

For Special Participation A, I used ChatGPT4o on several parts of HW7. Overall, it was helpful but revealed important limitations. ChatGPT4o excels at high-level conceptual explanations—providing clear insights on autoencoders, PCA, and sequence models—And it's also fairly strong with mathematical computations.

A notable issue occurred on Question 7(b), where ChatGPT4o initially gave an incorrect answer about decoder tokens during training and resisted correction when I first pointed out the mistake. This highlighted the need to persistently challenge responses that seem wrong rather than accepting them at face value.

Despite these challenges, ChatGPT4o answers are insightful and deepened my understanding.

I gave the following prompt to start

Help me to complete following questions

- ⌚ Sure, please send or paste the questions you'd like help with. Once I see them, I can assist you in completing them.

▢ ⌂ ⓘ ⓘ ⓘ ...

Question 3b:

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

We are minimizing:

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

Let's denote:

- $\mathbf{X} \in \mathbb{R}^{m \times n}$
- $W_1 \in \mathbb{R}^{k \times m}$: encoder matrix
- $W_2 \in \mathbb{R}^{m \times k}$: decoder matrix
- The reconstruction is: $\hat{\mathbf{X}} = W_2 W_1 \mathbf{X}$

Let's differentiate \mathcal{L} with respect to W_1 and W_2 and set gradients to zero (first-order optimality conditions):

✓ Gradient w.r.t. W_2 :

$$\nabla_{W_2} \mathcal{L} = -2(\mathbf{X} - W_2 W_1 \mathbf{X})(W_1 \mathbf{X})^\top = 0$$

This gives the optimality condition:

$$(\mathbf{X} - W_2 W_1 \mathbf{X})(W_1 \mathbf{X})^\top = 0$$

Derivation of the derivative with respect to w2 and w1 is both correct

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

Similarly calculate the other one:

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

✓ Gradient w.r.t. W_1 :

$$\nabla_{W_1} \mathcal{L} = -2W_2^\top (\mathbf{X} - W_2 W_1 \mathbf{X}) \mathbf{X}^\top = 0$$

This gives the second condition:

$$W_2^\top (\mathbf{X} - W_2 W_1 \mathbf{X}) \mathbf{X}^\top = 0$$

📌 (ii) Show that the principal components U_k satisfy the optimality conditions outlined in (i).

We're told the top k principal components are the eigenvectors corresponding to the largest k eigenvalues of $\mathbf{X}\mathbf{X}^\top$, and form matrix $\mathbf{U}_k \in \mathbb{R}^{m \times k}$ such that:

- $\mathbf{U}_k^\top \mathbf{U}_k = I_k$
- $\mathbf{X} \approx \mathbf{U}_k \mathbf{U}_k^\top \mathbf{X}$

Let's define:

- $W_1 = \mathbf{U}_k^\top$
- $W_2 = \mathbf{U}_k$

Then the reconstruction is:

$$W_2 W_1 \mathbf{X} = \mathbf{U}_k \mathbf{U}_k^\top \mathbf{X}$$

Now let's check if this choice satisfies both optimality conditions.

 **Condition 1:**

$$(\mathbf{X} - W_2 W_1 \mathbf{X})(W_1 \mathbf{X})^\top = (\mathbf{X} - \mathbf{U}_k \mathbf{U}_k^\top \mathbf{X})(\mathbf{U}_k^\top \mathbf{X})^\top$$

This simplifies to:

$$(\mathbf{X} - \mathbf{U}_k \mathbf{U}_k^\top \mathbf{X})(\mathbf{X}^\top \mathbf{U}_k) = 0$$

This is **true**, because \mathbf{U}_k projects \mathbf{X} onto the top k -dimensional subspace, and the residual $\mathbf{X} - \mathbf{U}_k \mathbf{U}_k^\top \mathbf{X}$ is orthogonal to the subspace spanned by \mathbf{U}_k .

 **Condition 1 is satisfied.**

 **Condition 2:**

$$W_2^\top (\mathbf{X} - W_2 W_1 \mathbf{X}) \mathbf{X}^\top = \mathbf{U}_k^\top (\mathbf{X} - \mathbf{U}_k \mathbf{U}_k^\top \mathbf{X}) \mathbf{X}^\top = 0$$

Again, the residual is orthogonal to \mathbf{U}_k , so the projection \mathbf{U}_k^\top eliminates it.

 **Condition 2 is also satisfied.**

 **Final Answer Summary**

(i) **First-order optimality conditions:**

$$(\mathbf{X} - W_2 W_1 \mathbf{X})(W_1 \mathbf{X})^\top = 0$$

$$W_2^\top (\mathbf{X} - W_2 W_1 \mathbf{X}) \mathbf{X}^\top = 0$$

(ii) **The principal components U_k satisfy the above conditions when:**

$$W_1 = U_k^\top, \quad W_2 = U_k$$

Let me know if you want this written in LaTeX format or in Overleaf-ready form.

Part 2 is also correct; ChatGPT gives a direct proof similar to the baseline answer.

Question 4:

Thanks! I've reviewed the "How to Train Your ResNet" ↗ blog and can help you answer the questions from **Question 4** of your homework. Here's a full draft you can use or adapt:

(a) Baseline vs. Final Training Time and Accuracy

The authors began with a **baseline ResNet-50 training configuration** on ImageNet that achieved a **top-1 accuracy of 67% in 100 epochs**, taking approximately **14 days** to train on a **single GPU**. Through successive optimization steps, they significantly improved this. By the end of the blog series, they achieved **93% top-5 accuracy and 76% top-1 accuracy**, while reducing the training time to **just 3 days** on a single GPU. These improvements came from tuning learning rate schedules, using better weight decay, and improving data augmentation and batch norm handling.

(b) What I Learned (≈ 100 words)

Reading the series helped me understand how crucial low-level training details are. Even small changes like using label smoothing, altering the learning rate schedule, or adjusting batch norm momentum can lead to big improvements in both convergence speed and final accuracy. I also learned that training efficiency isn't just about throwing more GPUs at the problem—it's about carefully tuning every part of the pipeline. The authors' systematic approach taught me the value of treating training as an optimization problem and of benchmarking and measuring each tweak carefully. It's a great example of empirical deep learning best practices.



(c) Most Interesting Approach (≈ 100 words)

The most interesting technique for me was **cosine learning rate annealing with warm restarts**. Unlike traditional step decay schedules, cosine annealing allowed the model to escape local minima and converge more smoothly, which significantly improved accuracy without increasing computation. I found it fascinating how a simple change to the learning rate schedule had such a large impact. It reflects the importance of optimization techniques beyond just the architecture. I also appreciated how the authors combined this with progressive image resizing to further speed up early training. It's something I'd like to try in my own projects.

Let me know if you'd like me to rephrase any of these in a more casual or more formal tone, or if you need a LaTeX or markdown version!

Decent reflection, summarizes the article well.

Question 7:

Great! Here's how you can answer **Question 7: Machine Translation** based on the prompt and figure provided.

(a) Design Change Issue (4pts)

| **State one problem with the proposed design change** where the encoder's hidden state h_t is passed as input to decoder timestep t , instead of just initializing the decoder with the final encoder hidden state h_T .

Answer:

Passing h_t into decoder timestep t breaks the **auto-regressive property** of the decoder. In standard encoder-decoder models, the decoder generates the output sequence step by step, using only previously generated tokens (or ground truth tokens during training) and an initial hidden state summarizing the full input. Feeding the encoder's h_t into each decoder step leaks information about the entire input sequence at every decoding step. This makes the model rely too heavily on the source sequence and not learn the proper conditional generation behavior, hurting generalization, especially during inference where h_t is no longer available.

Correct Answer

(b) Decoder Tokens During Training (3pts)

| What five tokens will be passed into the decoder during training for w_1, w_2, \dots, w_5 ?

In training, we use **teacher forcing**, so we input the *correct* previous tokens into the decoder at each step.

The true translation is:

| I saw a dog <eos>

We must also begin with the <sos> token.

Answer:

- $w_1 = \text{<sos>}$
- $w_2 = \text{I}$
- $w_3 = \text{saw}$
- $w_4 = \text{a}$
- $w_5 = \text{dog}$

| These are the inputs to the decoder RNN at each time step to predict: I, saw, a, dog, <eos>

The answer for b is incorrect, and the model refuses to reflect on the correct answer when i correct it at the first time

- ⌚ No — the solution shown in your screenshot is **not correct** based on the provided prompt.
 - ⌚ Thanks for pushing back — let's take a careful look together.
-

🔍 What does the question actually say?

In the example shown, the **correct translation** is:
“I see a dog”
...but the model incorrectly generated “I saw a dog”

And it asks:

What five tokens will be passed into the decoder during training?

✓ So what's the *correct ground-truth* sequence?

→ It's:

I see a dog <eos>

Because during **training**, we always use **teacher forcing**, i.e., we feed the model the **correct sequence**, regardless of what it may have generated.



(c) Decoder Tokens During Evaluation (3pts)

What five tokens would be passed into the decoder at evaluation time?

During evaluation (inference), we **don't have access to ground truth**. We start with `<sos>` and use the model's previous predictions as input for the next timestep.

Assume the model generates: I saw a dog `<eos>`

Answer:

- $w_1 = \text{<sos>}$
- $w_2 = \text{I}$
- $w_3 = \text{saw}$
- $w_4 = \text{a}$
- $w_5 = \text{dog}$

These are now sampled/generated tokens during inference and passed as input to the decoder at each step.

Correct answer,

Question 8:

Here are the answers for **Question 8: Self-supervised Linear Autoencoders** based on the regularized loss function in Eq. (6):

◆ (a) Implementation Questions

We are minimizing:

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

We'll treat this as a shallow autoencoder network in PyTorch.

(i) How many linear layers do you need?

Answer: 2

Explanation:

We need one linear layer for the **encoder** (which maps input \mathbf{x} to hidden representation $z = W_1 \mathbf{x}$) and one for the **decoder** (which reconstructs from z via $W_2 z$). That makes two total `nn.Linear` layers — no nonlinear activation is used since it's a **linear** autoencoder.

Exact correct answer,

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

Answer: `nn.MSELoss`

Explanation:

We're minimizing reconstruction error using the **Frobenius norm**, which corresponds to mean squared error between the original and reconstructed data points. So `nn.MSELoss` is the correct choice.

(iii) Which of the following would you need to optimize Eq. (6)? (Select all that apply)

Answers:

- Weight Decay
- SGD Optimizer

Explanation:

- Weight Decay implements the L2 regularization on W_1 and W_2 , matching the $\lambda \|W\|_F^2$ terms.
- SGD Optimizer is needed to perform the gradient updates.

Dropout / Layer Norm / Batch Norm are not used here, as the network is linear and we're not regularizing activations or doing normalization.

Exact correct answer,

◆ (b) Orthonormality Bias and Small λ

Do you think that using a small nonzero λ induces a bias towards orthonormal columns in W_2 ? Why or why not?

✓ Answer:

Yes, using a small but nonzero λ does induce an inductive bias toward orthonormal columns in W_2 , though not strictly enforcing it. Here's why:

Reasoning:

Let's consider the SVD decomposition:

- $W_1 = U_1 \Sigma_1 V_1^\top$
- $W_2 = U_2 \Sigma_2 V_2^\top$

The Frobenius norm regularization encourages the singular values σ_i of W_1 and W_2 to be **small**, and ideally **uniform** to minimize the sum $\sum \sigma_i^2$.

If all the singular values of W_2 are 1 and the matrix is square or wide/tall enough, then W_2 will have **orthonormal columns**.

The total regularization penalty becomes:

$$\|W_2\|_F^2 + \|W_1\|_F^2 = \sum_i \sigma_i^2(W_1) + \sum_j \sigma_j^2(W_2)$$

To minimize this while preserving information (i.e., minimizing reconstruction error), the solution tends to spread energy evenly across singular values — encouraging orthogonality.

Is the solution unique?

No, the solution is not strictly unique.

While the set of **principal subspaces** is uniquely determined by the top k singular vectors (assuming distinct eigenvalues), the choice of specific **orthonormal bases** (e.g., rotated versions) is **not unique**.

Conclusion:

- Small $\lambda > 0$ leads to **compact, low-norm weight matrices**, which biases toward orthonormal structures.
- It does not enforce orthonormality explicitly but nudges the solution in that direction for better generalization.

explaining the same phenomenon