||_2^2$. This bypasses the issue in the previous part because we generate $\widehat{\mathbf{x}}_t$ recursively using $\widehat{\mathbf{x}}_{t-1}$.



(d) Now consider the case where we only have measurements of $\mathbf{y}_t$ for all time steps. (Continue to assume that the $\mathbf{u}_t = \mathbf{0}$ for simplicity.) This is common in applications such as **object tracking**, where you only have a series of video frames, and the model has to track the movement of an object throughout

 All Rights Reserved. This may not be publicly shared without explicit permission.

the frames. **First, show how an RNN model can be setup to predict the next $\mathbf{y}_{t+1}$ in the sequence and how this RNN can be trained.** Afterwards, **modify the same computation graph to include the new loss function.**



Solution:

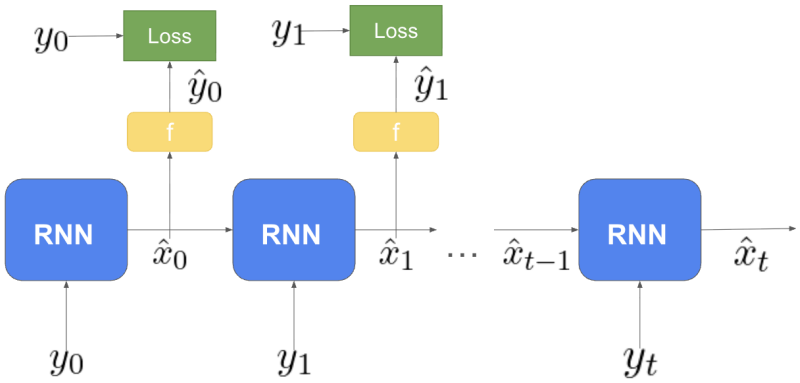
Since we assumed that the weights are timestep-invariant, RNN is a suitable model to learn the transitions given that both $\mathbf{u}_t$ and $\mathbf{x}_t$ are unknown. As we see in part (a), $\mathbf{y}_t$ is dependent on $\mathbf{u}_{t-1}$ and $\mathbf{x}_{t-1}$, we can rewrite the transition as the follows:

$$\mathbf{y}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) + \mathbf{w}_t$$

where f is a neural network that determines the transition. Here all we know is about $\mathbf{y}_t$. We can manipulate the algebra to show the dependency between $\mathbf{y}_t$ to make it into a one-step prediction setup:

$$\begin{aligned}
\mathbf{y}_t &= \mathbf{C}\mathbf{x}_t + \mathbf{w}_t \\
&= \mathbf{C}(\mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1}) + \mathbf{w}_t \\
&= \mathbf{C}(\mathbf{A} \cdot (\mathbf{C}^{-1}(\mathbf{y}_{t-1} - \mathbf{w}_{t-1})) + \mathbf{B}\mathbf{u}_{t-1}) + \mathbf{w}_t \\
&= \mathbf{C}\mathbf{A}\mathbf{C}^{-1}\mathbf{y}_{t-1} + \mathbf{C}\mathbf{B}\mathbf{u}_{t-1} + \text{noise}
\end{aligned}$$

which shows the dependency of $\mathbf{y}_t$ and the label at the previous timestep $\mathbf{y}_{t-1}$. This leads to the typical training paradigm of RNN, as depicted in the diagram below.



(e) To simplify computation, now assume that $\mathbf{w}_t = \mathbf{0}$. Given the input sequence $(\mathbf{u}_0, \cdots, \mathbf{u}_T)$ and output sequence $(\mathbf{y}_0, \cdots, \mathbf{y}_T)$, and assuming the initial state $\mathbf{x}_0 = 0$, **show that the output $\mathbf{y}_t$ can be**

 All Rights Reserved. This may not be publicly shared without explicit permission.

expressed as a *convolution* of the input sequence $\{\mathbf{u}_i\}_0^T$ with a kernel $K = \{K_i\}_0^T$:

$$\mathbf{y}_t = \sum_{i=0}^{T} K_{t-i}\mathbf{u}_i,$$

where any K_j for $j < 0$ is set to **0** to ensure causality. **Find K.**

Solution: From the recursion $x_{t+1} = Ax_t + Bu_t$, we get

$$x_1 = Bu_0, \quad x_2 = ABu_0 + Bu_1, \quad x_3 = A^2Bu_0 + ABu_1 + Bu_2, \dots$$

Hence,

$$y_0 = 0, \tag{4}$$
$$y_1 = CBu_0, \tag{5}$$
$$y_2 = CBu_1 + CABu_0, \tag{6}$$
$$y_3 = CBu_2 + CABu_1 + CA^2Bu_0, \tag{7}$$
$$\vdots \tag{8}$$

each y_t is given explicitly by summing all terms of the form $CA^{t-i-1}Bu_i$. This summation matches a discrete convolution $\{y\} = K * \{u\}$ with kernel K given by

$$K = (0, CB, CAB, CA^2B, \dots).$$

2. 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.

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

 All Rights Reserved. This may not be publicly shared without explicit permission.

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 $z = \text{encoder}(x)$.

(a) Designing AutoEncoders (Coding)

Please follow the instructions in this notebook. You will train autoencoders, denoising autoencoders, and masked autoencoders on a synthetic dataset and the MNIST dataset. Once you finished with the notebook,

- Answer the following questions in your submission of the written assignment:

(i) **Show your visualization** of the vanilla autoencoder with different latent representation sizes.

Solution:

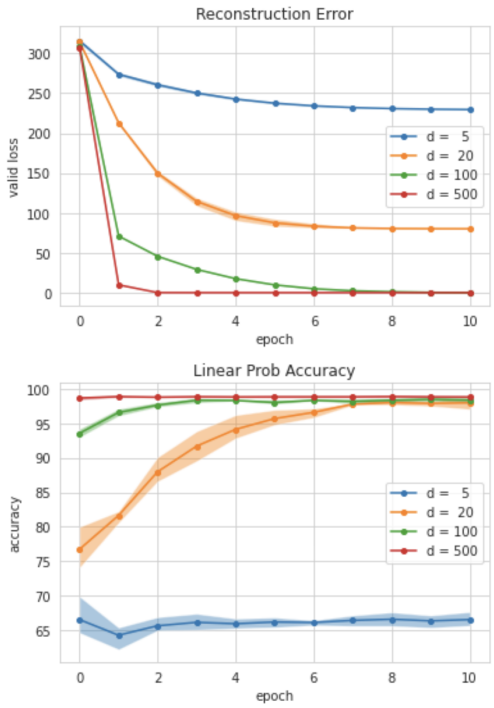


Figure 1: Visualization of the vanilla autoencoder with different latent representation sizes.

See Figure 1. Please refer to the solution notebook for the codes.

(ii) Based on your previous visualizations, answer this question: **How does changing the latent representation size of the autoencoder affect the model's performance in terms of reconstruction accuracy and linear probe accuracy? Why?**

Solution: Based on the given synthetic dataset, each data point has 100 dimensions, with 20 high-variance dimensions affecting the class label. The following observations can be made from the visualizations in Figure 1.

Firstly, the reconstruction error of the autoencoder decreases as the size of the latent representation increases. However, this reduction becomes marginal when the size of the latent representation exceeds the dimension of the data (100).

Secondly, the linear probe accuracy increases as the size of the latent representation increases. When the size of the latent representation is well above the dimension of the data (e.g., 500 vs.

 All Rights Reserved. This may not be publicly shared without explicit permission.

100), the linear probe accuracy is already close to 100% (about 98–99%) even without training the autoencoder; at size 100 it starts around 93–94% and reaches about 98% after training. However, when the size of the latent representation equals the number of interpretive dimensions (20), training the autoencoder becomes essential for achieving a high linear probe accuracy. The linear probe accuracy finally converges to $\geq$95% with training. On the other hand, if the size of the latent representation is as small as 5, it fails to capture all useful information in the input data, leading to significantly lower linear probe accuracy.

(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,

$$\mathcal{L}(W_1, W_2; \mathbf{X}) = ||\mathbf{X} - W_2 W_1 \mathbf{X}||_F^2 \tag{9}$$

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 $S = \mathrm{diag}(\sigma_1, \ldots, \sigma_\mathrm{k})$, 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. 13 would satisfy.**

Solution: We can compute the first order conditions for W_1 and W_2, respectively.
Important equations:

$$\|A\|_F^2 = tr(AA^T), \quad tr(ABC) = tr(BCA) = tr(CAB), \quad tr(A) = tr(A^T)$$

$$\nabla_A tr(AB) = B^T, \quad \nabla_B tr(AB) = A^T$$

Also, taking the matrix derivative of trace follows the product rule $(uv)' = u'v + uv'$. So for example (assuming all the matrices are of the right shape):

$$\begin{aligned}
\nabla_A tr(ABBA) &= \nabla_A tr(ABBC)|_{C=A} + \nabla_A tr(CBBA)|_{C=A} && \text{product rule of derivatives} \\
&= \nabla_A tr(ABBC)|_{C=A} + \nabla_A tr(ACBB)|_{C=A} && \text{cyclic rule} \\
&= (BBA)^T + (ABB)^T
\end{aligned}$$

Now just do the calculation.

$$\begin{aligned}
\nabla_{W_2} L &= \nabla_{W_2} tr(XX^T + W_2 W_1 XX^T W_1^T W_2^T - 2W_2 W_1 XX^T) \\
&= \cancel{\nabla_{W_2} tr(XX^T)} + \nabla_{W_2} tr(W_2 W_1 XX^T W_1^T W_2^T) - \nabla_{W_2} 2tr(W_2 W_1 XX^T) && \text{linearity} \\
&= 2(W_1 XX^T W_1^T W_2^T)^T - 2(W_1 XX^T)^T \\
&= 2(W_2 W_1 - I)XX^T W_1^T
\end{aligned}$$

[1] Baldi, Pierre, and Kurt Hornik. "Neural networks and principal component analysis: Learning from examples without local minima" (1989)

 All Rights Reserved. This may not be publicly shared without explicit permission.

Similarly calculate the other one:

$$\nabla_{W_1} L = 2W_2^T(W_2W_1 - I)XX^T$$

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

Solution: Do the SVD decomposition: $X = U\Sigma V^T$, and $D = \Sigma\Sigma^T$. Note that D is a diagonal matrix.

Then the first order condition states that

$$W_2^T(W_2W_1 - I)UDU^T = 0; \quad (W_2W_1 - I)UDU^TW_1^T = 0$$

Let $V_1 := W_1U, V_2 := U^TW_2$, then it further simplifies to

$$(V_2V_1 - I)DV_1^T = 0; \quad V_2^T(V_2V_1 - I)D = 0$$

If $W_1^T = W_2$ are the first k columns of U, then

$$V_1 = \left[\begin{array}{ccc|c} 1 & & & \\ & \ddots & & 0 \\ & & 1 & \end{array}\right] \tag{10}$$

and $V_2 = V_1^T$. Now direct computation finishes the proof.

3. 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,

$$\mathcal{L}(W_1, W_2; \mathbf{X}) = \frac{1}{n}||\mathbf{X} - W_2W_1\mathbf{X}||_F^2 \tag{11}$$

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:

$$\mathcal{L}_\lambda(W_1, W_2; \mathbf{X}) = \tfrac{1}{n}||\mathbf{X} - W_2W_1\mathbf{X}||_F^2 + \lambda\|W_1\|_F^2 + \lambda\|W_2\|_F^2. \tag{12}$$

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 (12) 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

 All Rights Reserved. This may not be publicly shared without explicit permission.

Solution: **2**, we would use two linear layers, one for the encoder, one for the decoder.

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

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

Solution: We should use **MSE-Loss** to train the model (reconstruction under l2-loss) since what we want is for each vector to be close to its reconstruction in a squared-error sense.

(iii) **Which of the following would you need to optimize** (12) **exactly as it is written? (Select all that are needed)**

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

Solution: We need to use **Weight Decay** to achieve the desired regularization and the problem states that we are doing "SGD-Style", training. Therefore we use the **SGD Optimizer**.

(b) **Do you think that the solution to** (12) **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?)

Solution: If there were no regularization terms, we know that all the optimizers have to have W_2W_1 acting like a projection matrix that projects onto the k largest singular vectors of X.

This means that the W_2W_1 to minimize the main loss has to be the identity when restricted to the subspace spanned by the k largest singular vectors of X.

Therefore we would expect W_1, W_2 be approximate psuedo-inverses of each other since they are not square, and the rank of either one is at most k.

Therefore regularizing by penalizing the Frobenius norms forces us to consider:

$$\begin{aligned}\|W_1\|_F^2 + \|W_2\|_F^2 &= \|\Sigma_1\|_F^2 + \|\Sigma_2\|_F^2 \\ &= \sum_{i=1}^{k} \left(\sigma_i^2 + \frac{1}{\sigma_i^2}\right)\end{aligned}$$

where W_1 is bringing the σ_i terms and its approximate pseudo-inverse W_2 is bringing the $\frac{1}{\sigma_i}$ for its singular values.

Minimizing $\frac{1}{\sigma^2} + \sigma^2$ by taking derivatives results in setting $0 = -\frac{2}{\sigma^3} + 2\sigma$ which has a unique non-negative real solution at $\sigma = 1$, and so this is the unique minimizer since clearly this expression goes to ∞ as $\sigma \to \infty$ or $\sigma \to 0$.

If the λ is small enough, then the optimization essentially decouples: the main loss forces W_2 and W_1 to be pseudoinverses and to have the product W_2W_1 project onto the supspace spanned by the k singular vectors of X whose singular values are largest; and the regularization term forces the individual W_2 and W_1 to have all nonzero singular values as close to 1 as possible.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Once $\sigma_i \approx 1$, the matrix has approximately orthonormal columns for W_2 and approximately orthonormal rows for W_1. You can see this by simply writing $W = U\Sigma V^\top$ and then noticing for a tall W that $W^\top W = V\Sigma^\top \Sigma V^\top \approx VIV^\top = I$ and similarly for $WW^\top$ for a wide W.

4. Self-Supervised Linear Purification

Consider a linear encoder — *square* weight matrix $W \in \mathbb{R}^{m\times m}$ — that we want to be a "purification" operation on $m-$dimensional feature vectors from a particular problem domain. We do this by using self-supervised learning to reconstruct n points of training data $\mathbf{X} \in \mathbb{R}^{m\times n}$ by minimizing the loss:

$$\mathcal{L}_1(W;\mathbf{X}) = \|\mathbf{X} - W\mathbf{X}\|_F^2 \tag{13}$$

While the trivial solution $W = \mathbf{I}$ can minimize the reconstruction loss (13), we will now see how weight-decay (or equivalently in this case, ridge-style regularization) can help us achieve non-trivial purification.

$$\mathcal{L}_2(W;\mathbf{X},\lambda) = \underbrace{\|\mathbf{X} - W\mathbf{X}\|_F^2}_{\text{Reconstruction Loss}} + \lambda \underbrace{\|W\|_F^2}_{\text{Regularization Loss}} \tag{14}$$

Note above that λ controls the relative weighting of the two losses in the optimization.

(a) Consider the simplified case for $m = 2$ with the following two candidate weight matrices:

$$W^{(\alpha)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W^{(\beta)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{15}$$

The training data matrix $\mathbf{X}$ is also given to you as follows:

$$\mathbf{X} = \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix} \tag{16}$$

i. **Compute the reconstruction loss and the regularization loss for the two encoders, and fill in the missing entries in the table below.**

Encoder	Reconstruction Loss	Regularization Loss
α	_____	_____
β	0.001	_____

Solution:

Encoder	Reconstruction Loss	Regularization Loss
α	0	2
β	0.001	1

ii. **For what values of the regularization parameter λ is the identity matrix $W^{(\alpha)}$ get higher loss $\mathcal{L}_2$ in (14), as compared to $W^{(\beta)}$?**

Solution: Plugging the numbers from the table into Eqn. 14, the β encoder is a better choice for

$$\lambda > 1 \times 10^{-3}.$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

This suggests that despite the encoder being square, regularization induces a bottleneck in the encoder, preventing it from learning an identity mapping.

(b) Now consider a generic square linear encoder $W \in \mathbb{R}^{m\times m}$ and the regularized objective $\mathcal{L}_2$ reproduced below for your convenience:

$$\mathcal{L}_2(W;\mathbf{X},\lambda) = \underbrace{\|\mathbf{X} - W\mathbf{X}\|_F^2}_{\text{Reconstruction Loss}} + \lambda \underbrace{\|W\|_F^2}_{\text{Regularization Loss}}$$

Assume $\sigma_1 > \cdots > \sigma_m \geq 0$ are the m singular values in $\mathbf{X}$, that the number of training points n is larger than the number of features m, and that $\mathbf{X}$ can be expressed in SVD coordinates as $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$.

i. You are given that the optimizing weight matrix for the regularized objective $\mathcal{L}_2$ above takes the following form. **Fill in the empty matrices below**.

$$\widehat{W} = \begin{bmatrix} \quad \end{bmatrix} \cdot \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2+\lambda} & & & \\ & \frac{\sigma_2^2}{\sigma_2^2+\lambda} & & \\ & & \ddots & \\ & & & \frac{\sigma_m^2}{\sigma_m^2+\lambda} \end{bmatrix} \cdot \begin{bmatrix} \quad \end{bmatrix} \tag{17}$$

Solution:

$$\hat{W} = \begin{bmatrix} \mathbf{U} \end{bmatrix} \cdot \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2+\lambda} & & & \\ & \frac{\sigma_2^2}{\sigma_2^2+\lambda} & & \\ & & \ddots & \\ & & & \frac{\sigma_m^2}{\sigma_m^2+\lambda} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{U}^\top \end{bmatrix}$$

ii. **Derive the above expression.**
(Hint: Can you understand $\mathcal{L}_2(W;\mathbf{X},\lambda)$ as a sum of m completely decoupled ridge-regression problems?)
(Hint: The Frobenius norm is equal to $\|A\|_F^2 := tr(AA^T)$, and it is invariant under orthogonal transform. That is, $\|A\|_F^2 = \|UAV^T\|_F^2$ for any orthogonal matrices U, V, and any rectangular matrix A, as long as U, A, V have compatible shapes.)
Solution: Sum of m completely decoupled ridge-regression approach:
Since the squared Forbenius norm is just the sum of squares of all the elements in the matrix, we can decouple them into sum of m l2 norms. Specifically:

$$\mathcal{L}_2(W;\mathbf{X},\lambda) = \|\mathbf{X} - W\mathbf{X}\|_F^2 + \lambda\|W\|_F^2 = \sum_{i=1}^{m} \|\mathbf{X}_i - W_i\mathbf{X}\|_2^2 + \lambda\|W_i\|_2^2$$

where $\mathbf{X}_i \in \mathbb{R}^{1\times n}$ and $W_i \in \mathbb{R}^{1\times m}$ are rows. We can also write it as:

$$\mathcal{L}_2(W;\mathbf{X},\lambda) = \sum_{i=1}^{m} \|\mathbf{X}_i^\top - \mathbf{X}^\top W_i^\top\|_2^2 + \lambda\|W_i^\top\|_2^2$$

which resembles the classical ridge regression optimization problem $\|y - X\beta\|_2^2 + \lambda\|\beta\|_2^2$ which can be minimized with $\hat{\beta} = (X^\top X + \lambda I)^{-1}X^\top y$.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Through pattern matching $\mathbf{X}_i^\top = y, \mathbf{X}^\top = X, W_i^\top = \beta$, we can arrive with the result:

$$\hat{W}_i^\top = (\mathbf{X}\mathbf{X}^\top + \lambda I)^{-1}\mathbf{X}\mathbf{X}_i^\top$$

$$\hat{W} = \mathbf{X}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top + \lambda I)^{-1}$$

Now substitute $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ to the above result, we have:

$$\hat{W} = \mathbf{U}\Sigma\mathbf{V}^\top(\mathbf{U}\Sigma\mathbf{V}^\top)^\top(\mathbf{U}\Sigma\mathbf{V}^\top(\mathbf{U}\Sigma\mathbf{V}^\top)^\top + \lambda I)^{-1}$$

$$\hat{W} = \mathbf{U}\Sigma\Sigma^\top\mathbf{U}^\top(\mathbf{U}\Sigma\Sigma^\top\mathbf{U}^\top + \lambda I)^{-1}$$

$$\hat{W} = \mathbf{U}\Sigma\Sigma^\top\mathbf{U}^\top\mathbf{U}(\Sigma\Sigma^\top + \lambda I)^{-1}\mathbf{U}^\top$$

$$\hat{W} = \mathbf{U}\Sigma\Sigma^\top(\Sigma\Sigma^\top + \lambda I)^{-1}\mathbf{U}^\top$$

Solution: Vector calculus approach:
Alternatively we can solve this problem by taking the gradient of the loss function with respect to the weight matrix and finding the optimal point by setting it to zero.

$$\nabla\mathcal{L}_2(W; \mathbf{X}, \lambda) = \nabla(\|\mathbf{X} - W\mathbf{X}\|_F^2 + \lambda\|W\|_F^2) = 0$$

Using property of the frobenius norm $\|A\|_F^2 = \operatorname{Tr}\left(A^T A\right)$ we can rewrite the objective function:

$$\|\mathbf{X} - W\mathbf{X}\|_F^2 + \lambda\|W\|_F^2 = \operatorname{Tr}\left((\mathbf{X} - W\mathbf{X})(\mathbf{X} - W\mathbf{X})^T\right) + \lambda\operatorname{Tr}\left(WW^T\right)$$

Next, expand out the terms in the trace and compute the gradient with respect to W.

$$\nabla_W\left[\operatorname{Tr}\left(\mathbf{X}\mathbf{X}^T\right) - \operatorname{Tr}\left(\mathbf{X}\mathbf{X}^T W\right) - \operatorname{Tr}\left(W\mathbf{X}\mathbf{X}^T\right) + \operatorname{Tr}\left(W\mathbf{X}\mathbf{X}^T W\right) + \lambda\operatorname{Tr}\left(WW^T\right)\right]$$

$$= -2\mathbf{X}\mathbf{X}^T + 2W\mathbf{X}\mathbf{X}^T + 2\lambda W = 0$$

Solving for W we get

$$W = \mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda I)^{-1}$$

The problem can then be completed by substituting in $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ as in the alternative solution.

Solution: More elegant solution using the invariance (geometric) properties of trace and Frobenius norm.
The Frobenius norm is invariant under orthogonal transformation.
Proof: The Frobenius norm is equal to $\|A\|_F^2 = tr(AA^T)$, and since the trace has the cyclic property, we have $\|A\|_F^2 = tr(AA^T) = tr(U^TUAA^T) = tr(UAA^TU^T) = \|UA\|_F^2$ for any orthogonal matrix U. Similarly $\|A\|_F^2 = \|AV^T\|_F^2$ for any orthogonal matrix V.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Next, we expand

$$\begin{aligned} L &= \|X - WX\|_F^2 + \lambda\|W\|_F^2 \\ &= \|U\Sigma V^T - WU\Sigma V^T\|_F^2 + \lambda\|W\|_F^2 \\ &= \|\cancel{V}(\Sigma - U^T WU\Sigma)\cancel{V^T}\|_F^2 + \lambda\|W\|_F^2 \\ &= \|\Sigma - U^T WU\Sigma\|_F^2 + \lambda\|\cancel{U}U^T WU\cancel{U^T}\|_F^2 \\ &= \|\Sigma - \tilde{W}\Sigma\|_F^2 + \lambda\|\tilde{W}\|_F^2 \end{aligned}$$

where we defined $\tilde{W} := U^T WU$, or alternatively,

$$W = U\tilde{W}U^T$$

Now since Σ is already diagonalized, minimizing $\|\Sigma - \tilde{W}\Sigma\|_F^2 + \lambda\|\tilde{W}\|_F^2$ is simple, since we can directly expand it:

$$\begin{aligned} L &= \sum_i (\sigma_i - \sigma_i\tilde{W}_{ii})^2 + \sum_{i\neq j}(\sigma_i\tilde{W}_{ij})^2 + \lambda\sum_i(\tilde{W}_{ii})^2 + \lambda\sum_{i\neq j}(\tilde{W}_{ij})^2 \\ &= \sum_i [(\sigma_i - \sigma_i\tilde{W}_{ii})^2 + \lambda(\tilde{W}_{ii})^2] + \sum_{i\neq j}(\sigma_i^2 + \lambda)\tilde{W}_{ij}^2 \end{aligned}$$

As we see, the loss splits into many quadratic losses, each one depending on a single entry of $\tilde{W}$. Minimizing each, we find $\tilde{W} = \operatorname{diag}\left(\frac{\sigma_i^2}{\sigma_i^2+\lambda}\right)$.

(c) You are given that the data matrix $\mathbf{X} \in \mathbb{R}^{8\times n}$ has the following singular values:

$$\{\sigma_i\} = \{10, 8, 4, 1, 0.5, 0.36, 0.16, 0.01\}$$

For what set of hyperparameter values λ can we guarantee that the learned purifier $\widehat{W}$ will preserve at least 80% of the feature directions corresponding to the first 3 singular vectors of X, while attenuating components in the remaining directions to at most 50% of their original strength?

(Hint: What are the two critical singular values to focus on?)

Solution:

For the first condition, we have

$$\frac{\sigma_i^2}{\sigma_i^2 + \lambda} \geq 0.8 \;\; \forall\, i \in \{1, 2, 3\}$$

Since σ_i are decreasing, we get

$$\frac{\sigma_3^2}{\sigma_3^2 + \lambda} \geq 0.8 \implies \frac{4^2}{4^2 + \lambda} \geq 0.8 \implies \lambda \leq 4$$

Similarly, solving for the second condition gives:

$$\frac{\sigma_4^2}{\sigma_4^2 + \lambda} \leq 0.6 \implies \frac{1}{1 + \lambda} \leq 0.5 \implies \lambda \geq 1$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

Thus,

$$\mathbf{1} \leq \lambda \leq \mathbf{4}$$

5. SSM Convolution Kernel

Background and Setup: Consider a discrete-time State-Space Model (SSM) of the form

$$x_{k+1} = Ax_k + Bu_k,$$
$$y_k = Cx_k + Du_k,$$

(a) **Convolution Kernel and the Output Equation.** Given that the sequence length is L (input: $(u_0, \cdots, u_L)$, output: $(y_0, \cdots, y_L)$) and assume $x_0 = 0$, **show** that the output y_k can be expressed as a *convolution* of the input sequence $\{u_\ell\}_0^L$ with a kernel $K = \{K_\ell\}_0^L$:

$$y_k = \sum_{\ell=0}^{L} K_\ell u_{k-\ell},$$

where any $u_{\leq 0}$ with a negative index is set to 0 (zero-padding). Also, **find** K.

Solution: From the SSM recursion $x_{k+1} = Ax_k + Bu_k$, we get

$$x_1 = Bu_0, \quad x_2 = ABu_0 + Bu_1, \quad x_3 = A^2Bu_0 + ABu_1 + Bu_2, \ldots$$

Hence,

$$y_0 = Du_0, \tag{18}$$
$$y_1 = Du_1 + CBu_0, \tag{19}$$
$$y_2 = Du_2 + CBu_1 + CABu_0, \tag{20}$$
$$y_3 = Du_3 + CBu_2 + CABu_1 + CA^2Bu_0, \tag{21}$$
$$\vdots \tag{22}$$

This summation matches a discrete convolution $\{y\} = K * \{u\}$ with kernel K given by

$$K = (D, CB, CAB, CA^2B, \ldots).$$

(b) **Concrete Examples.**

i. *Scalar Case:* Let $n = 1$, and set $A = \alpha$, $B = \beta$, $C = \gamma$, $D = \delta$. Use $\alpha = 0.8, \beta = 1, \gamma = 1.5$ and compute the kernel up to $L = 4$.

ii. *2D Case:* Let $A \in \mathbb{R}^{2\times 2}$ be, for instance,

$$A = \begin{pmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 0 \end{pmatrix}, \quad D = 0$$

Compute kernels up to $L = 3$ and briefly discuss how the kernel captures the "impulse response".

 All Rights Reserved. This may not be publicly shared without explicit permission.

Solution:

(i) *Scalar example:* For $\alpha = 0.8, \beta = 1, \gamma = 1.5, \delta = 0$, the kernel reads

$$K = \left(0, 1.5, 1.5 \cdot 0.8, 1.5 \cdot 0.8^2, 1.5 \cdot 0.8^3\right).$$

(ii) *2D example:*

$$CB = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1,$$

$$CAB = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0.7,$$

$$CA^2B = 0.51,$$

The kernel of length $L = 4$ is $\begin{pmatrix} 0.0 & 1.0 & 0.7 & 0.51 \end{pmatrix}$.

Intuitively, the output of a SSM can be thought of as the response function of a sequence of discretized input signals and the output response decays over time. This mirrors the decaying kernel values because they correspond to how much the output value at the current step depends on the input values in the past (the older input values have lower influence on the current output value).

(c) **Efficient Computation with Convolutions.** If we already know the kernel K, how much can we parallelize the computation of the output sequence $\{y_k\}$ for an input sequence $\{u_k\} \in \mathbb{R}^d$ of length L? What is the minimum critical path length of the computation? What about a naive, direct computation of y_k from the unrolled recursion?

Solution: Once the convolution kernel K is known, computing the discrete convolution $\{y\} = K * \{u\}$ can be performed by zero-padding in $\mathcal{O}(\log L \log n)$ time (critical path length of the computation graph assuming matrix multiplication takes $O(\log n)$).

In contrast, naive unrolling would require, summing over all previous inputs $\ell \leq k$. Over $k = 1, \ldots, L$, this leads to at least $\mathcal{O}(L)$ recurrent steps which cannot be parallelized, resulting in a total complexity of $\mathcal{O}(L \log n)$ (critical path length).

(d) **Efficient Kernel Computation.** Given A, B, C, how can we compute the kernel, K, efficiently? What are some strategies to parallelize kernel computation? You may assume $L = 2^N$ for some N for simplicity.

Solution: To fully make use of parallel computational power of GPU, we want to compute $X^L = (I, A, \cdots, A^{L-1})$ fast. We can apply divide-and-conquer idea on this:

$$X^L = (X^{L/2}, A^{L/2} X^{L/2})$$

$$A^L = A^{L/2} A^{L/2}$$

For each $L = 2^l$ with $l \in \{1, 2, \cdots, N\}$, we can compute X^L through this recurrent formula so that we can compute the whole kernel with a maximum computation depth of $O(\log n \log L)$ where $O(\log n)$ is the maximum depth of computation for matrix multiplication.

(e) **Adding structure to A.** Suppose A is a diagonal matrix. How can we leverage this structure to compute the kernel K more efficiently?

Solution: If A is diagonal, we no longer need to compute matrix multiplication, so we could precompute all $(I, A, \cdots, A^{L-1})$ in $O(\log L)$ time (instead of $O(\log n \log L)$). In practice, the matrix

 All Rights Reserved. This may not be publicly shared without explicit permission.

multiplication is often much more expensive than the theoretical limit $O(\log n)$, so removing the need to do matrix multiplication for computing the kernel speeds up things a lot more.

(f) **Diagonal-plus-low-rank (DPLR) structure** Now if A has the following form:

$$A = I_n + pp^\top,$$

where $A \in \mathbb{R}^{n\times n}, p \in \mathbb{R}^n$. How can we leverage this structure to compute the kernel K more efficiently?

Solution:

$$A^L = (I_n + pp^\top)^L = I_n + Lpp^\top + \binom{L}{2}(pp^\top)^2 + \cdots + (pp^\top)^L$$

We observe that $(pp^\top)^l = p(p^\top p)^{l-1}p^\top = pp^\top v^{l-1}$ where $v = \|p\|_2^2$. Simplifying the equation above yields

$$A^L = I_n + pp^\top \sum_{l=1}^{L}\left[\binom{L}{l} v^{l-1}\right]$$

It is easy to see that computing $\sum_{l=1}^{L}\left[\binom{L}{l} v^{l-1}\right]$ can be done in $\log(L)$ using divide-and-conquer with no matrix multiplication involved.

6. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework?**

Contributors: Kevin Li, Anant Sahai, Naman Jain, Kumar Krishna Agrawal, Linyuan Gong, Sheng Shen, Dhruv Shah.

 All Rights Reserved. This may not be publicly shared without explicit permission.