EECS 16B Designing Information Devices and Systems II
Fall 2021 UC Berkeley Homework 13

This homework is due on Friday, December 3, 2021, at 11:00PM. Self-grades and HW Resubmission are due on Tuesday, December 7, 2021, at 11:00PM.

1. **Reading Lecture Notes**

   Staying up to date with lectures is an important part of the learning process in this course. Here are links to the notes that you need to read for this week: Note 19, Note 20, Note 2j.

   (a) We know that a scalar function $f(x)$ can be quadratically approximated around a particular point $x = x_\star$ using Taylor's series expansion as follows:

   $$f(x) \approx f(x_\star) + \frac{\mathrm{d}f}{\mathrm{d}x}(x_\star)\cdot(x - x_\star) + \frac{1}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_\star)\cdot(x - x_\star)^2 \tag{1}$$

   What is the equivalent quadratic approximation of a multivariate scalar function $f(x,u)$ around a particular expansion point $(x_\star, u_\star)$? Note, here x and u are both scalars.

   (b) **Explain why the complex inner product must not be symmetric in its two arguments if we want it to make sense relative to projections.**

2. Linearization to help classification: discovering logistic regression and how to solve it

You can, in spirit, reduce the problem of linear multi-class classification to that of binary classification by picking vectors that correspond to each of the categories "X" as compared with all the other examples categorized into a hybrid synthetic category of "not-X". This will give rise to scores corresponding to each category with the winner selected by seeing which score is the highest. However, we will focus here on the binary problem since that is the conceptual heart of this approach. In the binary problem, we do not need to have two different scores for the two categories since they are mutually exclusive. So, we have only a single score and just look at its sign.

Anyway, as was discussed in lecture, the naive straightforward way of picking the decision boundary (by looking at the mean example of each category and drawing the perpendicular bisector) is not always the best. The included Jupyter Notebook includes synthetic examples that illustrate the different things that can happen so that you can better appreciate the pathway that leads us to discover logistic regression as a natural approach to solve this problem based on the conceptual building blocks that you have already seen.

It is no exaggeration to say that logistic regression is the default starting method for doing classification in machine learning contexts, the same way that straightforward linear regression is the default starting method for doing regression. A lot of other approaches can be viewed as being built on top of logistic regression. Consequently, getting to logistic regression is a nice ending-point for this part of the 16AB story.

Let's start by giving things some names. Consider trying to classify a set of measurements $\vec{x}_i$ with given labels ℓ_i. For the binary case of interest here, we will think of the labels as being "+" and "-". For expository convenience, and because we don't want to have to carry it around separately, we will fold our threshold implicitly into the weights by augmenting our given measurements with the constant "1" in the first position of each $\vec{x}_i$. Now, the classification rule becomes simple. We want to learn a vector of weights $\vec{w}$ so that we can deem any point with $\vec{x}_i^\top \vec{w} > 0$ as being a member of the "+" category and anything with $\vec{x}_i^\top \vec{w} < 0$ as being a member of the "-" category.

We will do this using a minimization in the spirit of least squares. Except, instead of necessarily using some sort of squared loss function, we will just consider a generic cost function that can depend on the label and the prediction score for the point. For the i-th data point in our training data, we will incur a cost $c(\vec{x}_i^\top \vec{w}, \ell_i)$ for a total cost that we want to minimize by picking the best $\vec{w}$:

$$\underset{\vec{w}}{\operatorname{argmin}} \quad c_{\text{total}}(\vec{w}) = \sum_{i=1}^{m} c(\vec{x}_i^\top \vec{w}, \ell_i) \tag{2}$$

Because the objective we want to minimize can be a nonlinear function, our goal is to solve this iteratively as a sequence of least-squares problems that we know how to solve.

Consider the following algorithm:

```
1: $\vec{w} = \vec{0}$                                              ▷ Initialize the weights to $\vec{0}$
2: while Not done do                                                ▷ Iterate towards solution
3:     Compute $\vec{w}^\top \vec{x}_i$                             ▷ Generate current estimated labels
4:     Compute $\frac{d}{d\vec{w}} c(\vec{w}^\top \vec{x}_i, \ell_i)$      ▷ Generate derivatives with respect to $\vec{w}$ of the cost for update step
5:     Compute $\frac{d^2}{d\vec{w}^2} c(\vec{w}^\top \vec{x}_i, \ell_i)$  ▷ Generate second derivatives of the cost for update step
6:     $\delta\vec{w} = \texttt{LeastSquares}(\cdot, \cdot)$      ▷ We will derive what to call least squares on
7:     $\vec{w} = \vec{w} + \delta\vec{w}$                          ▷ Update parameters
8: end while
9: Return $\vec{w}$
```

The key step above is figuring out with what arguments to call `LeastSquares` while only having the labels ℓ_i and the points $\vec{x}_i$.

Recall that when the function $\vec{f}(\vec{x}, \vec{y}) : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^m$ takes in vectors and outputs a vector, the relevant derivatives for linearization are also represented by matrices:

$$\frac{\partial \vec{f}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{3}$$

$$\frac{\partial \vec{f}}{\partial \vec{y}} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \cdots & \frac{\partial f_1}{\partial y_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial y_1} & \cdots & \frac{\partial f_m}{\partial y_k} \end{bmatrix}. \tag{4}$$

where

$$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}. \tag{5}$$

Then, the linearization (first-order expansion) becomes

$$\vec{f}(\vec{x}, \vec{y}) \approx \vec{f}(\vec{x}_0, \vec{y}_0) + \frac{\partial \vec{f}}{\partial \vec{x}}(\vec{x}_0, \vec{y}_0) \cdot (\vec{x} - \vec{x}_0) + \frac{\partial \vec{f}}{\partial \vec{y}}(\vec{x}_0, \vec{y}_0) \cdot (\vec{y} - \vec{y}_0). \tag{6}$$

(a) Now, suppose we wanted to approximate the cost for each data point

$$c_i(\vec{w}) = c(\vec{x}_i^\top \vec{w}, \ell_i) \tag{7}$$

where

$$\vec{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \tag{8}$$

in the neighborhood of a weight vector $\vec{w}_\star$. Our goal is to write out the first-order expression for approximating the cost function $c_i(\vec{w}_\star + \delta\vec{w})$. This should be something in vector/matrix form like you have seen for the approximation of nonlinear systems by linear systems. We don't want to take any second derivatives just yet — only first derivatives. We have outlined a skeleton for the derivation with some parts missing. Follow the guidelines in each sub-section.

(i) Comparing to eq. (6), we know that $c_i(\vec{w}_\star + \delta\vec{w}) \approx c_i(\vec{w}_\star) + \frac{\partial c_i}{\partial \vec{w}}(\vec{w}_\star) \cdot \delta\vec{w}$. **Write out the vector form of $\frac{\partial c_i}{\partial \vec{w}}(\vec{w}_\star)$.**

(ii) **Write out the partial derivative of $c_i(\vec{w})$ with respect to w_g, the g^{th} component of $\vec{w}$, i.e., find $\frac{\partial c_i(\vec{w})}{\partial w_g}$.**
You should leave the answer in terms of the expression $c'(p, \ell) = \frac{\mathrm{d}}{\mathrm{d}p} c(p, \ell)$. This "prime" notation can be useful since by itself c is a scalar valued function of two scalar arguments and so this c' is also a scalar valued function of two scalar arguments.
(HINT: Use the linearity of derivatives and sums to compute the partial derivatives with respect

to each of the w_g terms. Don't forget the chain rule and the fact that $\vec{x}_i^\top \vec{w} = \sum_{j=1}^n x_{ij}w_j = x_{ig}w_g + \sum_{j\neq g} x_{ij}w_j$.)

(iii) With what you had above, **can you fill in the missing part to express the row vector $\frac{\partial}{\partial \vec{w}} c_i(\vec{w})$?**

$$\frac{\partial}{\partial \vec{w}} c_i(\vec{w}) = c'(\vec{x}_i^\top \vec{w}, \ell_i)\underline{\qquad} \tag{9}$$

(b) Now, we want a better approximation that includes second derivatives. For a general function, we would look for

$$f(\vec{x}_0 + \delta\vec{x}) \approx f(\vec{x}_0) + \frac{\partial f}{\partial \vec{x}}(\vec{x}_0)\delta\vec{x} + \frac{1}{2}\delta\vec{x}^\top \left(\frac{\partial^2 f}{\partial \vec{x}^2}(\vec{x}_0)\right)\delta\vec{x} \tag{10}$$

where $\frac{\partial f}{\partial \vec{x}}(\vec{x}_0)$ is an appropriate row vector and, as you've seen in the note, $\frac{\partial^2 f}{\partial \vec{x}^2}(\vec{x}_0)$ is called the Hessian and represents the second derivatives.

(i) Comparing to eq. (10), we know that

$$c_i(\vec{w}_\star + \delta\vec{w}) \approx c_i(\vec{w}_\star) + \frac{\partial c_i}{\partial \vec{w}}(\vec{w}_\star)\cdot\delta\vec{w} + \frac{1}{2}\delta\vec{w}^\top \left(\frac{\partial^2 c_i}{\partial \vec{w}^2}(\vec{w}_\star)\right)\delta\vec{w} \tag{11}$$

Write out the matrix form of $\frac{\partial^2 c_i}{\partial \vec{w}^2}(\vec{w}_\star)$ in terms of the second partial derivatives $\frac{\partial^2 c_i}{\partial w_g \partial w_h}(\vec{w}_\star)$.

(ii) **Take the second derivatives of the cost $c_i(\vec{w})$, i.e. solve for $\frac{\partial^2 c_i(\vec{w})}{\partial w_g \partial w_h}$.**
You should leave the answer in terms of $c''(p,\ell) = \frac{\mathrm{d}^2}{\mathrm{d}p^2}c(p,\ell)$. This c'' is also a familiar scalar second derivative with respect to the scalar first argument of the cost function.
(HINT: You should use the answer to part (a) and just take another derivative. Once again, use the linearity of derivatives and sums to compute the partial derivatives with respect to each of the w_h terms. This will give you $\frac{\partial^2}{\partial w_g \partial w_h}$. Don't forget the chain rule and again use the fact that $\vec{x}_i^\top \vec{w} = \sum_{j=1}^n x_{ij}w_j = x_{ih}w_h + \sum_{j\neq h} x_{ij}w_j$.)

(iii) The expression in part (ii) is for the $(g,h)^{\text{th}}$ component of the second derivative. $\frac{1}{2}$ times this times δw_g times δw_h would give us that component's contribution to the second-derivative term in the approximation, and we have to sum this up over all g and h to get the total contribution of the second-derivative term in the approximation. Now, we want to group terms to restructure this into matrix-vector form by utilizing the outer-product form of matrix multiplication. **What should the space in the following expression be filled with?**

$$\frac{\partial^2}{\partial \vec{w}^2} c_i(\vec{w}) = c''(\vec{x}_i^\top \vec{w}, \ell_i)\underline{\qquad} \tag{12}$$

(c) Now we have successfully expressed the second order approximation of $c_i(\vec{w}_\star + \delta\vec{w})$. Since we eventually want to minimize the total cost $c_{\text{total}}(\vec{w}) = \sum_{i=1}^m c_i(\vec{w})$, **can you write out the second order approximation of $c_{\text{total}}(\vec{w}_\star + \delta\vec{w})$ using results from (a) and (b)?**

(d) Now in this part, we want to re-write $c_{\text{total}}(\vec{w}_\star + \delta\vec{w})$ in form of $C + \sum_{i=1}^m (\vec{q}_i^\top \delta\vec{w} - s_i)^2$. This kind of thing is called "completing the square."

(i) Let's first rewrite a general second order polynomial $f(x) = ax^2 + bx + c$ in the form of $f(x) = r + (qx - s)^2$. **Find q, r, s in terms of a, b, c.** This procedure is called "completing the square". Then, **argue that**

$$\underset{x}{\operatorname{argmin}}\left(ax^2 + bx + c\right) = \underset{x}{\operatorname{argmin}}(qx - s)^2 \tag{13}$$

(ii) Now rewrite $c_{\text{total}}(\vec{w}_\star + \delta\vec{w})$ in the form of $C + \sum_{i=1}^m (\vec{q}_i^\top \delta\vec{w} - s_i)^2$. **What are C, q_i, and s_i?**

(e) Consider a least squares problem:

$$\vec{x}^{\star} = \underset{\vec{x}}{\operatorname{argmin}} \|A\vec{x} - \vec{s}\|^2 \tag{14}$$

Show that:

$$\|A\vec{x} - \vec{s}\|^2 = \sum_{i=1}^{m} (\vec{a}_i{}^{\top}\vec{x} - s_i)^2 \tag{15}$$

where

$$A = \begin{bmatrix} - & \vec{a}_1^{\top} & - \\ - & \vec{a}_2^{\top} & - \\ & \vdots & \\ - & \vec{a}_m^{\top} & - \end{bmatrix}. \tag{16}$$

Use this to intepret your expression from Part (d) as a standard least squares problem. What are the rows of A?

(f) Consider the following cost functions:

- squared error: $c^{+}_{\text{sq}}(p) = (p-1)^2,\ c^{-}_{\text{sq}}(p) = (p+1)^2$;
- exponential: $c^{+}_{\exp}(p) = \mathrm{e}^{-p},\ c^{-}_{\exp}(p) = \mathrm{e}^{p}$;
- and logistic: $c^{+}_{\text{logistic}}(p) = \ln\big(1 + \mathrm{e}^{-p}\big),\ c^{-}_{\text{logistic}}(p) = \ln(1 + \mathrm{e}^{p})$.

Compute the first and second derivatives of the above expressions with respect to p.

(g) Run the Jupyter Notebook `logistic.ipynb` and answer the following questions.

(i) **In Example 2, why does mean classification fail?**

(ii) **In Example 3, for what data distributions does ordinary least squares fail?**

(iii) Run the code cells in Example 4. By performing updates to $\vec{w}$ according to what you derived in previous parts of the question, **how many iterations does it take for exponential and logistic regression to converge?**

Congratulations! You now know the basic optimization-theoretic perspective on logistic regression. After you understand the probability material in 70 and 126, you can understand the probabilistic perspective on it as well. After you understand the optimization material in 127, you will understand even more about the optimization-theoretic perspective on the problem including why this approach actually converges.

3. Latch

The circuit below is a type of latch, which is one of the fundamental components of memory in many digital systems. The latch is a bistable circuit, which means that there are two possible stable states: one representing a stored '1' bit and the other a stored '0' bit.

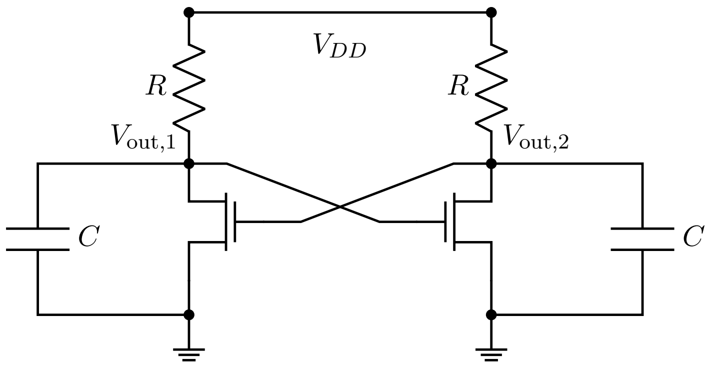


Figure 1: Simplified latch: the gate capacitances have been pulled out explicitly.

(a) To get a basic understanding of the stable equilibrium points for the latch, consider the following simplified circuit using the pure switch model for MOSFETs (and a threshold voltage of $\frac{V_{DD}}{2}$).

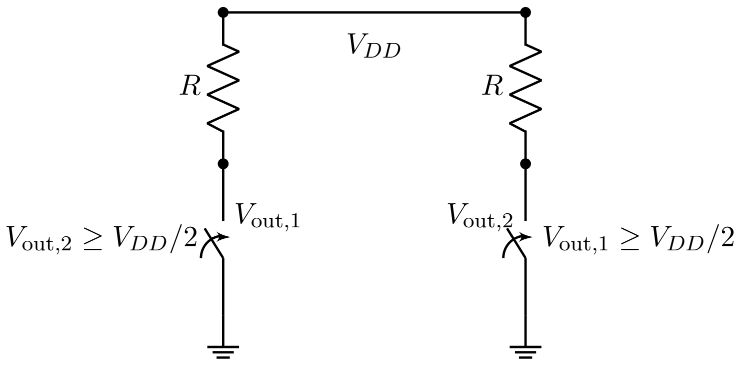

V_{DD}
R
R
$V_{\text{out},1}$
$V_{\text{out},2}$
$V_{\text{out},2} \geq V_{DD}/2$
$V_{\text{out},1} \geq V_{DD}/2$

Figure 2: Pure switch model for a latch

First assume that $V_{\text{out},1} = 0$. What is $V_{\text{out},2}$? Are the left and right switches open or closed?

	Open or V_{DD}	**Closed or 0**
Left Switch	○	○
Right Switch	○	○
$V_{\text{out},2}$	○	○

Suppose that $V_{\text{out},1} = V_{DD}$. What is $V_{\text{out},2}$? Are the left and right switches open or closed?

	Open or V_{DD}	**Closed or 0**
Left Switch	○	○
Right Switch	○	○
$V_{\text{out},2}$	○	○

(b) To get an understanding of latch dynamics, we will now break it down into smaller pieces. Below is one half of the latch circuit.

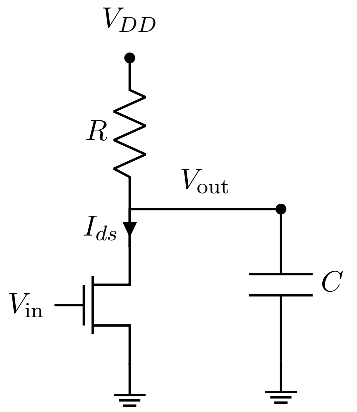
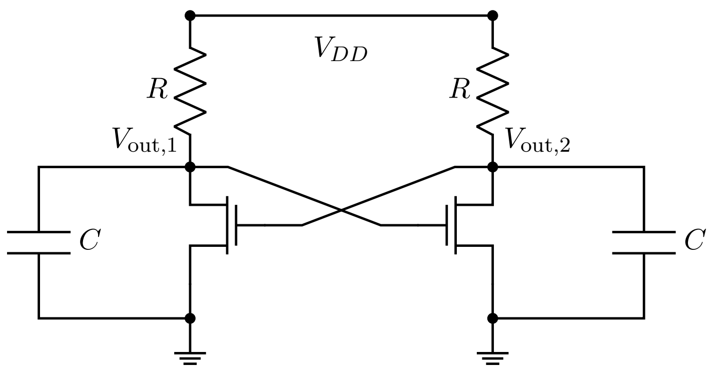


Figure 3: Latch half-circuit

Write a differential equation for the voltage V_{out} in terms of the drain to source current I_{ds}. Treat I_{ds} as some specified input signal and treat the transistor as a current source connected to ground. (i.e. There is no dependence on V_{in} in this part. In this part, treat the I_{ds} as a constant that you are given.)

(c)



For this circuit, we care about more detailed analog characteristics of the MOSFETs, so we will model their behavior more accurately as current sources that are controlled by their gate voltages V_{in} with the following equation:

$$I_{ds} = g(V_{\text{in}}) \tag{17}$$

Where $g(V_{\text{in}})$ is a some nonlinear function.

Using this I_{ds} expression together with the result from the previous part, write down a system of differential equations for $V_{\text{out},1}$ and $V_{\text{out},2}$ in vector form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} V_{\text{out},1}(t) \\ V_{\text{out},2}(t) \end{bmatrix} = \vec{f}\left(\begin{bmatrix} V_{\text{out},1}(t) \\ V_{\text{out},2}(t) \end{bmatrix}\right). \tag{18}$$

(Hint: Notice that the latch above can be constructed by taking two of the circuit in Figure 3, and connecting the V_{out} of one to the gate V_{in} of the other and vice-versa.)

(d) For the rest of this problem, assume that your analysis yields the following system of nonlinear differential equations:

$$\begin{bmatrix} \frac{\mathrm{d}V_{\mathrm{out},1}}{\mathrm{d}t} \\ \frac{\mathrm{d}V_{\mathrm{out},2}}{\mathrm{d}t} \end{bmatrix} = \begin{bmatrix} 1 - V_{\mathrm{out},1} - g(V_{\mathrm{out},2}) \\ 1 - V_{\mathrm{out},2} - g(V_{\mathrm{out},1}) \end{bmatrix} \tag{19}$$

Suppose that you put this latch into an ideal circuit simulator, and measure $g(V_{\mathrm{in}})$ and $\frac{\mathrm{d}g}{\mathrm{d}V}(V_{\mathrm{in}})$. The results from these measurements are shown in the graphs below. From your simulations, you also can see that for the following initial conditions, the latch voltages do not change over time:

$$\begin{bmatrix} V^*_{\mathrm{out},1} \\ V^*_{\mathrm{out},2} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \tag{20}$$

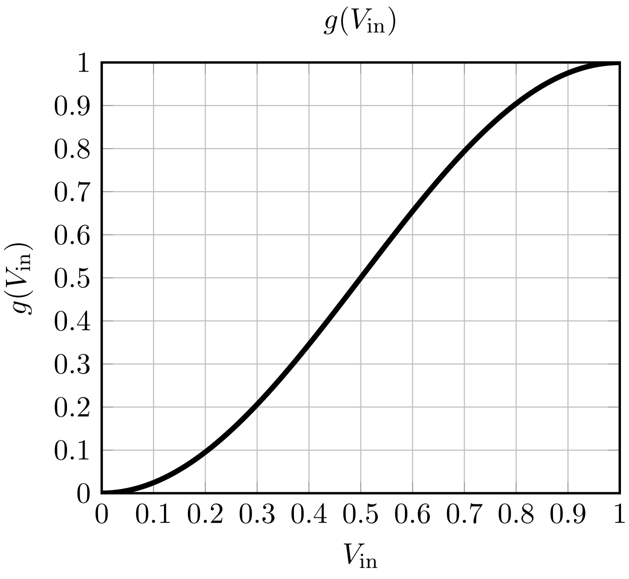
Use the graphs below to linearize the differential equations around the three equilibrium points.

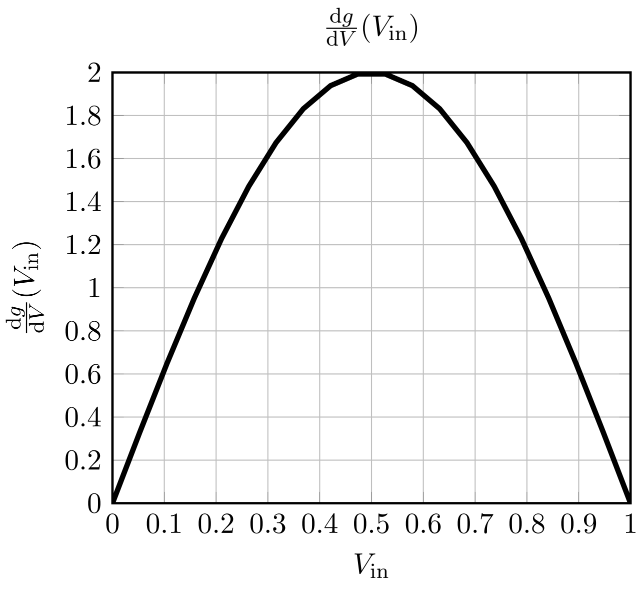
Write a linearized system of differential equations around each of those equilibrium points $\begin{bmatrix} V^*_{\mathrm{out},1} \\ V^*_{\mathrm{out},2} \end{bmatrix}$.

For which of the provided $\begin{bmatrix} V^*_{\mathrm{out},1} \\ V^*_{\mathrm{out},2} \end{bmatrix}$ points is the latch locally stable? For which of the provided points is the latch locally unstable? Why?

HINT 1: Use the following relationship for linear approximation.

$$\frac{\mathrm{d}\vec{f}}{\mathrm{d}\vec{v}}\left(\begin{bmatrix} V^*_{\mathrm{out},1} \\ V^*_{\mathrm{out},2} \end{bmatrix}\right) \cdot \begin{bmatrix} \delta V_1 \\ \delta V_2 \end{bmatrix} = \begin{bmatrix} -1 & -g'(V^*_{\mathrm{out},2}) \\ -g'(V^*_{\mathrm{out},1}) & -1 \end{bmatrix} \begin{bmatrix} \delta V_1 \\ \delta V_2 \end{bmatrix} \tag{21}$$





4. Extending Orthonormality to Complex Vectors

So far in the course, we have only dealt with real vectors. However, it is often useful to also think about complex vectors, as it allows for useful signal processing tools like the Discrete Fourier Transform (DFT). In this problem, we will extend several important properties of orthonormal matrices to the complex case.

The main difference is that the normal Euclidean inner product ($\langle \vec{u}, \vec{v} \rangle = \vec{u}^\top \vec{v}$) is no longer a valid way of defining lengths and angles when entries of vector are complex numbers, and we must define a new complex inner product for vectors $\vec{u}, \vec{v} \in \mathbb{C}^n$ ($\vec{u}$ and $\vec{v}$ have n complex number entries) as

$$\langle \vec{u}, \vec{v} \rangle = \overline{\vec{v}^\top} \vec{u} = \vec{v}^* \vec{u} = \sum_{i=1}^{n} u_i \overline{v_i} \tag{22}$$

where the * operation will complex conjugate and transpose its argument, and is aptly called the *conjugate transpose*. Note that for vectors with strictly real number entries, the complex inner product simplifies to the real inner product. In all the theorems you've seen in this class, you can replace every inner product with the complex inner product to show an analogous result for complex vectors: the least squares solution becomes $\hat{x} = (A^*A)^{-1}A^*\vec{b}$, the upper triangularization of A becomes $A = UTU^*$, the Spectral Theorem decomposes $A = A^*$ into $A = U\Lambda U^*$, and the SVD of A becomes $A = U\Sigma V^*$. In the next homework, we will reprove these results with the properties you will establish for the newly defined inner product.

(a) To get some practice computing complex inner products, what is $\left\langle \begin{bmatrix} 1+\mathrm{j} \\ 2 \end{bmatrix}, \begin{bmatrix} -3-\mathrm{j} \\ 2+\mathrm{j} \end{bmatrix} \right\rangle$ and $\left\langle \begin{bmatrix} -3-\mathrm{j} \\ 2+\mathrm{j} \end{bmatrix}, \begin{bmatrix} 1+\mathrm{j} \\ 2 \end{bmatrix} \right\rangle$? **Does the order of the vectors in the complex inner product matter i.e. is it commutative?**

(b) Let $U = \begin{bmatrix} | & & | \\ \vec{u}_1 & \cdots & \vec{u}_n \\ | & & | \end{bmatrix}$ be an n by n complex matrix, where its columns $\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_n$ form an orthonormal basis for $\mathbb{C}^n$, i.e.

$$\langle \vec{u}_j, \vec{u}_i \rangle = \vec{u}_i^* \vec{u}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{23}$$

Such a complex matrix is called *unitary* in math literature, to distinguish from real orthonormal matrices. **Show that $U^{-1} = U^*$, where U^* is the conjugate transpose of U.**

(c) **Show that U as defined in (b) preserves complex inner products, i.e. if $\vec{v}, \vec{w}$ are vectors of length n, then**

$$\langle \vec{v}, \vec{w} \rangle = \langle U\vec{v}, U\vec{w} \rangle \tag{24}$$

HINT: Note that $(AB)^* = B^*A^*$. This is because $(AB)^* = \overline{(AB)^\top} = \overline{B^\top A^\top} = \overline{B^\top}\ \overline{A^\top} = B^*A^*$. The reason conjugation distributes over matrix multiplication is because matrix multiplication is defined using addition and multiplication, which conjugation distributes over.

(d) **Show that if $\vec{u}_1, \ldots, \vec{u}_n$ are the columns of a unitary matrix U, they must be linearly independent.**

(Hint: Suppose $\vec{w} = \sum_{i=1}^{n} \alpha_i \vec{u}_i$, then first show that $\alpha_i = \langle \vec{w}, \vec{u}_i \rangle$. From here ask yourself whether a nonzero linear combination of the $\vec{u}_i$ could ever be identically zero.)

This basic fact shows how orthogonality is a very nice special case of linear independence.

(e) Now let V be another $n \times n$ matrix, where the columns of the matrix form an orthonormal basis for $\mathbb{C}^n$, i.e. V is unitary. **Show that the columns of the product UV also form an orthonormal basis for $\mathbb{C}^n$.**)

5. Using PyTorch to learn the color organ

One of the greatest recent developments in easy-to-use software packages is the easy availability of automatic differentiation. Although the underlying technology had been well established for over four decades (originally developed for control and scientific modeling applications in the context of differential equations), today packages like PyTorch expose this power to us in an easy to use way. This means that we as human engineers no longer have to worry about manually computing derivatives for any purpose other than taking exams, doing proofs, and basically learning material. In practical applications, the computer can do it for us without any bugs. As students whose careers will span the next four decades, we want you to consider this as a part of your engineering inheritance so that you can use it freely without thinking of it as being any more special than a hash table.

This problem and the accompanying Jupyter Notebook `color_organ_learning.ipynb` will show you how this power can be used to learn the value for the resistors in the color organ simply by having examples of where you want the LEDs to be on and off. This will build on the use of PyTorch that you will have seen in discussion.

(a) Let's start with an example low pass filter where, given a desired transfer function, we can determine a resistor value manually for our predicted transfer function such that the transfer functions match. In the interactive plot, we show a transfer function of a desired low pass filter (orange dotted line); we want to design our circuit (given a fixed capacitor value) such that predicted transfer function (blue solid line) is equivalent. For the following problems, we provide a function `evaluate_lp_circuit` that evaluates the transfer function magnitude given a resistor value. Note that these functions use `torch` functions instead of `numpy` as we will typically use `torch` tensors instead of `numpy` arrays in this notebook for training. **Use the slider to find a resistor value such that the predicted and desired transfer functions match and report the resistor value.**

The use of `torch` tensors instead of `numpy` arrays allows the package to do the bookkeeping behind the scenes that allows derivatives to be easily calculated. This will be important in later parts.

(b) Now, suppose that instead of seeing the entire transfer function that we want to match, we are only given some data about which frequencies lie in the pass band (i.e., which frequencies cause the LED on our color organ to be lit). Again, we can determine a resistor value manually. In the interactive plot, we show a transfer function of a low pass filter with its corresponding cutoff frequency. The table shows the desired behavior (red bars denote that the LED is on for a given frequency while black bars denote that the LED is off); we want to design the circuit such that the LEDs light up in the same way. **Use the slider to find a resistor value and corresponding cutoff frequency such that the predicted lights match the desired lights and report the corresponding resistor value and cutoff frequency.**

(c) Assume that we are able to query the desired transfer function directly (i.e., we can play a frequency and record the magnitude of the output). Can we *learn* the resistor value in the low pass circuit directly from this data (instead of manually designing the circuit as you did in the first part)?

The code for this part creates a model of the low pass filter in PyTorch, generates training data, and then trains the circuit using mean squared error loss until convergence (i.e., loss or gradients are very small) or the maximum number of training steps are reached. Here, we use the `torch.autograd.grad` function to automatically find the derivative of the loss with respect to the input (the resistor value of the low pass circuit). All we need to do is input the transfer function and define the loss!

The plots show how the transfer function of the learned circuit evolved during training, the loss surface, the derivative with respect to each training point at each iteration, and the total gradient at each training iteration. Note that the learned transfer function and resistor value change more slowly when the gradient is small, and more quickly when the gradient is large, but if we continue to iterate, we will converge to a local minimum in the loss surface. Try initializing the circuit with different resistor values (there is an optional argument for the circuit class constructor; if you leave it blank it will be initialized to a random value between 0 and 1000). **How long does the circuit take to converge with different initializations? Does it converge to the same value you found in the previous parts?** Try changing the learning rate (this parameter controls how far we step at each iteration in the direction of the negative gradient). **What values of `lr` cause training to diverge? What values cause the circuit to converge quickly?**

(d) Now, using the same loss function as in the previous part, let's try to learn the resistor value using only the binary data we have (LED on the color organ being on or off). **Change the code to use binary data instead of the transfer function magnitudes. What is the learned resistor value?** (Hints: Some potentially useful constants are defined at the start of the notebook. The loss function must take in floats (not boolean values).)

(e) What happened in the previous part? We did not converge to the same resistor value as we did in the previous part. Why? Because in trying to fit to the binary data, we can see that the positive and negative samples in our training data are pulling the resistor value in opposite directions (bottom left plot) and the final solution we end up at is where this "tug of war" situation achieves a balance. This balance need not end up where we would like it to. Can we fix this problem by adjusting the loss function? **Adjust the loss function such that the negative samples (where the LEDs should be off) are demanding an output other than 0 in a way that helps get the balance to end up where you want it. Can you find a loss function that yields the same resistor value as when training with the full transfer function?**

(f) Let's use what we learned in the previous parts to also learn our high pass filter from binary data. **Input the high pass filter transfer function into the high pass circuit module (use the same `loss_fn` as you found in the previous part) and fill in the code that updates the value of the resistor at each training step** (Hint: use the low pass filter as an example). **What is the learned resistor value?**

(g) Now let's extend the problem to a circuit with multiple parameters and learn both resistors for a band pass filter. **Input the band pass filter transfer function into the band pass circuit module.** (Hint: you can use the functions previously defined for high and low pass circuits). **Then complete the code for updating both resistors (note that `torch.autograd.grad` returns a tuple of gradients corresponding to each input). What are the learned resistor values?** What happens if the initial resistor values are far from the solution? **Try training with initial resistor values of 900 Ohms each. Does the circuit converge within the maximum number of training steps? How much longer does it take to converge? How large are the gradients and what does the loss surface look like when the resistor values are very far from the correct solution?**

(h) Now that we have tried learning the resistor values for a band pass filter directly from binary data, let's explore a different parametrization of our filter: the Bode Plot. Let's again start by trying to learn cutoff frequencies from samples of a transfer function using two ReLU functions. **Run the code. Does the resulting Bode plot match what you would draw given the underlying transfer function data? Do the cutoff frequencies match those corresponding to the resistor values that were found in the previous part?**

(i) Let's put all of these together to try and learn a color organ circuit from a low pass, high pass, and band pass circuits. Here, we train with a vector of size `(3, n)` which is *one-hot encoded*, meaning that for each of the n datapoints, one of the three values is 1 and the rest are 0. This encoding corresponds to our LEDs being on or off for one and only one of the three filters at a given frequency. **Train the color organ circuit and verify that the learned resistor values match those from the previous parts. Try initializing the resistors to different values; does it take longer or shorter to learn the entire color organ circuit than a single one of the filters (low pass, high pass, or band pass)?**

The final part of the notebook visualizes the computation graph that PyTorch is using to compute the derivatives of each transfer function with respect to the resistor values. See if you can match each operation in the graph to the transfer functions for each filter. Hopefully, this graph gives you an idea of how PyTorch can determine the partial derivatives that you have been using throughout the notebook. Congratulations, you now know how to use the considerable power of PyTorch to automatically differentiate arbitrary functions and find the corresponding local minima of these functions via gradient descent!

6. **Write Your Own Question And Provide a Thorough Solution.**

   Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning (from the bottom up) is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top level. We rarely ask you any homework questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself (e.g. making flashcards). But we don't want the same to be true about the highest level. As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams. Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't ever happen.

   You need to write your own question and provide a thorough solution to it. The scope of your question should roughly overlap with the scope of this entire problem set. This is because we want you to exercise your understanding of this material, and not earlier material in the course. However, feel free to combine material here with earlier material, and clearly, you don't have to engage with everything all at once. A problem that just hits one aspect is also fine.

   Note: One of the easiest ways to make your own problem is to modify an existing one. Ordinarily, we do not ask you to cite official course materials themselves as you solve problems. This is an exception. Because the problem making process involves creative inputs, you should be citing those here. It is a part of professionalism to give appropriate attribution.

   Just FYI: Another easy way to make your own question is to create a Jupyter part for a problem that had no Jupyter part given, or to add additional Jupyter parts to an existing problem with Jupyter parts. This often helps you learn, especially in case you have a programming bent.

7. **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? Write it down here where you'll need to remember it for the self-grade form.**

Contributors:

- Druv Pai.
- Kourosh Hakhamaneshi.
- Kuan-Yun Lee.
- Nathan Lambert.

- Sidney Buchbinder.
- Gaoyue Zhou.
- Anant Sahai.
- Pavan Bhargava.
- Ashwin Vangipuram.
- Michael Danielczuk.