Lab 2 Report:

Iris Classification with Regression

Name: Forest Tschirhart

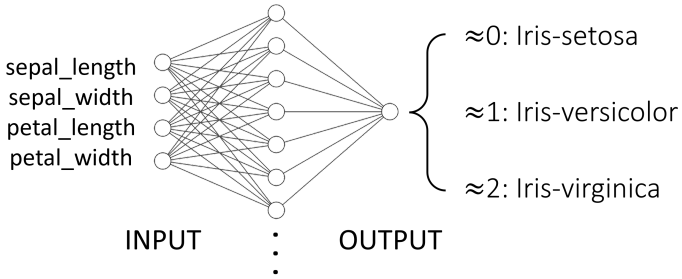
In [2]:
```
# Import neccessary packages

%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

import torch
```

In [3]:
```
from IPython.display import Image # For displaying images in colab jupyter c
```

In [4]:
```
Image('lab2_exercise1.png', width = 1000)
```

Out[4]:

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. 42

Prepare Data

In [5]:
```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# 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 scalir
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

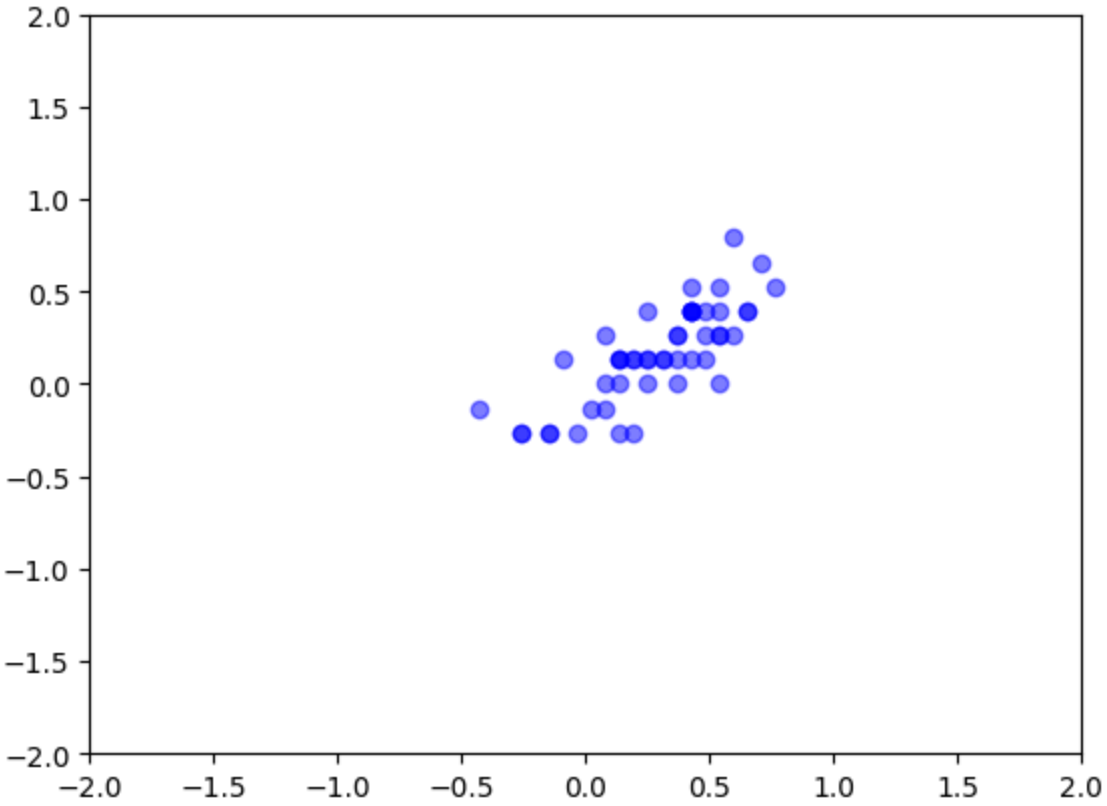
In [6]:
```
# Print the first 10 training samples for both features and targets

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

```
[[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
 [-1.14301691 -0.13197948 -1.34022653 -1.3154443 ]
 [-1.38535265  0.32841405 -1.39706395 -1.3154443 ]
 [-1.50652052  0.09821729 -1.2833891  -1.3154443 ]
 [-1.02184904  1.24920112 -1.34022653 -1.3154443 ]
 [-0.53717756  1.93979142 -1.16971425 -1.05217993]
 [-1.50652052  0.78880759 -1.34022653 -1.18381211]
 [-1.02184904  0.78880759 -1.2833891  -1.3154443 ]
 [-1.74885626 -0.36217625 -1.34022653 -1.3154443 ]
 [-1.14301691  0.09821729 -1.2833891  -1.44707648]] [0 0 0 0 0 0 0 0 0 0]
```

In [7]:
```
# Print the dimensions of features and targets

print(X_train.shape, y_train.shape)
plt.plot(X_train[50:100,2], X_train[50:100,3], 'o', color='blue', alpha=0.5)
plt.xlim(-2,2)
plt.ylim(-2,2)
```

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

Out[7]:
```
(-2.0, 2.0)
```



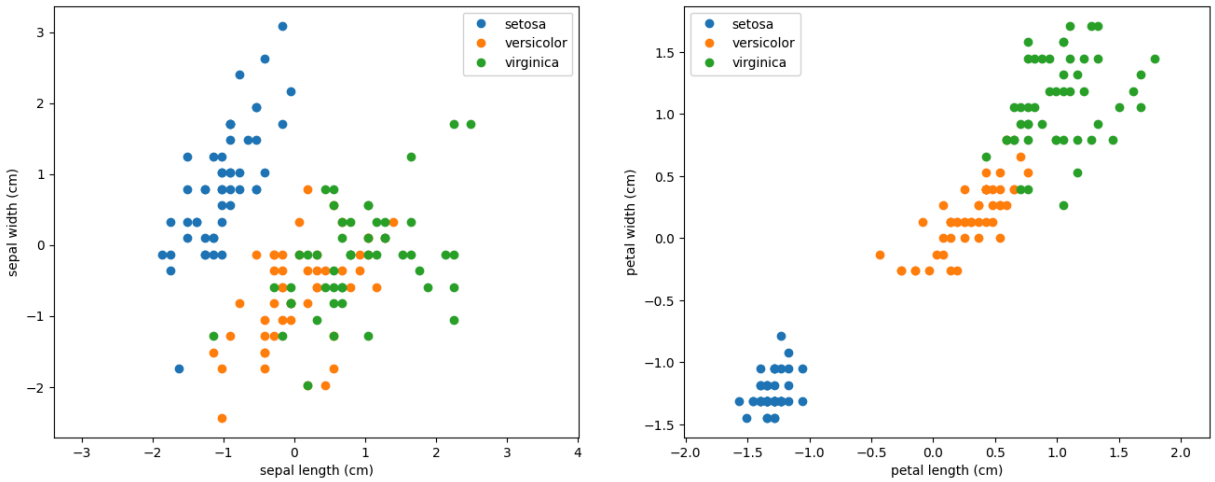
In [8]:
```
# feature_names contains name for each column in X_train
# For targets, 0 -> setosa, 1 -> versicolor, 2 -> virginica

print(feature_names, names)
```

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

In [9]:
```
# 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', 've
# 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 b
    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

# 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

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

Out[9]: <matplotlib.legend.Legend at 0x75e5e6070bc0>



Define Model

In [10]:
```
class irisClassification(torch.nn.Module):

    def __init__(self, input_dim, hidden_dim, output_dim, activation='sigmoi

        super(irisClassification, self).__init__()

        self.activation = activation
        self.layer1 = torch.nn.Linear(input_dim, hidden_dim)
        self.layer2 = torch.nn.Linear(hidden_dim, output_dim)
        self.sigmoid = torch.nn.Sigmoid()
        self.relu = torch.nn.ReLU()



    def forward(self, x):

        if self.activation == 'sigmoid':
            out = self.sigmoid(self.layer1(x))
            out = self.layer2(out)

        elif self.activation == 'relu':
            out = self.relu(self.layer1(x))
            out = self.layer2(out)
```

```
        return out
```

Define Hyperparameters

In [11]:
```
model = irisClassification(input_dim = 4, hidden_dim = 10, output_dim = 1, a

learning_rate = 0.09
epochs  = 50

# We will use gradient descent for our optimizer and Mean Squared Error Loss
loss_func = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate)
```

Identify Tracked Values

In [12]:
```
# follow models performance over each epoch. Identify a metric and track it
loss_tracker = []
```

Train Model

In [13]:
```
X_tens = torch.tensor(X_train, dtype=torch.float32)
y_tens = torch.tensor(y_train, dtype=torch.float32)
for epoch in range(epochs):

    optimizer.zero_grad()

    y_predict = model(X_tens).squeeze()

    loss = loss_func(y_predict, y_tens)
    loss_tracker.append(loss.item())


    loss.backward()
    optimizer.step()



    print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss.item():.4f}')
```

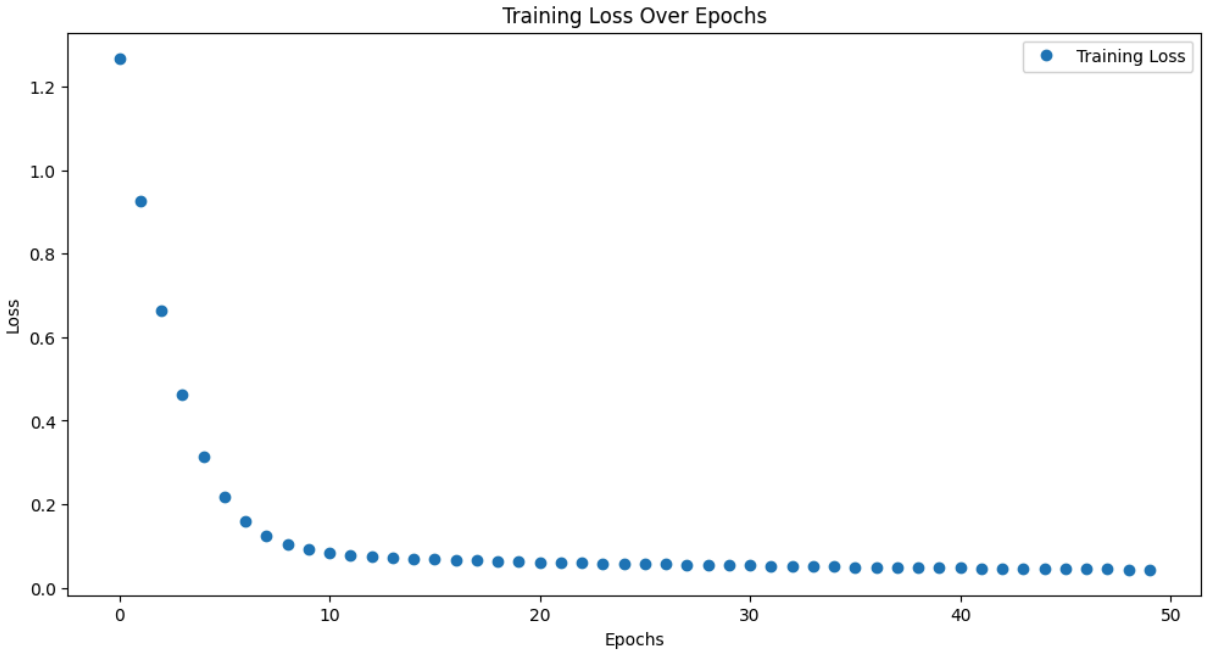
```
Epoch 1/50, Loss: 1.2684
Epoch 2/50, Loss: 0.9263
Epoch 3/50, Loss: 0.6649
Epoch 4/50, Loss: 0.4623
Epoch 5/50, Loss: 0.3153
Epoch 6/50, Loss: 0.2182
Epoch 7/50, Loss: 0.1585
Epoch 8/50, Loss: 0.1236
Epoch 9/50, Loss: 0.1033
Epoch 10/50, Loss: 0.0912
Epoch 11/50, Loss: 0.0834
Epoch 12/50, Loss: 0.0783
Epoch 13/50, Loss: 0.0747
Epoch 14/50, Loss: 0.0720
Epoch 15/50, Loss: 0.0698
Epoch 16/50, Loss: 0.0680
Epoch 17/50, Loss: 0.0665
Epoch 18/50, Loss: 0.0651
Epoch 19/50, Loss: 0.0638
Epoch 20/50, Loss: 0.0626
Epoch 21/50, Loss: 0.0615
Epoch 22/50, Loss: 0.0605
Epoch 23/50, Loss: 0.0595
Epoch 24/50, Loss: 0.0586
Epoch 25/50, Loss: 0.0577
Epoch 26/50, Loss: 0.0568
Epoch 27/50, Loss: 0.0560
Epoch 28/50, Loss: 0.0553
Epoch 29/50, Loss: 0.0545
Epoch 30/50, Loss: 0.0538
Epoch 31/50, Loss: 0.0531
Epoch 32/50, Loss: 0.0524
Epoch 33/50, Loss: 0.0518
Epoch 34/50, Loss: 0.0512
Epoch 35/50, Loss: 0.0506
Epoch 36/50, Loss: 0.0500
Epoch 37/50, Loss: 0.0494
Epoch 38/50, Loss: 0.0489
Epoch 39/50, Loss: 0.0483
Epoch 40/50, Loss: 0.0478
Epoch 41/50, Loss: 0.0473
Epoch 42/50, Loss: 0.0468
Epoch 43/50, Loss: 0.0464
Epoch 44/50, Loss: 0.0459
Epoch 45/50, Loss: 0.0455
Epoch 46/50, Loss: 0.0451
Epoch 47/50, Loss: 0.0447
Epoch 48/50, Loss: 0.0443
Epoch 49/50, Loss: 0.0439
Epoch 50/50, Loss: 0.0435
```

Visualize and Evaluate Model

In [14]:
```
# Plot your training loss throughout the training
# Include proper x and y labels for the plot
```

```
plt.figure(figsize=(12, 6))
plt.plot(loss_tracker, 'o',label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss Over Epochs')
plt.legend()
```

Out[14]: <matplotlib.legend.Legend at 0x75e5de35bbf0>



In [15]:
```
# Confirm that your model's training accuracy is >90%

with torch.no_grad():

    # Compare your model predictions with targets (y_train) to compute the t
    model_predictions = model(X_tens).numpy()
    #print(model_predictions)
    int_model_predictions = model_predictions.round().squeeze().astype(int)
    #print(int_model_predictions)
    #print(y_train)
    correct_predictions = int_model_predictions == y_train
    #print(correct_predictions)
    accuracy = correct_predictions.sum() / len(correct_predictions)
    print(f'Training Accuracy: {accuracy:.2f}')


# Training accuracy = (# of correct predictions) / (total # of training samp
# You can round the model predictions to integer (e.g. 0.34 -> 0, 1.78 -> 2)

# YOUR CODE HERE
```

```
Training Accuracy: 0.98
```

In [ ]: