HW0

Name: Mengying Lin.

SID: 3038737132.

2. Course Policies

Go to the course website and read the course policies carefully. Leave a followup in the Homework 0, Question 2 thread on Ed if you have any questions. Are the following situations violations of course policy? Write "Yes" or "No", and a short explanation for each.

(a) Alice and Bob work on a problem in a study group. They write up a solution together and submit it, noting on their submissions that they wrote up their homework answers together.

(b) Carol goes to a homework party and listens to Dan describe his approach to a problem on the board, taking notes in the process. She writes up her homework submission from her notes, crediting Dan.

(c) Erin gets frustrated by the fact that a homework problem given seems to have nothing in the lecture, notes, or discussion that is parallel to it. So, she starts searching for the problem online. She finds a solution to the homework problem on a website. She reads it and then, after she has understood it, writes her own solution using the same approach. She submits the homework with a citation to the website.

(d) Frank is having trouble with his homework and asks Grace for help. Grace lets Frank look at her written solution. Frank copies it onto his notebook and uses the copy to write and submit his homework, crediting Grace.

(e) Heidi has completed her homework. Her friend Irene has been working on a homework problem for hours, and asks Heidi for help. Heidi sends Irene her photos of her solution through an instant messaging service, and Irene uses it to write her own solution with a citation to Heidi.

a) Yes. Each students should write up their solutions individually.

b) No. She has credited external resources.

c) No. She finally made the information clear to herself and wrote up the homework on her own.

d) Yes. He shouldn't directly copy others' solutions.

e) No. It is OK to work in group but Heidi' shouldn't send her homework to others directly.

3. Gradient Descent Doesn't Go Nuts with Ill-Conditioning

Consider a linear regression problem with n training points and d features. When $n = d$, the feature matrix $F \in \mathbb{R}^{n \times n}$ has some maximum singular value α and an extremely tiny minimum singular value. We have noisy observations $\mathbf{y} = F\mathbf{w}^* + \epsilon$. If we compute $\hat{\mathbf{w}}_{inv} = F^{-1}\mathbf{y}$, then due to the tiny singular value of F and the presence of noise we observe that $\|\hat{\mathbf{w}}_{inv} - \mathbf{w}^*\|_2 = 10^{10}$.

Suppose instead of inverting the matrix we decide to use gradient descent instead. We run k iterations of gradient descent to minimize the loss $\ell(w) = \frac{1}{2}\|\mathbf{y} - F\mathbf{w}\|^2$ starting from $\mathbf{w}_0 = \mathbf{0}$. We use a learning rate η which is *small enough* that gradient descent cannot possibly diverge for the given problem. (**This is important. You will need to use this.**)

The gradient-descent update for $t > 0$ is:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \left(F^\top (F\mathbf{w}_{t-1} - \mathbf{y}) \right).$$

We are interested in the error $\|\mathbf{w}_k - \mathbf{w}^*\|_2^2$. We want to show that in the worst case, this error can grow at most linearly with iterations k and in particular $\|\mathbf{w}_k - \mathbf{w}^*\|_2 \leq k\eta\alpha\|\mathbf{y}\|_2 + \|\mathbf{w}^*\|_2$.

i.e. The error cannot go "nuts," at least not very fast.

For the purposes of the homework, you only have to prove the key idea, since the rest follows by applying induction and the triangle inequality.

Show that for $t > 0$, $\|\mathbf{w}_t\|_2 \leq \|\mathbf{w}_{t-1}\|_2 + \eta\alpha\|\mathbf{y}\|_2$.

(HINT: What do you know about $(I - \eta F^\top F)$ if gradient descent cannot diverge? What are its eigenvalues like? Use this fact.)

Proof: $\|w_t\|_2 = \|w_{t-1} - \eta(F^T(Fw_{t-1} - y)\|_2 = \|w_{t-1} - \eta F^T F w_{t-1} + \eta F^T y\|_2$

$\leq \|(I - \eta F^T F) w_{t-1}\|_2 + \eta \|F^T y\|_2$. ①

Now that α is the max singular value for F.

we have $\|F^T y\|_2 \leq \alpha \|y\|_2$. ... ②

In addition, $\|(I - \eta F^T F) w_{t-1}\|_2 \leq \|I - \eta F^T F\|_2 \|w_{t-1}\|_2$

$= |\sigma_{max}(I - \eta F^T F)| \|w_{t-1}\|_2$. ... ③

Assume η is small enough, we have

$\sigma_{max}(I - \eta F^T F) = 1 - \eta \sigma_{max}(F^T F) > 0$

Thus ③ $\leq \|w_{t-1}\|_2$. ... ④

According to ①, ②, ④, we have $\|w_t\|_2 \leq \|w_{t-1}\|_2 + \eta\alpha\|y\|_2$.

4. Regularization from the Augmentation Perspective

Assume $\mathbf{w}$ is a d-dimensional Gaussian random vector $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma)$ and Σ is symmetric positive-definite. Our model for how the $\{y_i\}$ training data is generated is

$$y = \mathbf{w}^\top \mathbf{x} + Z, \quad Z \sim \mathcal{N}(0, 1), \tag{1}$$

where the noise variables Z are independent of $\mathbf{w}$ and iid across training samples. Notice that all the training $\{y_i\}$ and the parameters $\mathbf{w}$ are jointly normal/Gaussian random variables conditioned on the training inputs $\{\mathbf{x}_i\}$. Let us define the standard data matrix and measurement vector:

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

In this model, the MAP estimate of $\mathbf{w}$ is given by the Tikhonov regularization counterpart of ridge regression:

$$\hat{\mathbf{w}} = (X^\top X + \Sigma^{-1})^{-1} X^\top \mathbf{y}, \tag{2}$$

In this question, we explore Tikhonov regularization from the data augmentation perspective.

Define the matrix Γ as a $d \times d$ matrix that satisfies $\Gamma^\top \Gamma = \Sigma^{-1}$. Consider the following augmented design matrix (data) $\hat{X}$ and augmented measurement vector $\hat{y}$:

$$\hat{X} = \begin{bmatrix} X \\ \Gamma \end{bmatrix} \in \mathbb{R}^{(n+d)\times d}, \quad \text{and} \quad \hat{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0}_d \end{bmatrix} \in \mathbb{R}^{n+d},$$

where $\mathbf{0}_d$ is the zero vector in $\mathbb{R}^d$. **Show that the ordinary least squares problem**

$$\operatorname*{argmin}_{\mathbf{w}} \|\hat{\mathbf{y}} - \hat{X}\mathbf{w}\|_2^2$$

has the same solution as (2).

(HINT: Feel free to just use the formula you know for the OLS solution. You don't have to rederive that. This problem is not intended to be hard or time consuming.)

Define $t(w) = \|\hat{y} - \hat{X}w\|_2^2$

$$= \sum_{i=1}^{n+d} (\hat{y}_i - \hat{x}_i w)^2 = \sum_{i=1}^{n} (y_i - x_i w)^2 + \sum_{i=n+1}^{n+d} (\hat{y}_i - \hat{x}_i w)^2$$

$$= (Xw - y)^T (Xw - y) + (\Gamma w)^T (\Gamma w).$$

$$= w^T X^T X w - y^T X w - w^T X^T y + y^T y + w^T \Gamma^T \Gamma w.$$

$$\frac{\partial t}{\partial w} = 2X^T X w - X^T y - X^T y + 2\Gamma^T \Gamma w$$

Let $\frac{\partial t}{\partial w} = 0$. $\quad w = (X^T X + \Gamma^T \Gamma)^{-1} X^T y$

$$= (X^T X + \Sigma)^{-1} X^T y.$$

So the solution is same as (2).

$(\hat{X}^T \hat{X})^{-1} \hat{X}^T \hat{y}$.

5. Vector Calculus Review

Let $\mathbf{x}, \mathbf{c} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$. For the following parts, before taking any derivatives, identify what the derivative looks like (is it a scalar, vector, or matrix?) and how we calculate each term in the derivative. Then carefully solve for an arbitrary entry of the derivative, then stack/arrange all of them to get the final result. Note that the convention we will use going forward is that vector derivatives of a scalar (with respect to a column vector) are expressed as a row vector, i.e. $\frac{\partial f}{\partial \mathbf{x}} = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}]$ since a row acting on a column gives a scalar. You may have seen alternative conventions before, but the important thing is that you need to understand the types of objects and how they map to the shapes of the multidimensional arrays we use to represent those types.

(a) Show $\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}^T\mathbf{c}) = \mathbf{c}^T$

(b) Show $\frac{\partial}{\partial \mathbf{x}}||\mathbf{x}||_2^2 = 2\mathbf{x}^T$

(c) Show $\frac{\partial}{\partial \mathbf{x}}(A\mathbf{x}) = A$

(d) Show $\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}^T A\mathbf{x}) = \mathbf{x}^T(A + A^T)$

(e) Under what condition is the previous derivative equal to $2\mathbf{x}^T A$?

a) $\frac{\partial}{\partial x}(x^Tc) = \left(\frac{\partial(x^Tc)}{\partial x_1}, \cdots \frac{\partial(x^Tc)}{\partial x_n}\right)$

Note $x^Tc = \sum_{i=1}^{n} x_i c_i$,

Therefore $\frac{\partial(x^Tc)}{\partial x_i} = c_i$. $\frac{\partial}{\partial x}(x^Tc) = (c_1, \cdots c_n)$

$= c^T$.

b) $\frac{\partial \|x\|_2^2}{\partial x} = \frac{\partial(\sum x_i^2)}{\partial x} = (2x_1, 2x_2 \cdots, 2x_n) = 2x^T$.

c) $\frac{\partial}{\partial x}(Ax) = \frac{\partial}{\partial x}\begin{pmatrix} \sum a_{1i}x_i \\ \vdots \\ \sum a_{ni}x_i \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x}\sum a_{1i}x_i \\ \vdots \\ \frac{\partial}{\partial x}\sum a_{ni}x_i \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$

d) $\frac{\partial(x^TAx)}{\partial x} = \frac{\partial(x^T(Ax))}{\partial x} = (Ax)^T + x^T\frac{\partial(Ax)}{\partial x}$

$= x^T(A^T + A)$.

e) Let $x^T(A + A^T) = 2x^TA$.

$x^TA + x^TA^T = 2x^TA$.

$x^TA^T = x^TA$.

$x^T(A^T - A) = 0_n$. for all $x \in \mathbb{R}^n$.

Therefore when $A^T = A$, i.e. A is symmetric,

the previous derivative equals to $2x^TA$.

6. ReLU Elbow Update under SGD

In this question we will explore the behavior of the ReLU nonlinearity with Stochastic Gradient Descent (SGD) updates. The hope is that this problem should help you build a more intuitive understanding for how SGD works and how it iteratively adjusts the learned function.

We want to model a 1D function $y = f(x)$ using a 1-hidden layer network with ReLU activations and no biases in the linear output layer. Mathematically, our network is

$$\hat{f}(x) = \mathbf{W}^{(2)}\Phi\left(\mathbf{W}^{(1)}x + \mathbf{b}\right)$$

where $x, y \in \mathbb{R}$, $\mathbf{b} \in \mathbb{R}^d$, $\mathbf{W}^{(1)} \in \mathbb{R}^{d\times 1}$, and $\mathbf{W}^{(2)} \in \mathbb{R}^{1\times d}$. We define our loss function to be the squared error,

$$\ell\left(x, y, \mathbf{W}^{(1)}, \mathbf{b}, \mathbf{W}^{(2)}\right) = \frac{1}{2}\left\|\hat{f}(x) - y\right\|_2^2.$$

For the purposes of this problem, we define the gradient of a ReLU at 0 to be 0.

(a) Let's start by examining the behavior of a single ReLU with a linear function of x as the input,

$$\phi(x) = \begin{cases} wx + b, & wx + b > 0 \\ 0, & \text{else} \end{cases}.$$

Notice that the slope of $\phi(x)$ is w in the non-zero domain.
We define a loss function $\ell(x, y, \phi) = \frac{1}{2}\|\phi(x) - y\|_2^2$. **Find the following:**

(i) The location of the 'elbow' e of the function, where it transitions from 0 to something else.

(ii) The derivative of the loss w.r.t. $\phi(x)$, namely $\frac{d\ell}{d\phi}$

(iii) The partial derivative of the loss w.r.t. w, namely $\frac{\partial \ell}{\partial w}$

(iv) The partial derivative of the loss w.r.t. b, namely $\frac{\partial \ell}{\partial b}$

a) i). let $\phi(x) = 0$.

$x = -\frac{b}{w}$. The elbow is $(-\frac{b}{w}, 0)$.

ii). $\frac{\partial \ell}{\partial \phi} = \frac{\partial\left[\frac{1}{2}\|\phi - y\|_2^2\right]}{\partial \phi} = \frac{1}{2}\frac{\partial(\phi^2 - 2\phi y + y^2)}{\partial \phi} = \phi - y.$

iii) $\ell = \begin{cases} \frac{1}{2}\|wx+b-y\|_2^2, & wx+b>0 \\ 0, & \text{else} \end{cases}$

$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial \phi}\cdot\frac{\partial \phi}{\partial w} = \begin{cases} (wx+b-y)\cdot x, & wx+b>0 \\ 0 & \text{else} \end{cases}$

iv) $\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial \phi}\cdot\frac{\partial \phi}{\partial b} = \begin{cases} (wx+b-y), & b > -wx \\ 0 & \text{else} \end{cases}$

(b) Now suppose we have some training point (x, y) such that $\phi(x) - y = 1$. In other words, the prediction $\phi(x)$ is 1 unit above the target y — we are too high and are trying to pull the function downward.

Describe what happpens to the slope and elbow of $\phi(x)$ when we perform gradient descent in the following cases:

(i) $\phi(x) = 0$.

(ii) $w > 0$, $x > 0$, **and** $\phi(x) > 0$. **It is fine to check the behavior of the elbow numerically in this case.**

(iii) $w > 0$, $x < 0$, **and** $\phi(x) > 0$.

(iv) $w < 0$, $x > 0$, **and** $\phi(x) > 0$. **It is fine to check the behavior of the elbow numerically in this case.**

Additionally, draw and label $\phi(x)$, the elbow, and the qualitative changes to the slope and elbow after a gradient update to w and b. You should label the elbow location and a candidate (x, y) pair. Remember that the update for some parameter vector $\mathbf{p}$ and loss ℓ under SGD is

$$\mathbf{p}' = \mathbf{p} - \lambda \nabla_{\mathbf{p}}(\ell),\ \lambda > 0.$$

b) Gradient descent:

$$\begin{cases} w \leftarrow w - \eta \frac{\partial L}{\partial w} \\ b \leftarrow b - \eta \frac{\partial L}{\partial b} \end{cases}$$

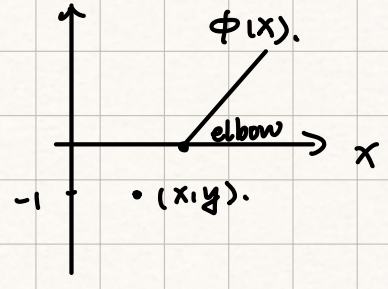
Denote the updated w and b as w', b'.

$$\begin{cases} \begin{cases} w' = w - \eta (wx+b-y)x = w - \eta x \\ b' = b - \eta (wx+b-y) = b - \eta \end{cases} & \text{①} \\ \begin{cases} w' = w \\ b' = b \end{cases} & \phi \le 0. \quad \text{②} \end{cases}$$

i. We have $wx+b = 0$.

Note in this case $\frac{\partial L}{\partial b}, \frac{\partial L}{\partial w} = 0$. so w and b remain unchanged.

The slope and the elbow do not change.

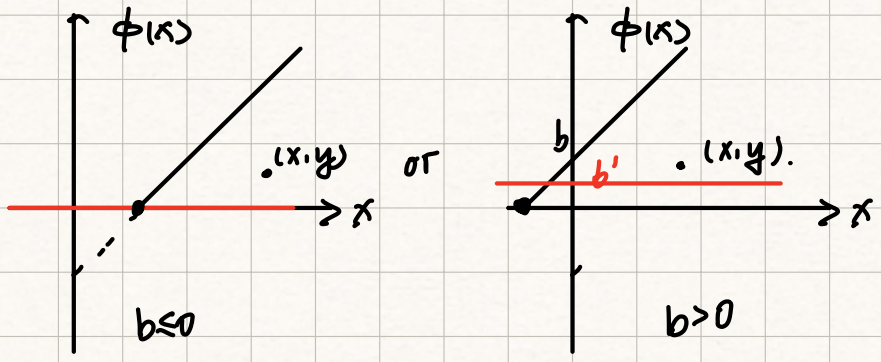


* Red line indicates the changed $\phi(x)$.

ii). As indicated in ①, $w' = w - \eta x$ and $x > 0$, so the slope is dropping.

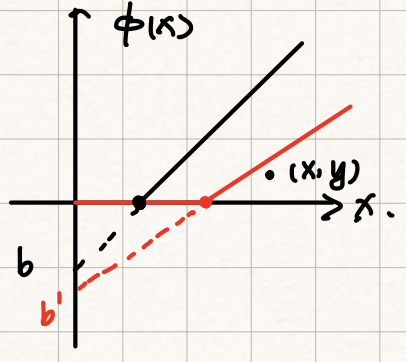
① If $w' = 0$. $\phi = \begin{cases} b' & (b' > 0) \\ 0 & (\text{else}) \end{cases}$. the elbow disappears.

Corresponding graphs:



② if $w' \neq 0$. Let $\phi = w'x + b' = 0$. $x = -\frac{b'}{w'} = -\frac{b - \eta}{w - \eta x}$.

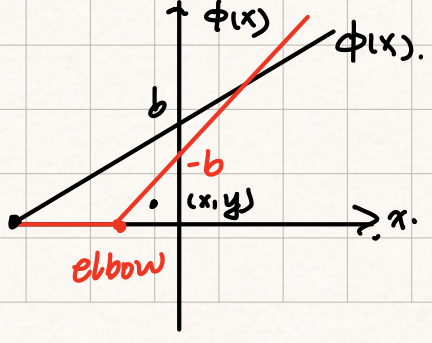
The elbow is moving towards right.



iii). $w' = w - \eta x > w > 0$, so the slope is increasing.

$b' = b - \eta$.

Let $\phi = w'x + b' = 0$, $x = -\frac{b - \eta}{w - \eta x}$.



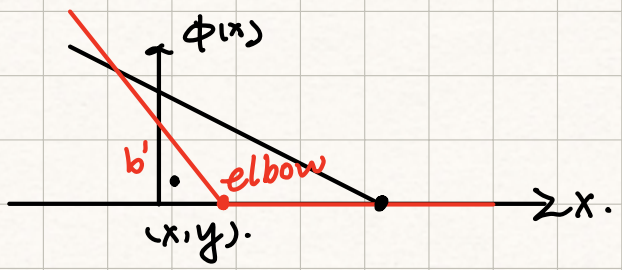
The elbow is moving towards right.

iv). $w' = w - \eta x < w$. so the slope is decreasing numerically.

$b' = b - \eta$

Let $\phi = w'x + b' = 0$. $x = -\frac{b-\eta}{w-\eta x}$.

The elbow is moving towards left.



(c) Now we return to the full network function $\hat{f}(x)$. **Derive the location e_i of the elbow of the i'th elementwise ReLU activation.**

(d) **Derive the new elbow location e_i' of the i'th elementwise ReLU activation after one stochastic gradient update with learning rate λ.**

c). Denote $W^{(1)} = \begin{pmatrix} w_1^{(1)} \\ \vdots \\ w_d^{(1)} \end{pmatrix}$ $b = \begin{pmatrix} b_1 \\ \vdots \\ b_d \end{pmatrix}$.

$$\ell\left(x, y, \mathbf{W}^{(1)}, \mathbf{b}, \mathbf{W}^{(2)}\right) = \frac{1}{2}\left\|\hat{f}(x) - y\right\|_2^2.$$

$$\hat{f}(x) = \mathbf{W}^{(2)}\Phi\left(\mathbf{W}^{(1)}x + \mathbf{b}\right)$$

Let $w_i^{(1)} e_i + b_i = 0$.

If $w_i^{(1)} \neq 0$, $e_i = -\frac{b_i}{w_i^{(1)}}$

d). $\frac{\partial \ell}{\partial w_i^{(1)}} = \frac{\partial \ell}{\partial \hat{f}} \cdot \frac{\partial \hat{f}}{\partial \phi} \cdot \frac{\partial \phi}{\partial w_i} \cdot \frac{\partial w^{(1)}}{\partial w_i^{(1)}}$

$= \begin{cases} (\hat{f}(x) - y) w_i^{(2)} x & (w_i^{(1)} x + b_i > 0) \\ 0 & (\text{else}). \end{cases}$

$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial \hat{f}} \cdot \frac{\partial \hat{f}}{\partial \phi} \cdot \frac{\partial \phi}{\partial b}$

$= \begin{cases} (\hat{f}(x) - y) w_i^{(2)} & (w_i^{(1)} x + b_i > 0) \\ 0 & (\text{else}). \end{cases}$

Hence if $w_i^{(1)} x + b_i > 0$

$\begin{cases} w_i^{(1)} \leftarrow w_i^{(1)} - \lambda (\hat{f}(x) - y) w_i^{(2)} x. \\ b_i \leftarrow b_i - \lambda (\hat{f}(x) - y) w_i^{(2)} \end{cases}$

let $w_i^{(1)} x + b_i = 0$.

$$x = -\frac{b_i - \lambda (\hat{f}(x) - y) w_i^{(2)}}{w_i^{(1)} - \lambda (\hat{f}(x) - y) w_i^{(2)} x}$$

Else if $w_i^{(1)} x + b_i \leq 0$.

$\begin{cases} w_i^{(1)} \leftarrow w_i^{(1)} \\ b_i \leftarrow b_i. \end{cases}$ $x = -\frac{b_i}{w_i^{(1)}}$.

a). The value I found : 200

b). The value I found : 200.

c) ① Yes

②

R_init(Ω)	Iterations	Final R (Ω).
0	Fail to train	
100	20321	197
200	0	200
300	27335	200
800	43670	200
1000	77121	200

③ It does not always coverge to the same value.

④ If lr is too small, (e.g. lr = 20, R_init = 1000), it will fail to converge within 1×10^5 iterations. If lr is too large (e.g. lr = 2×10^7), it diverges.

⑤ Generally, within a moderate interval (e.g. lr ∈ [200, 2×10^3]). the larger the learning rate is, the faster it converges.

I test on lr = 2×10^3, and it converges in 87 iterations.

d) learned value: R = 339 Ω

e) Yes, The loss I find is:

$$l(\hat{y}, y) = (1-y) \times \max(\hat{y} - \text{cutoff\_mag}, 0) + y \times \max(\text{cutoff\_mag} - \hat{y}, 0)$$

This loss has 2 term.

① Let us first look at $y \times \max(\text{cutoff\_mag} - \hat{y}, 0)$

For positive sample (y = 1), we only need the predicted value to be greater than threshold. If it does, we force the loss to equal to zero.

② $(1-y) \times \max(\hat{y} - \text{cutoff\_mag}, 0)$ aims at negative sample,

and the key idea is similar to ①.

f). The learned value is 32 Ω.

g). ① The learned value:

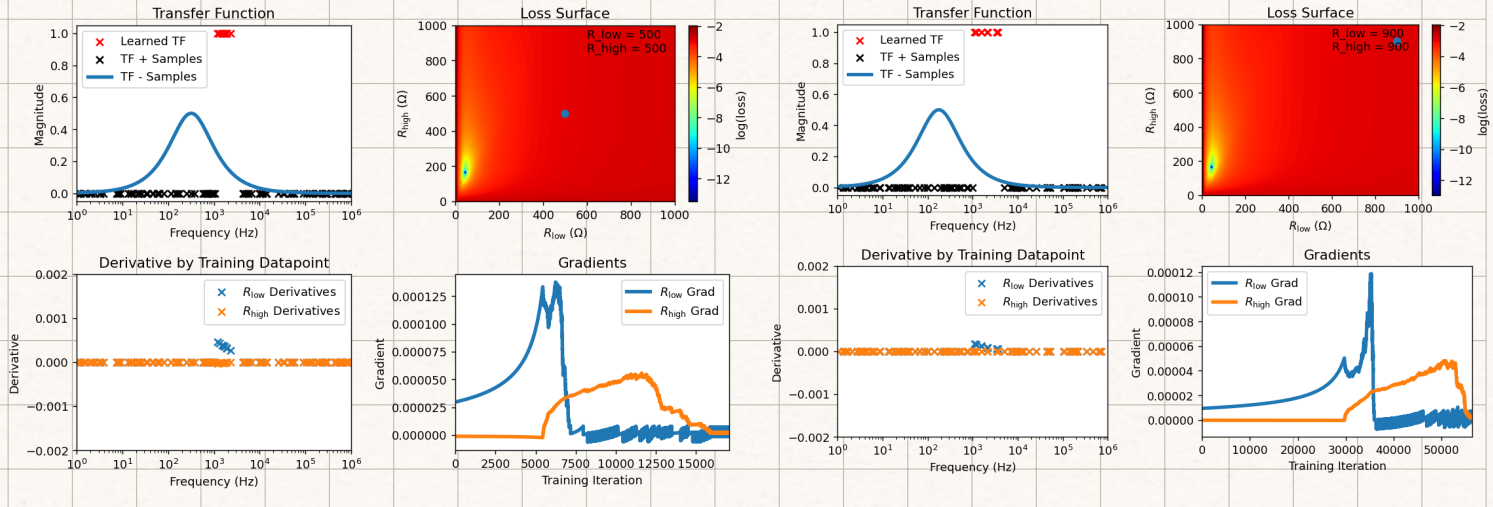
$$\begin{cases} R\_low = 40\,\Omega \\ R\_high = 160\,\Omega \end{cases}$$

② If the initial values are too far from the solution, the circuit may not converge.

When the initial values are both set to 900 Ω, the circuit converges with $R\_low = 40\,\Omega$, $R\_high = 161\,\Omega$.

③ When initialized to 500Ω, it only takes 17090 iterations, and <9873> iterations when set to 900Ω.

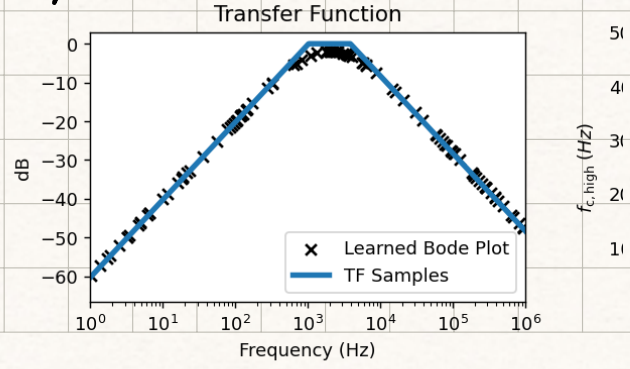
④ In this case loss is relatively high and the start point is far from low-loss region as shown in loss surface



Initialized with 500Ω.

Initialized with 900Ω.

h) ① The bode plot match the expected curve.



② The learned cutoff low pass frequency is 3864 Hz, and the high pass frequency is 1029 Hz, which are slightly different from the given 4000 Hz and 1000 Hz.

i) It takes similar amount of iterations because the parameters will all be updated in a singe interation, but if evaluated with computing time. it'll definitely takes longer.

8. 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.**

a) https://eecs16b.org/notes/fa21/note6.pdf

b)

Names	Xiong Fei
SID	3038733024.

c) 15 h

color_organ_learning

January 31, 2023

[1]:
```
!pip install ipympl torchviz
!pip install torch==1.13 --extra-index-url https://download.pytorch.org/whl/cpu
# restart your runtime after this step
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting ipympl
  Downloading ipympl-0.9.2-py2.py3-none-any.whl (510 kB)
                             510.3/510.3
KB 5.0 MB/s eta 0:00:00
Collecting torchviz
  Downloading torchviz-0.0.2.tar.gz (4.9 kB)
  Preparing metadata (setup.py) … done
Collecting matplotlib<4,>=3.4.0
  Downloading
matplotlib-3.6.3-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (9.4
MB)
                             9.4/9.4 MB
51.9 MB/s eta 0:00:00
Requirement already satisfied: pillow in /usr/local/lib/python3.8/dist-
packages (from ipympl) (7.1.2)
Requirement already satisfied: ipython-genutils in
/usr/local/lib/python3.8/dist-packages (from ipympl) (0.2.0)
Requirement already satisfied: traitlets<6 in /usr/local/lib/python3.8/dist-
packages (from ipympl) (5.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages
(from ipympl) (1.21.6)
Requirement already satisfied: ipython<9 in /usr/local/lib/python3.8/dist-
packages (from ipympl) (7.9.0)
Requirement already satisfied: ipywidgets<9,>=7.6.0 in
/usr/local/lib/python3.8/dist-packages (from ipympl) (7.7.1)
Requirement already satisfied: torch in /usr/local/lib/python3.8/dist-packages
(from torchviz) (1.13.1+cu116)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-
packages (from torchviz) (0.10.1)
Requirement already satisfied: pygments in /usr/local/lib/python3.8/dist-
packages (from ipython<9->ipympl) (2.6.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.8/dist-
```

```
packages (from ipython<9->ipympl) (0.2.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.8/dist-
packages (from ipython<9->ipympl) (4.4.2)
Collecting jedi>=0.10
  Downloading jedi-0.18.2-py2.py3-none-any.whl (1.6 MB)
                           1.6/1.6 MB
44.7 MB/s eta 0:00:00
Requirement already satisfied: pexpect in /usr/local/lib/python3.8/dist-
packages (from ipython<9->ipympl) (4.8.0)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.8/dist-
packages (from ipython<9->ipympl) (0.7.5)
Requirement already satisfied: prompt-toolkit<2.1.0,>=2.0.0 in
/usr/local/lib/python3.8/dist-packages (from ipython<9->ipympl) (2.0.10)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.8/dist-packages (from ipython<9->ipympl) (57.4.0)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in
/usr/local/lib/python3.8/dist-packages (from ipywidgets<9,>=7.6.0->ipympl)
(3.0.5)
Requirement already satisfied: widgetsnbextension~=3.6.0 in
/usr/local/lib/python3.8/dist-packages (from ipywidgets<9,>=7.6.0->ipympl)
(3.6.1)
Requirement already satisfied: ipykernel>=4.5.1 in
/usr/local/lib/python3.8/dist-packages (from ipywidgets<9,>=7.6.0->ipympl)
(5.3.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-
packages (from matplotlib<4,>=3.4.0->ipympl) (21.3)
Collecting contourpy>=1.0.1
  Downloading
contourpy-1.0.7-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (300
kB)
                           300.0/300.0
KB 9.9 MB/s eta 0:00:00
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.8/dist-packages (from matplotlib<4,>=3.4.0->ipympl)
(0.11.0)
Collecting fonttools>=4.22.0
  Downloading fonttools-4.38.0-py3-none-any.whl (965 kB)
                          965.4/965.4 KB
26.5 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.8/dist-packages (from matplotlib<4,>=3.4.0->ipympl)
(2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib<4,>=3.4.0->ipympl)
(3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib<4,>=3.4.0->ipympl)
```

```
(1.4.4)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.8/dist-packages (from torch->torchviz) (4.4.0)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.8/dist-
packages (from ipykernel>=4.5.1->ipywidgets<9,>=7.6.0->ipympl) (6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.8/dist-
packages (from ipykernel>=4.5.1->ipywidgets<9,>=7.6.0->ipympl) (6.0.4)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
/usr/local/lib/python3.8/dist-packages (from jedi>=0.10->ipython<9->ipympl)
(0.8.3)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.8/dist-
packages (from prompt-toolkit<2.1.0,>=2.0.0->ipython<9->ipympl) (1.15.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.8/dist-packages
(from prompt-toolkit<2.1.0,>=2.0.0->ipython<9->ipympl) (0.2.5)
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.8/dist-
packages (from widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (5.7.16)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.8/dist-
packages (from pexpect->ipython<9->ipympl) (0.7.0)
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.8/dist-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(23.2.1)
Requirement already satisfied: nbformat in /usr/local/lib/python3.8/dist-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(5.7.1)
Requirement already satisfied: nbconvert<6.0 in /usr/local/lib/python3.8/dist-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(5.6.1)
Requirement already satisfied: prometheus-client in
/usr/local/lib/python3.8/dist-packages (from
notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(0.15.0)
Requirement already satisfied: terminado>=0.8.1 in
/usr/local/lib/python3.8/dist-packages (from
notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(0.13.3)
Requirement already satisfied: Send2Trash in /usr/local/lib/python3.8/dist-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(1.8.0)
Requirement already satisfied: jinja2<=3.0.0 in /usr/local/lib/python3.8/dist-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(2.11.3)
Requirement already satisfied: jupyter-core>=4.4.0 in
/usr/local/lib/python3.8/dist-packages (from
```

```
notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(5.1.3)
Requirement already satisfied: MarkupSafe>=0.23 in
/usr/local/lib/python3.8/dist-packages (from jinja2<=3.0.0->notebook>=4.4.1->wid
getsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (2.0.1)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.8/dist-packages (from jupyter-core>=4.4.0->notebook>=4.4.
1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (2.6.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.8/dist-packages
(from nbconvert<6.0->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=
7.6.0->ipympl) (5.0.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.8/dist-packages (from nbconvert<6.0->notebook>=4.4.1->wid
getsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (0.4)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.8/dist-
packages (from nbconvert<6.0->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywid
gets<9,>=7.6.0->ipympl) (0.7.1)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.8/dist-packages (from nbconvert<6.0->notebook>=4.4.1->wid
getsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (0.8.4)
Requirement already satisfied: testpath in /usr/local/lib/python3.8/dist-
packages (from nbconvert<6.0->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywid
gets<9,>=7.6.0->ipympl) (0.6.0)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.8/dist-packages (from nbconvert<6.0->notebook>=4.4.1->wid
getsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (1.5.0)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.8/dist-
packages (from nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<
9,>=7.6.0->ipympl) (4.3.3)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.8/dist-
packages (from nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<
9,>=7.6.0->ipympl) (2.16.2)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.8/dist-
packages (from jsonschema>=2.6->nbformat->notebook>=4.4.1->widgetsnbextension~=3
.6.0->ipywidgets<9,>=7.6.0->ipympl) (22.2.0)
Requirement already satisfied: importlib-resources>=1.4.0 in
/usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbformat->notebook
>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (5.10.2)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
/usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbformat->notebook
>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (0.19.3)
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-
packages (from bleach->nbconvert<6.0->notebook>=4.4.1->widgetsnbextension~=3.6.0
->ipywidgets<9,>=7.6.0->ipympl) (0.5.1)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.8/dist-
packages (from importlib-resources>=1.4.0->jsonschema>=2.6->nbformat->notebook>=
4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (3.11.0)
Building wheels for collected packages: torchviz
```

```
  Building wheel for torchviz (setup.py) … done
  Created wheel for torchviz: filename=torchviz-0.0.2-py3-none-any.whl size=4151
sha256=4a07ca4dda0fb37c587e2f2e3eb3f2ee5f88ec1e372edf02cadb98529c3c31aa
  Stored in directory: /root/.cache/pip/wheels/05/7d/1b/8306781244e42ede119edbb0
53bdcda1c1f424ca226165a417
Successfully built torchviz
Installing collected packages: jedi, fonttools, contourpy, torchviz, matplotlib,
ipympl
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.2.2
    Uninstalling matplotlib-3.2.2:
      Successfully uninstalled matplotlib-3.2.2
Successfully installed contourpy-1.0.7 fonttools-4.38.0 ipympl-0.9.2 jedi-0.18.2
matplotlib-3.6.3 torchviz-0.0.2
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/, https://download.pytorch.org/whl/cpu
Collecting torch==1.13
  Downloading https://download.pytorch.org/whl/cpu/torch-1.13.0%2Bcpu-
cp38-cp38-linux_x86_64.whl (198.5 MB)
                           198.5/198.5
MB 5.3 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.8/dist-packages (from torch==1.13) (4.4.0)
Installing collected packages: torch
  Attempting uninstall: torch
    Found existing installation: torch 1.13.1+cu116
    Uninstalling torch-1.13.1+cu116:
ERROR: Operation cancelled by user
```

[1]:
```
import math
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
from torch.autograd import Variable
import tqdm

import IPython
from ipywidgets import interactive, widgets, Layout
from IPython.display import display, HTML
```

[2]:
```
print(torch.__version__, torch.cuda.is_available())
# Homework 0 does not require a GPU
```

```
1.13.1+cu116 False
```

[3]:
```
# enable matplotlib widgets;

# on Google Colab
from google.colab import output
output.enable_custom_widget_manager()

%matplotlib widget
```

[38]:
```
# Constants
cap_value = 1e-6          # Farads
R_init = 500              # Ohms
cutoff_mag = 1. / math.sqrt(2)
cutoff_dB = 20 * math.log10(cutoff_mag)
dataset_size = 1000
max_training_steps = 100000
```

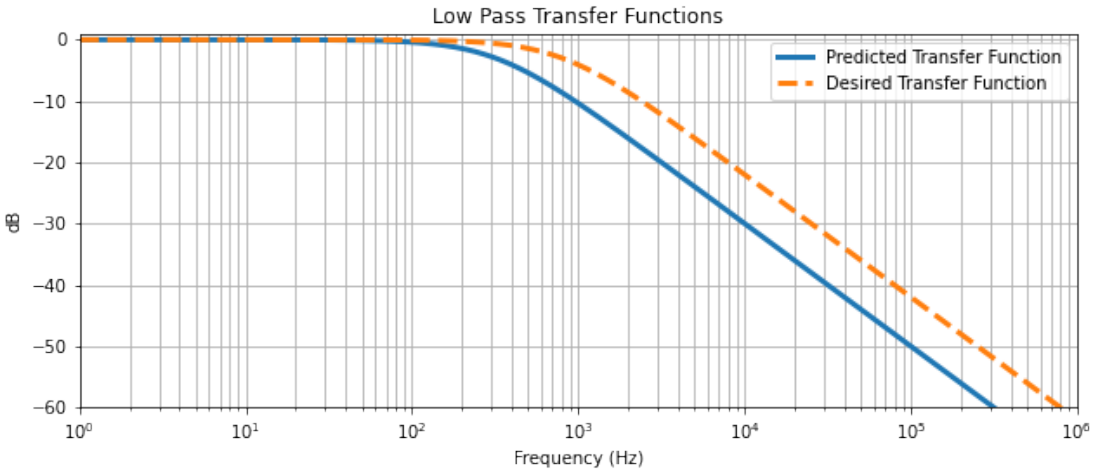
0.1 (a) Designing a Low Pass Filter by Matching Transfer Functions

[39]:
```
# Transfer function: evaluates magnitude of given frequencies for a resistor␣
 ↪value in the low pass circuit
def evaluate_lp_circuit(freqs, R_low):
    return 1. / torch.sqrt(1 + (R_low * cap_value * freqs) ** 2)
```

[40]:
```
# Plot transfer function for a given low pass circuit
fig = plt.figure(figsize=(9, 4))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
mags = 20 * torch.log10(evaluate_lp_circuit(ws, R_init))
R_low_des = 1 / (2 * math.pi * 800 * cap_value)
mags_des = 20 * torch.log10(evaluate_lp_circuit(ws, R_low_des))
tf, = plt.semilogx(ws / (2 * math.pi), mags, linewidth=3)
tf_des, = plt.semilogx(ws / (2 * math.pi), mags_des, linestyle="--",␣
 ↪linewidth=3)
plt.xlim([1, 1e6])
plt.ylim([-60, 1])
plt.title("Low Pass Transfer Functions")
plt.xlabel("Frequency (Hz)")
plt.ylabel("dB")
plt.grid(which="both")
leg = plt.legend(["Predicted Transfer Function", "Desired Transfer Function"])
plt.tight_layout()

# Main update function for interactive plot
def update_tfs(R=R_init):
    mags = 20 * torch.log10(evaluate_lp_circuit(ws, R))
    tf.set_data(ws / (2 * math.pi), mags)
    fig.canvas.draw_idle()
```

```
# Include sliders for relevant quantities
ip = interactive(update_tfs,
                 R=widgets.IntSlider(value=R_init, min=1, max=1000, step=1,␣
 ↪description="R", layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=500, description='R',␣
 ↪layout=Layout(width='100%'), max=1000, min=1), Out…
```

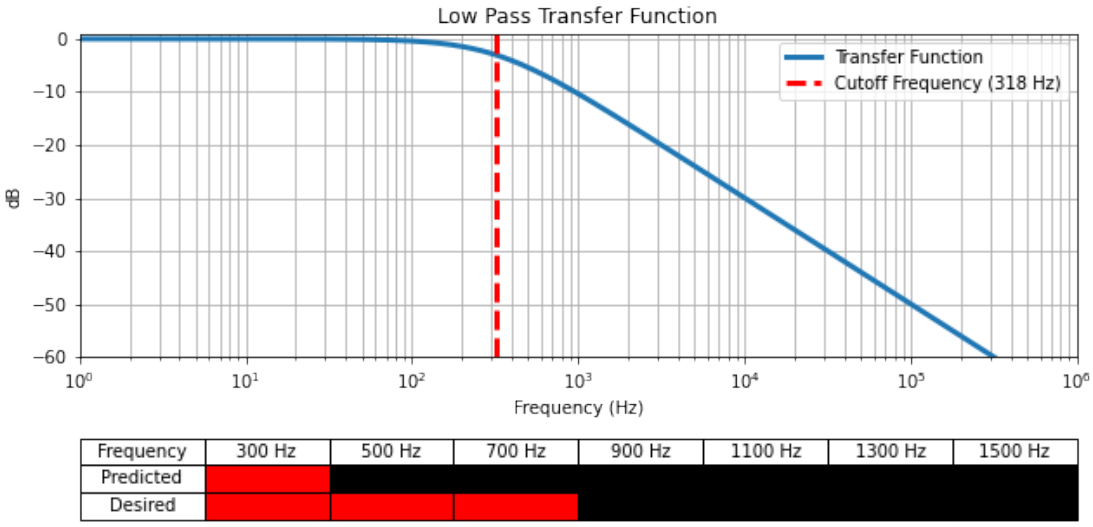


0.2 (b) Designing a Low pass Filter from Binary Data

[31]:
```
# Plot transfer function for a given low pass circuit
fig = plt.figure(figsize=(9, 5))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
mags = 20 * torch.log10(evaluate_lp_circuit(ws, R_init))
cutoff = ws[np.argmax(mags < cutoff_dB)]
tf, = plt.semilogx(ws / (2 * math.pi), mags, linewidth=3)
cut = plt.axvline(cutoff / (2 * math.pi), c="red", linestyle="--", linewidth=3)
plt.xlim([1, 1e6])
plt.ylim([-60, 1])
plt.title("Low Pass Transfer Function")
plt.xlabel("Frequency (Hz)")
plt.ylabel("dB")
plt.grid(which="both")
leg = plt.legend(["Transfer Function", f"Cutoff Frequency ({1 / (2 * math.pi *␣
 ↪R_init * cap_value):.0f} Hz)"])

# Plot table of LED on/off values (predicted and desired)
ws_test = 2 * math.pi * np.linspace(300, 1500, num=7)
table_txt = np.zeros((3, len(ws_test) + 1), dtype="U15")
```

```
table_txt[0, :] = ["Frequency"] + [f"{w / (2 * math.pi):.0f} Hz" for w in␣
↪ws_test]
table_txt[1:, 0] = ["Predicted", "Desired"]
table_colors = np.zeros_like(table_txt, dtype=(np.int32, (3,)))
table_colors[-1, 1:4] = (1, 0, 0)
table_colors[1, 1] = (1, 0, 0)
table_colors[:, :1] = (1, 1, 1)
table_colors[:1, :] = (1, 1, 1)
tab = plt.table(table_txt, table_colors, bbox=[0.0, -0.5, 1.0, 0.25],␣
↪cellLoc="center")
plt.tight_layout()

# Main update function for interactive plot
def update_lights(R=R_init):
    mags = 20 * torch.log10(evaluate_lp_circuit(ws, R))
    cutoff = ws[np.argmax(mags < cutoff_dB)]
    tf.set_data(ws / (2 * math.pi), mags)
    cut.set_xdata(cutoff / (2 * math.pi))
    for i, w in enumerate(ws_test):
        if w < cutoff:
            tab[(1, i+1)].set_facecolor((1, 0, 0))
        else:
            tab[(1, i+1)].set_facecolor((0, 0, 0))
    leg.get_texts()[1].set_text(f"Cutoff Frequency ({1 / (2 * math.pi * R *␣
↪cap_value):.0f} Hz)")
    fig.canvas.draw_idle()

# Include sliders for relevant quantities
ip = interactive(update_lights,
                 R=widgets.IntSlider(value=R_init, min=1, max=1000, step=1,␣
↪description="R", layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=500, description='R',␣
↪layout=Layout(width='100%'), max=1000, min=1), Out…
```



Frequency	300 Hz	500 Hz	700 Hz	900 Hz	1100 Hz	1300 Hz	1500 Hz
Predicted							
Desired							

0.3 (c) Learning a Low Pass Filter from Desired Transfer Function Samples

[32]:
```
# PyTorch model of the low pass circuit (for training)
class LowPassCircuit(nn.Module):
    def __init__(self, R=None):
        super().__init__()
        self.R = nn.Parameter(torch.tensor(R, dtype=float) if R is not None␣
 ↪else torch.rand(1) * 1000)

    # Note: the forward function is called automatically when the __call__␣
 ↪function of this object is called
    def forward(self, freqs):
        return evaluate_lp_circuit(freqs, self.R)

# Generate training data in a uniform log scale of frequences, then evaluate␣
 ↪using the true transfer function
def generate_lp_training_data(n):
    rand_ws = 2 * math.pi * torch.pow(10, torch.rand(n) * 6)
    labels = evaluate_lp_circuit(rand_ws, R_low_des)
    return rand_ws, labels

# Train a given low pass filter
def train_lp_circuit_tf(circuit, loss_fn, dataset_size, max_training_steps, lr):

    R_values = [float(circuit.R.data)]
    grad_values = [np.nan]
```

```
    train_data = generate_lp_training_data(dataset_size)
    print(f"Initial Resistor Value: R = {float(circuit.R.data):.0f}")
    iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
    for i in iter_bar:
        pred = circuit(train_data[0])
        loss = loss_fn(pred, train_data[1]).mean()
        grad = torch.autograd.grad(loss, circuit.R)
        with torch.no_grad():
            circuit.R -= lr * grad[0]

        R_values.append(float(circuit.R.data))
        grad_values.append(float(grad[0].data))
        iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f},␣
↪R={float(circuit.R.data):.0f}")
        if loss.data < 1e-6 or abs(grad[0].data) < 1e-6:
            break

    print(f"Final Resistor Value: R = {float(circuit.R.data):.0f}")
    return train_data, R_values, grad_values
```

[ ]:
```
# Create a circuit, use mean squared error loss w/ learning rate of 200
circuit = LowPassCircuit(200)
loss_fn = lambda x, y: (x - y) ** 2
lr = 200
train_data_low_tf, R_values_low_tf, grad_values_low_tf =␣
↪train_lp_circuit_tf(circuit, loss_fn, dataset_size, max_training_steps, lr)
```

```
Initial Resistor Value: R = 200
```

```
Training Iter:   0%|          | 0/100000 [00:00<?, ?it/s, Loss: 0.000, R=200]
```

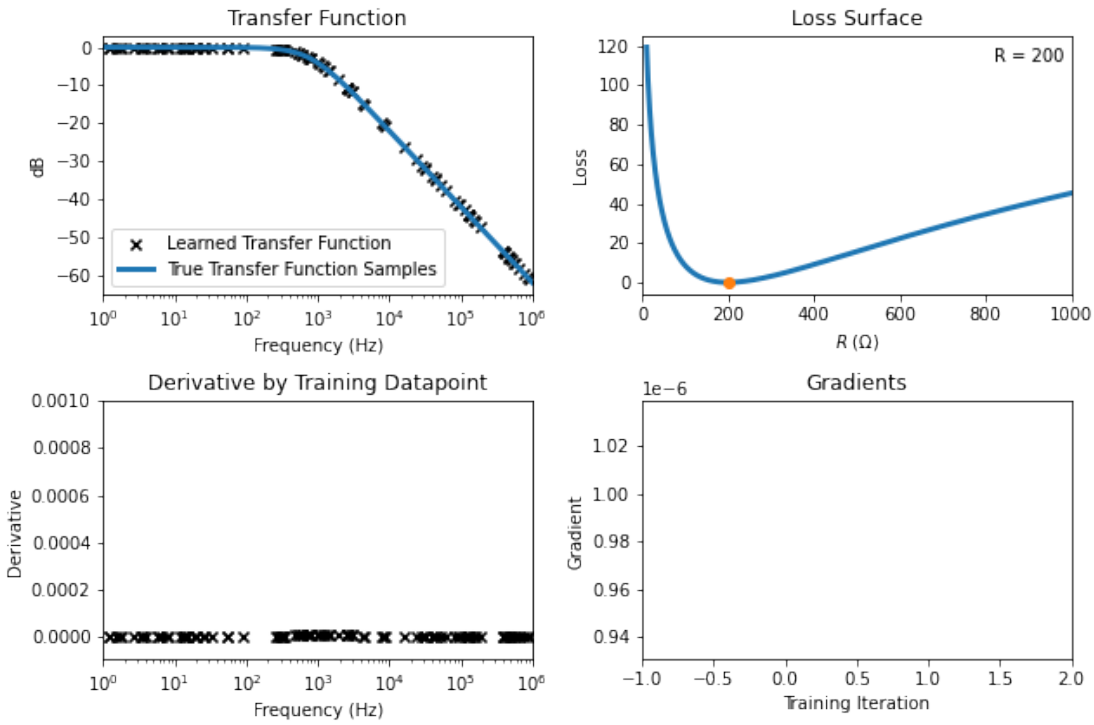
```
Final Resistor Value: R = 200
```

[ ]:
```
# Plot transfer function over training
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
subsample = int(dataset_size / 100)
ax1.scatter(train_data_low_tf[0][::subsample] / (2 * math.pi), 20 * torch.
↪log10(train_data_low_tf[1][::subsample]), c="k", marker="x")
learned_tf, = ax1.semilogx(ws / (2 * math.pi), 20 * torch.
↪log10(evaluate_lp_circuit(ws, R_values_low_tf[0])), linewidth=3)
ax1.set_xlim([1, 1e6])
ax1.set_title("Transfer Function")
ax1.set_xlabel("Frequency (Hz)")
ax1.set_ylabel("dB")
ax1.legend(["Learned Transfer Function", "True Transfer Function Samples"])
```

```
# Show loss surface over training
eval_pts = torch.arange(10, 1001, 1)
eval_vals = evaluate_lp_circuit(train_data_low_tf[0][:, None], eval_pts[None, :
 ↪])
loss_surface_mse = loss_fn(eval_vals, train_data_low_tf[1][:, None].
 ↪expand(eval_vals.shape))
ax2.plot(eval_pts, loss_surface_mse.sum(0), linewidth=3)
cur_loss, = ax2.plot(R_values_low_tf[0], loss_surface_mse[:,␣
 ↪int(R_values_low_tf[0] - 10)].sum(0), marker="o")
cur_loss_label = ax2.annotate(f"R = {R_values_low_tf[0]:.0f}", (0, 0),␣
 ↪xytext=(0.82, 0.9), textcoords='axes fraction')
ax2.set_title("Loss Surface")
ax2.set_xlim([0, 1000])
ax2.set_xlabel("$R \; (\Omega)$")
ax2.set_ylabel("Loss")

# Show loss contributions of each data point
cur_circuit = LowPassCircuit(R_values_low_tf[0])
data_losses = loss_fn(cur_circuit(train_data_low_tf[0][::subsample]),␣
 ↪(train_data_low_tf[1][::subsample]).float())
data_grads = torch.zeros(len(data_losses))
for i, dl in enumerate(data_losses):
    data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
data_grads_scat = ax3.scatter(train_data_low_tf[0][::subsample] / (2 * math.
 ↪pi), data_grads, marker="x", c="k")
ax3.set_xscale("log")
ax3.set_ylabel("Derivative")
ax3.set_xlim([1, 1e6])
ax3.set_ylim([-1e-4, 1e-3])
ax3.set_xlabel("Frequency (Hz)")
ax3.set_title("Derivative by Training Datapoint")

# Show total gradient at each training iteration
ax4.plot(np.arange(len(grad_values_low_tf)), grad_values_low_tf, linewidth=3)
cur_iter, = ax4.plot(0, grad_values_low_tf[0], marker="o")
cur_grad_label = ax4.annotate(f"Grad = {grad_values_low_tf[0]:.2e}", (0, 0),␣
 ↪xytext=(0.65, 0.9), textcoords='axes fraction')
ax4.set_xlabel("Training Iteration")
ax4.set_ylabel("Gradient")
ax4.set_title("Gradients")
ax4.set_xlim([-1, len(grad_values_low_tf)])

plt.tight_layout()

# Main update function for interactive plots
def update_iter_tf(t=0):
```

```
    learned_tf.set_data(ws / (2 * math.pi), 20 * torch.
↪log10(evaluate_lp_circuit(ws, R_values_low_tf[t])))
    cur_loss.set_data(R_values_low_tf[t], loss_surface_mse[:,␣
↪int(R_values_low_tf[t] - 10)].sum(0))
    cur_loss_label.set_text(f"R = {R_values_low_tf[t]:.0f}")
    cur_iter.set_data(t, grad_values_low_tf[t])
    cur_grad_label.set_text(f"Grad = {grad_values_low_tf[t]:.2e}")
    cur_circuit = LowPassCircuit(R_values_low_tf[t])
    data_losses = loss_fn(cur_circuit(train_data_low_tf[0][::subsample]),␣
↪(train_data_low_tf[1][::subsample]).float())
    data_grads = torch.zeros(len(data_losses))
    for i, dl in enumerate(data_losses):
        data_grads[i] = torch.autograd.grad(dl, cur_circuit.R,␣
↪retain_graph=True)[0]
    data_grads_scat.set_offsets(torch.stack((train_data_low_tf[0][::subsample] /
↪ (2 * math.pi), data_grads)).T)
    fig.canvas.draw_idle()

# Include sliders for relevant quantities
ip = interactive(update_iter_tf,
                 t=widgets.IntSlider(value=0, min=0, max=len(R_values_low_tf) -␣
↪1, step=1, description="Training Iteration", style={'description_width':␣
↪'initial'}, layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=0, description='Training Iteration',␣
↪layout=Layout(width='100%'), max=1,…
```



0.4 (d) Learning a Low Pass Filter from Binary Data with Mean Squared Error Loss

[41]:
```
# Train a given low pass filter from binary data
def train_lp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps,␣
↪lr):

    R_values = [float(circuit.R.data)]
    grad_values = [np.nan]
    train_data = generate_lp_training_data(dataset_size)
    print(f"Initial Resistor Value: R = {float(circuit.R.data):.0f}")
    iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
    for i in iter_bar:
        pred = circuit(train_data[0])
        ### YOUR CODE HERE
        loss = loss_fn(pred, (train_data[1] > cutoff_mag).float()).mean()
        # loss = loss_fn(?, ?).mean()
        ### END YOUR CODE
        grad = torch.autograd.grad(loss, circuit.R)
        with torch.no_grad():
            circuit.R -= lr * grad[0]

        R_values.append(float(circuit.R.data))
```

```
        grad_values.append(float(grad[0].data))
        iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f},␣
↪R={float(circuit.R.data):.0f}")
        if loss.data < 1e-6 or abs(grad[0].data) < 1e-6:
            break

    print(f"Final Resistor Value: R = {float(circuit.R.data):.0f}")
    return train_data, R_values, grad_values
```

[42]:
```
# Create a circuit, use MSE loss with learning rate of 200
circuit = LowPassCircuit(500)
loss_fn = lambda x, y: (x - y) ** 2 # x:pred
lr = 200
train_data_low_bin, R_values_low_bin, grad_values_low_bin =␣
↪train_lp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps,␣
↪lr)
```

```
Initial Resistor Value: R = 500
```

```
Training Iter:  61%|       | 61101/100000 [03:27<02:12, 294.31it/s, Loss:
0.016, R=361]
```

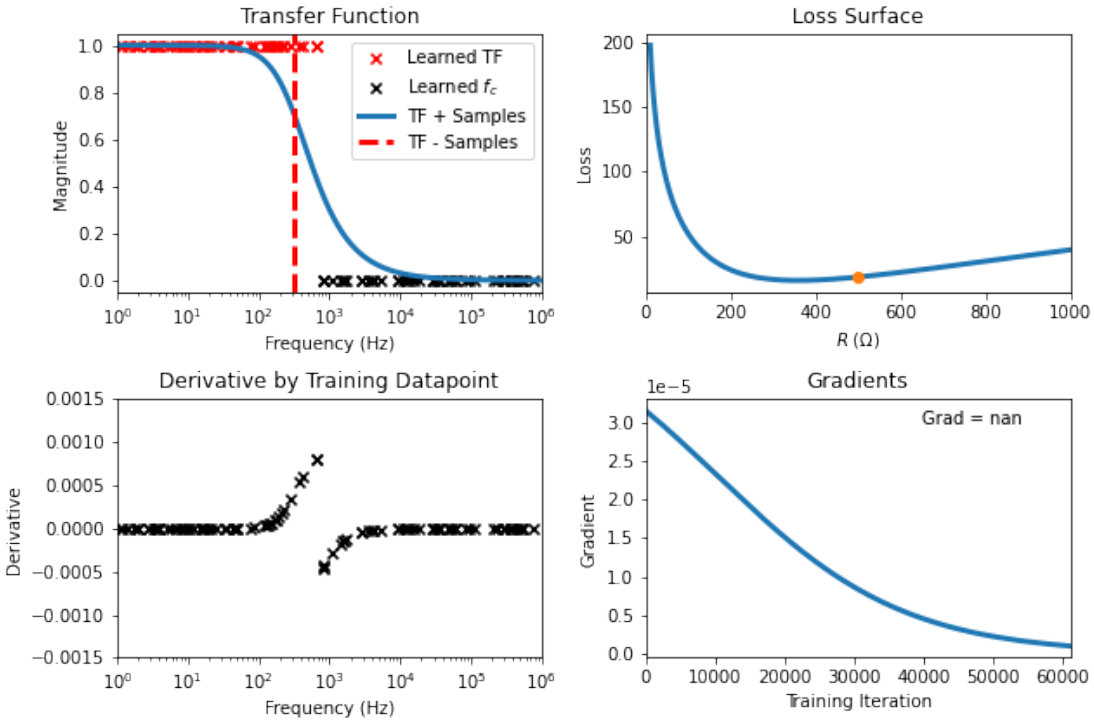
```
Final Resistor Value: R = 361
```

[43]:
```
# Plot transfer function over training
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
subsample = int(dataset_size / 100)
train_data_mask = train_data_low_bin[1][::subsample] > cutoff_mag
ax1.scatter(train_data_low_bin[0][::subsample][train_data_mask] / (2 * math.
↪pi), np.ones(train_data_mask.sum()), c="r", marker="x")
ax1.scatter(train_data_low_bin[0][::subsample][~train_data_mask] / (2 * math.
↪pi), np.zeros((~train_data_mask).sum()), c="k", marker="x")
mags = evaluate_lp_circuit(ws, R_values_low_bin[0])
learned_tf, = ax1.semilogx(ws / (2 * math.pi), mags, linewidth=3)
cutoff = ws[np.argmax(mags < cutoff_mag)]
cut = ax1.axvline(cutoff / (2 * math.pi), c="red", linestyle="--", linewidth=3)
ax1.set_xlim([1, 1e6])
ax1.set_title("Transfer Function")
ax1.set_xlabel("Frequency (Hz)")
ax1.set_ylabel("Magnitude")
ax1.legend(["Learned TF", "Learned $f_c$", "TF + Samples", "TF - Samples"])

# Show loss surface over training
eval_pts = torch.arange(10, 1001, 1)
eval_vals = evaluate_lp_circuit(train_data_low_bin[0][:, None], eval_pts[None, :
↪])
```

```
loss_surface_mse = loss_fn(eval_vals, (train_data_low_bin[1][:, None].
 ↪expand(eval_vals.shape) > cutoff_mag).float())
ax2.plot(eval_pts, loss_surface_mse.sum(0), linewidth=3)
cur_loss, = ax2.plot(R_values_low_bin[0], loss_surface_mse[:,␣
 ↪int(R_values_low_bin[0] - 10)].sum(0), marker="o")
cur_loss_label = ax2.annotate(f"R = {R_values_low_bin[0]:.0f}", (0, 0),␣
 ↪xytext=(0.82, 0.9), textcoords='axes fraction')
ax2.set_title("Loss Surface")
ax2.set_xlim([0, 1000])
ax2.set_xlabel("$R \; (\Omega)$")
ax2.set_ylabel("Loss")

# Show loss contributions of each data point
cur_circuit = LowPassCircuit(R_values_low_bin[0])
data_losses = loss_fn(cur_circuit(train_data_low_bin[0][::subsample]),␣
 ↪(train_data_low_bin[1][::subsample] > cutoff_mag).float())
data_grads = torch.zeros(len(data_losses))
for i, dl in enumerate(data_losses):
    data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
data_grads_scat = ax3.scatter(train_data_low_bin[0][::subsample] / (2 * math.
 ↪pi), data_grads, marker="x", c="k")
ax3.set_xscale("log")
ax3.set_ylabel("Derivative")
ax3.set_xlim([1, 1e6])
ax3.set_ylim([-1.5e-3, 1.5e-3])
ax3.set_xlabel("Frequency (Hz)")
ax3.set_title("Derivative by Training Datapoint")

# Show gradient at each training iteration
ax4.plot(np.arange(len(grad_values_low_bin)), grad_values_low_bin, linewidth=3)
cur_iter, = ax4.plot(0, grad_values_low_bin[0], marker="o")
cur_grad_label = ax4.annotate(f"Grad = {grad_values_low_bin[0]:.2e}", (0, 0),␣
 ↪xytext=(0.65, 0.9), textcoords='axes fraction')
ax4.set_xlabel("Training Iteration")
ax4.set_ylabel("Gradient")
ax4.set_title("Gradients")
ax4.set_xlim([-1, len(grad_values_low_bin)])

plt.tight_layout()

# Main update function for interactive plots
def update_iter_low_bin(t=0):
    mags = evaluate_lp_circuit(ws, R_values_low_bin[t])
    learned_tf.set_data(ws / (2 * math.pi), mags)
    cutoff = ws[np.argmax(mags < cutoff_mag)]
    cut.set_xdata(cutoff / (2 * math.pi))
```

```
    cur_loss.set_data(R_values_low_bin[t], loss_surface_mse[:,␣
↪int(R_values_low_bin[t] - 10)].sum(0))
    cur_loss_label.set_text(f"R = {R_values_low_bin[t]:.0f}")
    cur_iter.set_data(t, grad_values_low_bin[t])
    cur_grad_label.set_text(f"Grad = {grad_values_low_bin[t]:.2e}")
    cur_circuit = LowPassCircuit(R_values_low_bin[t])
    data_losses = loss_fn(cur_circuit(train_data_low_bin[0][::subsample]),␣
↪(train_data_low_bin[1][::subsample] > cutoff_mag).float())
    data_grads = torch.zeros(len(data_losses))
    for i, dl in enumerate(data_losses):
        data_grads[i] = torch.autograd.grad(dl, cur_circuit.R,␣
↪retain_graph=True)[0]
    data_grads_scat.set_offsets(torch.stack((train_data_low_bin[0][::subsample]␣
↪/ (2 * math.pi), data_grads)).T)
    fig.canvas.draw_idle()

# Include sliders for relevant quantities
ip = interactive(update_iter_low_bin,
                 t=widgets.IntSlider(value=0, min=0, max=len(R_values_low_bin)␣
↪- 1, step=1, description="Training Iteration", style={'description_width':␣
↪'initial'}, layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=0, description='Training Iteration',␣
↪layout=Layout(width='100%'), max=61…
```



0.5 (e) Learning a Low Pass Filter from Binary Data with a Different Loss

[ ]:
```
circuit = LowPassCircuit(500)
### YOUR CODE HERE
lr = 200
loss_fn = lambda x, y: (1-y) * torch.where(x-cutoff_mag>0,x-cutoff_mag, 0) + y␣
↪* torch.where(cutoff_mag-x>0,cutoff_mag-x, 0)# x:pred
# loss_fn = lambda x, y: ?
### END YOUR CODE
train_data_low_bin, R_values_low_bin, grad_values_low_bin =␣
↪train_lp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps,␣
↪lr)
```

```
Initial Resistor Value: R = 500
```

```
Training Iter:  76%|       | 76095/100000 [05:23<01:41, 235.03it/s, Loss:
0.000, R=200]
```

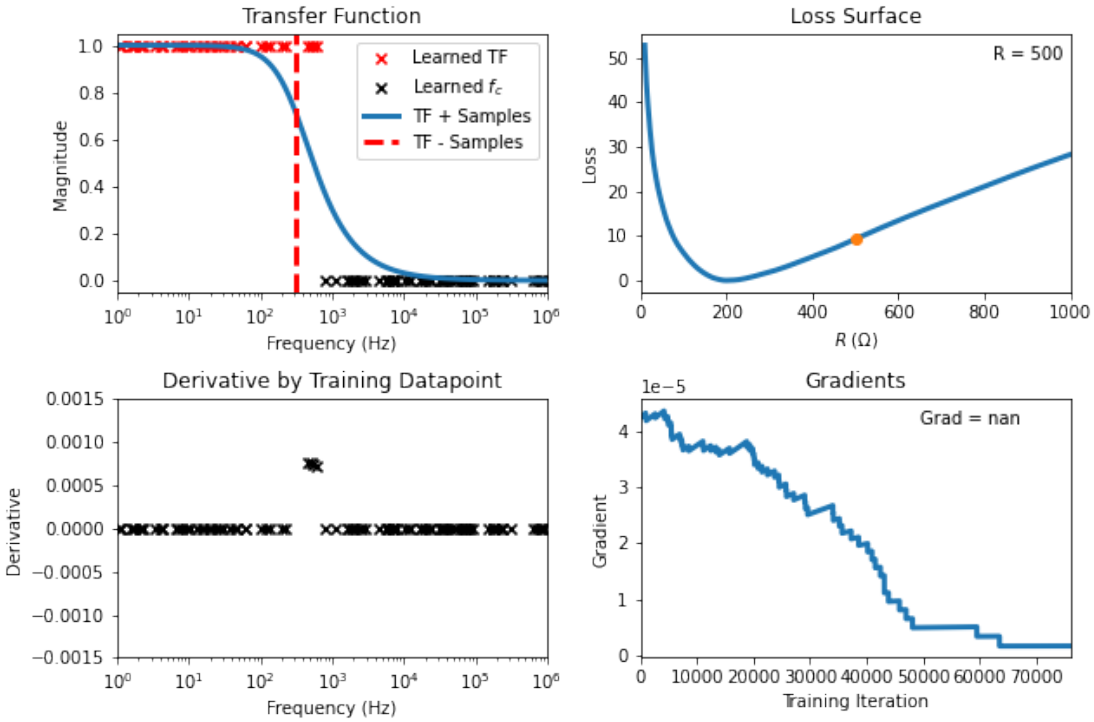
```
Final Resistor Value: R = 200
```

[ ]:
```
# Plot transfer function over training
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
subsample = int(dataset_size / 100)
train_data_mask = train_data_low_bin[1][::subsample] > cutoff_mag
ax1.scatter(train_data_low_bin[0][::subsample][train_data_mask] / (2 * math.
↪pi), np.ones(train_data_mask.sum()), c="r", marker="x")
ax1.scatter(train_data_low_bin[0][::subsample][~train_data_mask] / (2 * math.
↪pi), np.zeros((~train_data_mask).sum()), c="k", marker="x")
mags = evaluate_lp_circuit(ws, R_values_low_bin[0])
learned_tf, = ax1.semilogx(ws / (2 * math.pi), mags, linewidth=3)
cutoff = ws[np.argmax(mags < cutoff_mag)]
cut = ax1.axvline(cutoff / (2 * math.pi), c="red", linestyle="--", linewidth=3)
ax1.set_xlim([1, 1e6])
ax1.set_title("Transfer Function")
ax1.set_xlabel("Frequency (Hz)")
ax1.set_ylabel("Magnitude")
ax1.legend(["Learned TF", "Learned $f_c$", "TF + Samples", "TF - Samples"])

# Show loss surface over training
eval_pts = torch.arange(10, 1001, 1)
eval_vals = evaluate_lp_circuit(train_data_low_bin[0][:, None], eval_pts[None, :
↪])
```

```
loss_surface_mse = loss_fn(eval_vals, (train_data_low_bin[1][:, None].
 ↪expand(eval_vals.shape) > cutoff_mag).float())
ax2.plot(eval_pts, loss_surface_mse.sum(0), linewidth=3)
cur_loss, = ax2.plot(R_values_low_bin[0], loss_surface_mse[:,␣
 ↪int(R_values_low_bin[0] - 10)].sum(0), marker="o")
cur_loss_label = ax2.annotate(f"R = {R_values_low_bin[0]:.0f}", (0, 0),␣
 ↪xytext=(0.82, 0.9), textcoords='axes fraction')
ax2.set_title("Loss Surface")
ax2.set_xlim([0, 1000])
ax2.set_xlabel("$R \; (\Omega)$")
ax2.set_ylabel("Loss")

# Show loss contributions of each data point
cur_circuit = LowPassCircuit(R_values_low_bin[0])
data_losses = loss_fn(cur_circuit(train_data_low_bin[0][::subsample]),␣
 ↪(train_data_low_bin[1][::subsample] > cutoff_mag).float())
data_grads = torch.zeros(len(data_losses))
for i, dl in enumerate(data_losses):
    data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
data_grads_scat = ax3.scatter(train_data_low_bin[0][::subsample] / (2 * math.
 ↪pi), data_grads, marker="x", c="k")
ax3.set_xscale("log")
ax3.set_ylabel("Derivative")
ax3.set_xlim([1, 1e6])
ax3.set_ylim([-1.5e-3, 1.5e-3])
ax3.set_xlabel("Frequency (Hz)")
ax3.set_title("Derivative by Training Datapoint")

# Show gradient at each training iteration
ax4.plot(np.arange(len(grad_values_low_bin)), grad_values_low_bin, linewidth=3)
cur_iter, = ax4.plot(0, grad_values_low_bin[0], marker="o")
cur_grad_label = ax4.annotate(f"Grad = {grad_values_low_bin[0]:.2e}", (0, 0),␣
 ↪xytext=(0.65, 0.9), textcoords='axes fraction')
ax4.set_xlabel("Training Iteration")
ax4.set_ylabel("Gradient")
ax4.set_title("Gradients")
ax4.set_xlim([-1, len(grad_values_low_bin)])

plt.tight_layout()

# Main update function for interactive plots
def update_iter_low_bin(t=0):
    mags = evaluate_lp_circuit(ws, R_values_low_bin[t])
    learned_tf.set_data(ws / (2 * math.pi), mags)
    cutoff = ws[np.argmax(mags < cutoff_mag)]
    cut.set_xdata(cutoff / (2 * math.pi))
```

```
    cur_loss.set_data(R_values_low_bin[t], loss_surface_mse[:,␣
↪int(R_values_low_bin[t] - 10)].sum(0))
    cur_loss_label.set_text(f"R = {R_values_low_bin[t]:.0f}")
    cur_iter.set_data(t, grad_values_low_bin[t])
    cur_grad_label.set_text(f"Grad = {grad_values_low_bin[t]:.2e}")
    cur_circuit = LowPassCircuit(R_values_low_bin[t])
    data_losses = loss_fn(cur_circuit(train_data_low_bin[0][::subsample]),␣
↪(train_data_low_bin[1][::subsample] > cutoff_mag).float())
    data_grads = torch.zeros(len(data_losses))
    for i, dl in enumerate(data_losses):
        data_grads[i] = torch.autograd.grad(dl, cur_circuit.R,␣
↪retain_graph=True)[0]
    data_grads_scat.set_offsets(torch.stack((train_data_low_bin[0][::subsample]␣
↪/ (2 * math.pi), data_grads)).T)
    fig.canvas.draw_idle()

# Include sliders for relevant quantities
ip = interactive(update_iter_low_bin,
                 t=widgets.IntSlider(value=0, min=0, max=len(R_values_low_bin)␣
↪- 1, step=1, description="Training Iteration", style={'description_width':␣
↪'initial'}, layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=0, description='Training Iteration',␣
↪layout=Layout(width='100%'), max=76…
```



0.6 (f) Learning a High Pass Filter from Binary Data

[7]:
```
# Transfer function: evaluates magnitude of given frequencies for a resistor␣
↪value in the high pass circuit
def evaluate_hp_circuit(freqs, R_high):
    ### YOUR CODE HERE
    return torch.sqrt((R_high * cap_value * freqs) ** 2) / torch.sqrt(1 +␣
↪(R_high * cap_value * freqs) ** 2)
    # return ?
    ### END YOUR CODE

# PyTorch model of the high pass circuit (for training)
class HighPassCircuit(nn.Module):
    def __init__(self, R=None):
        super().__init__()
        self.R = nn.Parameter(torch.tensor(R, dtype=float) if R is not None␣
↪else torch.rand(1) * 1000)

    def forward(self, freqs):
        return evaluate_hp_circuit(freqs, self.R)

# Generate training data in a uniform log scale of frequences, then evaluate␣
↪using the true transfer function
R_high_des = 1 / (2 * math.pi * 5000 * cap_value)
def generate_hp_training_data(n):
    rand_ws = 2 * math.pi * torch.pow(10, torch.rand(n) * 6)
    labels = evaluate_hp_circuit(rand_ws, R_high_des)
    return rand_ws, labels

# Train a given low pass filter from binary data
def train_hp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps,␣
↪lr):

    R_values = [float(circuit.R.data)]
    grad_values = [np.nan]
    train_data = generate_hp_training_data(dataset_size)
    print(f"Initial Resistor Value: R = {float(circuit.R.data):.0f}")
    iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
    for i in iter_bar:
        pred = circuit(train_data[0])
        loss = loss_fn(pred, (train_data[1] > cutoff_mag).float()).mean()
        ### YOUR CODE HERE
        grad = torch.autograd.grad(loss, circuit.R)
        # grad = torch.autograd.grad(?, ?)
```

```
        ### END YOUR CODE
        with torch.no_grad():
            ### YOUR CODE HERE
            # circuit.R -= ?
            circuit.R -= lr * grad[0]
            ### END YOUR CODE

        R_values.append(float(circuit.R.data))
        grad_values.append(float(grad[0].data))
        iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f},␣
↪R={float(circuit.R.data):.0f}")
        if loss.data < 1e-6 or abs(grad[0].data) < 1e-6:
            break

    print(f"Final Resistor Value: R = {float(circuit.R.data):.0f}")
    return train_data, R_values, grad_values
```

[9]:
```
# Create a circuit, use loss_fn with learning rate of 1000
circuit = HighPassCircuit(500)
### YOUR CODE HERE
loss_fn = lambda x, y: (1-y) * torch.where(x-cutoff_mag>0,x-cutoff_mag, 0) + y␣
↪* torch.where(cutoff_mag-x>0,cutoff_mag-x, 0)# x:pred
### END YOUR CODE
lr = 1000
train_data_high_bin, R_values_high_bin, grad_values_high_bin =␣
↪train_hp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps,␣
↪lr)
```

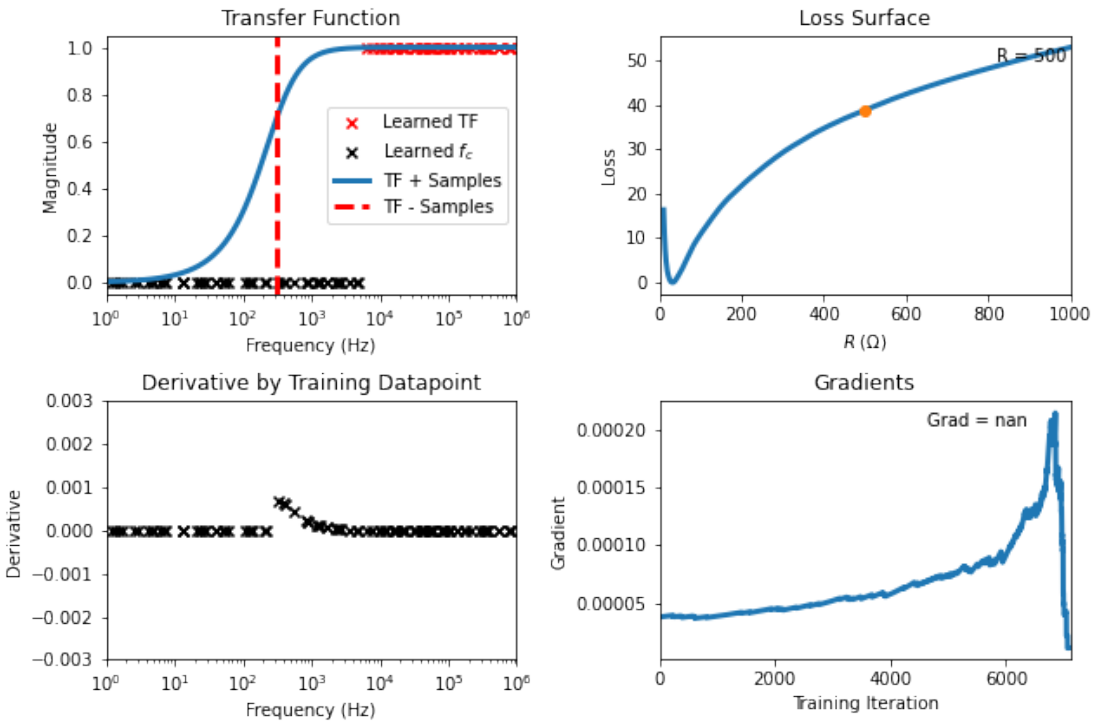
```
Initial Resistor Value: R = 500

Training Iter:   7%|          | 7115/100000 [00:26<05:42, 271.02it/s, Loss:
0.000, R=32]

Final Resistor Value: R = 32
```

[10]:
```
# Plot transfer function over training
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
subsample = int(dataset_size / 100)
train_data_mask = train_data_high_bin[1][::subsample] > cutoff_mag
ax1.scatter(train_data_high_bin[0][::subsample][train_data_mask] / (2 * math.
↪pi), np.ones(train_data_mask.sum()), c="r", marker="x")
ax1.scatter(train_data_high_bin[0][::subsample][~train_data_mask] / (2 * math.
↪pi), np.zeros((~train_data_mask).sum()), c="k", marker="x")
mags = evaluate_hp_circuit(ws, R_values_high_bin[0])
learned_tf, = ax1.semilogx(ws / (2 * math.pi), mags, linewidth=3)
cutoff = ws[np.argmax(mags > cutoff_mag)]
```

```
cut = ax1.axvline(cutoff / (2 * math.pi), c="red", linestyle="--", linewidth=3)
ax1.set_xlim([1, 1e6])
ax1.set_title("Transfer Function")
ax1.set_xlabel("Frequency (Hz)")
ax1.set_ylabel("Magnitude")
ax1.legend(["Learned TF", "Learned $f_c$", "TF + Samples", "TF - Samples"])

# Show loss surface over training
eval_pts = torch.arange(10, 1001, 1)
eval_vals = evaluate_hp_circuit(train_data_high_bin[0][:, None], eval_pts[None,␣
↪:])
loss_surface_mse = loss_fn(eval_vals, (train_data_high_bin[1][:, None].
↪expand(eval_vals.shape) > cutoff_mag).float())
ax2.plot(eval_pts, loss_surface_mse.sum(0), linewidth=3)
cur_loss, = ax2.plot(R_values_high_bin[0], loss_surface_mse[:,␣
↪int(R_values_high_bin[0] - 10)].sum(0), marker="o")
cur_loss_label = ax2.annotate(f"R = {R_values_high_bin[0]:.0f}", (0, 0),␣
↪xytext=(0.82, 0.9), textcoords='axes fraction')
ax2.set_title("Loss Surface")
ax2.set_xlim([0, 1000])
ax2.set_xlabel("$R \; (\Omega)$")
ax2.set_ylabel("Loss")

# Show loss contributions of each data point
cur_circuit = HighPassCircuit(R_values_high_bin[0])
data_losses = loss_fn(cur_circuit(train_data_high_bin[0][::subsample]),␣
↪(train_data_high_bin[1][::subsample] > cutoff_mag).float())
data_grads = torch.zeros(len(data_losses))
for i, dl in enumerate(data_losses):
    data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
data_grads_scat = ax3.scatter(train_data_high_bin[0][::subsample] / (2 * math.
↪pi), data_grads, marker="x", c="k")
ax3.set_xscale("log")
ax3.set_ylabel("Derivative")
ax3.set_xlim([1, 1e6])
ax3.set_ylim([-3e-3, 3e-3])
ax3.set_xlabel("Frequency (Hz)")
ax3.set_title("Derivative by Training Datapoint")

# Show gradient at each training iteration
ax4.plot(np.arange(len(grad_values_high_bin)), grad_values_high_bin,␣
↪linewidth=3)
cur_iter, = ax4.plot(0, grad_values_high_bin[0], marker="o")
cur_grad_label = ax4.annotate(f"Grad = {grad_values_high_bin[0]:.2e}", (0, 0),␣
↪xytext=(0.65, 0.9), textcoords='axes fraction')
ax4.set_xlabel("Training Iteration")
```

```
ax4.set_ylabel("Gradient")
ax4.set_title("Gradients")
ax4.set_xlim([-1, len(grad_values_high_bin)])

plt.tight_layout()

# Main update function for interactive plots
def update_iter_high_bin(t=0):
    mags = evaluate_hp_circuit(ws, R_values_high_bin[t])
    learned_tf.set_data(ws / (2 * math.pi), mags)
    cutoff = ws[np.argmax(mags > cutoff_mag)]
    cut.set_xdata(cutoff / (2 * math.pi))
    cur_loss.set_data(R_values_high_bin[t], loss_surface_mse[:,␣
↪int(R_values_high_bin[t] - 10)].sum(0))
    cur_loss_label.set_text(f"R = {R_values_high_bin[t]:.0f}")
    cur_iter.set_data(t, grad_values_high_bin[t])
    cur_grad_label.set_text(f"Grad = {grad_values_high_bin[t]:.2e}")
    cur_circuit = HighPassCircuit(R_values_high_bin[t])
    data_losses = loss_fn(cur_circuit(train_data_high_bin[0][::subsample]),␣
↪(train_data_high_bin[1][::subsample] > cutoff_mag).float())
    data_grads = torch.zeros(len(data_losses))
    for i, dl in enumerate(data_losses):
        data_grads[i] = torch.autograd.grad(dl, cur_circuit.R,␣
↪retain_graph=True)[0]
    data_grads_scat.set_offsets(torch.stack((train_data_high_bin[0][::
↪subsample] / (2 * math.pi), data_grads)).T)
    fig.canvas.draw_idle()

# Include sliders for relevant quantities
ip = interactive(update_iter_high_bin,
                 t=widgets.IntSlider(value=0, min=0, max=len(R_values_high_bin)␣
↪- 1, step=1, description="Training Iteration", style={'description_width':␣
↪'initial'}, layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=0, description='Training Iteration',␣
↪layout=Layout(width='100%'), max=71…
```



0.7 (g) Learning a Band Pass Filter from Binary Data

[16]:
```
# Transfer function: evaluates magnitude of given frequencies for resistor␣
 ↪values in the band pass circuit
def evaluate_bp_circuit(freqs, R_low, R_high):
    ### YOUR CODE HERE
    tmp = torch.sqrt((R_high * cap_value * freqs) ** 2) / torch.sqrt(1 +␣
 ↪(R_high * cap_value * freqs) ** 2)
    return 1 / torch.sqrt(1 + (R_low * cap_value * freqs) ** 2) * tmp
    ### END YOUR CODE

# PyTorch model of the band pass circuit (for training)
class BandPassCircuit(nn.Module):
    def __init__(self, R_low=None, R_high=None):
        super().__init__()
        self.R_low = nn.Parameter(torch.tensor(R_low, dtype=float) if R_low is␣
 ↪not None else torch.rand(1) * 1000)
        self.R_high = nn.Parameter(torch.tensor(R_high, dtype=float) if R_high␣
 ↪is not None else torch.rand(1) * 1000)

    def forward(self, freqs):
        return evaluate_bp_circuit(freqs, self.R_low, self.R_high)
```

```
# Generate training data in a uniform log scale of frequences, then evaluate␣
 ↪using true transfer function
R_low_des = 1 / (2 * math.pi * 4000 * cap_value)
R_high_des = 1 / (2 * math.pi * 1000 * cap_value)
def generate_bp_training_data(n):
    rand_ws = 2 * math.pi * torch.pow(10, torch.rand(n) * 6)
    labels = evaluate_bp_circuit(rand_ws, R_low_des, R_high_des)
    return rand_ws, labels

# Train a given low pass filter from binary data
def train_bp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps,␣
 ↪lr):

    R_values = [[float(circuit.R_low.data), float(circuit.R_high.data)]]
    grad_values = [[np.nan, np.nan]]
    train_data = generate_bp_training_data(dataset_size)
    print(f"Initial Resistor Values: R_low = {float(circuit.R_low.data):.0f},␣
 ↪R_high = {float(circuit.R_high.data):.0f}")
    iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
    for i in iter_bar:
        pred = circuit(train_data[0])
        loss = loss_fn(pred, (train_data[1] > cutoff_mag).float()).mean()
        ### YOUR CODE HERE
        grad = torch.autograd.grad(loss, [circuit.R_low, circuit.R_high])
        ### END YOUR CODE
        with torch.no_grad():
            ### YOUR CODE HERE
            # circuit.R_low -= ?
            # circuit.R_high -= ?
            circuit.R_low -= lr * grad[0]
            circuit.R_high -= lr * grad[1]
            ### END YOUR CODE

        R_values.append([float(circuit.R_low.data), float(circuit.R_high.data)])
        grad_values.append([float(grad[0].data), float(grad[1].data)])
        iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f},␣
 ↪R_low={float(circuit.R_low.data):.0f}, R_high={float(circuit.R_high.data):.
 ↪0f}")
        if loss.data < 1e-6 or (abs(grad[0].data) < 1e-6 and abs(grad[1].data)␣
 ↪< 1e-6):
            break

    print(f"Final Resistor Values: R_low = {float(circuit.R_low.data):.0f},␣
 ↪R_high = {float(circuit.R_high.data):.0f}")
    return train_data, R_values, grad_values
```

[26]:
```
# Create a circuit, use loss_fn with learning rate of 1000
circuit = BandPassCircuit(900, 900)
# circuit = BandPassCircuit(500, 500)
lr = 1000
loss_fn = lambda x, y: (1-y) * torch.where(x-cutoff_mag>0,x-cutoff_mag, 0) + y␣
↪* torch.where(cutoff_mag-x>0,cutoff_mag-x, 0)# x:pred
train_data_band_bin, R_values_band_bin, grad_values_band_bin =␣
↪train_bp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps,␣
↪lr)
```

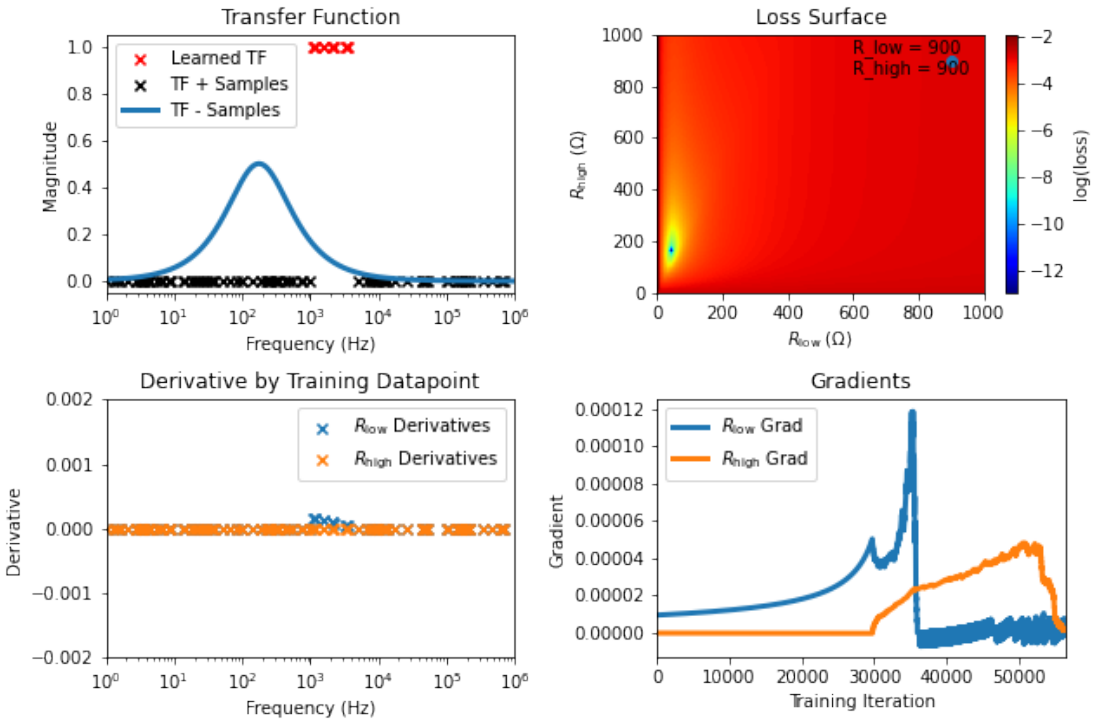
```
Initial Resistor Values: R_low = 900, R_high = 900

Training Iter:  56%|        | 56469/100000 [04:04<03:08, 231.28it/s, Loss:
0.000, R_low=40, R_high=161]

Final Resistor Values: R_low = 40, R_high = 161
```

[28]:
```
# Plot transfer function over training
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
subsample = int(dataset_size / 100)
train_data_mask = train_data_band_bin[1][::subsample] > cutoff_mag
ax1.scatter(train_data_band_bin[0][::subsample][train_data_mask] / (2 * math.
↪pi), np.ones(train_data_mask.sum()), c="r", marker="x")
ax1.scatter(train_data_band_bin[0][::subsample][~train_data_mask] / (2 * math.
↪pi), np.zeros((~train_data_mask).sum()), c="k", marker="x")
learned_tf, = ax1.semilogx(ws / (2 * math.pi), evaluate_bp_circuit(ws,␣
↪*R_values_band_bin[0]), linewidth=3)
ax1.set_xlim([1, 1e6])
ax1.set_title("Transfer Function")
ax1.set_xlabel("Frequency (Hz)")
ax1.set_ylabel("Magnitude")
ax1.legend(["Learned TF", "TF + Samples", "TF - Samples"])

# Show loss surfaces for BCE and MSE Loss
eval_pts = torch.stack(torch.meshgrid((torch.arange(0, 1000, 10), torch.
↪arange(0, 1000, 10)), indexing="ij"))
eval_vals = evaluate_bp_circuit(train_data_band_bin[0][:, None, None],␣
↪eval_pts[0][None, ...], eval_pts[1][None, ...])
loss_surface = loss_fn(eval_vals, (train_data_band_bin[1][..., None, None].
↪expand(eval_vals.shape) > cutoff_mag).float())
loss_surf = ax2.imshow(torch.log(loss_surface.mean(0)).T, cmap=plt.cm.jet,␣
↪extent=(0, 1000, 0, 1000), aspect="auto", origin="lower")
cur_loss, = ax2.plot(*R_values_band_bin[0], marker="o")
```

```
cur_loss_label = ax2.annotate(f"R_low = {R_values_band_bin[0][0]:.0f}\nR_high =␣
↪{R_values_band_bin[0][1]:.0f}", (0, 0), xytext=(0.6, 0.85), textcoords='axes␣
↪fraction')
ax2.set_title("Loss Surface")
ax2.set_xlabel("$R_\mathrm{low} \; (\Omega)$")
ax2.set_ylabel("$R_\mathrm{high} \; (\Omega)$")
fig.colorbar(loss_surf, ax=ax2, label="log(loss)")

# Show loss contributions of each data point
cur_circuit = BandPassCircuit(*R_values_band_bin[0])
data_losses = loss_fn(cur_circuit(train_data_band_bin[0][::subsample]),␣
↪(train_data_band_bin[1][::subsample] > cutoff_mag).float())
data_grads = torch.zeros((len(data_losses), 2))
for i, dl in enumerate(data_losses):
    data_grads[i] = torch.tensor(torch.autograd.grad(dl, (cur_circuit.R_low,␣
↪cur_circuit.R_high), retain_graph=True))
data_grads_scat1 = ax3.scatter(train_data_band_bin[0][::subsample] / (2 * math.
↪pi), data_grads[:, 0], marker="x")
data_grads_scat2 = ax3.scatter(train_data_band_bin[0][::subsample] / (2 * math.
↪pi), data_grads[:, 1], marker="x")
ax3.set_xscale("log")
ax3.set_ylabel("Derivative")
ax3.set_xlim([1, 1e6])
ax3.set_ylim([-2e-3, 2e-3])
ax3.set_xlabel("Frequency (Hz)")
ax3.set_title("Derivative by Training Datapoint")
ax3.legend(["$R_\mathrm{low}$ Derivatives", "$R_\mathrm{high}$ Derivatives"])

# Show gradient at each training iteration
ax4.plot(np.arange(len(grad_values_band_bin)), grad_values_band_bin,␣
↪linewidth=3)
cur_grad0, = ax4.plot(0, grad_values_band_bin[0][0], marker="o")
cur_grad1, = ax4.plot(0, grad_values_band_bin[0][1], marker="o")
ax4.set_xlabel("Training Iteration")
ax4.set_ylabel("Gradient")
ax4.set_title("Gradients")
ax4.set_xlim([-1, len(grad_values_band_bin)])
ax4.legend(["$R_\mathrm{low}$ Grad", "$R_\mathrm{high}$ Grad"])

plt.tight_layout()

# Main update function for interactive plots
def update_iter_band_bin(t=0):
    mags = evaluate_bp_circuit(ws, *R_values_band_bin[t])
    learned_tf.set_data(ws / (2 * math.pi), mags)
    cur_loss.set_data(*R_values_band_bin[t])
```

```
    cur_loss_label.set_text(f"R_low = {R_values_band_bin[t][0]:.0f}\nR_high =␣
↪{R_values_band_bin[t][1]:.0f}")
    cur_grad0.set_data(t, grad_values_band_bin[t][0])
    cur_grad1.set_data(t, grad_values_band_bin[t][1])
    cur_circuit = BandPassCircuit(*R_values_band_bin[t])
    data_losses = loss_fn(cur_circuit(train_data_band_bin[0][::subsample]),␣
↪(train_data_band_bin[1][::subsample] > cutoff_mag).float())
    data_grads = torch.zeros((len(data_losses), 2))
    for i, dl in enumerate(data_losses):
        data_grads[i] = torch.tensor(torch.autograd.grad(dl, (cur_circuit.
↪R_low, cur_circuit.R_high), retain_graph=True))
    data_grads_scat1.set_offsets(torch.stack((train_data_band_bin[0][::
↪subsample] / (2 * math.pi), data_grads[:, 0])).T)
    data_grads_scat2.set_offsets(torch.stack((train_data_band_bin[0][::
↪subsample] / (2 * math.pi), data_grads[:, 1])).T)
    fig.canvas.draw_idle()

# Include sliders for relevant quantities
ip = interactive(update_iter_band_bin,
                 t=widgets.IntSlider(value=0, min=0, max=len(R_values_band_bin)␣
↪- 1, step=1, description="Training Iteration", style={'description_width':␣
↪'initial'}, layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=0, description='Training Iteration',␣
↪layout=Layout(width='100%'), max=56…
```



0.8 (h) Learning a Band Pass Filter Bode Plot from Transfer Function Samples

[18]:
```
def evaluate_bp_bode(freqs, low_cutoff, high_cutoff):
    return -20 * nn.ReLU()(torch.log10(freqs / low_cutoff)) + -20 * nn.
 ↪ReLU()(torch.log10(high_cutoff / freqs))

# PyTorch model of the band pass bode plot
class BandPassBodePlot(nn.Module):
    def __init__(self, low_cutoff=None, high_cutoff=None):
        super().__init__()
        self.low_cutoff = nn.Parameter(torch.rand(1) * 5000 if low_cutoff is␣
 ↪None else torch.tensor(float(low_cutoff)))
        self.high_cutoff = nn.Parameter(torch.rand(1) * 5000 if high_cutoff is␣
 ↪None else torch.tensor(float(high_cutoff)))

    def forward(self, freqs):
        return evaluate_bp_bode(freqs, self.low_cutoff, self.high_cutoff)

# Train a given band pass bode plot
def train_bp_bode(bode, loss_fn, dataset_size, max_training_steps, lr):

    cutoff_values = [[float(bode.low_cutoff.data), float(bode.high_cutoff.
 ↪data)]]
    grad_values = [[np.nan, np.nan]]
    train_data = generate_bp_training_data(dataset_size)
    print(f"Initial Cutoff Values: f_c,l = {float(bode.low_cutoff.data / (2 *␣
 ↪math.pi)):.0f} Hz, f_c,h = {float(bode.high_cutoff.data / (2 * math.pi)):.
 ↪0f} Hz")
    iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
    for i in iter_bar:

        pred = bode(train_data[0])
        loss = loss_fn(pred, 20 * torch.log10(train_data[1])).mean()
        grad = torch.autograd.grad(loss, (bode.low_cutoff, bode.high_cutoff))
        with torch.no_grad():
            bode.low_cutoff -= lr * grad[0]
            bode.high_cutoff -= lr * grad[1]

        cutoff_values.append([float(bode.low_cutoff.data), float(bode.
 ↪high_cutoff.data)])
        grad_values.append([float(grad[0].data), float(grad[1].data)])
        iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f}, f_c,l =␣
 ↪{float(bode.low_cutoff.data / (2 * math.pi)):.0f} Hz, f_c,h = {float(bode.
 ↪high_cutoff.data / (2 * math.pi)):.0f} Hz")
```

```
        if loss.data < 1e-6 or (abs(grad[0].data) < 1e-6 and abs(grad[1].data)␣
↪< 1e-6):
            break

    print(f"Final Cutoff Values: f_c,l = {float(bode.low_cutoff.data / (2 *␣
↪math.pi)):.0f} Hz, f_c,h = {float(bode.high_cutoff.data / (2 * math.pi)):.
↪0f} Hz")
    return train_data, cutoff_values, grad_values
```

[19]:
```
bode = BandPassBodePlot()
loss_fn = lambda x, y: (x - y) ** 2    # MSE loss
lr = 1000
train_data_band_bode, cutoffs_band_bode, grad_values_band_bode =␣
↪train_bp_bode(bode, loss_fn, dataset_size, max_training_steps, lr)
```

```
Initial Cutoff Values: f_c,l = 681 Hz, f_c,h = 693 Hz
```

```
Training Iter:  61%|       | 61235/100000 [03:42<02:20, 275.43it/s, Loss:
0.905, f_c,l = 3864 Hz, f_c,h = 1029 Hz]
```

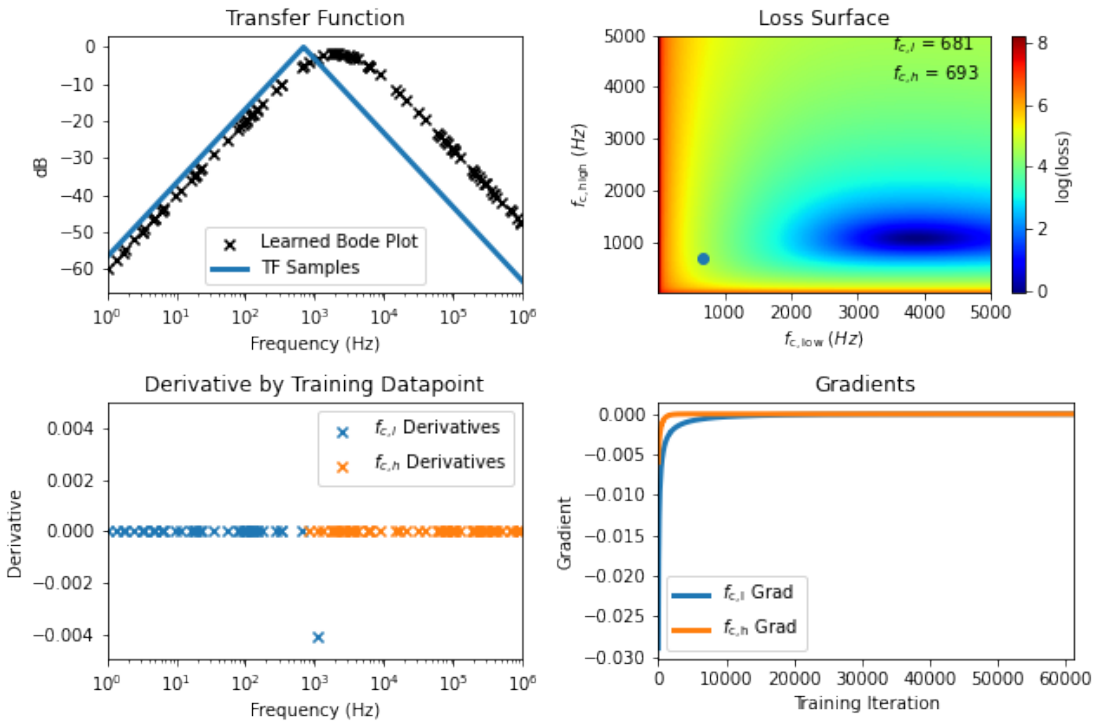
```
Final Cutoff Values: f_c,l = 3864 Hz, f_c,h = 1029 Hz
```

[21]:
```
# Plot transfer function over training
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
subsample = int(dataset_size / 100)
train_data_mask = train_data_band_bode[1][::subsample] > cutoff_mag
ax1.scatter(train_data_band_bode[0][::subsample]/ (2 * math.pi), 20 * torch.
↪log10(train_data_band_bode[1][::subsample]), c="k", marker="x")
learned_tf, = ax1.semilogx(ws / (2 * math.pi), evaluate_bp_bode(ws,␣
↪*cutoffs_band_bode[0]), linewidth=3)
ax1.set_xlim([1, 1e6])
ax1.set_title("Transfer Function")
ax1.set_xlabel("Frequency (Hz)")
ax1.set_ylabel("dB")
ax1.legend(["Learned Bode Plot", "TF Samples"])

# Show loss surfaces for BCE and MSE Loss
eval_pts = torch.stack(torch.meshgrid((torch.arange(1, 5001, 50), torch.
↪arange(1, 5001, 50)), indexing="ij"))
eval_vals = evaluate_bp_bode(train_data_band_bode[0][:, None, None], 2 * math.
↪pi * eval_pts[0][None, ...], 2 * math.pi * eval_pts[1][None, ...])
loss_surface = loss_fn(eval_vals, 20 * torch.log10(train_data_band_bode[1])[...
↪, None, None].expand(eval_vals.shape))
loss_surf = ax2.imshow(torch.log(loss_surface.mean(0)).T, cmap=plt.cm.jet,␣
↪extent=(1, 5000, 1, 5000), aspect="auto", origin="lower")
```

```
cur_loss, = ax2.plot(cutoffs_band_bode[0][0] / (2 * math.pi),␣
 ↪cutoffs_band_bode[0][1] / (2 * math.pi), marker="o")
cur_loss_label = ax2.annotate(f"$f_{{c,l}}$ = {cutoffs_band_bode[0][0]:.
 ↪0f}\n$f_{{c,h}}$ = {cutoffs_band_bode[0][1]:.0f}", (0, 0), xytext=(0.7, 0.
 ↪82), textcoords='axes fraction')
ax2.set_title("Loss Surface")
ax2.set_xlabel("$f_\mathrm{c,low} \; (Hz)$")
ax2.set_ylabel("$f_\mathrm{c,high} \; (Hz)$")
fig.colorbar(loss_surf, ax=ax2, label="log(loss)")

# Show loss contributions of each data point
cur_bode = BandPassBodePlot(*cutoffs_band_bode[0])
data_losses = loss_fn(cur_bode(train_data_band_bode[0][::subsample]), 20 *␣
 ↪torch.log10(train_data_band_bode[1][::subsample]))
data_grads = torch.zeros((len(data_losses), 2))
for i, dl in enumerate(data_losses):
    data_grads[i] = torch.tensor(torch.autograd.grad(dl, (cur_bode.low_cutoff,␣
 ↪cur_bode.high_cutoff), retain_graph=True))
data_grads_scat1 = ax3.scatter(train_data_band_bode[0][::subsample] / (2 * math.
 ↪pi), data_grads[:, 0], marker="x")
data_grads_scat2 = ax3.scatter(train_data_band_bode[0][::subsample] / (2 * math.
 ↪pi), data_grads[:, 1], marker="x")
ax3.set_xscale("log")
ax3.set_ylabel("Derivative")
ax3.set_xlim([1, 1e6])
ax3.set_ylim([-5e-3, 5e-3])
ax3.set_xlabel("Frequency (Hz)")
ax3.set_title("Derivative by Training Datapoint")
ax3.legend(["$f_{c,l}$ Derivatives", "$f_{c,h}$ Derivatives"])

# Show gradient at each training iteration
ax4.plot(np.arange(len(grad_values_band_bode)), grad_values_band_bode,␣
 ↪linewidth=3)
cur_grad0, = ax4.plot(0, grad_values_band_bode[0][0], marker="o")
cur_grad1, = ax4.plot(0, grad_values_band_bode[0][1], marker="o")
ax4.set_xlabel("Training Iteration")
ax4.set_ylabel("Gradient")
ax4.set_title("Gradients")
ax4.set_xlim([-1, len(grad_values_band_bode)])
ax4.legend(["$f_\mathrm{c,l}$ Grad", "$f_\mathrm{c,h}$ Grad"])

plt.tight_layout()

# Main update function for interactive plots
def update_iter_band_bode(t=0):
```

```
    learned_tf.set_data(ws / (2 * math.pi), evaluate_bp_bode(ws,␣
↪*cutoffs_band_bode[t]))
    cur_loss.set_data(cutoffs_band_bode[t][0] / (2 * math.pi),␣
↪cutoffs_band_bode[t][1] / (2 * math.pi))
    cur_loss_label.set_text(f"$f_{{c,l}}$ = {cutoffs_band_bode[t][0] / (2 *␣
↪math.pi):.0f}\n$f_{{c,h}}$ = {cutoffs_band_bode[t][1] / (2 * math.pi):.0f}")
    cur_grad0.set_data(t, grad_values_band_bode[t][0])
    cur_grad1.set_data(t, grad_values_band_bode[t][1])
    cur_bode = BandPassBodePlot(*cutoffs_band_bode[t])
    data_losses = loss_fn(cur_bode(train_data_band_bode[0][::subsample]), 20 *␣
↪torch.log10(train_data_band_bode[1][::subsample]))
    data_grads = torch.zeros((len(data_losses), 2))
    for i, dl in enumerate(data_losses):
        data_grads[i] = torch.tensor(torch.autograd.grad(dl, (cur_bode.
↪low_cutoff, cur_bode.high_cutoff), retain_graph=True))
    data_grads_scat1.set_offsets(torch.stack((train_data_band_bode[0][::
↪subsample] / (2 * math.pi), data_grads[:, 0])).T)
    data_grads_scat2.set_offsets(torch.stack((train_data_band_bode[0][::
↪subsample] / (2 * math.pi), data_grads[:, 1])).T)
    fig.canvas.draw_idle()

# Include sliders for relevant quantities
ip = interactive(update_iter_band_bode,
                 t=widgets.IntSlider(value=0, min=0, max=len(cutoffs_band_bode)␣
↪- 1, step=1, description="Training Iteration", style={'description_width':␣
↪'initial'}, layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=0, description='Training Iteration',␣
↪layout=Layout(width='100%'), max=61…
```



0.9 (i) Learn a Color Organ Circuit

[33]:
```
# PyTorch model of the color organ circuit
class ColorOrganCircuit(nn.Module):
    def __init__(self, R_low=None, R_high=None, R_band_low=None,␣
 ↪R_band_high=None):
        super().__init__()
        self.low = LowPassCircuit(R_low)
        self.high = HighPassCircuit(R_high)
        self.band = BandPassCircuit(R_band_low, R_band_high)

    def forward(self, freqs):
        return torch.stack((self.low(freqs), self.band(freqs), self.
 ↪high(freqs)))


# Generate training data in a uniform log scale of frequences, then evaluate␣
 ↪using the true transfer function
R_low_des = 1 / (2 * math.pi * 800 * cap_value)
R_band_low_des = 1 / (2 * math.pi * 4000 * cap_value)
R_band_high_des = 1 / (2 * math.pi * 1000 * cap_value)
R_high_des = 1 / (2 * math.pi * 5000 * cap_value)
def generate_co_training_data(n):
```

```
    rand_ws = 2 * math.pi * torch.pow(10, torch.rand(n) * 6)
    labels = torch.stack((evaluate_lp_circuit(rand_ws, R_low_des),␣
↪evaluate_bp_circuit(rand_ws, R_band_low_des, R_band_high_des),␣
↪evaluate_hp_circuit(rand_ws, R_high_des)))
    return rand_ws, labels

# Train a given color organ circuit
def train_co_circuit(circuit, loss_fn, dataset_size, max_training_steps, lr):

    R_values = [[float(circuit.low.R.data), float(circuit.band.R_low.data),␣
↪float(circuit.band.R_high.data), float(circuit.high.R.data)]]
    grad_values = [[np.nan, np.nan, np.nan, np.nan]]
    train_data = generate_co_training_data(dataset_size)
    print(f"Initial Resistor Values: LP: {float(circuit.low.R.data):.0f} Ohms,␣
↪BP (Low): {float(circuit.band.R_low.data):.0f} Ohms, BP (High):␣
↪{float(circuit.band.R_high.data):.0f} Ohms, HP: {float(circuit.high.R.data):.
↪0f} Ohms")

    iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
    for i in iter_bar:
        pred = circuit(train_data[0])
        loss = loss_fn(pred, (train_data[1] > cutoff_mag).float()).mean()
        grad = torch.autograd.grad(loss, (circuit.low.R, circuit.band.R_low,␣
↪circuit.band.R_high, circuit.high.R))
        with torch.no_grad():
            circuit.low.R -= lr * grad[0]
            circuit.band.R_low -= lr * grad[1]
            circuit.band.R_high -= lr * grad[2]
            circuit.high.R -= lr * grad[3]

        R_values.append([float(circuit.low.R.data), float(circuit.band.R_low.
↪data), float(circuit.band.R_high.data), float(circuit.high.R.data)])
        grad_values.append([float(grad[0].data), float(grad[1].data),␣
↪float(grad[2].data), float(grad[3].data)])
        iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f}, Rs =␣
↪{float(circuit.low.R.data):.0f}, {float(circuit.band.R_low.data):.0f},␣
↪{float(circuit.band.R_high.data):.0f}, {float(circuit.high.R.data):.0f}")
        if loss.data < 1e-6 or (abs(grad[0].data) < 1e-6 and abs(grad[1].data)␣
↪< 1e-6):
            break

    print(f"Final Resistor Values: LP: {float(circuit.low.R.data):.0f} Ohms, BP␣
↪(Low): {float(circuit.band.R_low.data):.0f} Ohms, BP (High): {float(circuit.
↪band.R_high.data):.0f} Ohms, HP: {float(circuit.high.R.data):.0f} Ohms")
```

```
    print(f"Final Cutoff Frequencies: LP: {1 / (2 * math.pi * cap_value *␣
↪float(circuit.low.R.data)):.0f} Hz, BP (Low): {1 / (2 * math.pi * cap_value␣
↪* float(circuit.band.R_low.data)):.0f} Hz, BP (High): {1 / (2 * math.pi *␣
↪cap_value * float(circuit.band.R_high.data)):.0f} Hz, HP: {1 / (2 * math.pi␣
↪* cap_value * float(circuit.high.R.data)):.0f} Hz")
    return train_data, R_values, grad_values
```

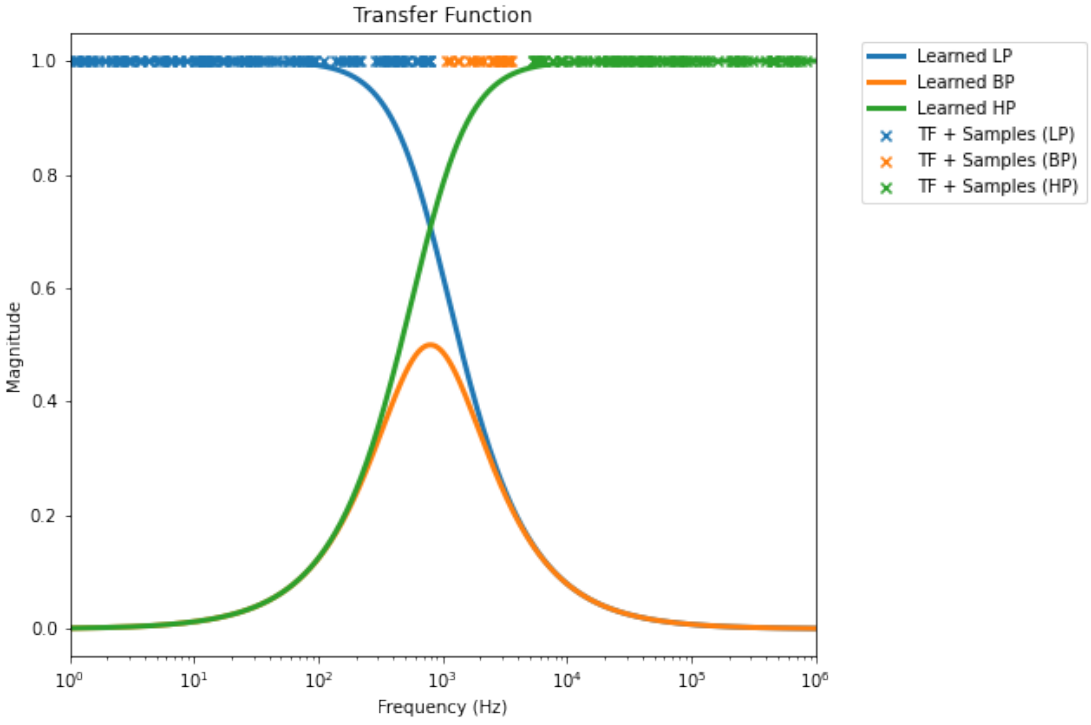
[34]:
```
co = ColorOrganCircuit(200, 200, 200, 200)
loss_fn = lambda x, y: (x - (0.3 + 0.7 * y)) ** 2   # weighted MSE loss
lr = 500
train_data_co, R_values_co, grad_values_co = train_co_circuit(co, loss_fn,␣
↪dataset_size, max_training_steps, lr)
```

```
Initial Resistor Values: LP: 200 Ohms, BP (Low): 200 Ohms, BP (High): 200 Ohms,
HP: 200 Ohms

Training Iter:   9%|          | 9207/100000 [00:43<07:13, 209.52it/s, Loss:
0.047, Rs = 189, 39, 178, 31]

Final Resistor Values: LP: 189 Ohms, BP (Low): 39 Ohms, BP (High): 178 Ohms, HP:
31 Ohms
Final Cutoff Frequencies: LP: 841 Hz, BP (Low): 4130 Hz, BP (High): 894 Hz, HP:
5213 Hz
```

[35]:
```
# Plot transfer function over training
fig, ax1 = plt.subplots(1, 1, figsize=(9, 6))
ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
subsample = int(dataset_size / 250)
train_data_mask = train_data_co[1][:, ::subsample] > cutoff_mag
learned_tf1, = ax1.semilogx(ws / (2 * math.pi), evaluate_lp_circuit(ws,␣
↪R_values_co[0][0]), linewidth=3)
learned_tf2, = ax1.semilogx(ws / (2 * math.pi), evaluate_bp_circuit(ws,␣
↪*R_values_co[0][1:3]), linewidth=3)
learned_tf3, = ax1.semilogx(ws / (2 * math.pi), evaluate_hp_circuit(ws,␣
↪R_values_co[0][-1]), linewidth=3)
ax1.scatter(train_data_co[0][::subsample][train_data_mask[0]] / (2 * math.pi),␣
↪np.ones(train_data_mask[0].sum()), c=learned_tf1.get_color(), marker="x")
ax1.scatter(train_data_co[0][::subsample][train_data_mask[1]] / (2 * math.pi),␣
↪np.ones(train_data_mask[1].sum()), c=learned_tf2.get_color(), marker="x")
ax1.scatter(train_data_co[0][::subsample][train_data_mask[2]] / (2 * math.pi),␣
↪np.ones(train_data_mask[2].sum()), c=learned_tf3.get_color(), marker="x")
# ax1.scatter(train_data_co[0][::subsample][(~train_data_mask).all(0)] / (2 *␣
↪math.pi), np.zeros((~(train_data_mask.any(0))).sum()), c="k", marker="x")
ax1.set_xlim([1, 1e6])
ax1.set_title("Transfer Function")
ax1.set_xlabel("Frequency (Hz)")
```

```
ax1.set_ylabel("Magnitude")
ax1.legend(["Learned LP", "Learned BP", "Learned HP",
            "TF + Samples (LP)", "TF + Samples (BP)", "TF + Samples (HP)",
            "TF - Samples"], bbox_to_anchor=(1.05, 1), loc='upper left', ncol=1)

plt.tight_layout()

# Main update function for interactive plots
def update_iter_co(t=0):
    learned_tf1.set_data(ws / (2 * math.pi), evaluate_lp_circuit(ws,␣
 ↪R_values_co[t][0]))
    learned_tf2.set_data(ws / (2 * math.pi), evaluate_bp_circuit(ws,␣
 ↪*R_values_co[t][1:3]))
    learned_tf3.set_data(ws / (2 * math.pi), evaluate_hp_circuit(ws,␣
 ↪R_values_co[t][-1]))
    fig.canvas.draw_idle()

# Include sliders for relevant quantities
ip = interactive(update_iter_co,
                 t=widgets.IntSlider(value=0, min=0, max=len(R_values_co) - 1,␣
 ↪step=1, description="Training Iteration", style={'description_width':␣
 ↪'initial'}, layout=Layout(width='100%')))
ip
```

```
interactive(children=(IntSlider(value=0, description='Training Iteration',␣
 ↪layout=Layout(width='100%'), max=92…
```



0.10 Visualizing the computation graph for the Color Organ

[36]:
```
from torchviz import make_dot
make_dot(co(generate_co_training_data(dataset_size)[0]), params=dict(co.
 ↪named_parameters()))
```

[36]: