CS547 HW6

Group 37

Yue Cui

Gaoyu Liu

Colab link:

Objective

In this assignment, we will explore the dependence of a multilayer feedforward network on

1) number of hidden layers;
2) dimensions of the internal layers;
3) activation functions (ReLU and sigmoid)

by a simple problem: estimating a real valued function

$$f(x) = x^3 - 0.5x^2$$

This is also known as hyperparameter tuning task.

Imports and Config

In [1]:

```
import torch
import numpy as np
import matplotlib
import pandas as pd
# %matplotlib notebook
import matplotlib.pyplot as plt
import scipy.stats
#from pandas.plotting import autocorrelation_plot
import matplotlib.offsetbox as offsetbox
from matplotlib.ticker import StrMethodFormatter
import graphviz
import itertools
import time
import sys


def saver(fname):
    plt.savefig(fname + ".png", bbox_inches="tight")


def legend(pos="bottom", ncol=3):
    if pos == "bottom":
        plt.legend(bbox_to_anchor=(0.5, -0.2),
                   loc='upper center',
                   facecolor="lightgray",
                   ncol=ncol)
    elif pos == "side":
        plt.legend(bbox_to_anchor=(1.1, 0.5),
                   loc='center left',
                   facecolor="lightgray",
                   ncol=1)


def textbox(txt, fname=None):
    plt.figure(figsize=(1, 1))
    plt.gca().add_artist(
        offsetbox.AnchoredText("\n".join(txt),
                               loc="center",
                               prop=dict(size=30)))
    plt.axis('off')
    if fname is not None:
        saver(fname)
    plt.show()
    plt.close()
```

In [ ]:

```
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 600

font = {#'family' : 'normal',
        'weight' : 'normal',
        'size'   : 6}


matplotlib.rc('font', **font)


colourWheel = [
    '#f5abb7',
    '#ff6961',
    # '#eb4035',

    # '#cb364a',
    '#e31a1c',
    '#be0f2d',
    '#67001f',
    #
    '#badde9',
    '#94d2ef',
    '#5091c0',
    '#355386',
    '#053061',
    # '#dfe7d7',
    '#a7d4c3',
    '#b0ce95',
    '#bbd96d',
    '#509a80',
    '#46776d',
    # '#3d6756',
    # '#329932',


    # 'b',

    # '#fb9a99',
    '#f0c986',
    '#dba880',
    '#b15928',
    '#8d4b45',
    '#ff7f00',
    #
    '#d5c0cf',
    '#b186a3',
    '#684e94',
    '#6a3d9a',
    '#2d1c4d',
    # '#ffff99',


    # '#b2182b',
    # '#d6604d',
    # '#f4a582',
    # '#fddbc7',
    # '#f7f7f7',
    # '#d1e5f0',
    # '#92c5de',
    # '#4393c3',
    # '#2166ac',
]
dashesStyles = [[3, 1],
                [5, 5],
                [1000, 1],
                [2, 1, 10, 1],
                [4, 1, 1, 1, 1, 1],
                [3, 5, 1, 5],
                [3, 5, 1, 5, 1, 5],
```

```
                 ]

# show the defined colorwheel
fig, ax = plt.subplots()

num_colors = len(colourWheel)
for j, n in enumerate(colourWheel):
    weight = None
    cn = n
    r1 = mpl.patches.Rectangle((0, num_colors - j - 1), 0.44, 1, color=cn)
    txt_0 = ax.text(.5, num_colors - j - .5, j, va='center', fontsize=10,
                    weight=weight)
    txt = ax.text(.55, num_colors - j - .5, '  ' + n, va='center', fontsize=10,
                  weight=weight)
    ax.add_patch(r1)
    # ax.add_patch(r2)
    ax.axhline(j, color='k')

ax.text(.22, j + 1.5, 'ColorWheel', ha='center', va='center')
#ax.text(1.5, j + 1.5, 'xkcd', ha='center', va='center')
ax.set_xlim(0, 0.75)
ax.set_ylim(0, j + 2)
ax.axis('off')

#plt.close()

# plt.clf()
def get_color_num(i, colors):
    return int((i % (len(colors) - 5)/5)) + \
           5*((i % (len(colors) - 5)) % 5) + 1
```

Prepare for Data

Define the target function and generate the data.

In [3]:
```
def f(x):
    return x**3 - 0.5 * x**2


X_train = np.linspace(-3, 3, 1001, dtype=np.float32)[:, np.newaxis]
y_train = f(X_train)

# convert to torch tensors
X_train = torch.from_numpy(X_train)
y_train = torch.from_numpy(y_train)
```

In [ ]:
```
```

Build Deep Model

In [4]:
```
print(f"PyTorch Version: {torch.__version__}")
print()
print(f"Python {sys.version}")
#print(f"Pandas {pd.__version__}")
#print(f"Scikit-Learn {sk.__version__}")
print("GPU is", "available" if torch.cuda.is_available() else "NOT AVAILABLE")

# check for GPU
if torch.cuda.is_available():
    X_train = X_train.cuda()
    y_train = y_train.cuda()
```

```
PyTorch Version: 1.7.1

Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)]
GPU is available
```

In [5]:

```python
def hidden_layer(g, layer_idx, nodes, prev_nodes=None):
    layer_node = ['a' + str(layer_idx + 1) + str(j + 1) for j in range(nodes)]
    for node in layer_node:
        g.node(node, label='')
    for node_0, node_1 in itertools.product(prev_nodes, layer_node):
        g.edge(node_0, node_1)
    return g, layer_node


def plot_network(
    input_features=['X'],
    nodes=[4],
    output_features=['y'],
):
    g = graphviz.Digraph()
    g.attr(rankdir='LR')
    # input layer
    for feature in input_features:
        g.node(feature)
    parent_nodes = input_features
    # hidden layer
    for i, item in enumerate(nodes):
        g, parent_nodes = hidden_layer(g, i, item, prev_nodes=parent_nodes)
    # output layer
    for feature in output_features:
        g.node(feature)
    for node_0, node_1 in itertools.product(parent_nodes, output_features):
        g.edge(node_0, node_1)
    #
    # g
    return g


# plot_network(input_features=['X'],
#                    nodes=[4,7,3],
#                    output_features=['y'],)
```

Several important setup of the models:

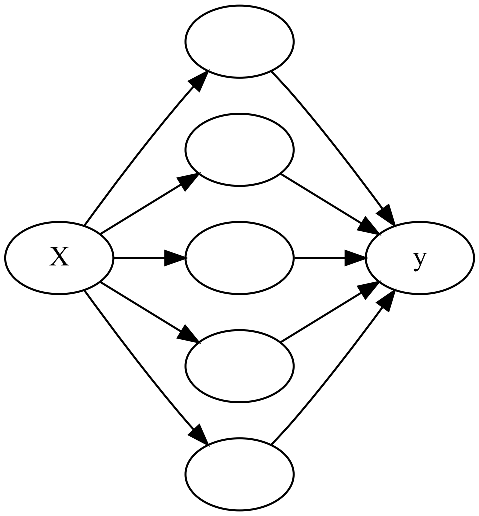
- Make **x** the only input for the neural network.
- Try **1, 2, 3, 4, and 5** hidden layer(s) with each layer consisting of various nodes.
- For all the hidden layers in the model, for simplicity, we keep **the number of nodes the same**.
- We assume that the feedforward networks are **fully connected**.

One Hidden Layer Model Class

In [6]:

```python
plot_network(
    input_features=['X'],
    nodes=[
        5,
    ],
    output_features=['y'],
)
```

Out[6]:



In [7]:

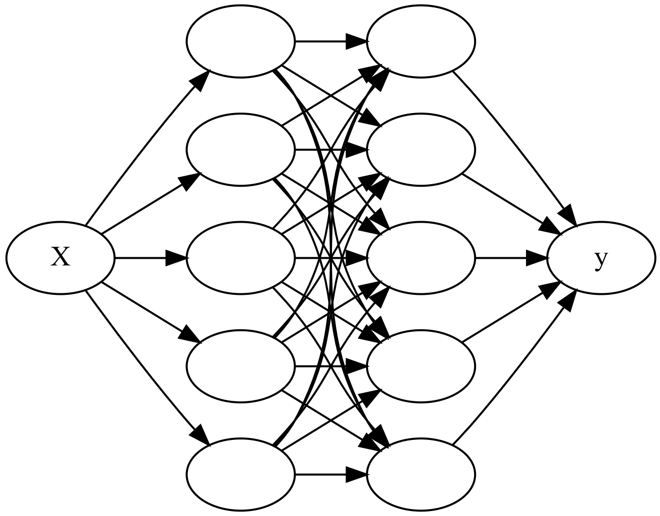
```
class SimpleFeedForward_1(torch.nn.Module):
    def __init__(
        self,
        max_nodes,
    ):
        # default to one-dimensional feature and response
        super().__init__()  # run init of torch.nn.Module
        self.max_nodes = max_nodes
        self.linear1 = torch.nn.Linear(1, max_nodes)
        self.linear2 = torch.nn.Linear(max_nodes, 1)
        self.sigmoid = torch.nn.Sigmoid()
        self.ReLU = torch.nn.ReLU()

    def forward(self, x, ReLU=True):
        # 1st hidden layer
        out = self.linear1(x)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # output layer
        out = self.linear2(out)
        #
        return out
```

Two Hidden Layer Model Class

In [8]:

```
plot_network(
    input_features=['X'],
    nodes=[5, 5],
    output_features=['y'],
)
```

Out[8]:



In [9]:

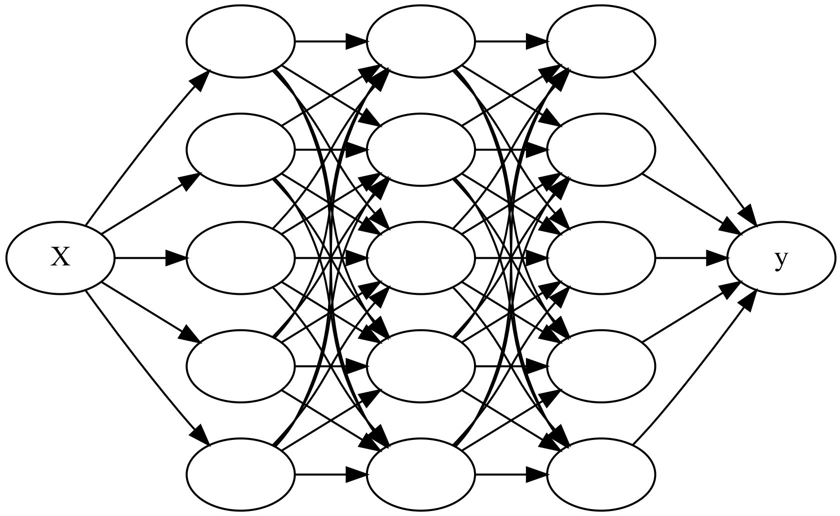
```
class SimpleFeedForward_2(torch.nn.Module):
    def __init__(
        self,
        max_nodes,
    ):
        # default to one-dimensional feature and response
        super().__init__()  # run init of torch.nn.Module
        self.max_nodes = max_nodes
        self.linear1 = torch.nn.Linear(1, max_nodes)
        self.linear2 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear3 = torch.nn.Linear(max_nodes, 1)
        self.sigmoid = torch.nn.Sigmoid()
        self.ReLU = torch.nn.ReLU()

    def forward(self, x, ReLU=True):
        # 1st hidden layer
        out = self.linear1(x)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 2nd layer
        out = self.linear2(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # output layer
        out = self.linear3(out)
        #
        return out
```

Three Hidden Layer Model Class

In [10…

```
plot_network(
    input_features=['X'],
    nodes=[5, 5, 5],
    output_features=['y'],
)
```

Out[10…



In [11…

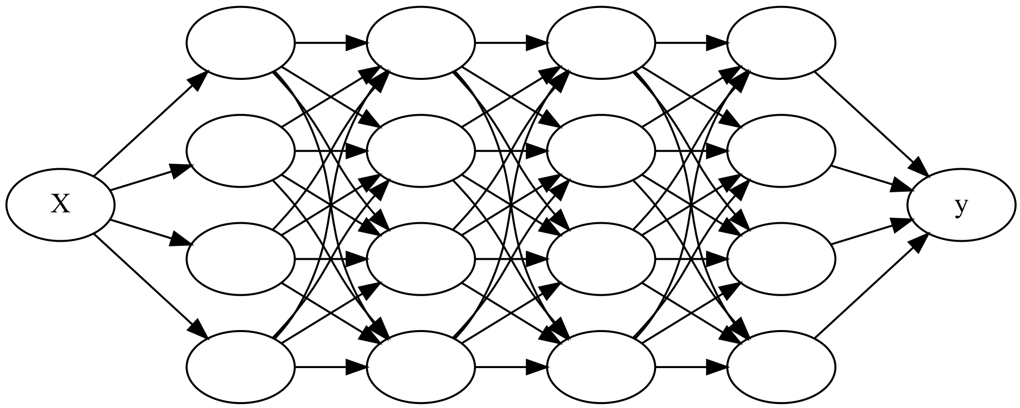
```
class SimpleFeedForward_3(torch.nn.Module):
    def __init__(
        self,
        max_nodes,
    ):
        # default to one-dimensional feature and response
        super().__init__()  # run init of torch.nn.Module
        self.max_nodes = max_nodes
        self.linear1 = torch.nn.Linear(1, max_nodes)
        self.linear2 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear3 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear4 = torch.nn.Linear(max_nodes, 1)
        self.sigmoid = torch.nn.Sigmoid()
        self.ReLU = torch.nn.ReLU()

    def forward(self, x, ReLU=True):
        # 1st hidden layer
        out = self.linear1(x)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 2nd layer
        out = self.linear2(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 3rd layer
        out = self.linear3(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # output layer
        out = self.linear4(out)
        #
        return out
```

Four Hidden Layers

In [12…

```
plot_network(
    input_features=['X'],
    nodes=[
        4, 4, 4, 4,
    ],
    output_features=['y'],
)
```

Out[12…



In [13…

```
class SimpleFeedForward_4(torch.nn.Module):
    def __init__(
        self,
        max_nodes,
    ):
        # default to one-dimensional feature and response
        super().__init__()  # run init of torch.nn.Module
        self.max_nodes = max_nodes
        self.linear1 = torch.nn.Linear(1, max_nodes)
        self.linear2 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear3 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear4 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear5 = torch.nn.Linear(max_nodes, 1)
        self.sigmoid = torch.nn.Sigmoid()
        self.ReLU = torch.nn.ReLU()

    def forward(self, x, ReLU=True):
        # 1st hidden layer
        out = self.linear1(x)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 2nd layer
        out = self.linear2(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 3rd layer
        out = self.linear3(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 4th layer
        out = self.linear4(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # output layer
        out = self.linear5(out)
        #
        return out
```

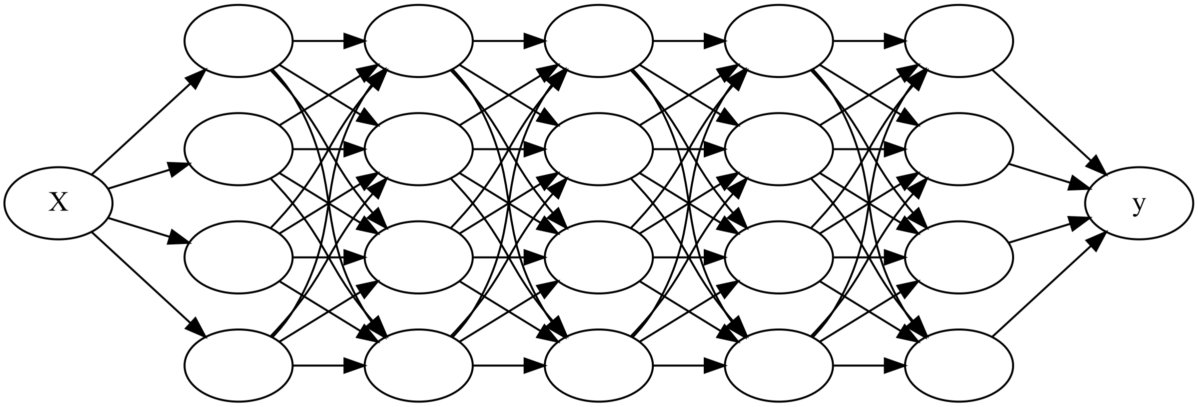
Five Hidden Layers

In [14…

```
plot_network(
    input_features=['X'],
    nodes=[
        4, 4, 4, 4, 4,
    ],
```

```
    output_features=['y'],
)
```

Out[14…



In [15…

```
class SimpleFeedForward_5(torch.nn.Module):
    def __init__(
        self,
        max_nodes,
    ):
        # default to one-dimensional feature and response
        super().__init__()  # run init of torch.nn.Module
        self.max_nodes = max_nodes
        self.linear1 = torch.nn.Linear(1, max_nodes)
        self.linear2 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear3 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear4 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear5 = torch.nn.Linear(max_nodes, max_nodes)
        self.linear6 = torch.nn.Linear(max_nodes, 1)
        self.sigmoid = torch.nn.Sigmoid()
        self.ReLU = torch.nn.ReLU()

    def forward(self, x, ReLU=True):
        # 1st hidden layer
        out = self.linear1(x)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 2nd layer
        out = self.linear2(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 3rd layer
        out = self.linear3(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 4th layer
        out = self.linear4(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # 5th layer
        out = self.linear5(out)
        if ReLU:
            out = self.ReLU(out)
        else:
            out = self.sigmoid(out)
        # output layer
        out = self.linear6(out)
        #
        return out
```

Training and Visualization Functions

In [24…

```
def train_model(
    initialweights,
    model,
    inputs=X_train,
    ground_truth=y_train,
    seed=0,
    learningRate=.01,
    MAX_iter=100000,
    loss_threshold=.01,
    show_results=True,
    ReLU=True,
):
    torch.manual_seed(seed)  # initialize for reproducibility
    model = model
    #     model.linear1.weight.data = torch.from_numpy(
    #         np.array(initialweights, dtype=np.float32))
    # check for CUDA
    if torch.cuda.is_available():
        model = model.cuda()
    # define the loss
    Loss = torch.nn.MSELoss()
    #
    losses = []
    errors = []
    # define the optimizer
    optimizer = torch.optim.SGD(model.parameters(),
                                lr=learningRate)  # gradient descent
    tic = time.perf_counter()
    for ctr in range(MAX_iter):

        # Clear gradient buffers because we don't want any gradient from previous epoch to carry forward,
        # dont want to cummulate gradients
        optimizer.zero_grad()

        # get output from the model, given the inputs
        outputs = model.forward(x=inputs, ReLU=ReLU)

        # get loss for the predicted output
        lossvalue = Loss(outputs, ground_truth)
        #error = (outputs > 0.5) != (labels == 1)
        # errors.append(torch.sum(error)/len(error))
        losses.append(lossvalue)
        if lossvalue < loss_threshold:
            break

        # get gradients w.r.t to parameters
        lossvalue.backward()
        # print(model.linear.weight.grad.item(),model.linear.bias.grad.item())

        # update parameters
        optimizer.step()
        if ctr % int(
                MAX_iter /
                10) == 0:  # print out data for 10 intermediate steps
            print("iteration {}: loss={:.5f}".format(ctr, lossvalue.item()))
    toc = time.perf_counter()

    if show_results:
        print("\n")
        print("M^{(1)}=\n",
              model.linear1.weight.data.cpu().numpy(), "\nB^{(1)}=",
              model.linear1.bias.data.cpu().numpy())
        print("\n")
        print("M^{(2)}=\n",
              model.linear2.weight.data.cpu().numpy(), "\nB^{(2)}=",
              model.linear2.bias.data.cpu().numpy())
        print("\n")
        try:
```

```
            print("M^{(3)}=\n",
                  model.linear3.weight.data.cpu().numpy(), "\nB^{(3)}=",
                  model.linear3.bias.data.cpu().numpy())
        except:
            print('No additonal layer transformations to show.')
        print("\n")
        try:
            print("M^{(4)}=\n",
                  model.linear4.weight.data.cpu().numpy(), "\nB^{(4)}=",
                  model.linear4.bias.data.cpu().numpy())
        except:
            print('No additonal layer transformations to show.')

    dt = toc - tic

    return model, losses, errors, dt
```

In [17…

```
def visualizer(
        model,
        losses,
        errors,
        dt,
        figsize=(3, 2.),
):
    #     outputs = model(inputs) > 0.5
    #     flags_correct = (labels == 1) == outputs
    #     accuracy = torch.sum(flags_correct)/len(flags_correct)
    #     flags = outputs.cpu().flatten()
    #     plt.figure()
    #     plt.scatter(xy[flags, 0], xy[flags, 1], color="red", label="1")
    #     plt.scatter(xy[~flags, 0], xy[~flags, 1], color="blue", label="0")
    #     title = []
    #     title.append("Computed Labels on Training Data")
    #     title.append("Accuracy={:.1%}".format(accuracy))
    #     plt.title("\n".join(title))
    #     plt.xlim(x_min, x_max)
    #     plt.ylim(x_min, x_max)
    #     legend("side")
    #     # saver("tt_123")
    #     plt.show()
    #     plt.close()

    plt.figure(figsize=figsize)
    plt.plot(losses, color="red")
    plt.xlabel("Iteration", fontsize=6)
    plt.ylabel("loss", fontsize=6)
    plt.ylim(-5., )
    title = []
    title.append("Losses")
    title.append("Elapsed time: {0:.02f} seconds".format(dt))
    plt.title("\n".join(title), fontsize=6)
    plt.axhline(0, linewidth=2, linestyle=":", color="black")
    plt.show()
    plt.close()

    #     plt.figure()
    #     plt.plot(errors, color="red")
    #     plt.xlabel("Iteration")
    #     plt.ylabel("percent wrong")
    #     plt.ylim(-0.05,)
    #     plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:.1%}'))
    #     plt.axhline(0, linewidth=2, linestyle=":", color="black")
    #     title = []
    #     title.append("Classification Errors")
    #     title.append("Elapsed time: {0:.02f} seconds".format(dt))
    #     plt.title("\n".join(title))
    #     plt.show()
    #     plt.close()

    plt.figure(figsize=figsize)
    plt.plot(X_train.cpu().detach().numpy(),
```

```
            model(X_train).cpu().detach().numpy(),
            color="red")
    plt.xlabel("X", fontsize=6)
    plt.ylabel("Prediction", fontsize=6)
    # plt.ylim(-0.1,)
    title = []
    title.append("Prediction")
    title.append("Elapsed time: {0:.02f} seconds".format(dt))
    plt.title("\n".join(title), fontsize=6)
    plt.axhline(0, linewidth=2, linestyle=":", color="black")
    plt.show()
    plt.close()
```

Experiments

This section studies the dependence on

- Number of hidden layers
- Dimension of internal layers (number of nodes in each hidden layer)
- Activation function: ReLU vs sigmoids

Before exploring the aspects above, it is also necessary to choose proper learning rates for the neural networks with different configurations.

To compare the performance of all models, a universal stopping criterion is selected. Namely, by preliminary experiments, a MSE of **0.01** is thought be a good stopping point for the iterations which update the model parameters. Provided the same stopping criterion for all models, the accuracy of all models measured by MSE loss can be regarded as the same. Thus, the training performance of the models can be compared by the training time and convergence of MSE with the iterations.

Preliminary Experiment

In this section, we find the proper learning rates for the model with various numbers layers. To save the computational time, we select the dimension of all the hidden layer to be 10.

The tested learning rate values include:

- 0.05
- 0.025
- 0.01
- 0.005
- 0.0025

Regardless of if all models reaches the target MSE, 0.01, a maximum of 100000 iterations is adopted.

Define Result DataFrame

In [ ]:

```
max_iters = 100000
lr_list = [.05, .025, .01, .005, .0025]
num_layers_list = [
    1,
    2,
    3,
    4,
    5,
]
nodes_list = [10]
activation_list = ['ReLU', 'Sigmoid']

# create the np.array and pd.DataFrame to save all the results
loss_array = np.empty((len(lr_list) * len(num_layers_list) * len(nodes_list) *
                       len(activation_list), max_iters))
loss_array[:] = np.NaN
display(loss_array.shape)

idxs = []
```

```
for activation in activation_list:
    for num_layers in num_layers_list:
        for nodes in nodes_list:
            for lr in lr_list:
                idxs.append(
                    str(num_layers) + '_' + str(nodes) + '_' + str(lr) + '_' +
                    activation)

cols = ['Layers', 'Nodes', 'Learning_rate', 'Activation', 'Time',] + \
    [i+1 for i in range(max_iters)]

model_res_df = pd.DataFrame(index=idxs, columns=cols)

for i in range(model_res_df.index.shape[0]):
    model_res_df.Layers.iloc[i] = num_layers_list[int(
        (i % (len(num_layers_list) * len(lr_list) * len(nodes_list))) /
        (len(lr_list) * len(nodes_list)))]
    model_res_df.Nodes.iloc[i] = nodes_list[int(
        (i % (len(lr_list) * len(nodes_list))) / len(lr_list))]
    model_res_df.Learning_rate.iloc[i] = lr_list[int(i % (len(lr_list)))]
    model_res_df.Activation.iloc[i] = activation_list[int(
        i / (len(num_layers_list) * len(lr_list) * len(nodes_list)))]

model_res_df_lr = model_res_df.copy()
display(model_res_df_lr.iloc[:10, :10])
```

Running Models

In [11…

```
def run_parametric(
    res_df,
    max_iter=max_iters,
    start = 0,
    end = -1,
):
    res_df = res_df.copy()
    # loop over all the tuples in the DF
    if end == -1:
        end = res_df.index.shape[0]
    for item in res_df.index[start:end]:
        layers = res_df.loc[item, 'Layers']
        nodes = res_df.loc[item, 'Nodes']
        lr = res_df.loc[item, 'Learning_rate']
        # choose model
        if layers == 1:
            model = SimpleFeedForward_1(nodes)
        elif layers == 2:
            model = SimpleFeedForward_2(nodes)
        elif layers == 3:
            model = SimpleFeedForward_3(nodes)
        elif layers == 4:
            model = SimpleFeedForward_4(nodes)
        elif layers == 5:
            model = SimpleFeedForward_5(nodes)
        else:
            print(f'Wrong number of layers. \n')
            continue
        # train the model
        print(f'===== Training model {item} ===== ')
        m, losses, errors, dt = \
            train_model(None,
                        model,
                        inputs=X_train,
                        ground_truth=y_train,
                        seed=0,
                        learningRate=lr,
                        MAX_iter=max_iter,
                        show_results=False,)
        print(f'\n')
        print(f'== Finished == \n')
        # save the results
        # running time
```

```python
        res_df.loc[item, 'Time'] = dt
        # losses
        losses_np = np.array([item.cpu().detach().numpy() for item in losses])
        res_df.loc[item].iloc[5:5 + losses_np.size] = losses_np
    #
    return res_df
```

In [22…

```python
model_res_df_lr = run_parametric(
    model_res_df_lr,
    max_iter=max_iters,
)
```

```
===== Training model 1_10_0.05_ReLU =====


== Finished ==

===== Training model 1_10_0.025_ReLU =====


== Finished ==

===== Training model 1_10_0.01_ReLU =====


== Finished ==

===== Training model 1_10_0.005_ReLU =====


== Finished ==

===== Training model 1_10_0.0025_ReLU =====


== Finished ==

===== Training model 2_10_0.05_ReLU =====


== Finished ==

===== Training model 2_10_0.025_ReLU =====


== Finished ==

===== Training model 2_10_0.01_ReLU =====


== Finished ==

===== Training model 2_10_0.005_ReLU =====


== Finished ==

===== Training model 2_10_0.0025_ReLU =====


== Finished ==

===== Training model 3_10_0.05_ReLU =====


== Finished ==

===== Training model 3_10_0.025_ReLU =====


== Finished ==
```

```
===== Training model 3_10_0.01_ReLU =====

== Finished ==
===== Training model 3_10_0.005_ReLU =====

== Finished ==
===== Training model 3_10_0.0025_ReLU =====

== Finished ==
===== Training model 4_10_0.05_ReLU =====

== Finished ==
===== Training model 4_10_0.025_ReLU =====

== Finished ==
===== Training model 4_10_0.01_ReLU =====

== Finished ==
===== Training model 4_10_0.005_ReLU =====

== Finished ==
===== Training model 4_10_0.0025_ReLU =====

== Finished ==
===== Training model 5_10_0.05_ReLU =====

== Finished ==
===== Training model 5_10_0.025_ReLU =====

== Finished ==
===== Training model 5_10_0.01_ReLU =====

== Finished ==
===== Training model 5_10_0.005_ReLU =====

== Finished ==
===== Training model 5_10_0.0025_ReLU =====

== Finished ==
===== Training model 1_10_0.05_Sigmoid =====

== Finished ==
===== Training model 1_10_0.025_Sigmoid =====
```

```
== Finished ==

===== Training model 1_10_0.01_Sigmoid =====

== Finished ==

===== Training model 1_10_0.005_Sigmoid =====

== Finished ==

===== Training model 1_10_0.0025_Sigmoid =====

== Finished ==

===== Training model 2_10_0.05_Sigmoid =====

== Finished ==

===== Training model 2_10_0.025_Sigmoid =====

== Finished ==

===== Training model 2_10_0.01_Sigmoid =====

== Finished ==

===== Training model 2_10_0.005_Sigmoid =====

== Finished ==

===== Training model 2_10_0.0025_Sigmoid =====

== Finished ==

===== Training model 3_10_0.05_Sigmoid =====

== Finished ==

===== Training model 3_10_0.025_Sigmoid =====

== Finished ==

===== Training model 3_10_0.01_Sigmoid =====

== Finished ==

===== Training model 3_10_0.005_Sigmoid =====

== Finished ==

===== Training model 3_10_0.0025_Sigmoid =====

== Finished ==

===== Training model 4_10_0.05_Sigmoid =====
```

```
== Finished ==

===== Training model 4_10_0.025_Sigmoid =====


== Finished ==

===== Training model 4_10_0.01_Sigmoid =====


== Finished ==

===== Training model 4_10_0.005_Sigmoid =====


== Finished ==

===== Training model 4_10_0.0025_Sigmoid =====


== Finished ==

===== Training model 5_10_0.05_Sigmoid =====


== Finished ==

===== Training model 5_10_0.025_Sigmoid =====


== Finished ==

===== Training model 5_10_0.01_Sigmoid =====


== Finished ==

===== Training model 5_10_0.005_Sigmoid =====


== Finished ==
```

Results

In [78…

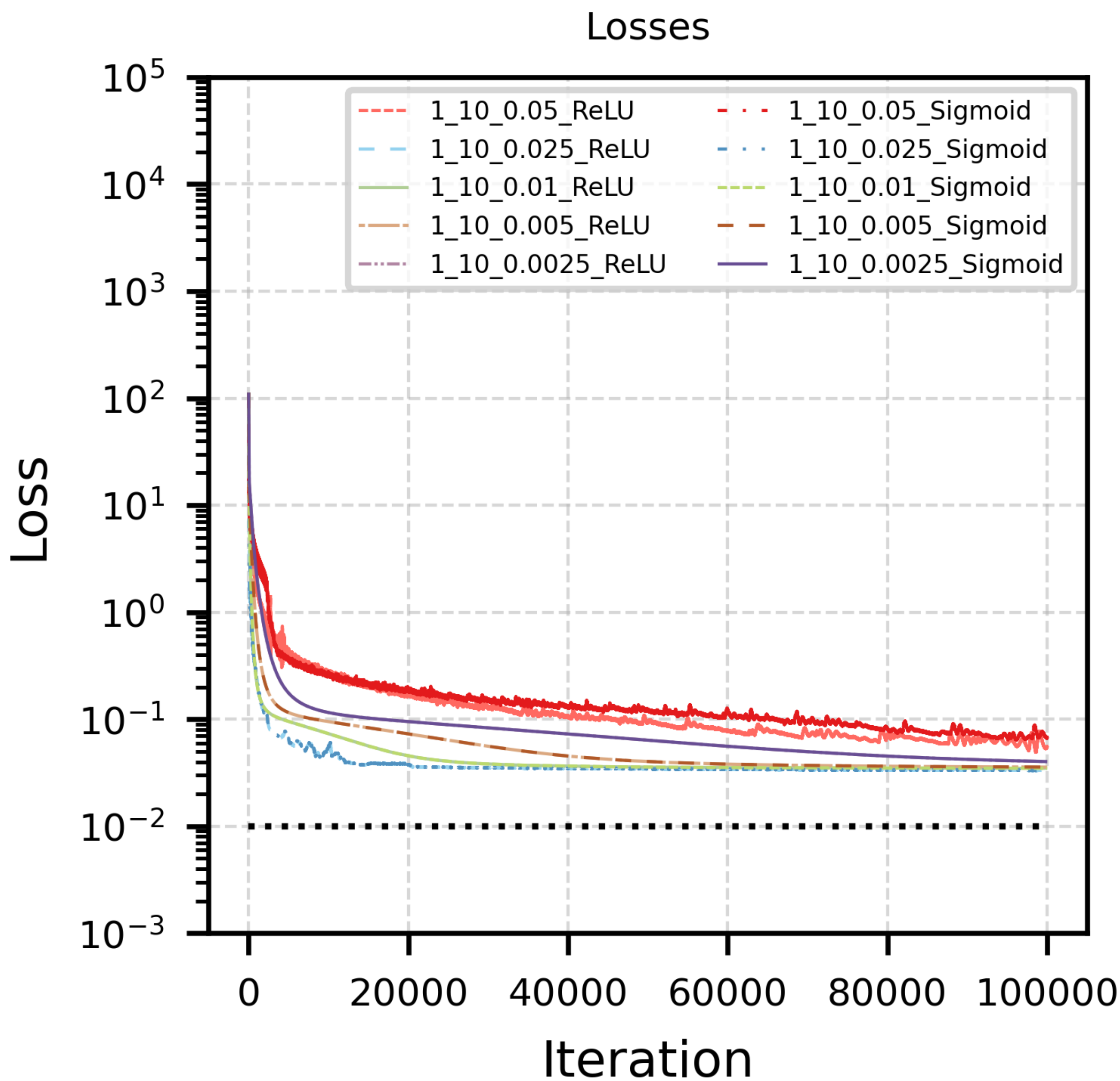
```python
def compare_models(
    res_df,
    fig_size=(2, 2),
    colors=colourWheel,
    dashes=dashesStyles,
    bar_width=0.5,
    loss_threshold = .01,
    time=True,
):
    layers_list = list(set(res_df.Layers))
    nodes_list = list(set(res_df.Nodes))
    lr_list = list(set(res_df.Learning_rate))
    #
    fig1, ax1 = plt.subplots(
        nrows=1,
        ncols=1,
        figsize=fig_size,
    )
    #
    if time:
        fig2, ax2 = plt.subplots(
            nrows=1,
            ncols=1,
            figsize=fig_size,
        )
    num_model = res_df.index.shape[0]
```

```python
    x = np.arange(num_model) * bar_width * 1.5  # the label locations
    #
    for i, model in enumerate(res_df.index):
        ax1.plot(
            res_df.iloc[i, 5:],
            color=colors[get_color_num(i, colors)],
            linestyle='-',
            dashes=dashes[i % len(dashes)],
            label=model,
            lw=.5,
        )
        try:
            rects = ax2.bar(x[i],
                            res_df.loc[model, 'Time'],
                            bar_width,
                            color=colors[get_color_num(i, colors)],
                            label=model)
        except:
            continue

    ax1.set_xlabel("Iteration", fontsize=8)
    ax1.set_ylabel("Loss", fontsize=8)
    ax1.set_yscale('log')
    ax1.set_ylim(1e-3, 1e5)
    title = []
    title.append("Losses")
    #title.append("Elapsed time: {0:.02f} seconds".format(dt))
    ax1.set_title("\n".join(title), fontsize=6)
    ax1.hlines(loss_threshold,
               0,
               res_df.columns[-1],
               linewidth=1,
               linestyle=":",
               color="black")
    ax1.grid(lw=.5, alpha=.5, ls='--')
    ax1.legend(loc=0, ncol=2, fontsize=4)

    try:
        ax2.set_xticks(x)
        ax2.set_xticklabels(res_df.index)
        ax2.grid(lw=.5, alpha=.5, ls='--')
        ax2.tick_params(axis='x', labelrotation=70)
        ax2.set_ylabel("Training Time (sec)", fontsize=8)
    except:
        return

    plt.show()
    plt.close()
```

One Hidden Layer

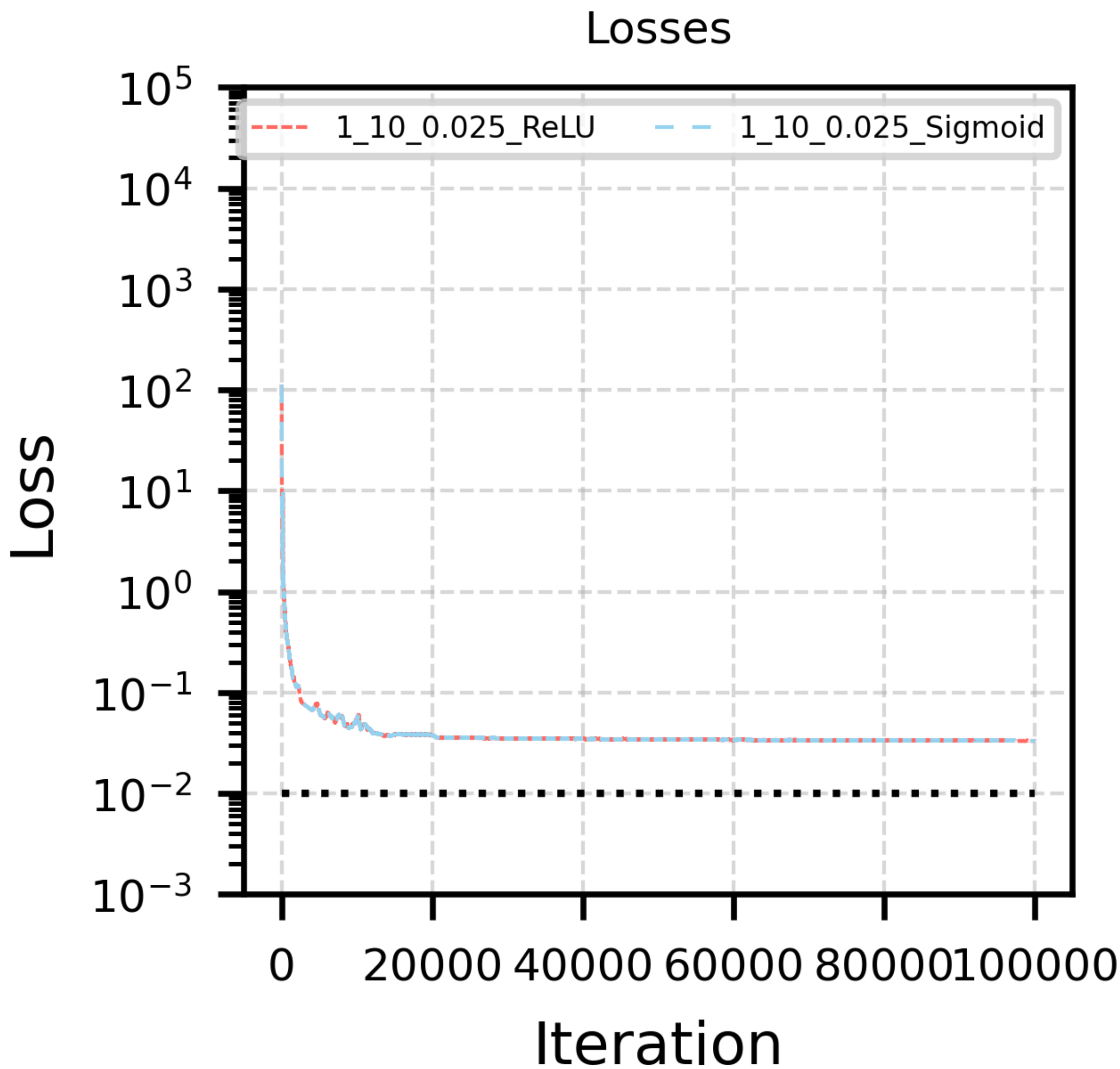
In [79…

```python
compare_models(
    model_res_df_lr[(model_res_df_lr.Layers == 1)],
    fig_size=(2.5, 2.5),
    colors=colourWheel,
    dashes=dashesStyles,
    time = False,
)
```

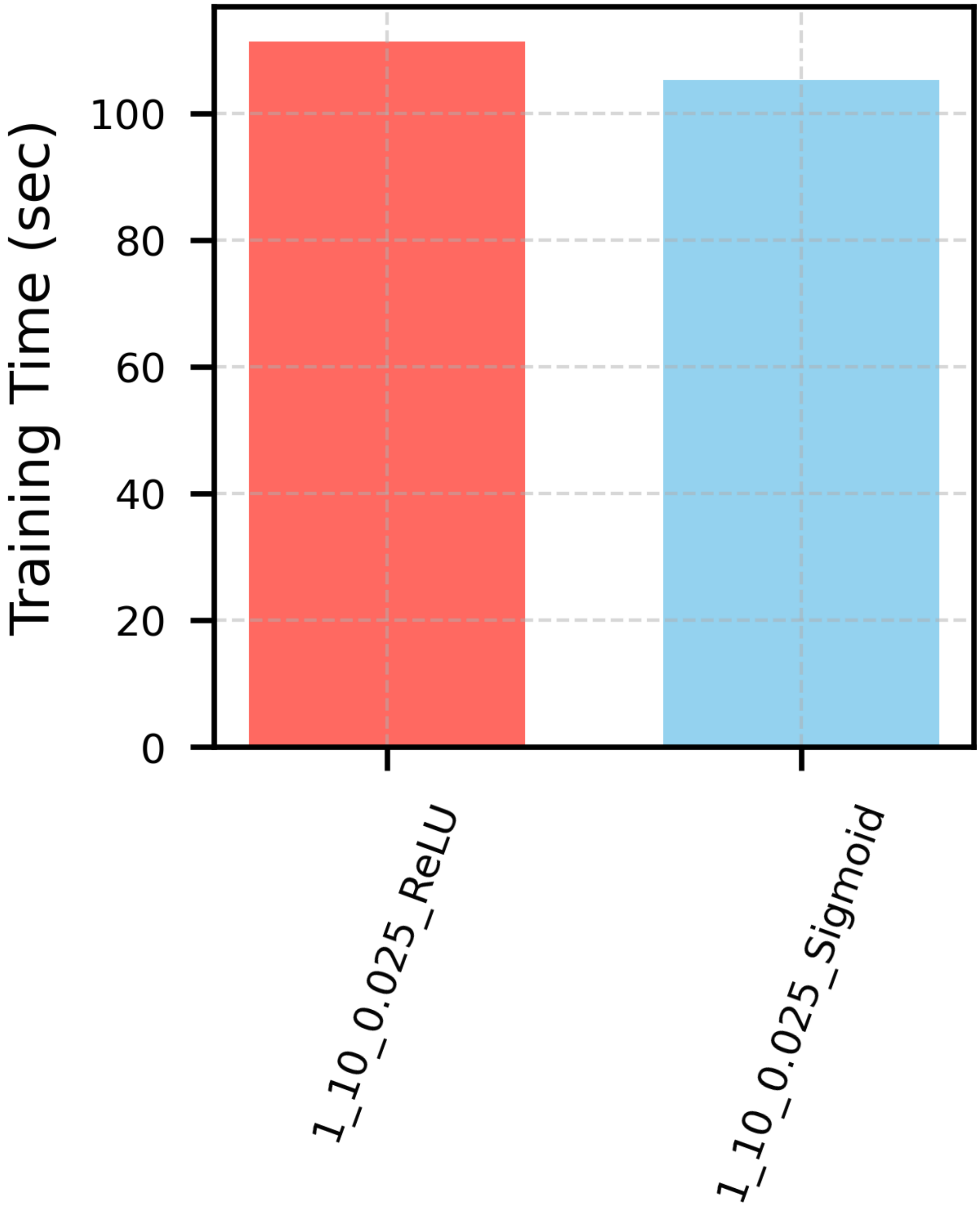


In [81…

```
compare_models(
    model_res_df_lr[(model_res_df_lr.Layers == 1) & (model_res_df_lr.Learning_rate == .025)],
    fig_size=(2, 2),
    colors=colourWheel,
    dashes=dashesStyles,
)
```



Losses
1_10_0.025_ReLU
1_10_0.025_Sigmoid
Loss
Iteration
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
0
20000
40000
60000
80000
100000

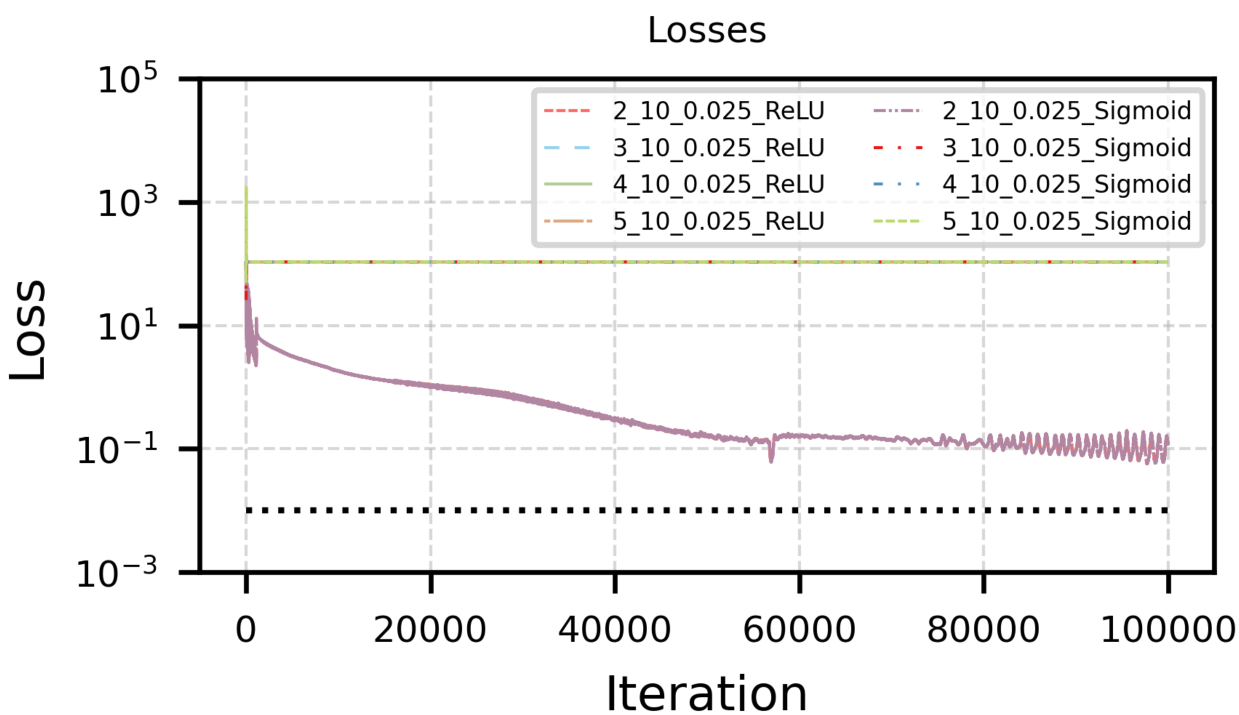


More than One Hidden Layers

In [ ]:
```
compare_models(
    model_res_df_lr[(model_res_df_lr.Layers > 1)
                    #& (model_res_df_lr.Activation == 'ReLU')
                    & (model_res_df_lr.Learning_rate >= .01)],
    fig_size=(3, 1.5),
    colors=colourWheel,
    dashes=dashesStyles,
    time=False,
)
```

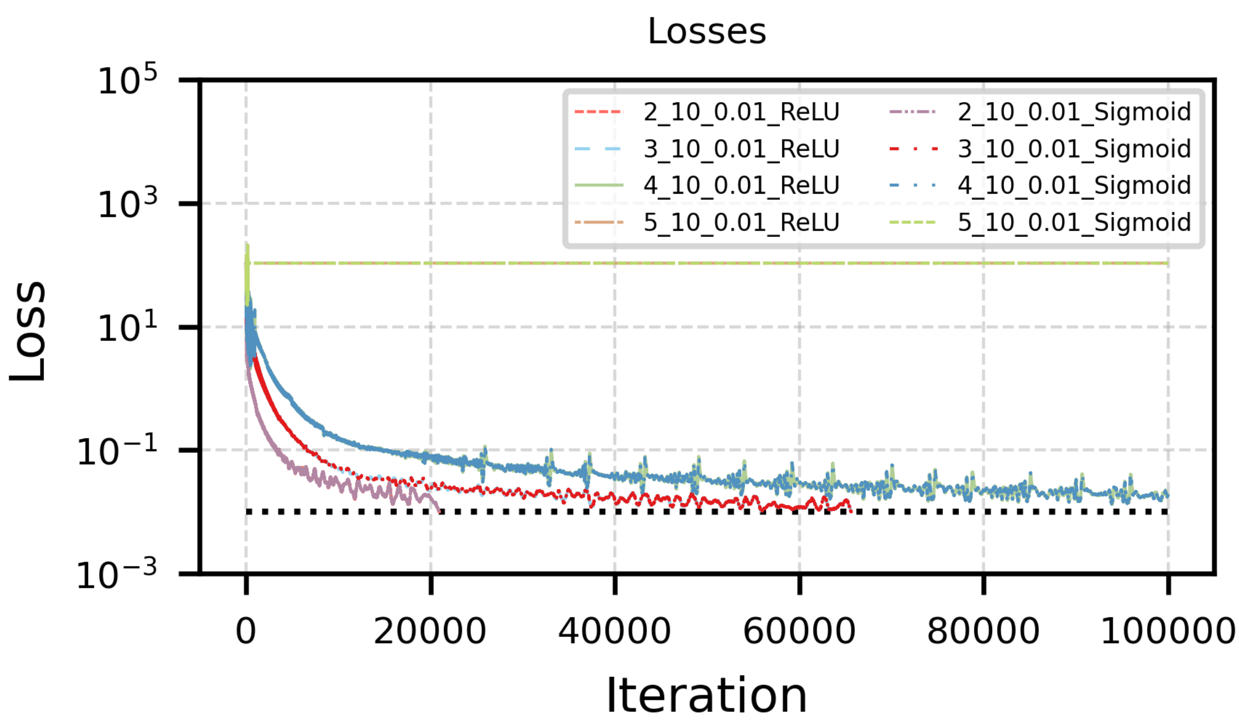
In [10…

```
compare_models(
    model_res_df_lr[(model_res_df_lr.Layers > 1)
                    #& (model_res_df_lr.Activation == 'ReLU')
                    & (model_res_df_lr.Learning_rate == .025)],
    fig_size=(3, 1.5),
    colors=colourWheel,
    dashes=dashesStyles,
    time=False,
)
```



In [10…

```
compare_models(
    model_res_df_lr[(model_res_df_lr.Layers > 1)
                    #& (model_res_df_lr.Activation == 'ReLU')
                    & (model_res_df_lr.Learning_rate == .01)],
    fig_size=(3, 1.5),
    colors=colourWheel,
    dashes=dashesStyles,
    time=False,
)
```



In [11…

```
model_res_df_lr[(model_res_df_lr.Layers == 5)
                    #& (model_res_df_lr.Activation == 'ReLU')
                    & (model_res_df_lr.Learning_rate == .0025)]
```

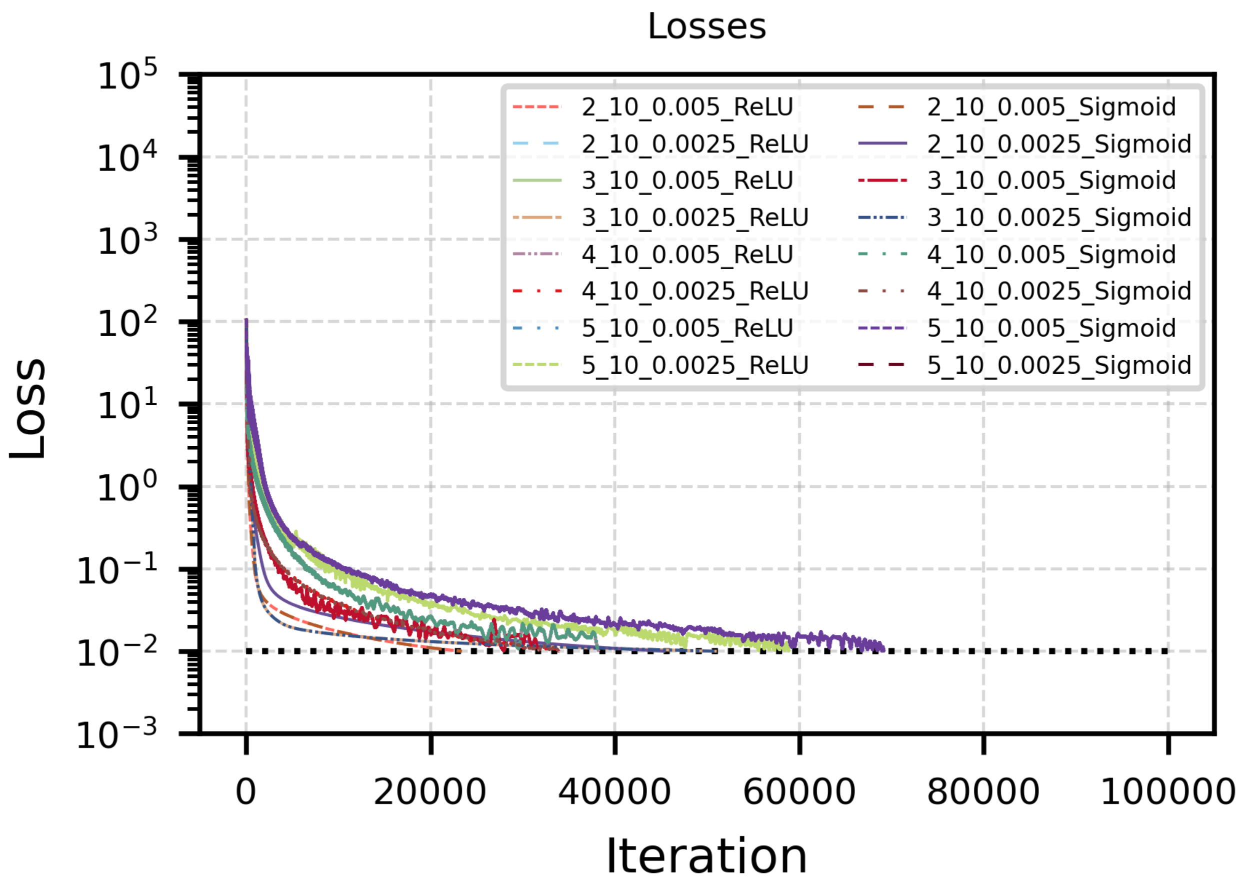
Out[11…

	Layers	Nodes	Learning_rate	Activation	Time	1	2	3	
5_10_0.0025_ReLU	5	10	0.0025	ReLU	141.344758	108.861961	108.819077	108.776672	108.73
5_10_0.0025_Sigmoid	5	10	0.0025	Sigmoid	NaN	NaN	NaN	NaN	

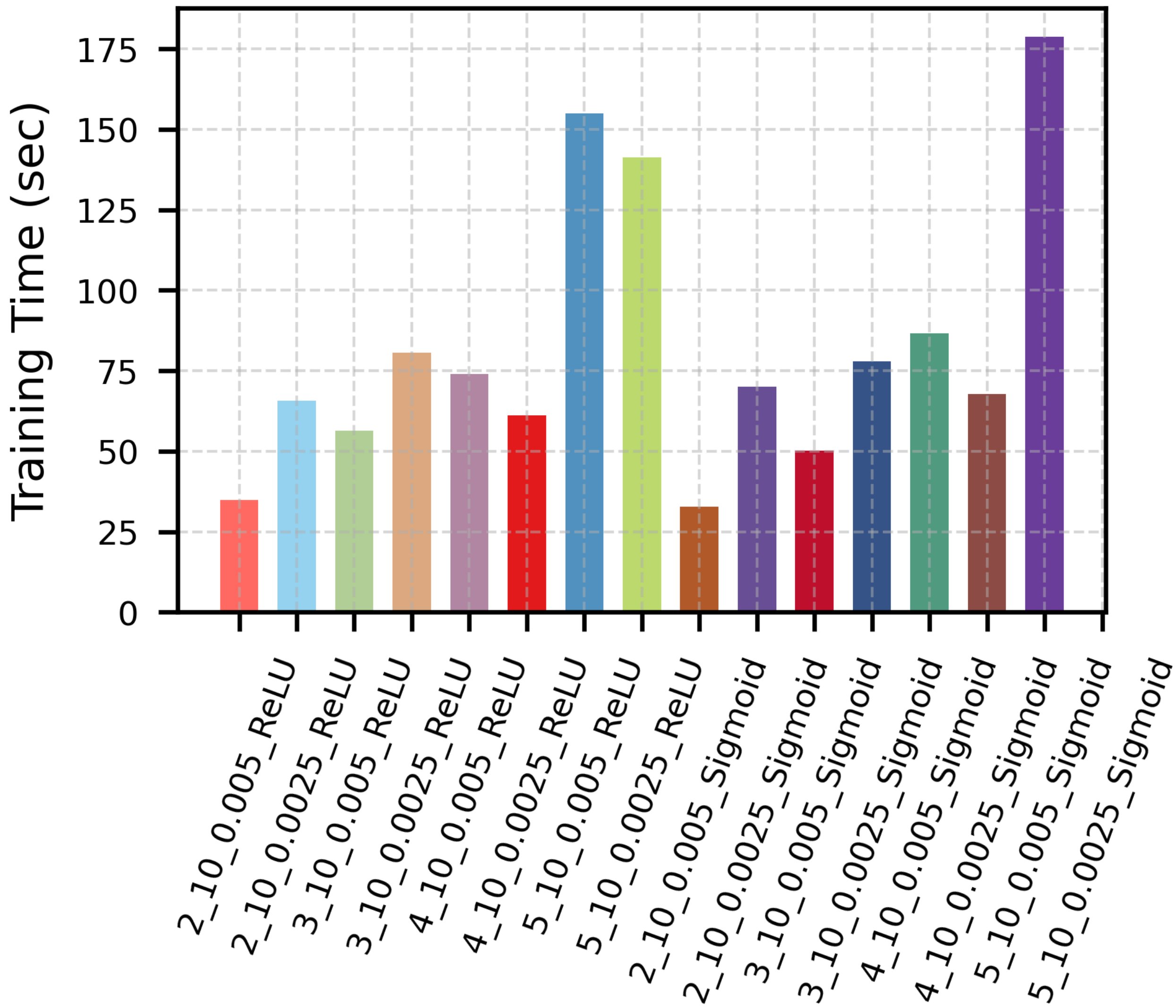
2 rows × 100005 columns

In [11…

```
compare_models(
    model_res_df_lr[(model_res_df_lr.Layers > 1)
                    #& (model_res_df_lr.Activation == 'ReLU')
                    & (model_res_df_lr.Learning_rate < .01)],
    fig_size=(3, 2),
    colors=colourWheel,
    dashes=dashesStyles,
    #time=False,
)
```

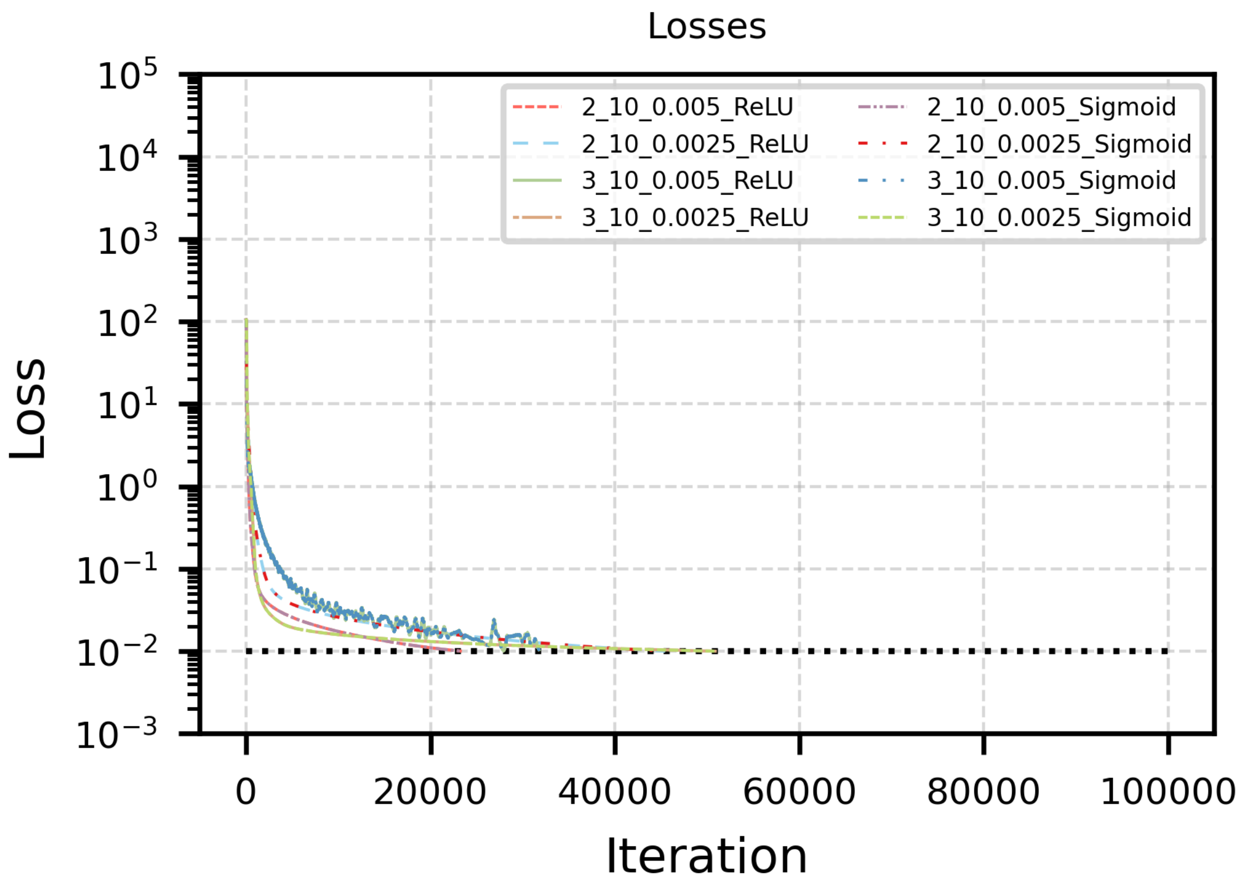


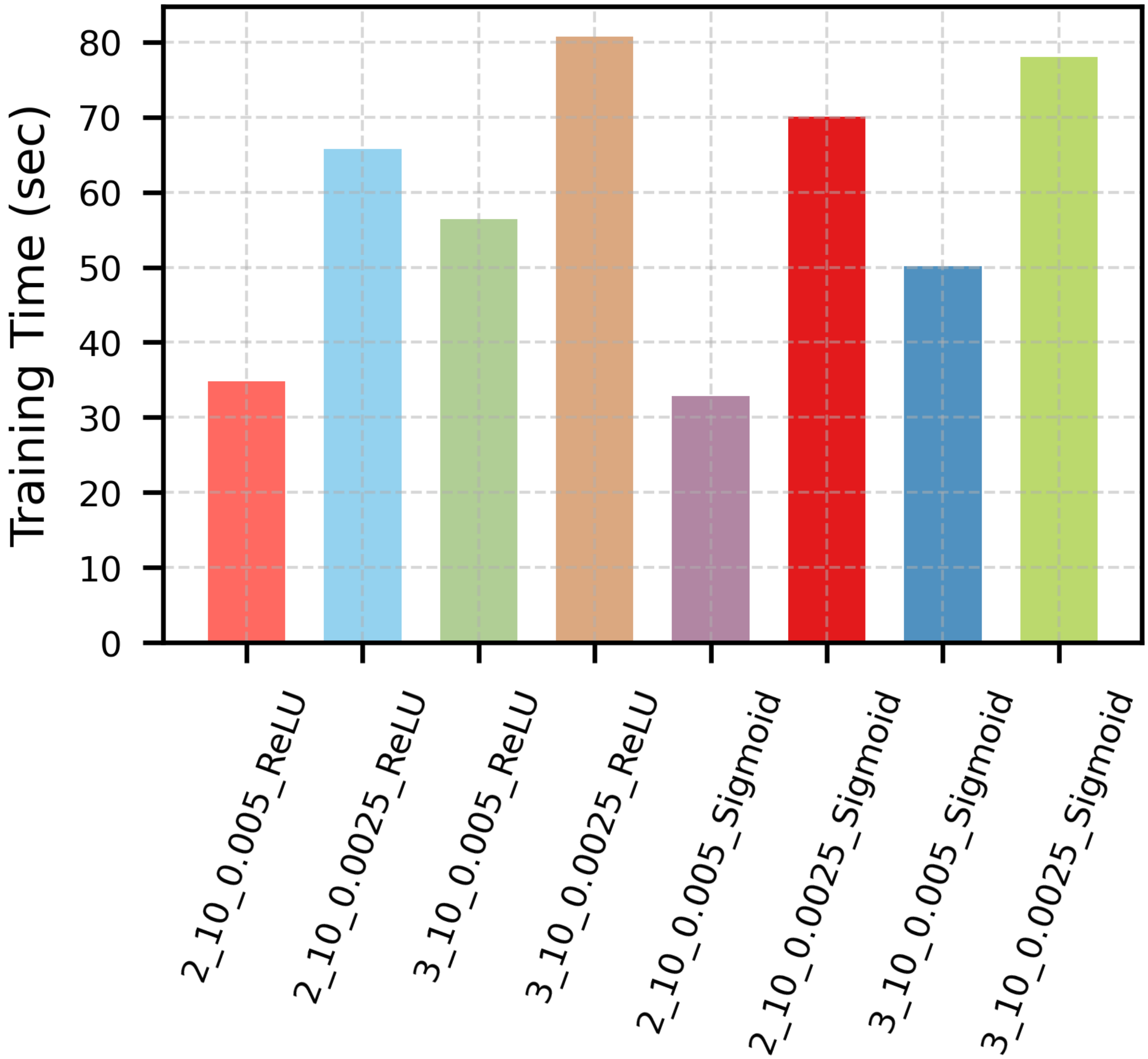
Losses
Loss
Iteration
2_10_0.005_ReLU
2_10_0.0025_ReLU
3_10_0.005_ReLU
3_10_0.0025_ReLU
4_10_0.005_ReLU
4_10_0.0025_ReLU
5_10_0.005_ReLU
5_10_0.0025_ReLU
2_10_0.005_Sigmoid
2_10_0.0025_Sigmoid
3_10_0.005_Sigmoid
3_10_0.0025_Sigmoid
4_10_0.005_Sigmoid
4_10_0.0025_Sigmoid
5_10_0.005_Sigmoid
5_10_0.0025_Sigmoid



In [11…

```
compare_models(
    model_res_df_lr[((model_res_df_lr.Layers == 2) | (model_res_df_lr.Layers == 3))
                    #& (model_res_df_lr.Activation == 'ReLU')
                    & (model_res_df_lr.Learning_rate < .01)],
    fig_size=(3, 2),
    colors=colourWheel,
    dashes=dashesStyles,
    #time=False,
)
```


Losses
Loss
Iteration
2_10_0.005_ReLU
2_10_0.0025_ReLU
3_10_0.005_ReLU
3_10_0.0025_ReLU
2_10_0.005_Sigmoid
2_10_0.0025_Sigmoid
3_10_0.005_Sigmoid
3_10_0.0025_Sigmoid
0
20000
40000
60000
80000
100000



Models with lr = 0.005

Define Result DF

In [11…

```
nodes_list = [5, 8, 10, 12, 15, 20]
max_iters = 100000
lr_list = [.005, ]
num_layers_list = [
    2,
    3,
    4,
]
activation_list = ['ReLU', 'Sigmoid']

# create the np.array and pd.DataFrame to save all the results
loss_array = np.empty((len(lr_list) * len(num_layers_list) * len(nodes_list) *
                       len(activation_list), max_iters))
loss_array[:] = np.NaN
display(loss_array.shape)

idxs = []
for activation in activation_list:
    for num_layers in num_layers_list:
        for nodes in nodes_list:
```

```python
            for lr in lr_list:
                idxs.append(
                    str(num_layers) + '_' + str(nodes) + '_' + str(lr) + '_' +
                    activation)

cols = ['Layers', 'Nodes', 'Learning_rate', 'Activation', 'Time', ] + \
    [i+1 for i in range(max_iters)]

model_res_df = pd.DataFrame(index=idxs, columns=cols)

for i in range(model_res_df.index.shape[0]):
    model_res_df.Layers.iloc[i] = num_layers_list[int(
        (i % (len(num_layers_list) * len(lr_list) * len(nodes_list))) /
        (len(lr_list) * len(nodes_list)))]
    #
    model_res_df.Nodes.iloc[i] = nodes_list[int(
        (i % (len(lr_list) * len(nodes_list))) / len(lr_list))]
    #
    model_res_df.Learning_rate.iloc[i] = lr_list[int(i % (len(lr_list)))]
    #
    model_res_df.Activation.iloc[i] = activation_list[int(
        i / (len(num_layers_list) * len(lr_list) * len(nodes_list)))]

model_res_df_all = model_res_df.copy()
display(model_res_df_all.iloc[:10, :10])
```

```
(36, 100000)
```

	Layers	Nodes	Learning_rate	Activation	Time	1	2	3	4	5
2_5_0.005_ReLU	2	5	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
2_8_0.005_ReLU	2	8	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
2_10_0.005_ReLU	2	10	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
2_12_0.005_ReLU	2	12	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
2_15_0.005_ReLU	2	15	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
2_20_0.005_ReLU	2	20	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
3_5_0.005_ReLU	3	5	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
3_8_0.005_ReLU	3	8	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
3_10_0.005_ReLU	3	10	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN
3_12_0.005_ReLU	3	12	0.005	ReLU	NaN	NaN	NaN	NaN	NaN	NaN

In [ ]:

```python

```

Running Models

In [ ]:

```python
model_res_df_all = run_parametric(
    model_res_df_all,
    max_iter=max_iters,
)
```

```
===== Training model 2_5_0.005_ReLU =====
iteration 0: loss=111.11900
iteration 10000: loss=0.57437
iteration 20000: loss=0.17069
iteration 30000: loss=0.08780
iteration 40000: loss=0.06164
iteration 50000: loss=0.04987
iteration 60000: loss=0.04158
iteration 70000: loss=0.03496
iteration 80000: loss=0.03163
iteration 90000: loss=0.02682
```

```
== Finished ==

===== Training model 2_8_0.005_ReLU =====
iteration 0: loss=109.45687
iteration 10000: loss=0.03568
iteration 20000: loss=0.01943
iteration 30000: loss=0.01696
iteration 40000: loss=0.01604
iteration 50000: loss=0.01602
iteration 60000: loss=0.01576
iteration 70000: loss=0.01590
iteration 80000: loss=0.01500
iteration 90000: loss=0.01494


== Finished ==

===== Training model 2_10_0.005_ReLU =====
iteration 0: loss=109.13280
iteration 10000: loss=0.01737
iteration 20000: loss=0.01101


== Finished ==

===== Training model 2_12_0.005_ReLU =====
iteration 0: loss=110.46222
iteration 10000: loss=0.07587
iteration 20000: loss=0.05400
iteration 30000: loss=0.04187
iteration 40000: loss=0.03302
iteration 50000: loss=0.02648
iteration 60000: loss=0.02259
iteration 70000: loss=0.02033
iteration 80000: loss=0.01729
iteration 90000: loss=0.01548


== Finished ==

===== Training model 2_15_0.005_ReLU =====
iteration 0: loss=108.95200
iteration 10000: loss=0.03504
iteration 20000: loss=0.01435


== Finished ==

===== Training model 2_20_0.005_ReLU =====
iteration 0: loss=107.68099
iteration 10000: loss=0.04956
iteration 20000: loss=0.03910
iteration 30000: loss=0.02491
iteration 40000: loss=0.02562
iteration 50000: loss=0.02261
iteration 60000: loss=0.01899
iteration 70000: loss=0.01634
iteration 80000: loss=0.01404
iteration 90000: loss=0.01260


== Finished ==

===== Training model 3_5_0.005_ReLU =====
iteration 0: loss=110.09887
iteration 10000: loss=0.09797
iteration 20000: loss=0.05297
iteration 30000: loss=0.04188
iteration 40000: loss=0.03203
iteration 50000: loss=0.02889
iteration 60000: loss=0.02369
iteration 70000: loss=0.02358
```

```
iteration 80000: loss=0.02574
iteration 90000: loss=0.02285


== Finished ==

===== Training model 3_8_0.005_ReLU =====
iteration 0: loss=106.74239
iteration 10000: loss=0.04604
iteration 20000: loss=0.02328
iteration 30000: loss=0.01857
iteration 40000: loss=0.01323
iteration 50000: loss=0.01286
iteration 60000: loss=0.01434
iteration 70000: loss=0.01177


== Finished ==

===== Training model 3_10_0.005_ReLU =====
iteration 0: loss=108.95901
iteration 10000: loss=0.02806
iteration 20000: loss=0.01652
iteration 30000: loss=0.01251


== Finished ==

===== Training model 3_12_0.005_ReLU =====
iteration 0: loss=108.77260
iteration 10000: loss=0.09192
iteration 20000: loss=0.06202
iteration 30000: loss=0.05196
iteration 40000: loss=0.03416
iteration 50000: loss=0.02857
iteration 60000: loss=0.02197
iteration 70000: loss=0.02142
iteration 80000: loss=0.01864
iteration 90000: loss=0.01416


== Finished ==

===== Training model 3_15_0.005_ReLU =====
iteration 0: loss=108.01099
iteration 10000: loss=0.02883
iteration 20000: loss=0.01363


== Finished ==

===== Training model 3_20_0.005_ReLU =====
iteration 0: loss=108.78493
iteration 10000: loss=0.01827


== Finished ==

===== Training model 4_5_0.005_ReLU =====
iteration 0: loss=108.99146
iteration 10000: loss=0.12405
iteration 20000: loss=0.05688
iteration 30000: loss=0.03080
iteration 40000: loss=0.01859
iteration 50000: loss=0.02570


== Finished ==

===== Training model 4_8_0.005_ReLU =====
iteration 0: loss=109.42415
iteration 10000: loss=0.11540
```

```
iteration 20000: loss=0.06037
iteration 30000: loss=0.04782
iteration 40000: loss=0.03999
iteration 50000: loss=0.03257
iteration 60000: loss=0.03414
iteration 70000: loss=0.02636
iteration 80000: loss=0.02445
iteration 90000: loss=0.02785


== Finished ==

===== Training model 4_10_0.005_ReLU =====
iteration 0: loss=108.23799
iteration 10000: loss=0.05917
iteration 20000: loss=0.02657
iteration 30000: loss=0.02176


== Finished ==

===== Training model 4_12_0.005_ReLU =====
iteration 0: loss=109.39426
iteration 10000: loss=0.06366
iteration 20000: loss=0.02373
iteration 30000: loss=0.01474
iteration 40000: loss=0.01144


== Finished ==

===== Training model 4_15_0.005_ReLU =====
iteration 0: loss=108.70986
iteration 10000: loss=0.04134
iteration 20000: loss=0.01483


== Finished ==

===== Training model 4_20_0.005_ReLU =====
iteration 0: loss=108.45959
iteration 10000: loss=0.05840
iteration 20000: loss=0.01531


== Finished ==

===== Training model 2_5_0.005_Sigmoid =====
iteration 0: loss=111.11900
iteration 10000: loss=0.57437
iteration 20000: loss=0.17069
iteration 30000: loss=0.08780
iteration 40000: loss=0.06164
iteration 50000: loss=0.04987
iteration 60000: loss=0.04158
iteration 70000: loss=0.03496
iteration 80000: loss=0.03163
iteration 90000: loss=0.02682


== Finished ==

===== Training model 2_8_0.005_Sigmoid =====
iteration 0: loss=109.45687
iteration 10000: loss=0.03568
iteration 20000: loss=0.01943
iteration 30000: loss=0.01696
iteration 40000: loss=0.01604
iteration 50000: loss=0.01602
iteration 60000: loss=0.01576
iteration 70000: loss=0.01590
iteration 80000: loss=0.01500
```

```
iteration 90000: loss=0.01494


== Finished ==

===== Training model 2_10_0.005_Sigmoid =====
iteration 0: loss=109.13280
iteration 10000: loss=0.01737
iteration 20000: loss=0.01101


== Finished ==

===== Training model 2_12_0.005_Sigmoid =====
iteration 0: loss=110.46222
iteration 10000: loss=0.07587
iteration 20000: loss=0.05400
iteration 30000: loss=0.04187
iteration 40000: loss=0.03302
iteration 50000: loss=0.02648
iteration 60000: loss=0.02259
iteration 70000: loss=0.02033
iteration 80000: loss=0.01729
iteration 90000: loss=0.01548


== Finished ==

===== Training model 2_15_0.005_Sigmoid =====
iteration 0: loss=108.95200
iteration 10000: loss=0.03504
iteration 20000: loss=0.01435


== Finished ==

===== Training model 2_20_0.005_Sigmoid =====
iteration 0: loss=107.68099
iteration 10000: loss=0.04956
iteration 20000: loss=0.03910
iteration 30000: loss=0.02491
iteration 40000: loss=0.02562
iteration 50000: loss=0.02261
iteration 60000: loss=0.01899
iteration 70000: loss=0.01634
iteration 80000: loss=0.01404
iteration 90000: loss=0.01260


== Finished ==

===== Training model 3_5_0.005_Sigmoid =====
iteration 0: loss=110.09887
iteration 10000: loss=0.09797
iteration 20000: loss=0.05297
iteration 30000: loss=0.04188
iteration 40000: loss=0.03203
iteration 50000: loss=0.02889
iteration 60000: loss=0.02369
iteration 70000: loss=0.02358
iteration 80000: loss=0.02574
iteration 90000: loss=0.02285


== Finished ==

===== Training model 3_8_0.005_Sigmoid =====
iteration 0: loss=106.74239
iteration 10000: loss=0.04604
iteration 20000: loss=0.02328
iteration 30000: loss=0.01857
iteration 40000: loss=0.01323
```

```
iteration 50000: loss=0.01286
iteration 60000: loss=0.01434
iteration 70000: loss=0.01177


== Finished ==
```

Results

In [ ]:
```
compare_models(
    model_res_df_ReLU.iloc[2:-1:5,:],
    fig_size=(3, 3),
    colors=colourWheel,
    dashes=dashesStyles,
)
```

In [ ]:
```
compare_models(
    model_res_df_ReLU.iloc[5:10],
    fig_size=(3, 3),
    colors=colourWheel,
    dashes=dashesStyles,
)
```

One Hidden Layer

In [ ]:
```
m, losses, errors, dt = \
    train_model(None,
                SimpleFeedForward_1(20),
                inputs=X_train,
                ground_truth=y_train,
                seed=0,
                learningRate=.001,
                MAX_iter=50000,
                show_results=False,)

visualizer(m, losses, errors, dt)
```

In [ ]:
```
torch.cuda.get_device_name(0)
```

Two Hidden Layers

In [ ]:
```
m, losses, errors, dt = \
    train_model(None,
                SimpleFeedForward_2(15),
                inputs=X_train,
                ground_truth=y_train,
                seed=0,
                learningRate=.02,
                MAX_iter=50000,
                show_results=False,)

visualizer(m, losses, errors, dt)
```

Three Hidden Layers

In [ ]:
```
m, losses, errors, dt = \
    train_model(None,
                SimpleFeedForward_3(10),
                inputs=X_train,
                ground_truth=y_train,
                seed=0,
                learningRate=.01,
                MAX_iter=50000,
```

```
                show_results=True,)

visualizer(m, losses, errors, dt)
```

In [ ]:

```
```