feedforward

October 11, 2023

1 Implementing a Neural Network

In this exercise we will develop a neural network with fully-connected layers to perform a regression task and a classification task. You need to implement two classes, `DenseLayer` and `Feedforward`, and the `train` function in `implementation.py`

[ ]:
```
# A bit of setup

import numpy as np
import torch
import matplotlib.pyplot as plt

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloa Can't pickle local`DenseLayer` and `Feedforward`.  object␣
 ↪'DenseLayer.__init__.<locals>.<lambda>'ding external modules
# see http://stackoverflow.com/questions/1907993/
 ↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
%autosave 60

# TODO: please install `torchvision` into the cs137 environment if you have not␣
 ↪done so.
#!conda install --yes -c pytorch torchvision
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
Autosaving every 60 seconds
```

1.1 1. Implement the two classes and one function

Question 1 (10 points): implement the two classes (`DenseLayer` and `Feedforward`) and the `train` function. The autograder can check the correcness of your implementation of the two classes

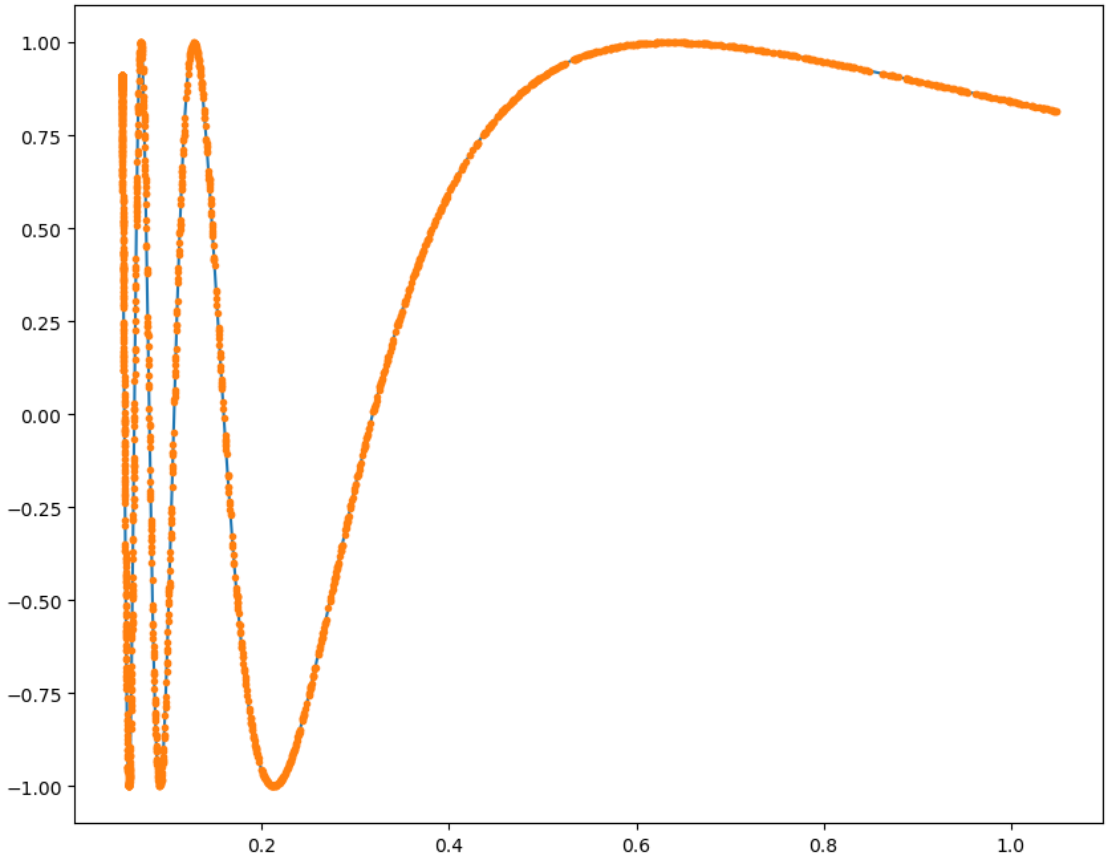
but NOT the train function. Each of the two classes worths 3 points, and the `train` function worth 4 points.

1.2 2. A toy regression problem

By this step, you should already have implemented all necessary classes and functions in `implementation.py`. In this task, we have a toy regression problem. Please check the data below and think about this question: is overfitting an issue for this problem? (This particular question has no points, but it might be useful for you to solve this problem).

[ ]:
```
def target_func(x):
    y = np.sin(1 / x)
    return y

# initialize training and validation sets.
x_train = np.power(np.random.random_sample([2000, 1]), 4) + 0.05
y_train = target_func(x_train)

x_val = np.power(np.random.random_sample([2000, 1]), 4) + 0.05
y_val = target_func(x_val)


# plot the function and data points
# Do you worry about overfitting problem?
sort_ind = np.argsort(x_train[:, 0])
plt.plot(x_train[sort_ind, 0], y_train[sort_ind, 0])
plt.plot(x_train[sort_ind, 0], y_train[sort_ind, 0], '.')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x28c402f50>]
```



1.2.1 Train the model

Question 2 (5 points): Now let's run your `train` function on this data. You will need to save your model to a file and submit it. We will test your saved model. You will earn * 2 points if your validation MSE is smaller than 0.1 * 4 points if your validation MSE is smaller than 0.05 * 5 points if your validation MSE is smaller than 0.01

[ ]:
```
# NOTE: neglect this cell if you are not using google colab
#from google.colab import drive
#import sys
#drive.mount('/content/drive')

# NOTE: you need to use your own path to add the implementation to the python␣
↪path
# so you can import functions from implementation.py
#sys.path.append('/content/drive/MyDrive/cs137assignments/assignment2')
```

[ ]:
```
from implementation import Feedforward, train

hidden_sizes = [32, 32]
```

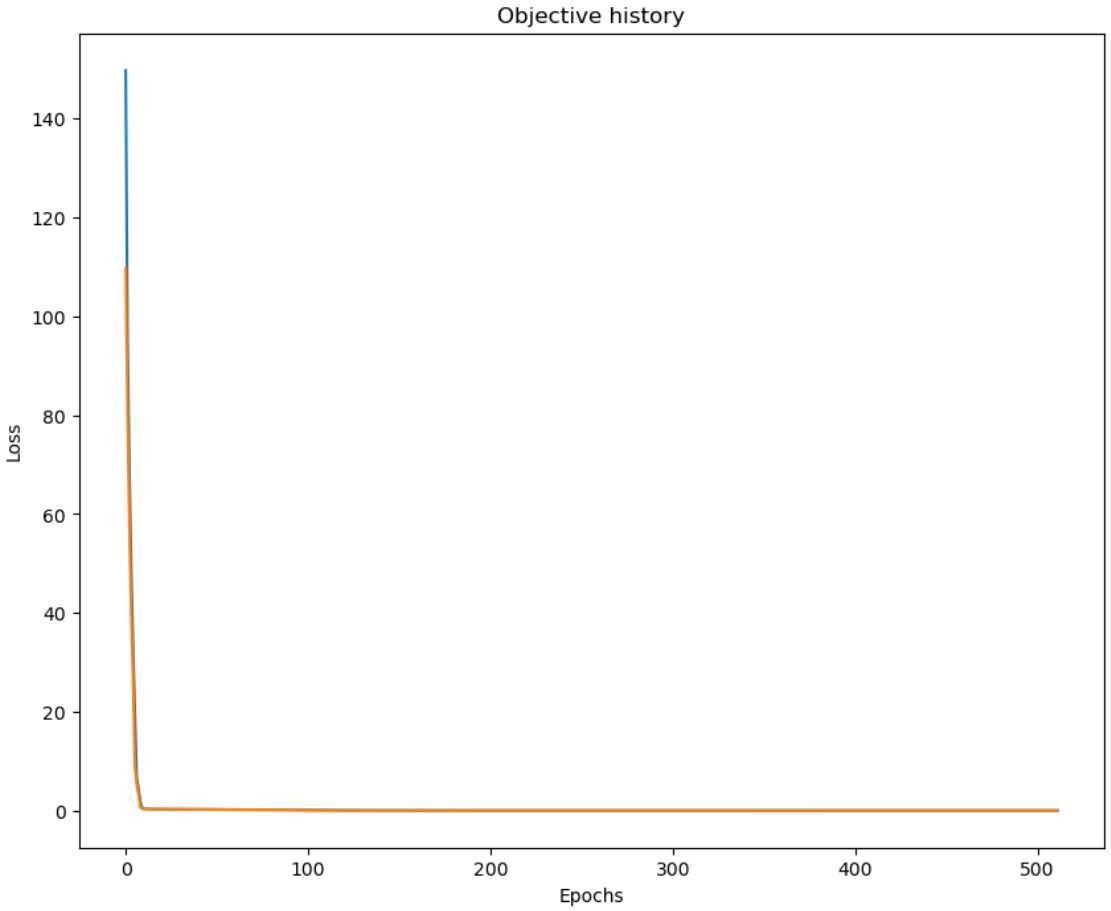
```
depth = len(hidden_sizes) + 1
reg_weight = 1e-5
num_train_epochs = 512
batch_size = 32

task_type = "regression"
input_size = 1
output_size = 1
learning_rate = 1e-3

model = Feedforward(task_type, input_size, depth, hidden_sizes, output_size,␣
 ↪reg_weight)

model,history = train(x_train, y_train, x_val, y_val, model,
                      num_train_epochs=num_train_epochs,
                      batch_size=batch_size,
                      lr=learning_rate,
                      task_type=task_type)
```

```
Training finished.
Final training loss: 0.0009
Final validation loss: 0.0001
Final validation MSE: 0.0001
```

1.2.2 Plot training trace

[ ]:
```
# Plot the loss function and train / validation accuracies

%matplotlib inline
plt.plot(history['loss'], label='train')
plt.plot(history['val_loss'], label='val')
plt.title('Objective history')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

[ ]:
```
Text(0, 0.5, 'Loss')
```

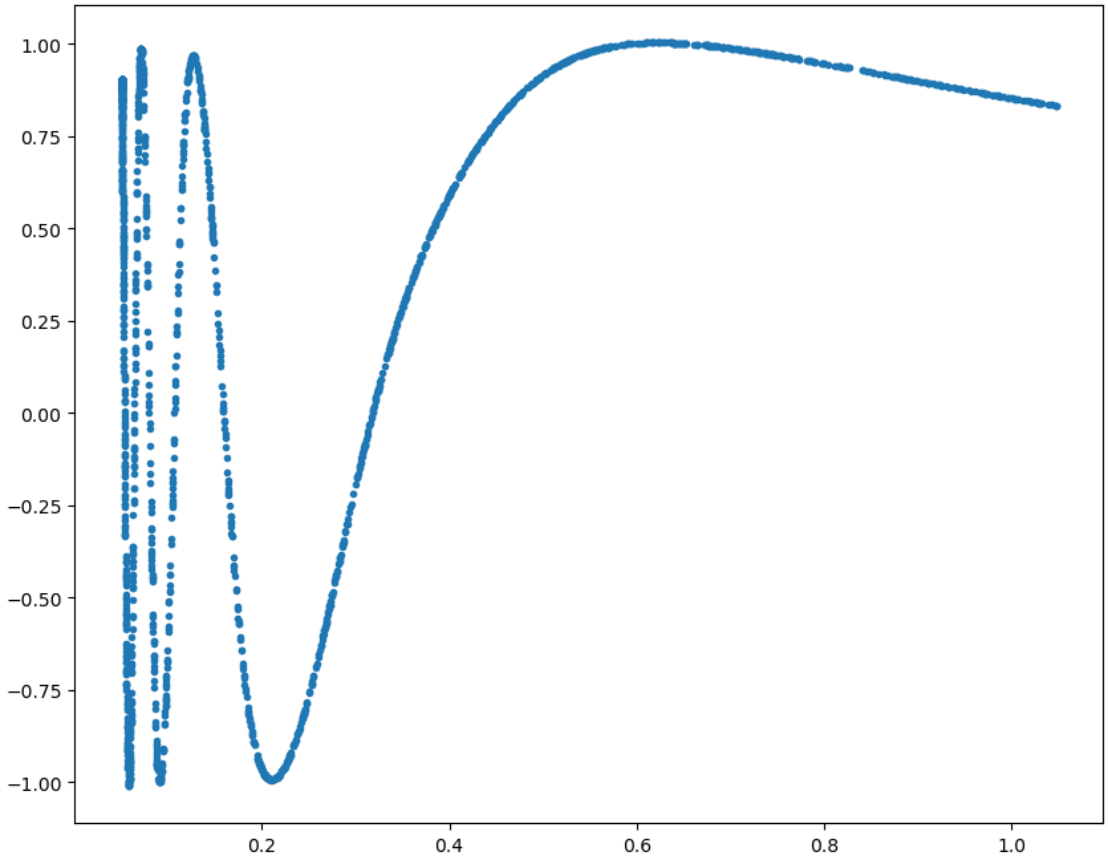


1.2.3 Save the model and plot the learned function

Please save your model as follows. Make sure the saved model can be loaded correctly with `torch.load_model('sin_inv_x.sav')`. *If your model cannot be loaded, you will get zero points for this problem.*

[ ]:
```
torch.save(model, 'sin_inv_x.sav')
model = torch.load('sin_inv_x.sav')

# need to set the model to evaluation mode
model.eval()

with torch.no_grad():
    y_pred = model(torch.tensor(x_val, dtype = torch.float32))

plt.plot(x_val[:, 0], y_pred.numpy()[:, 0], '.')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x28fc7cbd0>]
```



1.3 3. Classifying hand-written digits

This task is a classification problem on the famous MNIST dataset.

1.3.1 Load the data

[ ]:
```
from torchvision import datasets as dts
from torchvision.transforms import ToTensor

def transform(x):
    return ToTensor()(x).flatten()


traindt = dts.MNIST(
    root = 'data',
    train = True,
    transform = transform,
    download = True,
)
testdt = dts.MNIST(
```

```
    root = 'data',
    train = False,
    transform = transform
)

x_tr   = traindt.data.numpy().reshape(-1, 28 * 28)
x_test = testdt.data.numpy().reshape(-1, 28 * 28)
y_tr   = traindt.targets.numpy()
y_test = testdt.targets.numpy()
```

[ ]:
```
import torch.nn.functional as F
from sklearn.model_selection import train_test_split

# separate a validation set
x_train, x_val, y_train, y_val = train_test_split(x_tr, y_tr,
                                                  train_size=0.8,
                                                  stratify=y_tr)

print('Shape of training input: ', x_train.shape)
print('Shape of training labels: ', y_train.shape)
print('Shape of validation input: ', x_val.shape)
print('Shape of validation labels: ', y_val.shape)
print('Shape of test input: ', x_test.shape)
print('Shape of test labels: ', y_test.shape)
print('Number of channels: ', np.max(y_train) + 1)
```

```
Shape of training input:  (48000, 784)
Shape of training labels:  (48000,)
Shape of validation input:  (12000, 784)
Shape of validation labels:  (12000,)
Shape of test input:  (10000, 784)
Shape of test labels:  (10000,)
Number of channels:  10
```

1.3.2 Train a classification model

Question 3 (10 points): train a classification model for the MNIST dataset.

A neural network with your initial setting may not work straightforwardly, so you will need to diagnose the problems in the training procedure. One strategy for getting insight into what's wrong is to plot the loss function and the accuracies on the training and validation sets during optimization.

Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

Tuning. Tuning the hyperparameters and developing intuition for how they affect the final performance is a large part of using Neural Networks. This task gives you a chance to gain some

experience. Below, you should experiment with different values of the various hyperparameters, including

- the achitecture (depth, hidden layer sizes, and activation functions),
- optimization (optimization algorithm, learning rate, learning rate decay, batch size, and numer of training epochs)
- regularization (regularization weight, dropout).

Expected results. You goal in this exercise is to get a result as good as you can, with a fully-connected Neural Network. Feel free implement your own techniques (e.g. PCA to reduce dimensionality, or adding dropout, or adding features to the solver, etc.). You should be aim to achieve a classification accuracy of greater than 80% on the test results.

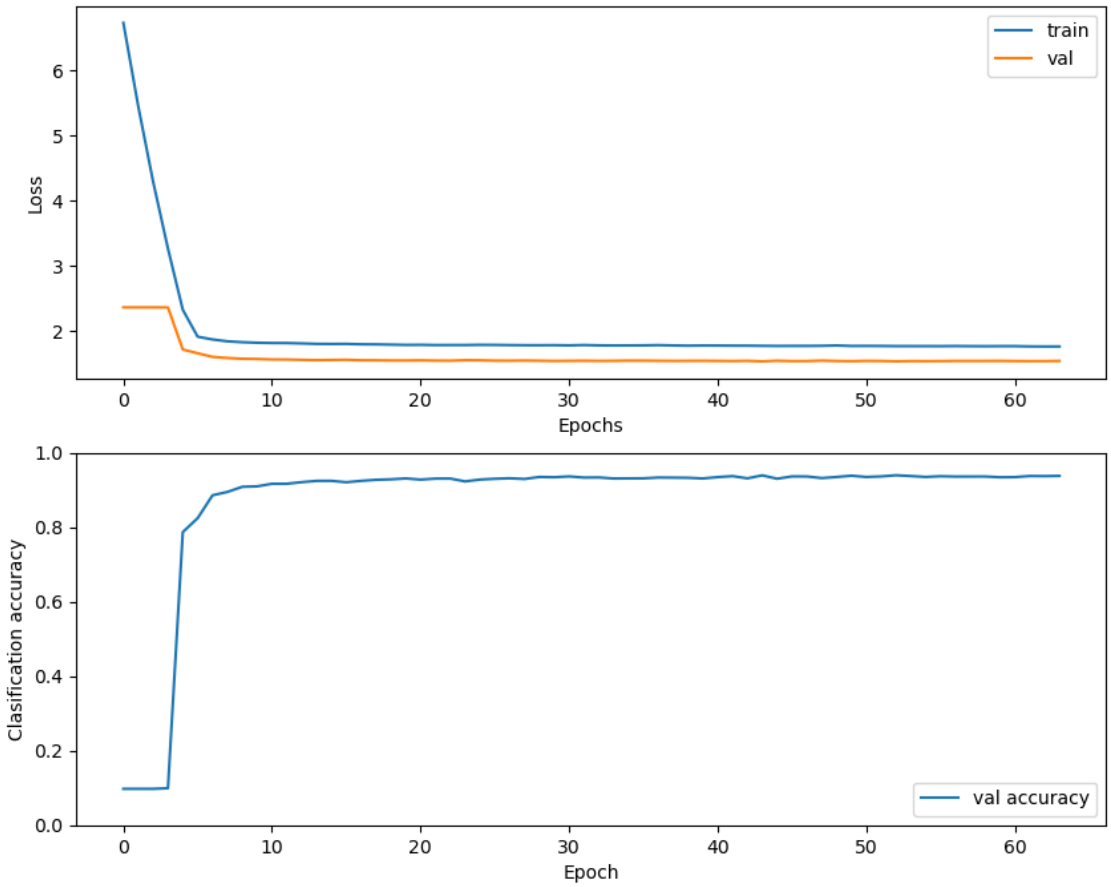
Grading. You will get * 5 points if your model has an accuracy over 0.75 * 8 points if your model has an accuracy over 0.85 * 10 points if your model has an accuracy over 0.92

[ ]:
```
from implementation import train

hidden_sizes = [768]
depth = len(hidden_sizes) + 1
num_train_epochs = 64
batch_size = 256

task_type = "classification"
input_size = x_train.shape[1]
output_size = np.max(y_train) + 1
reg_weight = 1e-2
learning_rate = 1e-3

model = Feedforward(task_type, input_size, depth, hidden_sizes, output_size,␣
 ↪reg_weight)

model, history = train(x_train,
                       y_train,
                       x_val,
                       y_val,
                       model,
                       num_train_epochs=num_train_epochs,
                       batch_size=batch_size,
                       lr=learning_rate,
                       task_type=task_type)
```

```
Training finished.
Final training loss: 1.7633
Final validation loss: 1.5415
Final validation accuracy: 0.9379
```

[ ]:
```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
```

```
plt.plot(history['loss'], label='train')
plt.plot(history['val_loss'], label='val')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(history['accuracy'], label='val accuracy')
#plt.plot(history.history['val_accuracy'], label='val')
plt.ylim(0.0, 1.0)
plt.xlabel('Epoch')
plt.ylabel('Clasification accuracy')
plt.legend()
plt.show()
```



[ ]:
```
from vis_utils import visualize_grid

with torch.no_grad():
    W1 = model.layers[0].W.numpy()
```

```
W1 = W1.transpose()
print(W1.shape)
W1 = np.reshape(W1, [W1.shape[0], 28, 28])

plt.imshow(visualize_grid(W1))
```

```
(768, 784)
```

[ ]:
```
<matplotlib.image.AxesImage at 0x2b3f98050>
```

Save the model and test it. Finally, save the model and reload it to make sure the model can be saved correctly. Test the model on the test set and show the accuracy.

[ ]:
```
torch.save(model, 'mnist_cls.sav')
model = torch.load('mnist_cls.sav')

model.eval()

print(x_test.shape)
with torch.no_grad():
    y_pred = model(torch.tensor(x_test.astype(np.float32)))
    acc = np.mean(y_test == np.argmax(y_pred.numpy(), axis=1))
print('The test accuracy is ', acc)
```

```
(10000, 784)
The test accuracy is  0.9397
```

1.4 Question (5 points): what have you learned from this assignment?

Please summarize your model tuning experience into three points below. If possible, please elaborate after your summarization.

Answer:

- Typically, the learning rate has the largest impact on the model's performance, but getting to the right value can be difficult as sometimes the optimal range can be very narrow (in this case 1e-3 worked well, but 1e-2 or 1e-4).
- In my experience tuning the regularization weight impacted the output much less, and had a wider optimal range.
- The number and size of the layers did impact the model's performance a lot, but was a lot harder to iterate through to find the right combination, and took a lot of trial and error.