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?**

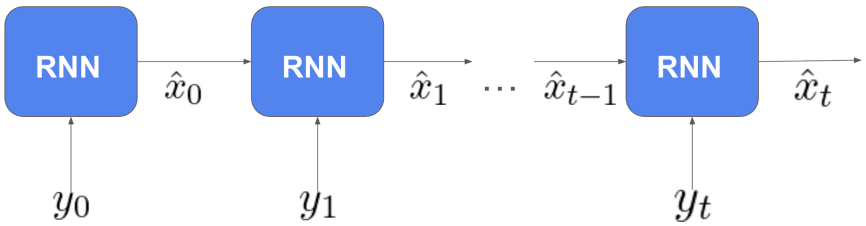
(b) Assume that the $\mathbf{u}_t = \mathbf{0}$ for simplicity. We know from Kalman filtering that if we know the $\mathbf{A}, \mathbf{B}, \mathbf{C}$ matrices, it is possible to create a steady-state filter which approximately tracks that state $\mathbf{x}_t$ from (1) given only the sequence of observations $\mathbf{y}_t$ 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:

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

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.**

(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.**

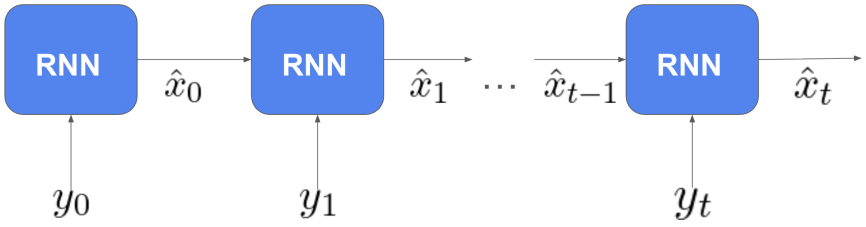
 All Rights Reserved. This may not be publicly shared without explicit permission.



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 $\mathbf{B'}$.

(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 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.**



(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 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 $\mathbf{0}$ to ensure causality. **Find K.**

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.

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

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.

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

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

(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.

(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?**

(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,

$$\mathcal{L}(W_1, W_2; \mathbf{X}) = ||\mathbf{X} - W_2 W_1 \mathbf{X}||_F^2 \tag{4}$$

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 $S = \text{diag}(\sigma_1, \ldots, \sigma_k)$, 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.

[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.

(i) **Write out the first order optimality conditions that the minima of Eq. 7 would satisfy.**

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

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,

$$\mathcal{L}(W_1, W_2; \mathbf{X}) = \frac{1}{n}||\mathbf{X} - W_2W_1\mathbf{X}||_F^2 \tag{5}$$

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:

$$\mathcal{L}_\lambda(W_1, W_2; \mathbf{X}) = \tfrac{1}{n}||\mathbf{X} - W_2W_1\mathbf{X}||_F^2 + \lambda\|W_1\|_F^2 + \lambda\|W_2\|_F^2. \tag{6}$$

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 $W_1 = U_1\Sigma_1V_1^\top$ and $W_2 = U_2\Sigma_2V_2^\top$. 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?)

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

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:

$$\mathcal{L}_1(W; \mathbf{X}) = \|\mathbf{X} - W\mathbf{X}\|_F^2 \tag{7}$$

While the trivial solution $W = \mathbf{I}$ can minimize the reconstruction loss (7), we will now see how weight-decay (or equivalently in this case, ridge-style regularization) can help us achieve non-trivial purification.

$$\mathcal{L}_2(W; \mathbf{X}, \lambda) = \underbrace{\|\mathbf{X} - W\mathbf{X}\|_F^2}_{\text{Reconstruction Loss}} + \lambda \underbrace{\|W\|_F^2}_{\text{Regularization Loss}} \tag{8}$$

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:

$$W^{(\alpha)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W^{(\beta)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{9}$$

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

$$\mathbf{X} = \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix} \tag{10}$$

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	_____

ii. **For what values of the regularization parameter λ is the identity matrix $W^{(\alpha)}$ get higher loss $\mathcal{L}_2$ in (8), as compared to $W^{(\beta)}$?**

(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

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

following form. **Fill in the empty matrices below**.

$$\widehat{W} = \begin{bmatrix} & \end{bmatrix} \cdot \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2+\lambda} & & & \\ & \frac{\sigma_2^2}{\sigma_2^2+\lambda} & & \\ & & \ddots & \\ & & & \frac{\sigma_m^2}{\sigma_m^2+\lambda} \end{bmatrix} \cdot \begin{bmatrix} & \end{bmatrix} \tag{11}$$

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.)

(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 $\mathbf{X}$, 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?)

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.

(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,

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

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

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

(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?

(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.

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

(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?

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.