Lab 2 Report:

Iris Classification with Regression

Name:

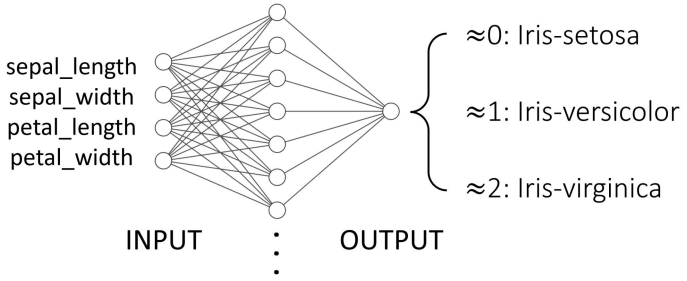
In [ ]:
```
# Import neccessary packages

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import torch
```

In [ ]:
```
from IPython.display import Image # For displaying images in colab jupyter cell
```

In [ ]:
```
Image('lab2_exercise1.png', width = 1000)
```

Out[ ]:

Exercise 1: Iris Classification using Regression



In this exercise, you will train a neural network with a single hidden layer consisting of linear neurons to perform regression on iris datasets.

Your goal is to achieve a training accuracy of >90% under 50 epochs.

You are free to experiment with different data normalization methods, size of the hidden layer, learning rate and epochs.

You can round the output value to an integer (e.g. 0.34 -> 0, 1.78 -> 2) to compute the model accuracy.

Demonstrate the performance of your model via plotting the training loss and printing out the training accuracy.

Prepare Data

In [ ]:
```
from sklearn.datasets import load_iris

# iris dataset is available from scikit-learn package
iris = load_iris()

# Load the X (features) and y (targets) for training
```

```
x_train = iris['data']
y_train = iris['target']

# Load the name labels for features and targets
feature_names = iris['feature_names']
names = iris['target_names']

# Feel free to perform additional data processing here (e.g. standard scaling)
def scale_data(arr):
    num = arr - np.mean(arr)
    den = np.std(arr)
    scaled_data = num/den

    return scaled_data

x_train = scale_data(x_train)
```

In [ ]:
```
# Print the first 10 training samples for both features and targets

print(x_train[:10, :], y_train[:10])
```

```
[[ 0.82858665  0.01798522 -1.04592915 -1.65388022]
 [ 0.72726147 -0.23532773 -1.04592915 -1.65388022]
 [ 0.62593629 -0.13400255 -1.09659174 -1.65388022]
 [ 0.5752737  -0.18466514 -0.99526657 -1.65388022]
 [ 0.77792406  0.06864781 -1.04592915 -1.65388022]
 [ 0.98057441  0.22063558 -0.89394139 -1.55255505]
 [ 0.5752737  -0.03267737 -1.04592915 -1.60321764]
 [ 0.77792406 -0.03267737 -0.99526657 -1.65388022]
 [ 0.47394852 -0.28599032 -1.04592915 -1.65388022]
 [ 0.72726147 -0.18466514 -0.99526657 -1.70454281]] [0 0 0 0 0 0 0 0 0 0]
```

In [ ]:
```
# Print the dimensions of features and targets

print(x_train.shape, y_train.shape)
```

```
(150, 4) (150,)
```

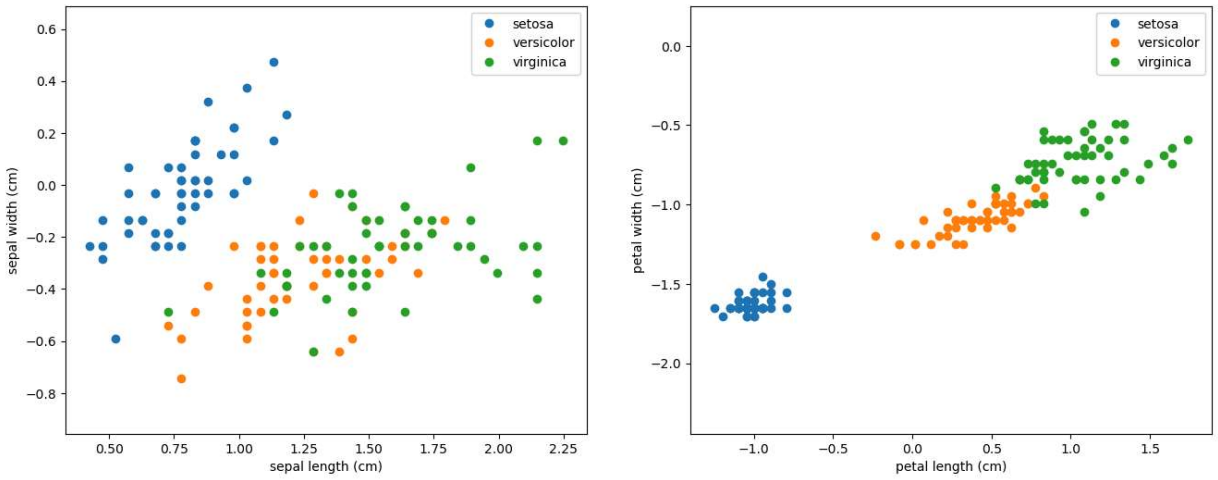
In [ ]:
```
# feature_names contains name for each column in x_train
# For targets, 0 -> setosa, 1 -> versicolor, 2 -> virginica

print(feature_names, names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
```

In [ ]:
```
# We can visualize the dataset before training

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# enumerate picks up both the index (0, 1, 2) and the element ('setosa', 'versicolo
# loop 1: target = 0, target_name = 'setosa'
# loop 2: target = 1, target_name = 'versicolor' etc

for target, target_name in enumerate(names):

    # Subset the rows of x_train that fall into each flower category using boolean
```

```
    X_plot = x_train[y_train == target]

    # Plot the sepal length versus sepal width for the flower category
    ax1.plot(X_plot[:, 0], X_plot[:, 1], linestyle='none', marker='o', label=target

# Label the plot
ax1.set_xlabel(feature_names[0])
ax1.set_ylabel(feature_names[1])
ax1.axis('equal')
ax1.legend()

# Repeat the above process but with petal length versus petal width
for target, target_name in enumerate(names):

    X_plot = x_train[y_train == target]

    ax2.plot(X_plot[:, 2], X_plot[:, 3], linestyle='none', marker='o', label=target

ax2.set_xlabel(feature_names[2])
ax2.set_ylabel(feature_names[3])
ax2.axis('equal')
ax2.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x2ba24637190>



Define Model

In [ ]:
```
class irisClassification(torch.nn.Module): # set of rules to differentiate between

    # Create layers and connect them to create a model (i.e. just stating they exis
    def __init__(self, input_dim, hidden_layer, output_dim):

        super(irisClassification, self).__init__()

        # Attributes (to get an attribute: model.layer1)
        self.layer1 = torch.nn.Linear(input_dim, hidden_layer) # saying it's a line
        self.layer2 = torch.nn.Linear(hidden_layer, output_dim)

    # Then define each step from beginning to end - i.e. tell the layer what to do
    def forward(self, x):
```

```
        # first layer
        x = self.layer1(x)
        # x = torch.nn.functional.relu(x) # applies activation function

        # second layer
        x = self.layer2(x)
        # out = torch.nn.functional.relu(x)

        return x
```

Define Hyperparameters

In [ ]:
```
# Hyperparameters: the optimizer, loss function, learning rate, epochs (# of iterat

# the optimizer minimizes the loss throughout the epochs by changing the connection

model = irisClassification(input_dim = 4, hidden_layer = 15, output_dim = 1)

learning_rate = 0.101 # aka the size of the steps descending the slope towards a mi
epochs  = 49 # needs to be under 50 for this assignment

# Mean Squared Error Loss function
loss_func = torch.nn.MSELoss()

# optimizing with gradient descent
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate)

if torch.cuda.is_available():
    model.cuda()
```

Identify Tracked Values

In [ ]:
```
# follow models performance over each epoch. Identify a metric and track it over ep

train_loss_list = []
```

Train Model

In [ ]:
```
# data = pd.DataFrame(data=iris.data,
#                     columns= iris.feature_names)
# data
```

In [ ]:
```
# # target is the type of iris each flower belongs to
# print(iris.target_names, iris['target'], sep = '\n')
```

In [ ]:
```
x_train = torch.from_numpy(x_train).float()
y_train = torch.from_numpy(y_train).float()

# loss is how far away the model's prediction was from each training data point
# the goal is to get as close to iris['target'] as possible (loss = 0 --> 100% accu
```

```
for epoch in range(epochs):
    optimizer.zero_grad()
    outputs = model(x_train)
    # print(outputs.shape, targets.shape)
    loss = loss_func(outputs, y_train.unsqueeze(dim=1))
    train_loss_list.append(loss.item())
    loss.backward()
    optimizer.step()
    print('epoch {}, loss {}'.format(epoch, loss.item()))
```

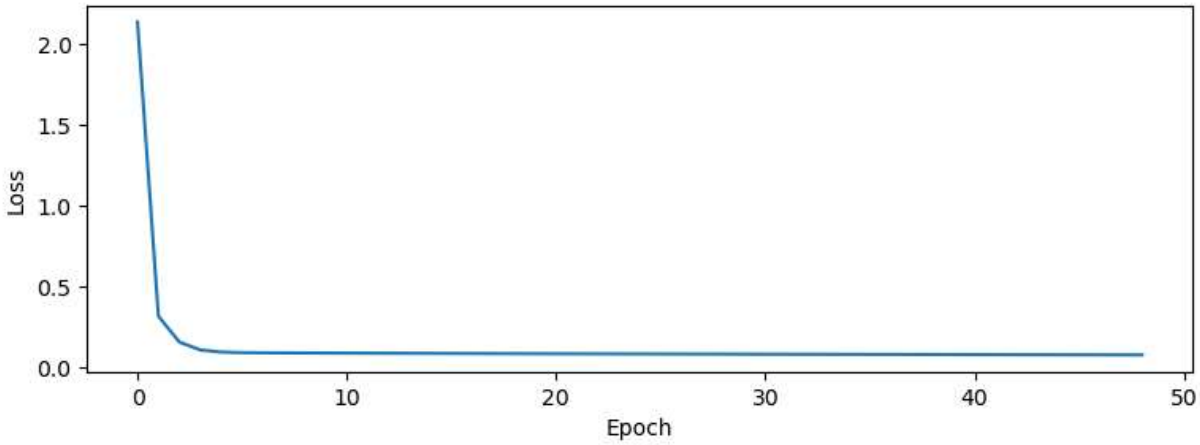
```
epoch 0, loss 2.138712167739868
epoch 1, loss 0.3161795437335968
epoch 2, loss 0.15635685622692108
epoch 3, loss 0.10726194083690643
epoch 4, loss 0.09398657083511353
epoch 5, loss 0.09052824974060059
epoch 6, loss 0.08944196999073029
epoch 7, loss 0.08889169245958328
epoch 8, loss 0.08846224099397659
epoch 9, loss 0.08806367218494415
epoch 10, loss 0.0876767560839653
epoch 11, loss 0.08729729056358337
epoch 12, loss 0.08692426979541779
epoch 13, loss 0.08655733615159988
epoch 14, loss 0.08619628846645355
epoch 15, loss 0.08584092557430267
epoch 16, loss 0.0854911133646965
epoch 17, loss 0.08514667302370071
epoch 18, loss 0.08480748534202576
epoch 19, loss 0.0844733938574791
epoch 20, loss 0.08414427936077118
epoch 21, loss 0.08381997793912888
epoch 22, loss 0.08350037783384323
epoch 23, loss 0.08318536728620529
epoch 24, loss 0.08287481963634491
epoch 25, loss 0.08256861567497253
epoch 26, loss 0.0822666585445404
epoch 27, loss 0.08196882158517838
epoch 28, loss 0.08167503774166107
epoch 29, loss 0.08138518035411835
epoch 30, loss 0.08109914511442184
epoch 31, loss 0.08081687986850739
epoch 32, loss 0.08053825795650482
epoch 33, loss 0.08026320487260818
epoch 34, loss 0.07999163120985031
epoch 35, loss 0.07972347736358643
epoch 36, loss 0.07945864647626877
epoch 37, loss 0.07919708639383316
epoch 38, loss 0.07893868535757065
epoch 39, loss 0.07868340611457825
epoch 40, loss 0.0784311518073082
epoch 41, loss 0.07818190008401871
epoch 42, loss 0.07793554663658142
epoch 43, loss 0.07769204676151276
epoch 44, loss 0.07745133340358734
epoch 45, loss 0.077213354408741
epoch 46, loss 0.07697803527116776
epoch 47, loss 0.07674534618854523
epoch 48, loss 0.07651522010564804
```

Visualize and Evaluate Model

In [ ]:
```
# Plot your training loss throughout the training
# Include proper x and y labels for the plot
```

```
plt.figure(figsize=(9, 3))

plt.plot(train_loss_list)
plt.xlabel('Epoch', fontsize = 10)
plt.ylabel('Loss', fontsize = 10)
```

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



In [ ]:
```
for point in zip(np.round(train_loss_list).tolist(), y_train):
    print(f"Prdctn: {point[0]}, Target: {point[1]}")
```

```
Prdctn: 2.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
Prdctn: 0.0, Target: 0.0
```

In [ ]:

```
# Confirm that your model's training accuracy is >90%

# Compare your model predictions with targets (y_train) to compute the training acc
# Training accuracy = (# of correct predictions) / (total # of training samples)
# You can round the model predictions to integer (e.g. 0.34 -> 0, 1.78 -> 2)
```

```
compare = zip(np.round(train_loss_list).tolist(), iris['target'])
counter = 0

with torch.no_grad():

    for point in compare:
        if point[0] == point[1]:
            counter = counter + 1

    accuracy = (counter / epochs)*100
    print(f"{counter} correct predictions of {epochs} epochs at {learning_rate} lea
```

```
48 correct predictions of 49 epochs at 0.101 learning rate.
Model accuracy: 97.95918367346938%.
```

In [ ]: