



Deep Neural Networks(EECS182)
Homework0

1. Surveys

I have submitted the surveys, my SID is 3038745426.

2. Course Policies

(a) Yes. (b) No. (c) No. (d) Yes. (e) Yes.

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

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

$$\begin{aligned}w_t &= w_{t-1} - \eta(F^T(Fw_{t-1} - y)) \\&= (I - \eta F^T F)w_{t-1} + \eta F^T y\end{aligned}$$

When the gradient descent cannot diverge, the eigenvalues of $(I - \eta F^T F)$ are smaller than 1.

Combining with the Absolute value inequalities, we can derive that

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

And since when $n = d$, the singular value of feature matrix $F \in \mathbb{R}^{n \times n}$ is not greater than α , $\|\eta F^T\| \leq \eta\alpha$

Since $I - \eta F^T F$ is a square matrix, it can be decomposed to $U\Sigma V^*$, where U and V^* are both orthogonal matrices. So that $\|I - \eta F^T F w_{t-1}\|_2 = \|U\Sigma V^* x w_{t-1}\| = \|\Sigma w_{t-1}\| \leq \|w_{t-1}\|$.

Therefore,

$$\|w_t\|_2 \leq \|w_{t-1}\|_2 + \eta\alpha \|y\|_2$$

4. Regularization from the Augmentation Perspective

We can derive that,

$$\begin{aligned}\hat{X} &= \begin{bmatrix} x_1^T \\ x_2^T \\ \dots \\ x_n^T \\ \gamma_1^T \\ \gamma_2^T \\ \dots \\ \gamma_d^T \end{bmatrix} \in \mathbb{R}^{(n+d) \times d}, \hat{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} \in \mathbb{R}^{n+d} \\ \hat{X}^T y &= \begin{bmatrix} X^T & \Gamma^T \end{bmatrix} \begin{bmatrix} X \\ 0_d \end{bmatrix} = X^T X \\ \hat{X}^T \hat{X} &= \begin{bmatrix} X^T & \Gamma^T \end{bmatrix} \begin{bmatrix} X \\ \Gamma \end{bmatrix}\end{aligned}$$

Since the X and Γ are both square matrix, the result of $\begin{bmatrix} X^T & \Gamma^T \end{bmatrix} \begin{bmatrix} X \\ \Gamma \end{bmatrix}$ is $X^T X + \Gamma^T \Gamma = X^T X + \Sigma^{-1}$

To find the \hat{w} to minimize the $\|\hat{y} - \hat{X}w\|_2^2$, it is known from the OLS solution that the following formula holds

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

which is the same as (2)

5. Vector Calculus Review

According to the fully differential equations, we know that for a scalar f and a $m * n$ matrix X , and since in the question, the vector derivatives of a scalar are expressed as a row vector, we have $df = \sum_{i=1}^m \sum_{j=1}^n \frac{\partial f}{\partial X_{ij}} dX_{ij} = tr(\frac{\partial f}{\partial x}) dX$.

(a)

Let $f = x^T c$, so that

$$df = d(x^T c) = dx^T c = tr(dx^T c) = tr(c^T dx^T)$$

Therefore, $\frac{\partial f}{\partial x} = c^T$

(b)

Let $f = \|x\|_2^2 = x^T x$, $df = d(x^T x) = dx^T x + x^T dx = tr(dx^T x + x^T dx) = tr(dx^T x) + tr(x^T dx) = tr(x^T dx) + tr(x^T dx) = tr(2x^T dx)$, therefore, $\frac{\partial f}{\partial x} = 2x^T$

(c)

Let $f = Ax$, $df = d(Ax) = Adx = tr(Adx)$, therefore, $\frac{\partial f}{\partial x} = A$

(d)

Let $f = x^T Ax$, $df = dx^T Ax + x^T Adx = tr(dx^T Ax + x^T Adx) = tr(dx^T Ax) + tr(x^T Adx) = tr((Ax)^T dx) + tr(x^T Adx) = tr(x^T (A + A^T) dx)$, therefore, $\frac{\partial f}{\partial x} = x^T (A + A^T)$

(e)

When $A = A^T$, the previous derivative equal to $2x^T A$

6. ReLU Elbow Update under SGD

(a)

(i)

The location of elbow is the point that make $w x + b < 0$ change to $w x + b > 0$, which is $-\frac{b}{w}$

(ii)

$$\begin{aligned} l &= \frac{1}{2} (\phi(x) - y)^T (\phi(x) - y) \\ dl &= \frac{1}{2} [d(\phi(x) - y)^T (\phi(x) - y) + (\phi(x) - y)^T d(\phi(x) - y)] \\ &= \frac{1}{2} [tr(d(\phi(x) - y)^T (\phi(x) - y)) + tr(\phi(x) - y)^T d(\phi(x))] \\ &= \frac{1}{2} [tr((\phi(x) - y)^T d(\phi(x) - y)) + tr(\phi(x) - y)^T d(\phi(x) - y)] \\ &= (\phi(x) - y)^T d(\phi(x) - y) \end{aligned}$$

so that

$$\frac{dl}{d\phi} = \begin{cases} (\phi(x) - y)^T & wx + b > 0 \\ 0 & else \end{cases}$$

(iii)

From (ii), we know that

$$dl = (\phi(x) - y)^T d\phi(x) = (\phi(x) - y)^T d(wx + b) = (\phi(x) - y)^T x dw$$

Therefore

$$\frac{\partial l}{\partial w} = \begin{cases} x^T (\phi(x) - y) & wx + b > 0 \\ 0 & else \end{cases}$$

(iv)

From (ii), we know that

$$dl = (\phi(x) - y)^T d\phi(x) = (\phi(x) - y)^T d(wx + b) = (\phi(x) - y)^T db$$

Therefore

$$\frac{\partial l}{\partial b} = \begin{cases} \phi(x) - y & wx + b > 0 \\ 0 & else \end{cases}$$

(b)

The gradient descent update formula of w and b is as below.

$$w_{t+1} \leftarrow w_t - \lambda \frac{\partial l}{\partial w}$$

$$b_{t+1} \leftarrow b_t - \lambda \frac{\partial l}{\partial b}$$

where the λ is the step size.

(i)

When $\phi(x) = 0$, $\frac{\partial l}{\partial w} = 0$, $\frac{\partial l}{\partial b} = 0$, $w_{t+1} \leftarrow w_t$, $b_{t+1} \leftarrow b_t$. The elbow and the slope will not change. The image is shown as figure1.

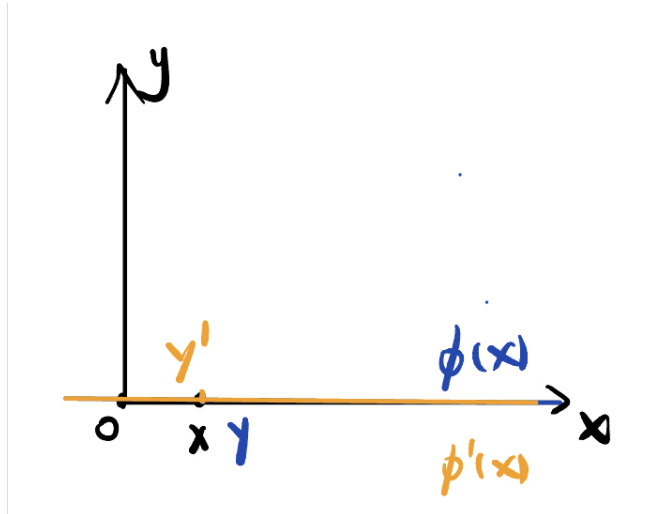


Figure 1: The elbow and slope will not change during updating.

(ii)

When $w > 0, x > 0, \phi(x) > 0$, $\frac{\partial l}{\partial w} = x^T(\phi(x) - y)$, $\frac{\partial l}{\partial b} = \phi(x) - y$, $w_{t+1} \leftarrow w_t - \lambda x^T(\phi(x) - y) = w_t - \lambda x^T$, $b_{t+1} \leftarrow b_t - \lambda(\phi(x) - y) = b_t - \lambda$.

The slope will decrease, and the b will decrease, too. In figure 2, we can see that the elbow moves left during update. The corresponding y decreases.

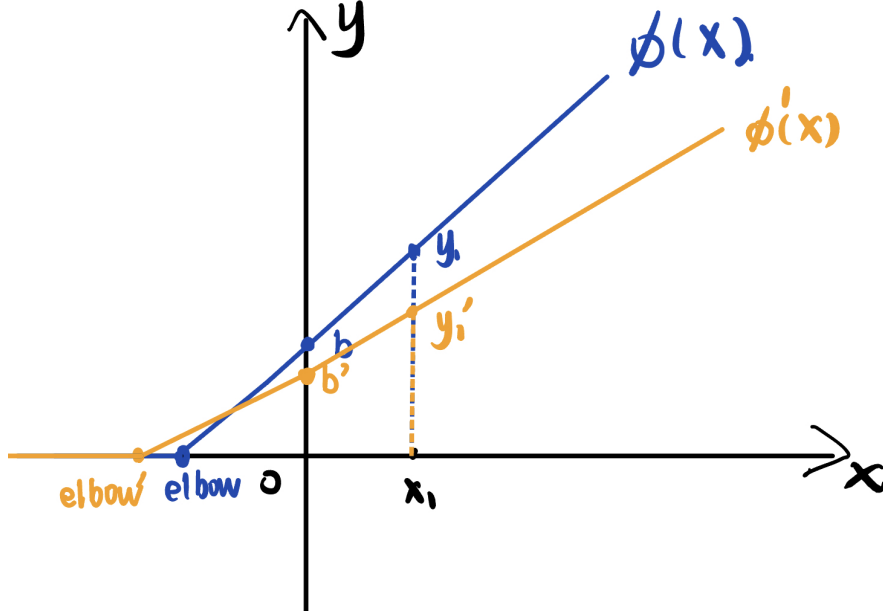


Figure 2: The elbow will move left and slope will decrease during updating.

(iii)

When $w > 0, x < 0, \phi(x) > 0$, $\frac{\partial l}{\partial w} = x^T(\phi(x) - y)$, $\frac{\partial l}{\partial b} = \phi(x) - y$, $w_{t+1} \leftarrow w_t - \lambda x^T(\phi(x) - y) = w_t - \lambda x^T$, $b_{t+1} \leftarrow b_t - \lambda(\phi(x) - y) = b_t - \lambda$.

The slope will increase while the b will decrease. In figure 3, we can see that the elbow moves right during update. The corresponding y decreases.

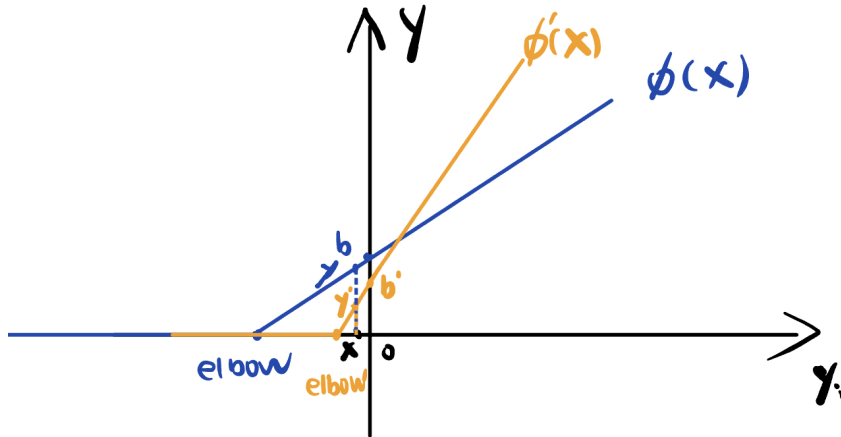


Figure 3: The elbow will move right and slope will decrease during updating.

(iv)

When $w < 0, x > 0, \phi(x) > 0$, $\frac{\partial l}{\partial w} = x^T(\phi(x) - y)$, $\frac{\partial l}{\partial b} = \phi(x) - y$, $w_{t+1} \leftarrow w_t - \lambda x^T(\phi(x) - y) = w_t - \lambda x^T$, $b_{t+1} \leftarrow b_t - \lambda(\phi(x) - y) = b_t - \lambda$.

The slope the b will both decrease. In figure 4, we can see that the elbow moves left during update. The corresponding y decreases.

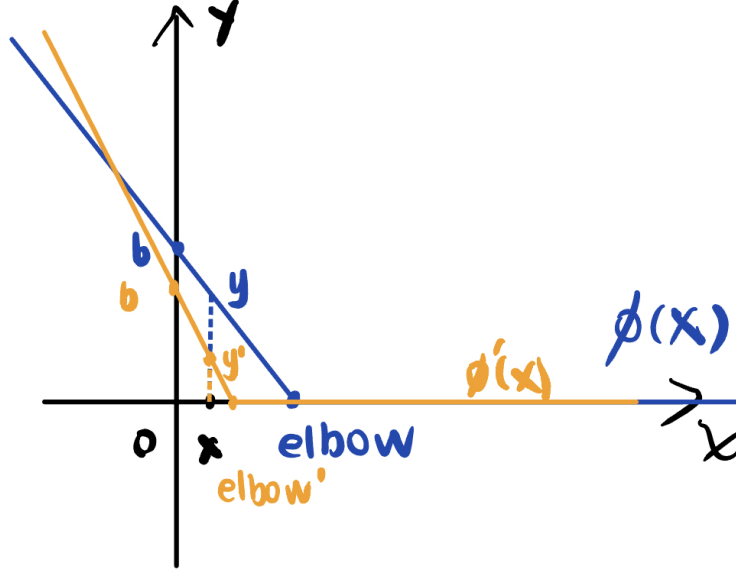


Figure 4: The elbow will move left and slope will decrease during updating.

(c)

$$e_i = -\frac{b^i}{w_i^{(1)}}$$

(d) When $W^{(1)}x + b \leq 0$, elbow will not change, $e_i^{(')} = e_i = -\frac{b^i}{w_i^{(1)}}$, otherwise, $\frac{\partial(W^{(1)}x+b)}{\partial W_i^1} = x$, $\frac{\partial(W^{(1)}x+b)}{\partial b_i} = 1$, after one gradient update, $w_i^{(i)'} = w_i^{(1)} - \lambda x$, $b^{i'} = b^i - \lambda$. Therefore, the new elbow $e_i' = \frac{w_i^{(1)} - \lambda x}{b^i - \lambda}$

7. Using PyTorch to Learn the Color Organ

(a) The resistor value is 200 such that the predicted and desired transfer functions match.

(b) The resistor value is 200 and the corresponding cutoff frequency is 829Hz.

(c) Yes, we can learn the resistor value by means of neural network.

The circuit take 4 minutes and 28 seconds to converge, and the final value of R is 200, which is the same as the value I found in the previous part.

When the value of lr is 20000000, it cause the training to diverge.

When the value of lr is 200000, it converged in a flash.

(d) The learned resistor value is 320.

(e) I used the cross entropy to be the loss function, which is $loss_fn = \lambda x, y : (torch.exp(x) - torch.exp(1.1y)) ** 2$, and the predicted value is 243, which is close to the real value.

(f) The learned resistor value is 30.

(g)

(h) Yes, it does. Yes.

(i) Under the same learning rate, the larger the initial resistor's value is, the longer training time it takes.

8. Homework Process and Study Group

(a) [Pytorch Tutorial](#)

(b)

Name: Chuan Chen

SID: 3038743333

(c) Approximately 15 hours.

9. Code Appendix

```
1 # -*- coding: utf-8 -*-
2 """Color_organ_learning.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1e4EIkoy_qX0kOBodgkgUZS-3wuSYhYHS
8 """
9
10 !pip install ipympl torchviz
11 !pip install torch==1.13 --extra-index-url https://download.pytorch.org/whl/cpu
12 # restart your runtime after this step
13
14 import math
15 import matplotlib.pyplot as plt
16 import numpy as np
17 import torch
18 import torch.nn as nn
19 from torch.autograd import Variable
20 import tqdm
21
22 import IPython
23 from ipywidgets import interactive, widgets, Layout
24 from IPython.display import display, HTML
25 # import os
26 # os.kill(os.getpid(), 9)
27
28 print(torch.__version__, torch.cuda.is_available())
29 # Homework 0 does not require a GPU
30
31 # Commented out IPython magic to ensure Python compatibility.
32 # enable matplotlib widgets;
33
34 # on Google Colab
35 from google.colab import output
36 output.enable_custom_widget_manager()
37
38 # %matplotlib widget
39
40 # Constants
41 cap_value = 1e-6          # Farads
42 R_init = 500              # Ohms
43 cutoff_mag = 1. / math.sqrt(2)
44 cutoff_dB = 20 * math.log10(cutoff_mag)
45 dataset_size = 1000
46 max_training_steps = 100000
47
48 """## (a) Designing a Low Pass Filter by Matching Transfer Functions"""
49
50 # Transfer function: evaluates magnitude of given frequencies for a resistor value
51                        # in the low pass circuit
52
53 def evaluate_lp_circuit(freqs, R_low):
54     return 1. / torch.sqrt(1 + (R_low * cap_value * freqs) ** 2)
55
56 # Plot transfer function for a given low pass circuit
57 fig = plt.figure(figsize=(9, 4))
58 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
59 mags = 20 * torch.log10(evaluate_lp_circuit(ws, R_init))
60 R_low_des = 1 / (2 * math.pi * 800 * cap_value)
61 mags_des = 20 * torch.log10(evaluate_lp_circuit(ws, R_low_des))
```

```

60 tf, = plt.semilogx(ws / (2 * math.pi), mags, linewidth=3)
61 tf_des, = plt.semilogx(ws / (2 * math.pi), mags_des, linestyle="--", linewidth=3)
62 plt.xlim([1, 1e6])
63 plt.ylim([-60, 1])
64 plt.title("Low Pass Transfer Functions")
65 plt.xlabel("Frequency (Hz)")
66 plt.ylabel("dB")
67 plt.grid(which="both")
68 leg = plt.legend(["Predicted Transfer Function", "Desired Transfer Function"])
69 plt.tight_layout()
70
71 # Main update function for interactive plot
72 def update_tfs(R=R_init):
73     mags = 20 * torch.log10(evaluate_lp_circuit(ws, R))
74     tf.set_data(ws / (2 * math.pi), mags)
75     fig.canvas.draw_idle()
76
77 # Include sliders for relevant quantities
78 ip = interactive(update_tfs,
79                  R=widgets.IntSlider(value=R_init, min=1, max=1000, step=1,
                                      description="R", layout=Layout(width='
100%'))))
80 ip
81
82 """## (b) Designing a Low pass Filter from Binary Data"""
83
84 # Plot transfer function for a given low pass circuit
85 fig = plt.figure(figsize=(9, 5))
86 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
87 mags = 20 * torch.log10(evaluate_lp_circuit(ws, R_init))
88 cutoff = ws[np.argmax(mags < cutoff_dB)]
89 tf, = plt.semilogx(ws / (2 * math.pi), mags, linewidth=3)
90 cut = plt.axvline(cutoff / (2 * math.pi), c="red", linestyle="--", linewidth=3)
91 plt.xlim([1, 1e6])
92 plt.ylim([-60, 1])
93 plt.title("Low Pass Transfer Function")
94 plt.xlabel("Frequency (Hz)")
95 plt.ylabel("dB")
96 plt.grid(which="both")
97 leg = plt.legend(["Transfer Function", f"Cutoff Frequency ({1 / (2 * math.pi *
R_init * cap_value):.0f} Hz)"])
98
99 # Plot table of LED on/off values (predicted and desired)
100 ws_test = 2 * math.pi * np.linspace(300, 1500, num=7)
101 table_txt = np.zeros((3, len(ws_test) + 1), dtype="U15")
102 table_txt[0, :] = ["Frequency"] + [f"{w / (2 * math.pi):.0f} Hz" for w in ws_test]
103 table_txt[1:, 0] = ["Predicted", "Desired"]
104 table_colors = np.zeros_like(table_txt, dtype=(np.int32, (3,)))
105 table_colors[-1, 1:4] = (1, 0, 0)
106 table_colors[1, 1] = (1, 0, 0)
107 table_colors[:, :1] = (1, 1, 1)
108 table_colors[:1, :] = (1, 1, 1)
109 tab = plt.table(table_txt, table_colors, bbox=[0.0, -0.5, 1.0, 0.25], cellLoc="
center")
110 plt.tight_layout()
111
112 # Main update function for interactive plot
113 def update_lights(R=R_init):
114     mags = 20 * torch.log10(evaluate_lp_circuit(ws, R))
115     cutoff = ws[np.argmax(mags < cutoff_dB)]
116     tf.set_data(ws / (2 * math.pi), mags)

```



```

117     cut.set_xdata(cutoff / (2 * math.pi))
118     for i, w in enumerate(ws_test):
119         if w < cutoff:
120             tab[(1, i+1)].set_facecolor((1, 0, 0))
121         else:
122             tab[(1, i+1)].set_facecolor((0, 0, 0))
123     leg.get_texts()[1].set_text(f"Cutoff Frequency ({1 / (2 * math.pi * R *
124                                     cap_value):.0f} Hz)")
125
126     fig.canvas.draw_idle()
127
128 # Include sliders for relevant quantities
129 ip = interactive(updateLights,
130                  R=widgets.IntSlider(value=R_init, min=1, max=1000, step=1,
131                                     description="R", layout=Layout(width='
132                                     100%'))))
133
134 """## (c) Learning a Low Pass Filter from Desired Transfer Function Samples"""
135
136 # PyTorch model of the low pass circuit (for training)
137 class LowPassCircuit(nn.Module):
138     def __init__(self, R=None):
139         super().__init__()
140         self.R = nn.Parameter(torch.tensor(R, dtype=float) if R is not None else
141                                torch.rand(1) * 1000)
142
143     # Note: the forward function is called automatically when the __call__ function
144     # of this object is called
145     def forward(self, freqs):
146         return evaluate_lp_circuit(freqs, self.R)
147
148 # Generate training data in a uniform log scale of frequencies, then evaluate using
149 # the true transfer function
150 def generate_lp_training_data(n):
151     rand_ws = 2 * math.pi * torch.pow(10, torch.rand(n) * 6)
152     labels = evaluate_lp_circuit(rand_ws, R_low_des)
153     return rand_ws, labels
154
155 # Train a given low pass filter
156 def train_lp_circuit_tf(circuit, loss_fn, dataset_size, max_training_steps, lr):
157
158     R_values = [float(circuit.R.data)]
159     grad_values = [np.nan]
160     train_data = generate_lp_training_data(dataset_size)
161     print(f"Initial Resistor Value: R = {float(circuit.R.data):.0f}")
162     iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
163     for i in iter_bar:
164         pred = circuit(train_data[0])
165         loss = loss_fn(pred, train_data[1]).mean()
166         grad = torch.autograd.grad(loss, circuit.R)
167         with torch.no_grad():
168             circuit.R -= lr * grad[0]
169
170     R_values.append(float(circuit.R.data))
171     grad_values.append(float(grad[0].data))
172     iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f}, R={float(circuit.R
173                                     .data):.0f}")
174
175     if loss.data < 1e-6 or abs(grad[0].data) < 1e-6:
176         break
177
178     print(f"Final Resistor Value: R = {float(circuit.R.data):.0f}")

```

```

171     return train_data, R_values, grad_values
172
173 # Create a circuit, use mean squared error loss w/ learning rate of 200
174 circuit = LowPassCircuit(1000)
175 loss_fn = lambda x, y: (x - y) ** 2
176 lr = 20000
177 train_data_low_tf, R_values_low_tf, grad_values_low_tf = train_lp_circuit_tf(
178     circuit, loss_fn, dataset_size,
179     max_training_steps, lr)
180
181 # Plot transfer function over training
182 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
183 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
184 subsample = int(dataset_size / 100)
185 ax1.scatter(train_data_low_tf[0][:subsample] / (2 * math.pi), 20 * torch.log10(
186     train_data_low_tf[1][:subsample]), c="
187     k", marker="x")
188
189 learned_tf, = ax1.semilogx(ws / (2 * math.pi), 20 * torch.log10(evaluate_lp_circuit
190     (ws, R_values_low_tf[0])), linewidth=3)
191
192 ax1.set_xlim([1, 1e6])
193 ax1.set_title("Transfer Function")
194 ax1.set_xlabel("Frequency (Hz)")
195 ax1.set_ylabel("dB")
196 ax1.legend(["Learned Transfer Function", "True Transfer Function Samples"])
197
198 # Show loss surface over training
199 eval_pts = torch.arange(10, 1001, 1)
200 eval_vals = evaluate_lp_circuit(train_data_low_tf[0][:, None], eval_pts[None, :])
201 loss_surface_mse = loss_fn(eval_vals, train_data_low_tf[1][:, None].expand(
202     eval_vals.shape))
203
204 ax2.plot(eval_pts, loss_surface_mse.sum(0), linewidth=3)
205 cur_loss, = ax2.plot(R_values_low_tf[0], loss_surface_mse[:, int(R_values_low_tf[0]
206     - 10)].sum(0), marker="o")
207
208 cur_loss_label = ax2.annotate(f"R = {R_values_low_tf[0]:.0f}", (0, 0), xytext=(0.82
209     , 0.9), textcoords='axes fraction')
210
211 ax2.set_title("Loss Surface")
212 ax2.set_xlim([0, 1000])
213 ax2.set_xlabel("$R \ ; \ (\Omega)$")
214 ax2.set_ylabel("Loss")
215
216 # Show loss contributions of each data point
217 cur_circuit = LowPassCircuit(R_values_low_tf[0])
218 data_losses = loss_fn(cur_circuit(train_data_low_tf[0][:subsample]), (
219     train_data_low_tf[1][:subsample]).
220     float())
221
222 data_grads = torch.zeros(len(data_losses))
223 for i, dl in enumerate(data_losses):
224     data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
225 data_grads_scatter = ax3.scatter(train_data_low_tf[0][:subsample] / (2 * math.pi),
226     data_grads, marker="x", c="k")
227
228 ax3.set_xscale("log")
229 ax3.set_ylabel("Derivative")
230 ax3.set_xlim([1, 1e6])
231 ax3.set_ylim([-1e-4, 1e-3])
232 ax3.set_xlabel("Frequency (Hz)")
233 ax3.set_title("Derivative by Training Datapoint")
234
235 # Show total gradient at each training iteration
236 ax4.plot(np.arange(len(grad_values_low_tf)), grad_values_low_tf, linewidth=3)
237 cur_iter, = ax4.plot(0, grad_values_low_tf[0], marker="o")
238 cur_grad_label = ax4.annotate(f"Grad = {grad_values_low_tf[0]:.2e}", (0, 0), xytext

```

```

                                                    =(0.65, 0.9), textcoords='axes fraction
                                                    ')
221 ax4.set_xlabel("Training Iteration")
222 ax4.set_ylabel("Gradient")
223 ax4.set_title("Gradients")
224 ax4.set_xlim([-1, len(grad_values_low_tf)])
225
226 plt.tight_layout()
227
228 # Main update function for interactive plots
229 def update_iter_tf(t=0):
230     learned_tf.set_data(ws / (2 * math.pi), 20 * torch.log10(evaluate_lp_circuit(ws
231                                     , R_values_low_tf[t])))
232     cur_loss.set_data(R_values_low_tf[t], loss_surface_mse[:, int(R_values_low_tf[t]
233                                     - 10)].sum(0))
234     cur_loss_label.set_text(f"R = {R_values_low_tf[t]:.0f}")
235     cur_iter.set_data(t, grad_values_low_tf[t])
236     cur_grad_label.set_text(f"Grad = {grad_values_low_tf[t]:.2e}")
237     cur_circuit = LowPassCircuit(R_values_low_tf[t])
238     data_losses = loss_fn(cur_circuit(train_data_low_tf[0][::subsample]), (
239                                     train_data_low_tf[1][::subsample])).
240                                     float()
241     data_grads = torch.zeros(len(data_losses))
242     for i, dl in enumerate(data_losses):
243         data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
244     data_grads_scatter.set_offsets(torch.stack((train_data_low_tf[0][::subsample] / (2
245                                     * math.pi), data_grads)).T)
246     fig.canvas.draw_idle()
247
248 # Include sliders for relevant quantities
249 ip = interactive(update_iter_tf,
250                 t=widgets.IntSlider(value=0, min=0, max=len(R_values_low_tf) - 1,
251                                     step=1, description="Training Iteration
252                                     ", style={'description_width': 'initial
253                                     '}, layout=Layout(width='100%'))))
254
255 ip
256
257 """## (d) Learning a Low Pass Filter from Binary Data with Mean Squared Error Loss
258 """
259
260 # Train a given low pass filter from binary data
261 def train_lp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps, lr)
262 :
263
264     R_values = [float(circuit.R.data)]
265     grad_values = [np.nan]
266     train_data = generate_lp_training_data(dataset_size)
267     print(f"Initial Resistor Value: R = {float(circuit.R.data):.0f}")
268     iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
269     for i in iter_bar:
270         pred = circuit(train_data[0])
271         ### YOUR CODE HERE
272
273         loss = loss_fn(pred, (train_data[1] > cutoff_mag).float()).mean()
274         ### END YOUR CODE
275         grad = torch.autograd.grad(loss, circuit.R)
276         with torch.no_grad():
277             circuit.R -= lr * grad[0]
278
279     R_values.append(float(circuit.R.data))

```

```

269         grad_values.append(float(grad[0].data))
270         iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f}, R={float(circuit.R
271                                     .data):.0f}")
272         if loss.data < 1e-6 or abs(grad[0].data) < 1e-6:
273             break
274
275         print(f"Final Resistor Value: R = {float(circuit.R.data):.0f}")
276         return train_data, R_values, grad_values
277
278 # Create a circuit, use MSE loss with learning rate of 200
279 circuit = LowPassCircuit(500)
280 loss_fn = lambda x, y: (x - y) ** 2
281 lr = 20000
282 train_data_low_bin, R_values_low_bin, grad_values_low_bin = train_lp_circuit_binary
283                                     (circuit, loss_fn, dataset_size,
284                                     max_training_steps, lr)
285
286 # Plot transfer function over training
287 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
288 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
289 subsample = int(dataset_size / 100)
290 train_data_mask = train_data_low_bin[1][:subsample] > cutoff_mag
291 ax1.scatter(train_data_low_bin[0][:subsample][train_data_mask] / (2 * math.pi), np
292                                     .ones(train_data_mask.sum()), c="r",
293                                     marker="x")
294 ax1.scatter(train_data_low_bin[0][:subsample][~train_data_mask] / (2 * math.pi),
295                                     np.zeros((~train_data_mask).sum()), c="
296                                     k", marker="x")
297
298 mags = evaluate_lp_circuit(ws, R_values_low_bin[0])
299 learned_tf, = ax1.semilogx(ws / (2 * math.pi), mags, linewidth=3)
300 cutoff = ws[np.argmax(mags < cutoff_mag)]
301 cut = ax1.axvline(cutoff / (2 * math.pi), c="red", linestyle="--", linewidth=3)
302 ax1.set_xlim([1, 1e6])
303 ax1.set_title("Transfer Function")
304 ax1.set_xlabel("Frequency (Hz)")
305 ax1.set_ylabel("Magnitude")
306 ax1.legend(["Learned TF", "Learned $f_c$", "TF + Samples", "TF - Samples"])
307
308 # Show loss surface over training
309 eval_pts = torch.arange(10, 1001, 1)
310 eval_vals = evaluate_lp_circuit(train_data_low_bin[0][:, None], eval_pts[None, :])
311 loss_surface_mse = loss_fn(eval_vals, (train_data_low_bin[1][:, None].expand(
312                                     eval_vals.shape) > cutoff_mag).float())
313
314 ax2.plot(eval_pts, loss_surface_mse.sum(0), linewidth=3)
315 cur_loss, = ax2.plot(R_values_low_bin[0], loss_surface_mse[:, int(R_values_low_bin[
316                                     0] - 10)].sum(0), marker="o")
317 cur_loss_label = ax2.annotate(f"R = {R_values_low_bin[0]:.0f}", (0, 0), xytext=(0.
318                                     82, 0.9), textcoords='axes fraction')
319
320 ax2.set_title("Loss Surface")
321 ax2.set_xlim([0, 1000])
322 ax2.set_xlabel("$R \backslash ; (\backslash \Omega)$")
323 ax2.set_ylabel("Loss")
324
325 # Show loss contributions of each data point
326 cur_circuit = LowPassCircuit(R_values_low_bin[0])
327 data_losses = loss_fn(cur_circuit(train_data_low_bin[0][:subsample]), (
328                                     train_data_low_bin[1][:subsample] >
329                                     cutoff_mag).float())
330
331 data_grads = torch.zeros(len(data_losses))
332 for i, dl in enumerate(data_losses):
333     data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]

```

```

318 data_grads_scatter = ax3.scatter(train_data_low_bin[0][::subsample] / (2 * math.pi),
                                   data_grads, marker="x", c="k")
319 ax3.set_xscale("log")
320 ax3.set_ylabel("Derivative")
321 ax3.set_xlim([1, 1e6])
322 ax3.set_ylim([-1.5e-3, 1.5e-3])
323 ax3.set_xlabel("Frequency (Hz)")
324 ax3.set_title("Derivative by Training Datapoint")
325
326 # Show gradient at each training iteration
327 ax4.plot(np.arange(len(grad_values_low_bin)), grad_values_low_bin, linewidth=3)
328 cur_iter, = ax4.plot(0, grad_values_low_bin[0], marker="o")
329 cur_grad_label = ax4.annotate(f"Grad = {grad_values_low_bin[0]:.2e}", (0, 0),
                               xytext=(0.65, 0.9), textcoords='axes
                               fraction')
330 ax4.set_xlabel("Training Iteration")
331 ax4.set_ylabel("Gradient")
332 ax4.set_title("Gradients")
333 ax4.set_xlim([-1, len(grad_values_low_bin)])
334
335 plt.tight_layout()
336
337 # Main update function for interactive plots
338 def update_iter_low_bin(t=0):
339     mags = evaluate_lp_circuit(ws, R_values_low_bin[t])
340     learned_tf.set_data(ws / (2 * math.pi), mags)
341     cutoff = ws[np.argmax(mags < cutoff_mag)]
342     cut.set_xdata(cutoff / (2 * math.pi))
343     cur_loss.set_data(R_values_low_bin[t], loss_surface_mse[:, int(R_values_low_bin
344                                     [t] - 10)].sum(0))
345     cur_loss_label.set_text(f"R = {R_values_low_bin[t]:.0f}")
346     cur_iter.set_data(t, grad_values_low_bin[t])
347     cur_grad_label.set_text(f"Grad = {grad_values_low_bin[t]:.2e}")
348     cur_circuit = LowPassCircuit(R_values_low_bin[t])
349     data_losses = loss_fn(cur_circuit(train_data_low_bin[0][::subsample]), (
350                                     train_data_low_bin[1][::subsample] >
351                                     cutoff_mag).float())
352     data_grads = torch.zeros(len(data_losses))
353     for i, dl in enumerate(data_losses):
354         data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
355     data_grads_scatter.set_offsets(torch.stack((train_data_low_bin[0][::subsample] / (
356                                     2 * math.pi), data_grads)).T)
357     fig.canvas.draw_idle()
358
359 # Include sliders for relevant quantities
360 ip = interactive(update_iter_low_bin,
361                  t=widgets.IntSlider(value=0, min=0, max=len(R_values_low_bin) - 1,
362                                     step=1, description="Training
363                                     Iteration", style={'description_width':
364                                     'initial'}, layout=Layout(width='100%'
365                                     )))
366
367 ip
368
369 """## (e) Learning a Low Pass Filter from Binary Data with a Different Loss"""
370
371 circuit = LowPassCircuit(500)
372 ### YOUR CODE HERE
373 loss_fn = lambda x, y: (torch.exp(x) - torch.exp(1.1*y))**2
374 ### END YOUR CODE
375 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)

367
368 # Plot transfer function over training
369 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
370 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
371 subsample = int(dataset_size / 100)
372 train_data_mask = train_data_low_bin[1][:subsample] > cutoff_mag
373 ax1.scatter(train_data_low_bin[0][:subsample][train_data_mask] / (2 * math.pi), np
    .ones(train_data_mask.sum()), c="r",
    marker="x")
374 ax1.scatter(train_data_low_bin[0][:subsample][~train_data_mask] / (2 * math.pi),
    np.zeros((~train_data_mask).sum()), c="
    k", marker="x")

375 mags = evaluate_lp_circuit(ws, R_values_low_bin[0])
376 learned_tf, = ax1.semilogx(ws / (2 * math.pi), mags, linewidth=3)
377 cutoff = ws[np.argmax(mags < cutoff_mag)]
378 cut = ax1.axvline(cutoff / (2 * math.pi), c="red", linestyle="--", linewidth=3)
379 ax1.set_xlim([1, 1e6])
380 ax1.set_title("Transfer Function")
381 ax1.set_xlabel("Frequency (Hz)")
382 ax1.set_ylabel("Magnitude")
383 ax1.legend(["Learned TF", "Learned $f_c$", "TF + Samples", "TF - Samples"])
384
385 # Show loss surface over training
386 eval_pts = torch.arange(10, 1001, 1)
387 eval_vals = evaluate_lp_circuit(train_data_low_bin[0][:, None], eval_pts[None, :])
388 loss_surface_mse = loss_fn(eval_vals, (train_data_low_bin[1][:, None].expand(
    eval_vals.shape) > cutoff_mag).float())
389 ax2.plot(eval_pts, loss_surface_mse.sum(0), linewidth=3)
390 cur_loss, = ax2.plot(R_values_low_bin[0], loss_surface_mse[:, int(R_values_low_bin[
    0] - 10)].sum(0), marker="o")
391 cur_loss_label = ax2.annotate(f"R = {R_values_low_bin[0]:.0f}", (0, 0), xytext=(0.
    82, 0.9), textcoords='axes fraction')
392 ax2.set_title("Loss Surface")
393 ax2.set_xlim([0, 1000])
394 ax2.set_xlabel("$R \ ; \ (\Omega)$")
395 ax2.set_ylabel("Loss")
396
397 # Show loss contributions of each data point
398 cur_circuit = LowPassCircuit(R_values_low_bin[0])
399 data_losses = loss_fn(cur_circuit(train_data_low_bin[0][:subsample]), (
    train_data_low_bin[1][:subsample] >
    cutoff_mag).float())
400 data_grads = torch.zeros(len(data_losses))
401 for i, dl in enumerate(data_losses):
402     data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
403 data_grads_scatter = ax3.scatter(train_data_low_bin[0][:subsample] / (2 * math.pi),
    data_grads, marker="x", c="k")
404 ax3.set_xscale("log")
405 ax3.set_ylabel("Derivative")
406 ax3.set_xlim([1, 1e6])
407 ax3.set_ylim([-1.5e-3, 1.5e-3])
408 ax3.set_xlabel("Frequency (Hz)")
409 ax3.set_title("Derivative by Training Datapoint")
410
411 # Show gradient at each training iteration
412 ax4.plot(np.arange(len(grad_values_low_bin)), grad_values_low_bin, linewidth=3)
413 cur_iter, = ax4.plot(0, grad_values_low_bin[0], marker="o")
414 cur_grad_label = ax4.annotate(f"Grad = {grad_values_low_bin[0]:.2e}", (0, 0),
    xytext=(0.65, 0.9), textcoords='axes

```

```

fraction')
415 ax4.set_xlabel("Training Iteration")
416 ax4.set_ylabel("Gradient")
417 ax4.set_title("Gradients")
418 ax4.set_xlim([-1, len(grad_values_low_bin)])
419
420 plt.tight_layout()
421
422 # Main update function for interactive plots
423 def update_iter_low_bin(t=0):
424     mags = evaluate_lp_circuit(ws, R_values_low_bin[t])
425     learned_tf.set_data(ws / (2 * math.pi), mags)
426     cutoff = ws[np.argmax(mags < cutoff_mag)]
427     cut.set_xdata(cutoff / (2 * math.pi))
428     cur_loss.set_data(R_values_low_bin[t], loss_surface_mse[:, int(R_values_low_bin
429                                                                [t] - 10)].sum(0))
429     cur_loss_label.set_text(f"R = {R_values_low_bin[t]:.0f}")
430     cur_iter.set_data(t, grad_values_low_bin[t])
431     cur_grad_label.set_text(f"Grad = {grad_values_low_bin[t]:.2e}")
432     cur_circuit = LowPassCircuit(R_values_low_bin[t])
433     data_losses = loss_fn(cur_circuit(train_data_low_bin[0][::subsample]), (
434                                                                train_data_low_bin[1][::subsample] >
435                                                                cutoff_mag).float())
434     data_grads = torch.zeros(len(data_losses))
435     for i, dl in enumerate(data_losses):
436         data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
437     data_grads_scatter.set_offsets(torch.stack((train_data_low_bin[0][::subsample] / (
438                                                                2 * math.pi), data_grads)).T)
438     fig.canvas.draw_idle()
439
440 # Include sliders for relevant quantities
441 ip = interactive(update_iter_low_bin,
442                  t=widgets.IntSlider(value=0, min=0, max=len(R_values_low_bin) - 1,
443                                     step=1, description="Training
444                                     Iteration", style={'description_width':
445                                     'initial'}, layout=Layout(width='100%'
446                                     )))
443 ip
444
445 """## (f) Learning a High Pass Filter from Binary Data"""
446
447 # Transfer function: evaluates magnitude of given frequencies for a resistor value
448 # in the high pass circuit
449 def evaluate_hp_circuit(freqs, R_high):
450     """ YOUR CODE HERE """
451     return (R_high * cap_value * freqs) / torch.sqrt(1 + (R_high * cap_value * freqs
452                                                         ) ** 2)
453     """ END YOUR CODE """
454
455 # PyTorch model of the high pass circuit (for training)
456 class HighPassCircuit(nn.Module):
457     def __init__(self, R=None):
458         super().__init__()
459         self.R = nn.Parameter(torch.tensor(R, dtype=float) if R is not None else
460                                torch.rand(1) * 1000)
461
462     def forward(self, freqs):
463         return evaluate_hp_circuit(freqs, self.R)
464
465 # Generate training data in a uniform log scale of frequencies, then evaluate using

```



```

the true transfer function
463 R_high_des = 1 / (2 * math.pi * 5000 * cap_value)
464 def generate_hp_training_data(n):
465     rand_ws = 2 * math.pi * torch.pow(10, torch.rand(n) * 6)
466     labels = evaluate_hp_circuit(rand_ws, R_high_des)
467     return rand_ws, labels
468
469 # Train a given low pass filter from binary data
470 def train_hp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps, lr)
471     :
472     R_values = [float(circuit.R.data)]
473     grad_values = [np.nan]
474     train_data = generate_hp_training_data(dataset_size)
475     print(f"Initial Resistor Value: R = {float(circuit.R.data):.0f}")
476     iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
477     for i in iter_bar:
478         pred = circuit(train_data[0])
479         loss = loss_fn(pred, (train_data[1] > cutoff_mag).float()).mean()
480         ### YOUR CODE HERE
481         grad = torch.autograd.grad(loss, circuit.R)
482         ### END YOUR CODE
483         with torch.no_grad():
484             ### YOUR CODE HERE
485             circuit.R -= lr*grad[0]
486             ### END YOUR CODE
487
488         R_values.append(float(circuit.R.data))
489         grad_values.append(float(grad[0].data))
490         iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f}, R={float(circuit.R
491             .data):.0f}")
492         if loss.data < 1e-6 or abs(grad[0].data) < 1e-6:
493             break
494
495     print(f"Final Resistor Value: R = {float(circuit.R.data):.0f}")
496     return train_data, R_values, grad_values
497
498 # Create a circuit, use loss_fn with learning rate of 1000
499 circuit = HighPassCircuit(500)
500 ### YOUR CODE HERE
501 lambda x, y: (torch.exp(x) - torch.exp(1.1*y))**2
502 ### END YOUR CODE
503 lr = 1000
504 train_data_high_bin, R_values_high_bin, grad_values_high_bin =
505     train_hp_circuit_binary(circuit,
506         loss_fn, dataset_size,
507         max_training_steps, lr)
508
509 # Plot transfer function over training
510 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
511 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
512 subsample = int(dataset_size / 100)
513 train_data_mask = train_data_high_bin[1][:subsample] > cutoff_mag
514 ax1.scatter(train_data_high_bin[0][:subsample][train_data_mask] / (2 * math.pi),
515             np.ones(train_data_mask.sum()), c="r",
516             marker="x")
517 ax1.scatter(train_data_high_bin[0][:subsample][~train_data_mask] / (2 * math.pi),
518             np.zeros((~train_data_mask).sum()), c="
519             k", marker="x")
520 mags = evaluate_hp_circuit(ws, R_values_high_bin[0])
521 learned_tf, = ax1.semilogx(ws / (2 * math.pi), mags, linewidth=3)

```



```

514 cutoff = ws[np.argmax(mags > cutoff_mag)]
515 cut = ax1.axvline(cutoff / (2 * math.pi), c="red", linestyle="--", linewidth=3)
516 ax1.set_xlim([1, 1e6])
517 ax1.set_title("Transfer Function")
518 ax1.set_xlabel("Frequency (Hz)")
519 ax1.set_ylabel("Magnitude")
520 ax1.legend(["Learned TF", "Learned  $f_c$ ", "TF + Samples", "TF - Samples"])
521
522 # Show loss surface over training
523 eval_pts = torch.arange(10, 1001, 1)
524 eval_vals = evaluate_hp_circuit(train_data_high_bin[0][:, None], eval_pts[None, :])
525 loss_surface_mse = loss_fn(eval_vals, (train_data_high_bin[1][:, None].expand(
526                                     eval_vals.shape) > cutoff_mag).float())
526 ax2.plot(eval_pts, loss_surface_mse.sum(0), linewidth=3)
527 cur_loss, = ax2.plot(R_values_high_bin[0], loss_surface_mse[:, int(
528                                     R_values_high_bin[0] - 10)].sum(0),
529                                     marker="o")
528 cur_loss_label = ax2.annotate(f"R = {R_values_high_bin[0]:.0f}", (0, 0), xytext=(0.
529                                     82, 0.9), textcoords='axes fraction')
529
529 ax2.set_title("Loss Surface")
530 ax2.set_xlim([0, 1000])
531 ax2.set_xlabel(" $R \ ; \ (\Omega)$ ")
532 ax2.set_ylabel("Loss")
533
534 # Show loss contributions of each data point
535 cur_circuit = HighPassCircuit(R_values_high_bin[0])
536 data_losses = loss_fn(cur_circuit(train_data_high_bin[0][::subsample]), (
537                                     train_data_high_bin[1][::subsample] >
538                                     cutoff_mag).float())
537 data_grads = torch.zeros(len(data_losses))
538 for i, dl in enumerate(data_losses):
539     data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
540 data_grads_scatter = ax3.scatter(train_data_high_bin[0][::subsample] / (2 * math.pi),
541                                     data_grads, marker="x", c="k")
541
541 ax3.set_xscale("log")
542 ax3.set_ylabel("Derivative")
543 ax3.set_xlim([1, 1e6])
544 ax3.set_ylim([-3e-3, 3e-3])
545 ax3.set_xlabel("Frequency (Hz)")
546 ax3.set_title("Derivative by Training Datapoint")
547
548 # Show gradient at each training iteration
549 ax4.plot(np.arange(len(grad_values_high_bin)), grad_values_high_bin, linewidth=3)
550 cur_iter, = ax4.plot(0, grad_values_high_bin[0], marker="o")
551 cur_grad_label = ax4.annotate(f"Grad = {grad_values_high_bin[0]:.2e}", (0, 0),
552                                     xytext=(0.65, 0.9), textcoords='axes
553                                     fraction')
552
552 ax4.set_xlabel("Training Iteration")
553 ax4.set_ylabel("Gradient")
554 ax4.set_title("Gradients")
555 ax4.set_xlim([-1, len(grad_values_high_bin)])
556
557 plt.tight_layout()
558
559 # Main update function for interactive plots
560 def update_iter_high_bin(t=0):
561     mags = evaluate_hp_circuit(ws, R_values_high_bin[t])
562     learned_tf.set_data(ws / (2 * math.pi), mags)
563     cutoff = ws[np.argmax(mags > cutoff_mag)]
564     cut.set_xdata(cutoff / (2 * math.pi))
565     cur_loss.set_data(R_values_high_bin[t], loss_surface_mse[:, int(

```

```

566         R_values_high_bin[t] - 10)].sum(0))
567 cur_iter.set_data(t, grad_values_high_bin[t])
568 cur_grad_label.set_text(f"Grad = {grad_values_high_bin[t]:.2e}")
569 cur_circuit = HighPassCircuit(R_values_high_bin[t])
570 data_losses = loss_fn(cur_circuit(train_data_high_bin[0][::subsample]), (
571     train_data_high_bin[1][::subsample] >
572     cutoff_mag).float())
573 data_grads = torch.zeros(len(data_losses))
574 for i, dl in enumerate(data_losses):
575     data_grads[i] = torch.autograd.grad(dl, cur_circuit.R, retain_graph=True)[0]
576 data_grads_scatter.set_offsets(torch.stack((train_data_high_bin[0][::subsample] /
577     (2 * math.pi), data_grads)).T)
578 fig.canvas.draw_idle()
579
580 # Include sliders for relevant quantities
581 ip = interactive(update_iter_high_bin,
582     t=widgets.IntSlider(value=0, min=0, max=len(R_values_high_bin) - 1
583         , step=1, description="Training
584         Iteration", style={'description_width':
585         'initial'}, layout=Layout(width='100%'
586         )))
587
588 ip
589
590 """## (g) Learning a Band Pass Filter from Binary Data"""
591
592 # Transfer function: evaluates magnitude of given frequencies for resistor values
593 # in the band pass circuit
594 def evaluate_bp_circuit(freqs, R_low, R_high):
595     """ YOUR CODE HERE """
596     return evaluate_lp_circuit(freqs=evaluate_hp_circuit(freqs, R_high), R_low =
597         R_low)
598
599 """ END YOUR CODE """
600
601 # PyTorch model of the band pass circuit (for training)
602 class BandPassCircuit(nn.Module):
603     def __init__(self, R_low=None, R_high=None):
604         super().__init__()
605         self.R_low = nn.Parameter(torch.tensor(R_low, dtype=float) if R_low is not
606             None else torch.rand(1) * 1000)
607         self.R_high = nn.Parameter(torch.tensor(R_high, dtype=float) if R_high is
608             not None else torch.rand(1) * 1000)
609
610     def forward(self, freqs):
611         return evaluate_bp_circuit(freqs, self.R_low, self.R_high)
612
613 # Generate training data in a uniform log scale of frequencies, then evaluate using
614 # true transfer function
615 R_low_des = 1 / (2 * math.pi * 4000 * cap_value)
616 R_high_des = 1 / (2 * math.pi * 1000 * cap_value)
617 def generate_bp_training_data(n):
618     rand_ws = 2 * math.pi * torch.pow(10, torch.rand(n) * 6)
619     labels = evaluate_bp_circuit(rand_ws, R_low_des, R_high_des)
620     return rand_ws, labels
621
622 # Train a given low pass filter from binary data
623 def train_bp_circuit_binary(circuit, loss_fn, dataset_size, max_training_steps, lr)
624     :
625
626     R_values = [[float(circuit.R_low.data), float(circuit.R_high.data)]]

```

```

612 grad_values = [[np.nan, np.nan]]
613 train_data = generate_bp_training_data(dataset_size)
614 print(f"Initial Resistor Values: R_low = {float(circuit.R_low.data):.0f},
        R_high = {float(circuit.R_high.data):.0f}")
615 iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
616 for i in iter_bar:
617     pred = circuit(train_data[0])
618     loss = loss_fn(pred, (train_data[1] > cutoff_mag).float()).mean()
619     ### YOUR CODE HERE
620     grad = torch.autograd.grad(loss, [circuit.R_low, circuit.R_high])
621     ### END YOUR CODE
622     with torch.no_grad():
623         ### YOUR CODE HERE
624         circuit.R_low -= lr*grad[0]
625         circuit.R_high -= lr*grad[1]
626         ### END YOUR CODE
627
628     R_values.append([float(circuit.R_low.data), float(circuit.R_high.data)])
629     grad_values.append([float(grad[0].data), float(grad[1].data)])
630     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}")
631     if loss.data < 1e-6 or (abs(grad[0].data) < 1e-6 and abs(grad[1].data) < 1e
        -6):
632         break
633
634     print(f"Final Resistor Values: R_low = {float(circuit.R_low.data):.0f}, R_high
        = {float(circuit.R_high.data):.0f}")
635     return train_data, R_values, grad_values
636
637 # Create a circuit, use loss_fn with learning rate of 1000
638 circuit = BandPassCircuit(500, 500)
639 lr = 1000
640 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)
641
642 # Plot transfer function over training
643 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
644 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
645 subsample = int(dataset_size / 100)
646 train_data_mask = train_data_band_bin[1][:subsample] > cutoff_mag
647 ax1.scatter(train_data_band_bin[0][:subsample][train_data_mask] / (2 * math.pi),
        np.ones(train_data_mask.sum()), c="r",
        marker="x")
648 ax1.scatter(train_data_band_bin[0][:subsample][~train_data_mask] / (2 * math.pi),
        np.zeros((~train_data_mask).sum()), c="
        k", marker="x")
649 learned_tf, = ax1.semilogx(ws / (2 * math.pi), evaluate_bp_circuit(ws, *
        R_values_band_bin[0]), linewidth=3)
650 ax1.set_xlim([1, 1e6])
651 ax1.set_title("Transfer Function")
652 ax1.set_xlabel("Frequency (Hz)")
653 ax1.set_ylabel("Magnitude")
654 ax1.legend(["Learned TF", "TF + Samples", "TF - Samples"])
655
656 # Show loss surfaces for BCE and MSE Loss
657 eval_pts = torch.stack(torch.meshgrid((torch.arange(0, 1000, 10), torch.arange(0,
        1000, 10))), indexing="ij"))

```

```

658 eval_vals = evaluate_bp_circuit(train_data_band_bin[0][:, None, None], eval_pts[0][
        None, ...], eval_pts[1][None, ...])
659 loss_surface = loss_fn(eval_vals, (train_data_band_bin[1][..., None, None].expand(
        eval_vals.shape) > cutoff_mag).float())
660 loss_surf = ax2.imshow(torch.log(loss_surface.mean(0)).T, cmap=plt.cm.jet, extent=(
        0, 1000, 0, 1000), aspect="auto",
        origin="lower")
661 cur_loss, = ax2.plot(*R_values_band_bin[0], marker="o")
662 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')
663 ax2.set_title("Loss Surface")
664 ax2.set_xlabel("$R_{\mathrm{low}} \ ; \ (\Omega)$")
665 ax2.set_ylabel("$R_{\mathrm{high}} \ ; \ (\Omega)$")
666 fig.colorbar(loss_surf, ax=ax2, label="log(loss)")
667
668 # Show loss contributions of each data point
669 cur_circuit = BandPassCircuit(*R_values_band_bin[0])
670 data_losses = loss_fn(cur_circuit(train_data_band_bin[0][::subsample]), (
        train_data_band_bin[1][::subsample] >
        cutoff_mag).float())
671 data_grads = torch.zeros((len(data_losses), 2))
672 for i, dl in enumerate(data_losses):
673     data_grads[i] = torch.tensor(torch.autograd.grad(dl, (cur_circuit.R_low,
        cur_circuit.R_high), retain_graph=True)
        )
674 data_grads_scatter1 = ax3.scatter(train_data_band_bin[0][::subsample] / (2 * math.pi),
        data_grads[:, 0], marker="x")
675 data_grads_scatter2 = ax3.scatter(train_data_band_bin[0][::subsample] / (2 * math.pi),
        data_grads[:, 1], marker="x")
676 ax3.set_xscale("log")
677 ax3.set_ylabel("Derivative")
678 ax3.set_xlim([1, 1e6])
679 ax3.set_ylim([-2e-3, 2e-3])
680 ax3.set_xlabel("Frequency (Hz)")
681 ax3.set_title("Derivative by Training Datapoint")
682 ax3.legend(["$R_{\mathrm{low}}$ Derivatives", "$R_{\mathrm{high}}$ Derivatives"])
683
684 # Show gradient at each training iteration
685 ax4.plot(np.arange(len(grad_values_band_bin)), grad_values_band_bin, linewidth=3)
686 cur_grad0, = ax4.plot(0, grad_values_band_bin[0][0], marker="o")
687 cur_grad1, = ax4.plot(0, grad_values_band_bin[0][1], marker="o")
688 ax4.set_xlabel("Training Iteration")
689 ax4.set_ylabel("Gradient")
690 ax4.set_title("Gradients")
691 ax4.set_xlim([-1, len(grad_values_band_bin)])
692 ax4.legend(["$R_{\mathrm{low}}$ Grad", "$R_{\mathrm{high}}$ Grad"])
693
694 plt.tight_layout()
695
696 # Main update function for interactive plots
697 def update_iter_band_bin(t=0):
698     mags = evaluate_bp_circuit(ws, *R_values_band_bin[t])
699     learned_tf.set_data(ws / (2 * math.pi), mags)
700     cur_loss.set_data(*R_values_band_bin[t])
701     cur_loss_label.set_text(f"R_low = {R_values_band_bin[t][0]:.0f}\nR_high = {
        R_values_band_bin[t][1]:.0f}")
702     cur_grad0.set_data(t, grad_values_band_bin[t][0])
703     cur_grad1.set_data(t, grad_values_band_bin[t][1])
704     cur_circuit = BandPassCircuit(*R_values_band_bin[t])

```

```

705     data_losses = loss_fn(cur_circuit(train_data_band_bin[0][::subsample]), (
                                                train_data_band_bin[1][::subsample] >
                                                cutoff_mag).float())
706     data_grads = torch.zeros((len(data_losses), 2))
707     for i, dl in enumerate(data_losses):
708         data_grads[i] = torch.tensor(torch.autograd.grad(dl, (cur_circuit.R_low,
                                                                cur_circuit.R_high), retain_graph=True)
                                     )
709     data_grads_scat1.set_offsets(torch.stack((train_data_band_bin[0][::subsample] /
                                                (2 * math.pi), data_grads[:, 0])).T)
710     data_grads_scat2.set_offsets(torch.stack((train_data_band_bin[0][::subsample] /
                                                (2 * math.pi), data_grads[:, 1])).T)
711     fig.canvas.draw_idle()
712
713     # Include sliders for relevant quantities
714     ip = interactive(update_iter_band_bin,
715                     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%'
716 ip
717
718     """## (h) Learning a Band Pass Filter Bode Plot from Transfer Function Samples"""
719
720     def evaluate_bp_bode(freqs, low_cutoff, high_cutoff):
721         return -20 * nn.ReLU()(torch.log10(freqs / low_cutoff)) + -20 * nn.ReLU()(torch
                                                .log10(high_cutoff / freqs))
722
723     # PyTorch model of the band pass bode plot
724     class BandPassBodePlot(nn.Module):
725         def __init__(self, low_cutoff=None, high_cutoff=None):
726             super().__init__()
727             self.low_cutoff = nn.Parameter(torch.rand(1) * 5000 if low_cutoff is None
                                                else torch.tensor(float(low_cutoff)))
728             self.high_cutoff = nn.Parameter(torch.rand(1) * 5000 if high_cutoff is None
                                                else torch.tensor(float(high_cutoff)))
729
730             def forward(self, freqs):
731                 return evaluate_bp_bode(freqs, self.low_cutoff, self.high_cutoff)
732
733     # Train a given band pass bode plot
734     def train_bp_bode(bode, loss_fn, dataset_size, max_training_steps, lr):
735
736         cutoff_values = [[float(bode.low_cutoff.data), float(bode.high_cutoff.data)]]
737         grad_values = [[np.nan, np.nan]]
738         train_data = generate_bp_training_data(dataset_size)
739         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")
740         iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
741         for i in iter_bar:
742
743             pred = bode(train_data[0])
744             loss = loss_fn(pred, 20 * torch.log10(train_data[1])).mean()
745             grad = torch.autograd.grad(loss, (bode.low_cutoff, bode.high_cutoff))
746             with torch.no_grad():
747                 bode.low_cutoff -= lr * grad[0]
748                 bode.high_cutoff -= lr * grad[1]
749

```

```

750         cutoff_values.append([float(bode.low_cutoff.data), float(bode.high_cutoff.
751                                data)])
752         grad_values.append([float(grad[0].data), float(grad[1].data)])
753         iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f}, f_c,l = {float(
754                                bode.low_cutoff.data / (2 * math.pi)):.
755                                0f} Hz, f_c,h = {float(bode.high_cutoff
756                                .data / (2 * math.pi)):.0f} Hz")
757         if loss.data < 1e-6 or (abs(grad[0].data) < 1e-6 and abs(grad[1].data) < 1e
758                                -6):
759             break
760         print(f"Final Cutoff Values: f_c,l = {float(bode.low_cutoff.data / (2 * math.pi
761                                )):.0f} Hz, f_c,h = {float(bode.
762                                high_cutoff.data / (2 * math.pi)):.0f}
763                                Hz")
764         return train_data, cutoff_values, grad_values
765
766 bode = BandPassBodePlot()
767 loss_fn = lambda x, y: (x - y) ** 2 # MSE loss
768 lr = 1000
769 train_data_band_bode, cutoffs_band_bode, grad_values_band_bode = train_bp_bode(bode
770                                         , loss_fn, dataset_size,
771                                         max_training_steps, lr)
772
773 # Plot transfer function over training
774 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(9, 6))
775 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
776 subsample = int(dataset_size / 100)
777 train_data_mask = train_data_band_bode[1][:subsample] > cutoff_mag
778 ax1.scatter(train_data_band_bode[0][:subsample] / (2 * math.pi), 20 * torch.log10(
779     train_data_band_bode[1][:subsample]),
780             c="k", marker="x")
781
782 learned_tf, = ax1.semilogx(ws / (2 * math.pi), evaluate_bp_bode(ws, *
783     cutoffs_band_bode[0]), linewidth=3)
784
785 ax1.set_xlim([1, 1e6])
786 ax1.set_title("Transfer Function")
787 ax1.set_xlabel("Frequency (Hz)")
788 ax1.set_ylabel("dB")
789 ax1.legend(["Learned Bode Plot", "TF Samples"])
790
791 # Show loss surfaces for BCE and MSE Loss
792 eval_pts = torch.stack(torch.meshgrid((torch.arange(1, 5001, 50), torch.arange(1,
793     5001, 50)), indexing="ij"))
794 eval_vals = evaluate_bp_bode(train_data_band_bode[0][:, None, None], 2 * math.pi *
795     eval_pts[0][None, ...], 2 * math.pi *
796     eval_pts[1][None, ...])
797
798 loss_surface = loss_fn(eval_vals, 20 * torch.log10(train_data_band_bode[1])[...,
799     None, None].expand(eval_vals.shape))
800
801 loss_surf = ax2.imshow(torch.log(loss_surface.mean(0)).T, cmap=plt.cm.jet, extent=(
802     1, 5000, 1, 5000), aspect="auto",
803     origin="lower")
804
805 cur_loss, = ax2.plot(cutoffs_band_bode[0][0] / (2 * math.pi), cutoffs_band_bode[0][
806     1] / (2 * math.pi), marker="o")
807
808 cur_loss_label = ax2.annotate(f"$f_{{c,l}}$ = {cutoffs_band_bode[0][0]:.0f}\n$f_{{c
809     ,h}}$ = {cutoffs_band_bode[0][1]:.0f}",
810     (0, 0), xytext=(0.7, 0.82), textcoords
811     ='axes fraction')
812
813 ax2.set_title("Loss Surface")
814 ax2.set_xlabel("$f_{\mathrm{c,low}} \backslash ;$ (Hz)$")
815 ax2.set_ylabel("$f_{\mathrm{c,high}} \backslash ;$ (Hz)$")
816 fig.colorbar(loss_surf, ax=ax2, label="log(loss)")

```

```

788
789 # Show loss contributions of each data point
790 cur_bode = BandPassBodePlot(*cutoffs_band_bode[0])
791 data_losses = loss_fn(cur_bode(train_data_band_bode[0][::subsample]), 20 * torch.
                                log10(train_data_band_bode[1][::
                                subsample]))
792 data_grads = torch.zeros((len(data_losses), 2))
793 for i, dl in enumerate(data_losses):
794     data_grads[i] = torch.tensor(torch.autograd.grad(dl, (cur_bode.low_cutoff,
                                cur_bode.high_cutoff), retain_graph=
                                True))
795 data_grads_scat1 = ax3.scatter(train_data_band_bode[0][::subsample] / (2 * math.pi)
                                , data_grads[:, 0], marker="x")
796 data_grads_scat2 = ax3.scatter(train_data_band_bode[0][::subsample] / (2 * math.pi)
                                , data_grads[:, 1], marker="x")
797 ax3.set_xscale("log")
798 ax3.set_ylabel("Derivative")
799 ax3.set_xlim([1, 1e6])
800 ax3.set_ylim([-5e-3, 5e-3])
801 ax3.set_xlabel("Frequency (Hz)")
802 ax3.set_title("Derivative by Training Datapoint")
803 ax3.legend([" $f_{c,l}$  Derivatives", " $f_{c,h}$  Derivatives"])
804
805 # Show gradient at each training iteration
806 ax4.plot(np.arange(len(grad_values_band_bode)), grad_values_band_bode, linewidth=3)
807 cur_grad0, = ax4.plot(0, grad_values_band_bode[0][0], marker="o")
808 cur_grad1, = ax4.plot(0, grad_values_band_bode[0][1], marker="o")
809 ax4.set_xlabel("Training Iteration")
810 ax4.set_ylabel("Gradient")
811 ax4.set_title("Gradients")
812 ax4.set_xlim([-1, len(grad_values_band_bode)])
813 ax4.legend([" $f_{c,l}$  Grad", " $f_{c,h}$  Grad"])
814
815 plt.tight_layout()
816
817 # Main update function for interactive plots
818 def update_iter_band_bode(t=0):
819     learned_tf.set_data(ws / (2 * math.pi), evaluate_bp_bode(ws, *cutoffs_band_bode
                                [t]))
820     cur_loss.set_data(cutoffs_band_bode[t][0] / (2 * math.pi), cutoffs_band_bode[t]
                                [1] / (2 * math.pi))
821     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}")
822     cur_grad0.set_data(t, grad_values_band_bode[t][0])
823     cur_grad1.set_data(t, grad_values_band_bode[t][1])
824     cur_bode = BandPassBodePlot(*cutoffs_band_bode[t])
825     data_losses = loss_fn(cur_bode(train_data_band_bode[0][::subsample]), 20 *
                                torch.log10(train_data_band_bode[1][::
                                subsample]))
826     data_grads = torch.zeros((len(data_losses), 2))
827     for i, dl in enumerate(data_losses):
828         data_grads[i] = torch.tensor(torch.autograd.grad(dl, (cur_bode.low_cutoff,
                                cur_bode.high_cutoff), retain_graph=
                                True))
829     data_grads_scat1.set_offsets(torch.stack((train_data_band_bode[0][::subsample]
                                / (2 * math.pi), data_grads[:, 0])).T)
830     data_grads_scat2.set_offsets(torch.stack((train_data_band_bode[0][::subsample]
                                / (2 * math.pi), data_grads[:, 1])).T)
831     fig.canvas.draw_idle()

```



```

832
833 # Include sliders for relevant quantities
834 ip = interactive(update_iter_band_bode,
835                  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%'
                                     )))
836 ip
837
838 """## (i) Learn a Color Organ Circuit"""
839
840 # PyTorch model of the color organ circuit
841 class ColorOrganCircuit(nn.Module):
842     def __init__(self, R_low=None, R_high=None, R_band_low=None, R_band_high=None):
843         super().__init__()
844         self.low = LowPassCircuit(R_low)
845         self.high = HighPassCircuit(R_high)
846         self.band = BandPassCircuit(R_band_low, R_band_high)
847
848     def forward(self, freqs):
849         return torch.stack((self.low(freqs), self.band(freqs), self.high(freqs)))
850
851 # Generate training data in a uniform log scale of frequencies, then evaluate using
852                                     the true transfer function
853 R_low_des = 1 / (2 * math.pi * 800 * cap_value)
854 R_band_low_des = 1 / (2 * math.pi * 4000 * cap_value)
855 R_band_high_des = 1 / (2 * math.pi * 1000 * cap_value)
856 R_high_des = 1 / (2 * math.pi * 5000 * cap_value)
857 def generate_co_training_data(n):
858     rand_ws = 2 * math.pi * torch.pow(10, torch.rand(n) * 6)
859     labels = torch.stack((evaluate_lp_circuit(rand_ws, R_low_des),
860                           evaluate_bp_circuit(rand_ws,
861                                                R_band_low_des, R_band_high_des),
862                           evaluate_hp_circuit(rand_ws, R_high_des)
863                           )))
864
865     return rand_ws, labels
866
867 # Train a given color organ circuit
868 def train_co_circuit(circuit, loss_fn, dataset_size, max_training_steps, lr):
869
870     R_values = [[float(circuit.low.R.data), float(circuit.band.R_low.data), float(
871                 circuit.band.R_high.data), float(
872                 circuit.high.R.data)]]
873
874     grad_values = [[np.nan, np.nan, np.nan, np.nan]]
875     train_data = generate_co_training_data(dataset_size)
876     print(f"Initial Resistor Values: LP: {float(circuit.low.R.data):.0f} Ohms, BP (
877           Low): {float(circuit.band.R_low.data):.
878           0f} Ohms, BP (High): {float(circuit.
879           band.R_high.data):.0f} Ohms, HP: {float(
880           circuit.high.R.data):.0f} Ohms")
881
882     iter_bar = tqdm.trange(max_training_steps, desc="Training Iter")
883     for i in iter_bar:
884         pred = circuit(train_data[0])
885         loss = loss_fn(pred, (train_data[1] > cutoff_mag).float()).mean()
886         grad = torch.autograd.grad(loss, (circuit.low.R, circuit.band.R_low,
887                                           circuit.band.R_high, circuit.high.R))
888
889         with torch.no_grad():
890             circuit.low.R -= lr * grad[0]

```



```

877         circuit.band.R_low -= lr * grad[1]
878         circuit.band.R_high -= lr * grad[2]
879         circuit.high.R -= lr * grad[3]
880
881     R_values.append([float(circuit.low.R.data), float(circuit.band.R_low.data),
882                     float(circuit.band.R_high.data), float(
883                         circuit.high.R.data)])
884     grad_values.append([float(grad[0].data), float(grad[1].data), float(grad[2]
885                             .data), float(grad[3].data)])
886     iter_bar.set_postfix_str(f"Loss: {float(loss.data):.3f}, Rs = {float(
887         circuit.low.R.data):.0f}, {float(
888             circuit.band.R_low.data):.0f}, {float(
889                 circuit.band.R_high.data):.0f}, {float(
890                     circuit.high.R.data):.0f}")
891     if loss.data < 1e-6 or (abs(grad[0].data) < 1e-6 and abs(grad[1].data) < 1e
892         -6):
893         break
894     print(f"Final Resistor Values: LP: {float(circuit.low.R.data):.0f} Ohms, BP (
895         Low): {float(circuit.band.R_low.data):.
896         0f} Ohms, BP (High): {float(circuit.
897             band.R_high.data):.0f} Ohms, HP: {float(
898                 circuit.high.R.data):.0f} Ohms")
899     print(f"Final Cutoff Frequencies: LP: {1 / (2 * math.pi * cap_value * float(
900         circuit.low.R.data)):.0f} Hz, BP (Low):
901         {1 / (2 * math.pi * cap_value * float(
902             circuit.band.R_low.data)):.0f} Hz, BP (
903             High): {1 / (2 * math.pi * cap_value *
904                 float(circuit.band.R_high.data)):.0f}
905             Hz, HP: {1 / (2 * math.pi * cap_value *
906                 float(circuit.high.R.data)):.0f} Hz")
907     return train_data, R_values, grad_values
908
909 co = ColorOrganCircuit(200, 200, 200, 200)
910 loss_fn = lambda x, y: (x - (0.3 + 0.7 * y)) ** 2 # weighted MSE loss
911 lr = 500
912 train_data_co, R_values_co, grad_values_co = train_co_circuit(co, loss_fn,
913     dataset_size, max_training_steps, lr)
914
915 # Plot transfer function over training
916 fig, ax1 = plt.subplots(1, 1, figsize=(9, 6))
917 ws = 2 * math.pi * 10 ** torch.linspace(0, 6, 1000)
918 subsample = int(dataset_size / 250)
919 train_data_mask = train_data_co[1][:, ::subsample] > cutoff_mag
920 learned_tf1, = ax1.semilogx(ws / (2 * math.pi), evaluate_lp_circuit(ws, R_values_co
921     [0][0]), linewidth=3)
922 learned_tf2, = ax1.semilogx(ws / (2 * math.pi), evaluate_bp_circuit(ws, *
923     R_values_co[0][1:3]), linewidth=3)
924 learned_tf3, = ax1.semilogx(ws / (2 * math.pi), evaluate_hp_circuit(ws, R_values_co
925     [0][-1]), linewidth=3)
926 ax1.scatter(train_data_co[0][:subsample][train_data_mask[0]] / (2 * math.pi), np.
927     ones(train_data_mask[0].sum()), c=
928     learned_tf1.get_color(), marker="x")
929 ax1.scatter(train_data_co[0][:subsample][train_data_mask[1]] / (2 * math.pi), np.
930     ones(train_data_mask[1].sum()), c=
931     learned_tf2.get_color(), marker="x")
932 ax1.scatter(train_data_co[0][:subsample][train_data_mask[2]] / (2 * math.pi), np.
933     ones(train_data_mask[2].sum()), c=
934     learned_tf3.get_color(), marker="x")
935 # ax1.scatter(train_data_co[0][:subsample][(~train_data_mask).all(0)] / (2 * math.
936     pi), np.zeros((~(train_data_mask.any(0))

```

```

908                                     )).sum()), c="k", marker="x")
909 ax1.set_xlim([1, 1e6])
910 ax1.set_title("Transfer Function")
911 ax1.set_xlabel("Frequency (Hz)")
912 ax1.set_ylabel("Magnitude")
913 ax1.legend(["Learned LP", "Learned BP", "Learned HP",
914           "TF + Samples (LP)", "TF + Samples (BP)", "TF + Samples (HP)",
915           "TF - Samples"], bbox_to_anchor=(1.05, 1), loc='upper left', ncol=1)
916 plt.tight_layout()
917
918 # Main update function for interactive plots
919 def update_iter_co(t=0):
920     learned_tf1.set_data(ws / (2 * math.pi), evaluate_lp_circuit(ws, R_values_co[t]
921                                                                [0]))
922     learned_tf2.set_data(ws / (2 * math.pi), evaluate_bp_circuit(ws, *R_values_co[t
923                                                                [1:3]))
924     learned_tf3.set_data(ws / (2 * math.pi), evaluate_hp_circuit(ws, R_values_co[t
925                                                                [-1]))
926     fig.canvas.draw_idle()
927
928 # Include sliders for relevant quantities
929 ip = interactive(update_iter_co,
930                 t=widgets.IntSlider(value=0, min=0, max=len(R_values_co) - 1, step
931                                     =1, description="Training Iteration",
932                                     style={'description_width': 'initial'},
933                                     layout=Layout(width='100%')))
934
935 ip
936
937 """## Visualizing the computation graph for the Color Organ"""
938
939 from torchviz import make_dot
940 make_dot(co(generate_co_training_data(dataset_size)[0]), params=dict(co.
941                                     named_parameters()))

```