Jonathon Nguyen

ID: 801093003

Homework 2

In [1]:

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from torchvision import datasets
import time
```

In [2]:

```
# Find the normalized of the input tensor.
def normalized_data(tensor):
    mean = torch.mean(tensor)
    std = torch.std(tensor)
    new_tensor = (tensor - mean) / std

    return new_tensor

def training_loop(epochs, optimizer, model, loss_fn, training_vars, validation_vars,
                  training_prices, validation_prices):
    training_losses = []
    val_losses = []
    for epoch in range(1, epochs + 1):
        # Validation model and loss
        loss_val_values = 0
        with torch.no_grad():
            prices_p_val = torch.squeeze(model(validation_vars))
            loss_val = loss_fn(prices_p_val, validation_prices)

        val_losses.append(float(loss_val))

        # Training model and loss
        prices_p_train = torch.squeeze(model(training_vars))
        loss_train = loss_fn(prices_p_train, training_prices)

        # Set the new params.
        optimizer.zero_grad()
        loss_train.backward()
        optimizer.step()

        training_losses.append(float(loss_train))

        # Print out the losses every 10 epoch
        if epoch % 10 == 0 or epoch == 1:
            print(f'Epoch {epoch}: Training Loss: {float(loss_train)}, Validation Loss: {f

    return training_losses, val_losses
```

In [3]:

```
NUM_EPOCHS = 300

# Read the data from the provided CSV files
housing = pd.DataFrame(pd.read_csv("Housing.csv"))

# Split the data into the input vars and the prices.
names_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
prices = housing['price']
```

```
# Find the length of the column.
num_samples = len(prices)
num_val = int(0.2 * num_samples)

# Generate the random indices with 80% training and 20% Validation
random_indices = torch.randperm(num_samples)
training_indices = random_indices[:-num_val]
validation_indices = random_indices[-num_val:]

input_vars = torch.tensor(housing[names_vars].values).float()
training_tensor = normalized_data(input_vars[training_indices])
validation_tensor = normalized_data(input_vars[validation_indices])

# Convert the prices to a tensor.
prices = normalized_data(torch.tensor(prices.values).float())
price_training = prices[training_indices]
price_validation = prices[validation_indices]
```

In [4]:

```
# Model with one hidden layer of 8
model = nn.Sequential(
        nn.Linear(5, 8),
        nn.Tanh(),
        nn.Linear(8, 1))

optimizer = optim.SGD(model.parameters(), lr=.01)
loss_function = nn.MSELoss()
```

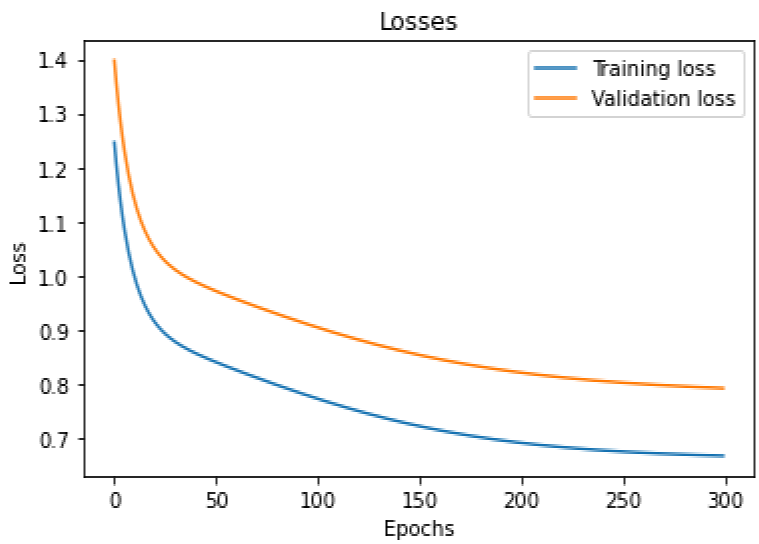
In [5]:

```
train_loss, val_loss = training_loop(NUM_EPOCHS, optimizer, model, loss_function, training
                                     validation_tensor, price_training, price_validation)
```

```
Epoch 1: Training Loss: 1.2460602521896362, Validation Loss: 1.3977521657943726
Epoch 10: Training Loss: 1.0101455450057983, Validation Loss: 1.151967167854309
Epoch 20: Training Loss: 0.9181671142578125, Validation Loss: 1.054456353187561
Epoch 30: Training Loss: 0.8796026706695557, Validation Loss: 1.0130778551101685
Epoch 40: Training Loss: 0.857615172863006, Validation Loss: 0.989709198474884
Epoch 50: Training Loss: 0.8409616947174072, Validation Loss: 0.9724629521369934
Epoch 60: Training Loss: 0.8261151313781738, Validation Loss: 0.9574528932571411
Epoch 70: Training Loss: 0.812021791934967, Validation Loss: 0.9434162974357605
Epoch 80: Training Loss: 0.798433780670166, Validation Loss: 0.9299797415733337
Epoch 90: Training Loss: 0.7853594422340393, Validation Loss: 0.917076051235199
Epoch 100: Training Loss: 0.7728792428970337, Validation Loss: 0.904740571975708
Epoch 110: Training Loss: 0.7610804438591003, Validation Loss: 0.8930349946022034
Epoch 120: Training Loss: 0.7500353455543518, Validation Loss: 0.8820186853408813
Epoch 130: Training Loss: 0.739793598651886, Validation Loss: 0.8717361688613892
Epoch 140: Training Loss: 0.7303813695907593, Validation Loss: 0.8622138500213623
Epoch 150: Training Loss: 0.7218034267425537, Validation Loss: 0.853459656238559
Epoch 160: Training Loss: 0.7140458226203918, Validation Loss: 0.845465600952148
Epoch 170: Training Loss: 0.7070803642272949, Validation Loss: 0.8382095694541931
Epoch 180: Training Loss: 0.700867235660553, Validation Loss: 0.8316590189933777
Epoch 190: Training Loss: 0.6953589916229248, Validation Loss: 0.8257730603218079
Epoch 200: Training Loss: 0.690502405166626, Validation Loss: 0.8205056190490723
Epoch 210: Training Loss: 0.6862422227859497, Validation Loss: 0.8158075213432312
Epoch 220: Training Loss: 0.6825219988822937, Validation Loss: 0.8116279244422913
Epoch 230: Training Loss: 0.6792863011360168, Validation Loss: 0.807917058467865
Epoch 240: Training Loss: 0.6764819025993347, Validation Loss: 0.804625928401947
Epoch 250: Training Loss: 0.6740583777427673, Validation Loss: 0.8017081618309021
Epoch 260: Training Loss: 0.6719690561294556, Validation Loss: 0.7991204261779785
Epoch 270: Training Loss: 0.670170783996582, Validation Loss: 0.7968224883079529
Epoch 280: Training Loss: 0.6686248779296875, Validation Loss: 0.7947779893875122
Epoch 290: Training Loss: 0.667296290397644, Validation Loss: 0.7929537892341614
Epoch 300: Training Loss: 0.6661543250083923, Validation Loss: 0.7913205623626709
```

In [6]:
```
# Plotting the Losses

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Loss")

# Plot the model and the actual values.
plt.plot(train_loss, label='Training loss')
plt.plot(val_loss, label='Validation loss')
plt.legend()
plt.title("Losses")
```

Out[6]:
```
Text(0.5, 1.0, 'Losses')
```



In [7]:
```
# Model with three hidden layer of 8, 32, 10
model_new = nn.Sequential(
        nn.Linear(5, 8),
        nn.Tanh(),
        nn.Linear(8, 32),
        nn.Tanh(),
        nn.Linear(32, 10),
        nn.Tanh(),
        nn.Linear(10, 1))

optimizer = optim.SGD(model_new.parameters(), lr=.01)

train_loss, val_loss = training_loop(NUM_EPOCHS, optimizer, model_new, nn.MSELoss(), trair
                                     validation_tensor, price_training, price_validation)
```

```
Epoch 1: Training Loss: 1.008755087852478, Validation Loss: 1.1300957202911377
Epoch 10: Training Loss: 0.9911390542984009, Validation Loss: 1.1144399642944336
Epoch 20: Training Loss: 0.9779357314109802, Validation Loss: 1.1026506423950195
Epoch 30: Training Loss: 0.967503309249879, Validation Loss: 1.0931150913238525
Epoch 40: Training Loss: 0.9581463932991028, Validation Loss: 1.0843251943588257
Epoch 50: Training Loss: 0.9491453766822815, Validation Loss: 1.0756783485412598
Epoch 60: Training Loss: 0.9401654005050659, Validation Loss: 1.0669103860855103
Epoch 70: Training Loss: 0.9310250878334045, Validation Loss: 1.0578820705413818
Epoch 80: Training Loss: 0.9216070175170898, Validation Loss: 1.0485001802444458
Epoch 90: Training Loss: 0.9118227958679199, Validation Loss: 1.0386887788772583
Epoch 100: Training Loss: 0.901601254940033, Validation Loss: 1.028381586074829
Epoch 110: Training Loss: 0.8908839225769043, Validation Loss: 1.0175195932388306
Epoch 120: Training Loss: 0.8796263337135315, Validation Loss: 1.0060539245605469
Epoch 130: Training Loss: 0.8677998185157776, Validation Loss: 0.9939488172531128
Epoch 140: Training Loss: 0.8553949594497681, Validation Loss: 0.9811856150627136
Epoch 150: Training Loss: 0.8424268364906311, Validation Loss: 0.9677679538726807
```

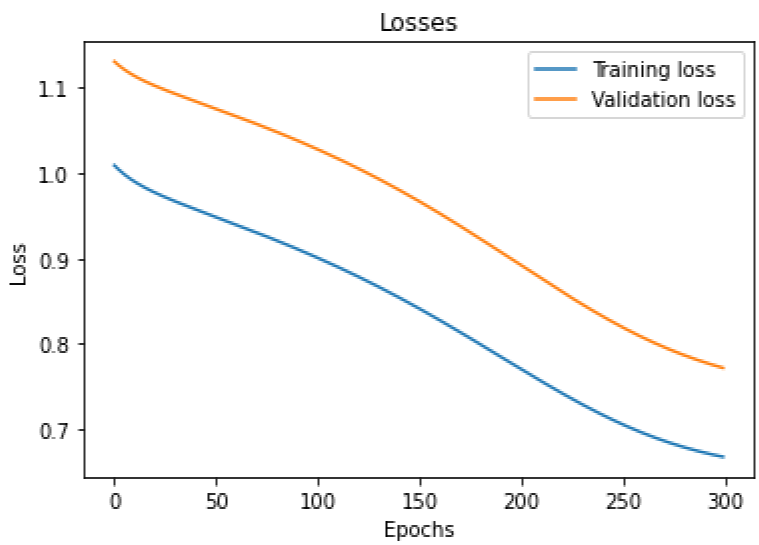
```
Epoch 160: Training Loss: 0.8289375901222229, Validation Loss: 0.9537258148193359
Epoch 170: Training Loss: 0.81500244140625, Validation Loss: 0.9391213655471802
Epoch 180: Training Loss: 0.8007311224937439, Validation Loss: 0.9240509867668152
Epoch 190: Training Loss: 0.7862692475318909, Validation Loss: 0.9086489677429199
Epoch 200: Training Loss: 0.7717961072921753, Validation Loss: 0.8930847644805908
Epoch 210: Training Loss: 0.7575180530548096, Validation Loss: 0.877559006214l418
Epoch 220: Training Loss: 0.7436574101448059, Validation Loss: 0.8622939586639404
Epoch 230: Training Loss: 0.730439305305481, Validation Loss: 0.8475205898284912
Epoch 240: Training Loss: 0.7180732488632202, Validation Loss: 0.8334622979164124
Epoch 250: Training Loss: 0.7067372798919678, Validation Loss: 0.820317804813385
Epoch 260: Training Loss: 0.6965621113777161, Validation Loss: 0.808246910572052
Epoch 270: Training Loss: 0.6876233220100403, Validation Loss: 0.7973576784133911
Epoch 280: Training Loss: 0.6799374222755432, Validation Loss: 0.787703275680542
Epoch 290: Training Loss: 0.673466682434082, Validation Loss: 0.7792821526527405
Epoch 300: Training Loss: 0.6681283116340637, Validation Loss: 0.7720456719398499
```

In [8]:

```
# Plotting the Losses

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Loss")

# Plot the model and the actual values.
plt.plot(train_loss, label='Training loss')
plt.plot(val_loss, label='Validation loss')
plt.legend()
plt.title("Losses")
```

Out[8]:

```
Text(0.5, 1.0, 'Losses')
```



Problem 2

In [9]:

```
from torchvision import transforms

transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4915, 0.4823, 0.4468),
                         (0.2470, 0.2435, 0.2616))
])

def training_loop(epochs, optimizer, model, loss_fn, train_loader, val_loader):
    training_losses = []
    val_losses = []
    accuracies = []
    for epoch in range(1, epochs + 1):
```

```python
        # Temp vars for use in finding the accuracy.
        correct_labels = 0
        count = 0
        loss_val_value = 0
        with torch.no_grad():
            for imgs, labels in val_loader:
                # Pass imgs through the model and find the loss.
                output = model(imgs.view(imgs.shape[0], -1))
                loss_val = loss_fn(output, labels)
                loss_val_value += float(loss_val)

                # Find the accurcey of the model.
                _, predicted = torch.max(output, dim=1)
                count += labels.shape[0]
                correct_labels += int((predicted == labels).sum())

            # Store the loss and accuracy.
            loss_val_value /= count
            val_losses.append(loss_val_value)
            accuracies.append(correct_labels/count)

        loss_train_value = 0
        for imgs, labels in train_loader:
            # Pass imgs through the model and find the loss.
            output = model(imgs.view(imgs.shape[0], -1))
            loss_train = loss_fn(output, labels)
            loss_train_value += float(loss_train)

            # Adject the params
            optimizer.zero_grad()
            loss_train.backward()
            optimizer.step()

        # Store the loss
        loss_train_value /= count
        training_losses.append(loss_train_value)

        # Print out the loss every 10 epoch
        if epoch % 10 == 0 or epoch == 1:
            print(f"Epoch: {epoch}, Training Loss: {loss_train}, Validation Loss: {loss_va

    return training_losses, val_losses, accuracies
```

In [10]:

```python
# Download the cifar10 dataset.
data = '.\cifar10'
cirfar10_train = datasets.CIFAR10(data, train=True, download=True, transform=transforms)
cirfar10_val = datasets.CIFAR10(data, train=False, download=True, transform=transforms)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [11]:

```python
NUM_EPOCHS = 300
LEARNING_RATE = 1e-2
BATCH_SIZE = 1024

# Neural Net with one hidden layer of 512
model = nn.Sequential(
        nn.Linear(3072, 512),
        nn.Tanh(),
        nn.Linear(512, 10),
        nn.LogSoftmax(dim=1))

loss = nn.NLLLoss()
```

```
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE)

# Load the data into a dataloader.
train_loader = torch.utils.data.DataLoader(cirfar10_train, batch_size=BATCH_SIZE, shuffle=
val_loader = torch.utils.data.DataLoader(cirfar10_val, batch_size=BATCH_SIZE, shuffle=Fals
```

In [12]:

```
# Using time to time the training.
start_time = time.time()
training_losses, val_losses, accuracies = training_loop(NUM_EPOCHS, optimizer, model, loss
end_time = time.time()

# Report the final stats about the training.
print(" ")
print(f"Final Loss: {training_losses[-1]}, Final Accuracy: {accuracies[-1] * 100}%")
print(f"Training Time: {(end_time - start_time):.2f} seconds")
```

```
Epoch: 1, Training Loss: 1.9678707122802734, Validation Loss: 2.2674553394317627, Accurac
y: 13.19%
Epoch: 10, Training Loss: 1.680593490600586, Validation Loss: 1.7601591348648071, Accurac
y: 39.629999999999995%
Epoch: 20, Training Loss: 1.683751106262207, Validation Loss: 1.7104140520095825, Accurac
y: 41.81%
Epoch: 30, Training Loss: 1.6511505842208862, Validation Loss: 1.6814672946929932, Accurac
y: 42.76%
Epoch: 40, Training Loss: 1.5423791408538818, Validation Loss: 1.661741018295288, Accurac
y: 43.730000000000004%
Epoch: 50, Training Loss: 1.5605401992797852, Validation Loss: 1.6429061889648438, Accurac
y: 44.24%
Epoch: 60, Training Loss: 1.4842920303344727, Validation Loss: 1.6270273923873901, Accurac
y: 45.07%
Epoch: 70, Training Loss: 1.4811780452728271, Validation Loss: 1.614139199256897, Accurac
y: 45.440000000000005%
Epoch: 80, Training Loss: 1.4788289070129395, Validation Loss: 1.5981080532073975, Accurac
y: 46.02%
Epoch: 90, Training Loss: 1.4786642789840698, Validation Loss: 1.5835200548171997, Accurac
y: 46.6%
Epoch: 100, Training Loss: 1.4138902425765991, Validation Loss: 1.5733616352081299, Accura
cy: 47.28%
Epoch: 110, Training Loss: 1.4385255575180054, Validation Loss: 1.561989665031433, Accurac
y: 47.44%
Epoch: 120, Training Loss: 1.383408546447754, Validation Loss: 1.5483051538467407, Accurac
y: 47.85%
Epoch: 130, Training Loss: 1.3222304582595825, Validation Loss: 1.5401769876480103, Accura
cy: 48.14%
Epoch: 140, Training Loss: 1.2957812547683716, Validation Loss: 1.5294804573059082, Accura
cy: 48.49%
Epoch: 150, Training Loss: 1.2477396726608276, Validation Loss: 1.5214554071426392, Accura
cy: 48.480000000000004%
Epoch: 160, Training Loss: 1.270158052444458, Validation Loss: 1.5157904624938965, Accurac
y: 49.02%
Epoch: 170, Training Loss: 1.1731148958206177, Validation Loss: 1.5065337419509888, Accura
cy: 49.18%
Epoch: 180, Training Loss: 1.2165769338607788, Validation Loss: 1.502706527709961, Accurac
y: 49.36%
Epoch: 190, Training Loss: 1.215359091758728, Validation Loss: 1.497601866722107, Accurac
y: 49.45%
Epoch: 200, Training Loss: 1.1286588907241821, Validation Loss: 1.5001407861709595, Accura
cy: 49.480000000000004%
Epoch: 210, Training Loss: 1.1143525838851929, Validation Loss: 1.4920920133590698, Accura
cy: 49.79%
Epoch: 220, Training Loss: 1.0544027090072632, Validation Loss: 1.4892915487289429, Accura
cy: 49.97%
Epoch: 230, Training Loss: 1.06126868724823, Validation Loss: 1.4929190874099731, Accurac
y: 49.86%
```

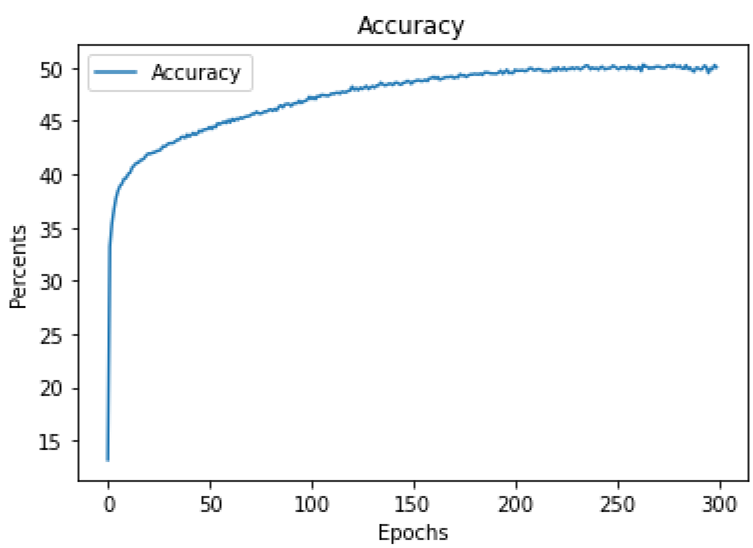
```
Epoch: 240, Training Loss: 1.0108919143676758, Validation Loss: 1.4876394271850586, Accura
cy: 49.9%
Epoch: 250, Training Loss: 0.9650559425354004, Validation Loss: 1.4945276975631714, Accura
cy: 50.1%
Epoch: 260, Training Loss: 0.9646227955818176, Validation Loss: 1.4951883554458618, Accura
cy: 49.89%
Epoch: 270, Training Loss: 0.9307659268379211, Validation Loss: 1.4912934303283691, Accura
cy: 50.09%
Epoch: 280, Training Loss: 0.9296294450759888, Validation Loss: 1.4971216917037964, Accura
cy: 50.080000000000005%
Epoch: 290, Training Loss: 0.9564180970191956, Validation Loss: 1.49622642993927, Accurac
y: 50.0%
Epoch: 300, Training Loss: 0.8289459347724915, Validation Loss: 1.4961400032043457, Accura
cy: 50.029999999999994%

Final Loss: 0.004228309345245361, Final Accuracy: 50.029999999999994%
Training Time: 4272.34 seconds
```

In [14]:

```
# Plotting the accuracy of the model.

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Percents")

for i, x in enumerate(accuracies):
    accuracies[i] = x * 100


plt.plot(accuracies, label='Accuracy')
plt.legend()
plt.title("Accuracy")
```

Out[14]:

```
Text(0.5, 1.0, 'Accuracy')
```



In [15]:

```
# Two new layers in the model.
model_new = nn.Sequential(
            nn.Linear(3072, 512),
            nn.Tanh(),
            nn.Linear(512, 1024),
            nn.Tanh(),
            nn.Linear(1024, 256),
            nn.Tanh(),
            nn.Linear(256, 10),
            nn.LogSoftmax(dim=1))
```

```
loss = nn.NLLLoss()
optimizer = optim.SGD(model_new.parameters(), lr=LEARNING_RATE)

# Using new model in the loop. Timing it with the same method.
start_time = time.time()
training_losses, val_losses, accuracies = training_loop(NUM_EPOCHS, optimizer, model_new,
end_time = time.time()

# Report the final stats about the training.
print(" ")
print(f"Final Loss: {training_losses[-1]}, Final Accuracy: {accuracies[-1] * 100}%")
print(f"Training Time: {(end_time - start_time):.2f} seconds")
```

```
Epoch: 1, Training Loss: 2.1452176570892334, Validation Loss: 2.3103466033935547, Accurac
y: 10.32%
Epoch: 10, Training Loss: 1.8333007097244263, Validation Loss: 1.8492969274520874, Accurac
y: 36.25%
Epoch: 20, Training Loss: 1.7433573007583618, Validation Loss: 1.7603672742843628, Accurac
y: 39.42%
Epoch: 30, Training Loss: 1.7012865543365479, Validation Loss: 1.7154326438903809, Accurac
y: 41.14%
Epoch: 40, Training Loss: 1.6453821659088135, Validation Loss: 1.6865911483764648, Accurac
y: 42.49%
Epoch: 50, Training Loss: 1.6486808061599731, Validation Loss: 1.66253662109375, Accuracy:
43.169999999999995%
Epoch: 60, Training Loss: 1.5862213373184204, Validation Loss: 1.6396384239196777, Accurac
y: 43.89%
Epoch: 70, Training Loss: 1.540399193763733, Validation Loss: 1.6203588247299194, Accurac
y: 45.04%
Epoch: 80, Training Loss: 1.5035895109176636, Validation Loss: 1.5962070226669312, Accurac
y: 45.98%
Epoch: 90, Training Loss: 1.5043388605117798, Validation Loss: 1.5754191875457764, Accurac
y: 46.69%
Epoch: 100, Training Loss: 1.3994776010513306, Validation Loss: 1.566023588180542, Accurac
y: 47.24%
Epoch: 110, Training Loss: 1.3454256057739258, Validation Loss: 1.5401884317398071, Accura
cy: 48.04%
Epoch: 120, Training Loss: 1.262939453125, Validation Loss: 1.551008939743042, Accuracy: 4
8.03%
Epoch: 130, Training Loss: 1.2566801309585571, Validation Loss: 1.5210061073303223, Accura
cy: 48.38%
Epoch: 140, Training Loss: 1.234034538269043, Validation Loss: 1.5160892009735107, Accurac
y: 49.02%
Epoch: 150, Training Loss: 1.1209686994552612, Validation Loss: 1.5157856941223145, Accura
cy: 49.09%
Epoch: 160, Training Loss: 1.1806663274765015, Validation Loss: 1.5796059370040894, Accura
cy: 47.24%
Epoch: 170, Training Loss: 1.1065318584442139, Validation Loss: 1.545309066772461, Accurac
y: 48.980000000000004%
Epoch: 180, Training Loss: 1.1540173292160034, Validation Loss: 1.6169922351837158, Accura
cy: 47.910000000000004%
Epoch: 190, Training Loss: 1.0791571140289307, Validation Loss: 1.5610172748565674, Accura
cy: 48.79%
Epoch: 200, Training Loss: 0.8759352564811707, Validation Loss: 1.6208983659744263, Accura
cy: 47.68%
Epoch: 210, Training Loss: 0.9494306445121765, Validation Loss: 1.6281899213790894, Accura
cy: 48.86%
Epoch: 220, Training Loss: 1.048671841621399, Validation Loss: 1.6302698850631714, Accurac
y: 48.68%
Epoch: 230, Training Loss: 0.7789023518562317, Validation Loss: 1.7281850576400757, Accura
cy: 47.870000000000005%
Epoch: 240, Training Loss: 0.7309607863426208, Validation Loss: 1.852726697921753, Accurac
y: 46.87%
Epoch: 250, Training Loss: 0.7191540002822876, Validation Loss: 1.8119103908538818, Accura
cy: 47.02%
```

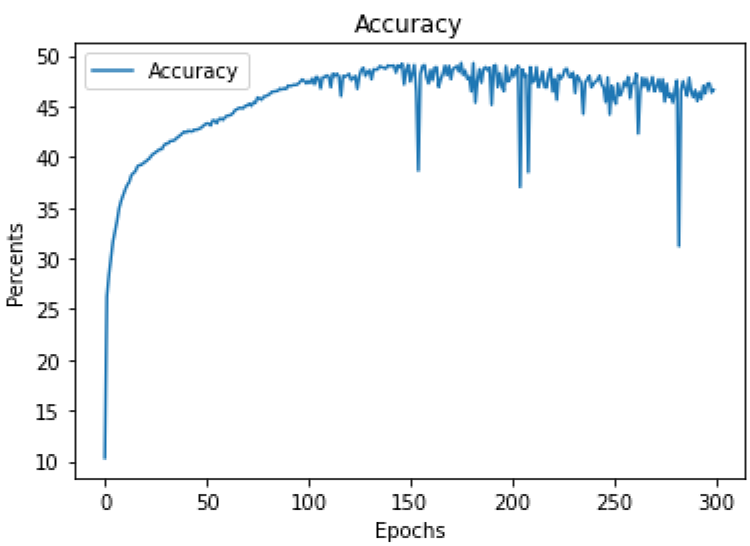
```
Epoch: 260, Training Loss: 0.6168233752250671, Validation Loss: 1.8419313430786133, Accura
cy: 47.17%
Epoch: 270, Training Loss: 0.5479784607887268, Validation Loss: 1.839116096496582, Accurac
y: 47.88%
Epoch: 280, Training Loss: 0.5738944411277771, Validation Loss: 2.102342367172241, Accurac
y: 45.36%
Epoch: 290, Training Loss: 0.5162743330001831, Validation Loss: 2.0567572116851807, Accura
cy: 45.9%
Epoch: 300, Training Loss: 2.5052082538604736, Validation Loss: 2.034482479095459, Accurac
y: 46.61%

Final Loss: 0.002601168116927147, Final Accuracy: 46.61%
Training Time: 4473.68 seconds
```

In [17]:

```
# Plotting the accuracy of the model.

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Percents")

for i, x in enumerate(accuracies):
    accuracies[i] = x * 100

plt.plot(accuracies, label='Accuracy')
plt.legend()
plt.title("Accuracy")
```

Out[17]:

```
Text(0.5, 1.0, 'Accuracy')
```



In [35]:

```
# Model with one hidden layer of 8
model_p1p1 = nn.Sequential(
        nn.Linear(5, 8),
        nn.Tanh(),
        nn.Linear(8, 1))

# Model with three hidden layer of 8, 32, 10
model__p1p2 = nn.Sequential(
                nn.Linear(5, 8),
                nn.Tanh(),
                nn.Linear(8, 32),
                nn.Tanh(),
                nn.Linear(32, 10),
                nn.Tanh(),
                nn.Linear(10, 1))
```

```python
from ptflops import get_model_complexity_info
import warnings
warnings.filterwarnings("ignore")

macs, params = get_model_complexity_info(model_p1p1, (436, 5), as_strings=True,
 print_per_layer_stat=False, verbose=False)
# print out the computational cost and the Model size.
print("Problem 1 Part 1")
print("Model size: " + params)

print("")

macs, params = get_model_complexity_info(model__p1p2, (436, 5), as_strings=True,
 print_per_layer_stat=False, verbose=False)
# print out the computational cost and the Model size.
print("Problem 1 Part 2")
print("Model size: " + params)

print("")

macs, params = get_model_complexity_info(model, (1, 3072), as_strings=True,
 print_per_layer_stat=False, verbose=False)
# print out the computational cost and the Model size.
print("Problem 2 Part 1")
print("Model size: " + params)

print("")

macs, params = get_model_complexity_info(model_new, (1, 3072), as_strings=True,
 print_per_layer_stat=False, verbose=False)
# print out the computational cost and the Model size.
print("Problem 2 Part 2")
print("Model size: " + params)
```

```
Problem 1 Part 1
Model size: 57

Problem 1 Part 2
Model size: 677

Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflop
s can affect your code!
Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflop
s can affect your code!
Problem 2 Part 1
Model size: 1.58 M

Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflop
s can affect your code!
Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflop
s can affect your code!
Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflop
s can affect your code!
Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflop
s can affect your code!
Problem 2 Part 2
Model size: 2.36 M
```

In [ ]: