Assignment 5:

Name: Sumedh Koppula

UID: 117386066

Please submit to ELMS

- a PDF containing all outputs (by executing **Run all**)
- your ipynb notebook containing all the code

I understand the policy on academic integraty (collaboration and the use of online material). Please sign your name here: Sumedh Reddy Koppula

```
# import the necessary packages
import numpy as np
import gzip, os
from urllib.request import urlretrieve
from random import random
from math import exp
from random import seed
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

Part 1: Backpropagation in Neural Networks (20 Points)

Overview

Artificial Neural Networks are computational learning systems that uses a network of functions to understand and translate a data train_features of one form into a desired output, usually in another form. The concept of the artificial neural network was inspired by human biology and the way neurons of the human brain function together to understand inputs from human senses.

A simple neural network consists of train_features Layer, Hidden Layer and Output Layer. To train these the network, we will use Backpropagation algorithm. Backpropagation is the cornerstone of modern neural networks. To understand the algorithm in details, here is a mathematical description in the Chapter 2 of *How the backpropagation algorithm works from Neural Networks and Deep Learning* (http://neuralnetworksanddeeplearning.com/chap2.html).

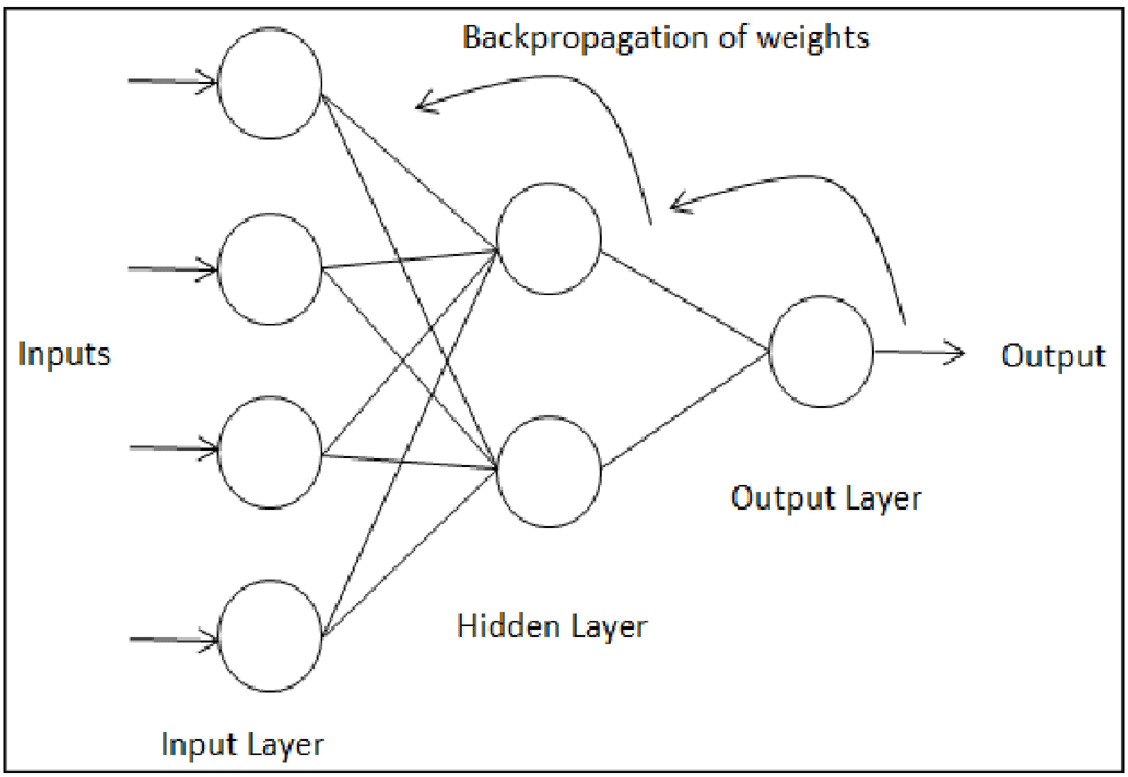
In this part, you are required to implement the following architecture and write training code of a neural network from scratch using the numpy library alone.

Architecture Definition :

- An train_features Layer with the following 2-dimensions:
- 0: Batch Size
- 1: 784 = 28*28 pixels
- A hidden layer with 500 units
- A second hidden layer with 50 units
- An output layer with 10 units

There are five major steps to the implementation:

1. Define neural network: initialize_network()
2. Forward Propagation: pre_activation(), sigmoid_activation(), forward_propagation()
3. Backpropagation: backward_propagate_error()
4. Loss function and updation of weights_vector (SGD): update_weights_vector()
5. Training: train()



Data

```
# Download Data -- run this cell only one time per runtime
!gdown 11SpETIc56PReKuaUKEwWDvdkiynyyGFA
!unzip "/content/MNISTArchive.zip" -d "/content/"
!gzip -d "/content/t10k-labels-idx1-ubyte.gz"
!gzip -d "/content/t10k-images-idx3-ubyte.gz"
```

```
!gzip -d "/content/train-labels-idx1-ubyte.gz"
!gzip -d "/content/train-images-idx3-ubyte.gz"
```

Helper Functions:

Code (10 pts)

```
def read_mnist(path=None):
    r"""Return (train_images, train_labels, test_images, test_labels).

    Args:
        path (str): Directory containing MNIST. Default is
            /home/USER/data/mnist or C:\Users\USER\data\mnist.
            Create if nonexistant. Download any missing files.

    Returns:
        Tuple of (train_images, train_labels, test_images,
test_labels), each
            a matrix. Rows are examples. Columns of images are pixel
values.
            Columns of labels are a onehot encoding of the correct
class.
    """
    url = 'http://yann.lecun.com/exdb/mnist/'
    files = ['train-images-idx3-ubyte.gz',
             'train-labels-idx1-ubyte.gz',
             't10k-images-idx3-ubyte.gz',
             't10k-labels-idx1-ubyte.gz']

    if path is None:
        # Set path to /home/USER/data/mnist or C:\Users\USER\data\
mnist
        path = os.path.join(os.path.expanduser('~'), 'data', 'mnist')

    # Create path if it doesn't exist
    os.makedirs(path, exist_ok=True)

    # Download any missing files
    for file in files:
        if file not in os.listdir(path):
            urlretrieve(url + file, os.path.join(path, file))
            print("Downloaded %s to %s" % (file, path))

    def _images(path):
        """Return images loaded locally."""
        with gzip.open(path) as f:
            # First 16 bytes are magic_number, n_imgs, n_rows, n_cols
            pixels = np.frombuffer(f.read(), 'B', offset=16)
        return pixels.reshape(-1, 784).astype('float32') / 255
```

```
    def _labels(path):
        """Return labels loaded locally."""
        with gzip.open(path) as f:
            # First 8 bytes are magic_number, n_labels
            integer_labels = np.frombuffer(f.read(), 'B', offset=8)

        def _onehot(integer_labels):
            """Return matrix whose rows are onehot encodings of
integers."""
            n_rows = len(integer_labels)
            n_cols = integer_labels.max() + 1
            onehot = np.zeros((n_rows, n_cols), dtype='uint8')
            onehot[np.arange(n_rows), integer_labels] = 1
            return onehot

        return _onehot(integer_labels)

    train_images = _images(os.path.join(path, files[0]))
    train_labels = _labels(os.path.join(path, files[1]))
    test_images = _images(os.path.join(path, files[2]))
    test_labels = _labels(os.path.join(path, files[3]))

    return train_images, train_labels, test_images, test_labels

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
     network = list()

     ## Write your code. Initialize hidden layer here.
     hidden_layer = [{'weights_vector':[random() for i in
range(n_inputs + 1)]} for i in range(n_hidden)]
     network.append(hidden_layer)

     ## Write your code. Initialize output_layer layer here.
     output_layer = [{'weights_vector':[random() for i in
range(n_hidden + 1)]} for i in range(n_outputs)]
     network.append(output_layer)
     return network

def initialize_network_mnist(train_data, y, n_hidden, neuron_size,
n_inputs= None, n_outputs= None):
     if n_inputs is not None:
          input_value = n_inputs
     else:
          input_value = train_features.shape[1]
     if n_outputs is not None:
          output_value = n_outputs
     else:
          output_value = y.shape[1]
     biases = []
```

```
    weights_vector = []
    weights_vector.append(np.random.randn(input_value,
neuron_size[0]) * np.sqrt(0.006))
    for i in range(n_hidden):

    weights_vector.append(np.random.randn(weights_vector[i].shape[1],
neuron_size[i + 1]) * np.sqrt(0.006))

    weights_vector.append(np.random.randn(weights_vector[-
1].shape[1], output_value) * np.sqrt(0.006))

    for i in range(len(weights_vector)):
        biases.append(np.random.randn(weights_vector[i].shape[1], )
* np.sqrt(0.006))

    return weights_vector, biases

# Calculate neuron activation for an train_features
def pre_activation(weights_vector, inputs):
    activation = weights_vector[-1]
    for i in range(len(weights_vector)-1):
        ## Write code here. compute activation: Wx+b
        activation += weights_vector[i] * inputs[i]

    return activation

def sigmoid_activation(activation):
    ## write code. implement sigmoid function
    out_sigmoid = 1.0 / (1.0 + exp(-activation))
    return out_sigmoid

# Calculate the derivative of a neuron output
def sigmoid_derivative(output):
    ## write code. implement sigmoid function
    out_sigmoid_deriv = output * (1.0 - output)
    return out_sigmoid_deriv

# Relu activation function
def ReLU(x):
    relu_ = np.maximum(0, x)
    return relu_

# Relu derivative function
def dReLU(x):
    relu_derivative = 1 * (x > 0)
    return relu_derivative

# Softmax activation function
def softmax(z):
    z = z - np.max(z, axis=1).reshape(z.shape[0], 1)
```

```
     softmax_ = np.exp(z) / np.sum(np.exp(z),
axis=1).reshape(z.shape[0], 1)
     return softmax_

# Shuffle the train_features and expected
def shuffle(train_features, expected):
    idx = [i for i in range(train_features.shape[0])]
    np.random.shuffle(idx)
    train_features = train_features[idx]
    expected = expected[idx]
    return train_features, expected

# Reverse the list
def reverse_list(sample_list):
    reversed_list = sample_list.copy()
    reversed_list.reverse()
    return reversed_list

# Forward Propagation:
def forward_propagation(network, row):
     inputs = row
     for layer in network:

          new_inputs = []
          ## write you code here.
          ## for each hidden neuron this 'layer', compute \
          ## pre_activation, sigmoid_activation and save then output
in 'new_inputs.'
          for neuron in layer:
               activation = pre_activation(neuron['weights_vector'],
inputs)
               neuron['output'] = sigmoid_activation(activation)
               new_inputs.append(neuron['output'])

          inputs = new_inputs
     return inputs

# Backpropagation:
def backward_propagate_error(network, ground_truth):
     for i in reversed(range(len(network))):
          layer = network[i]
          errors = list()
          if i != len(network)-1:
               ## write your code here.
               ## compute error for all the hidden layer and append
it to errors to keep track.
               for j in range(len(layer)):
                    error = 0.0
                    for neuron in network[i + 1]:
                         error += (neuron['weights_vector'][j] *
```

```
neuron['delta'])
                        errors.append(error)
                    #print("error computed for hidden layer")
            else:
                    ## write your code here.
                    ## compute error for the output layer using
ground_truth and append it to errors to keep track.
                    for j in range(len(layer)):
                        neuron = layer[j]
                        errors.append(ground_truth[j] -
neuron['output'])

                    #print("error computed for output layer")

            for j in range(len(layer)):
                    neuron = layer[j]
                    neuron['delta'] = errors[j] *
sigmoid_derivative(neuron['output'])
                    #print("delta computed for neuron")


# Stochastic GD for weight updation:
def update_weights_vector(network, row, l_rate):
        for i in range(len(network)):
            inputs = row[:-1]
            if i != 0:
                    ## write your code here.
                    ## pass activation i.e. neuron['output'] from previous
layer as train_features to current layer 'i'
                    inputs = [neuron['output'] for neuron in network[i -
1]]
                    #print("train_features computed for hidden layer")

            for neuron in network[i]:
                    for j in range(len(inputs)):
                        ## write you code here.
                        ## update the weights_vector between each
train_features and each neuron.
                        # Handle index out of range error
                        if j < len(neuron['weights_vector']):
                            neuron['weights_vector'][j] += l_rate *
neuron['delta'] * inputs[j]

                    ## write you code here.
                    ## update the bias vector
                    neuron['weights_vector'][-1] += l_rate *
neuron['delta']
                    #print("bias updated")

# Train a network for a fixed number of epochs
```

```
def train(network, train, l_rate, n_epoch, n_outputs):
	sum_error_lst = []
	for epoch in range(n_epoch):
		sum_error = 0
		for row in train:
			outputs = forward_propagation(network, row)
			expected = [0 for i in range(n_outputs)]
			expected[int(row[-1])] = 1
			#expected[row[-1]] = 1
			sum_error += sum([(expected[i]-outputs[i])**2 for i in
range(len(expected))])
			backward_propagate_error(network, expected)
			update_weights_vector(network, row, l_rate)
		sum_error_lst.append(sum_error)
		print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate,
sum_error))

	return sum_error_lst

# Feed forward for MNIST dataset
def feed_forward_mnist(x_, y_, weights_vector, bias_vector):
	layer_output_ = []
	activation_layer_ = []

	assert x_.shape[1] == weights_vector[0].shape[0]
	layer_output_.append(x_.dot(weights_vector[0]) + bias_vector[0])
	activation_layer_.append(ReLU(layer_output_[0]))

	for i in range(1, len(weights_vector)):
		assert activation_layer_[i - 1].shape[1] ==
weights_vector[i].shape[0]
		layer_output_.append(activation_layer_[i -
1].dot(weights_vector[i]) + bias_vector[i])
		activation_layer_.append(ReLU(layer_output_[i]))

	error = activation_layer_[-1] - y_
	return error, activation_layer_, layer_output_, weights_vector,
bias_vector, x_, y_

# Backpropagation for MNIST dataset
def back_propagation_mnist(activation_layer_, layer_output_,
weights_vector, bias_vector, batch, error, learning_rate, x):
	d_cost = (1 / batch) * error
	values = []
	b_values = []
	delete_weights = []
	delete_bias = []
	reverse_relu_activation = reverse_list(activation_layer_)
	reverse_relu_layer = reverse_list(layer_output_)
	reverse_weights = reverse_list(weights_vector)
```

```
    reverse_bias = reverse_list(bias_vector)
    delete_weights.append(np.dot(d_cost.T,
reverse_relu_activation[1]).T)
    val = np.dot((d_cost), reverse_weights[0].T) *
dReLU(reverse_relu_layer[1])
    values.append(val)
    delete_weights.append((np.dot(val.T,
reverse_relu_activation[2])).T)

    for i in range(len(weights_vector) - 3):
        val = np.dot(values[i], reverse_weights[i + 1].T) *
dReLU(reverse_relu_layer[i + 2])
        values.append(val)
        delete_weights.append((np.dot(values[i + 1].T,
reverse_relu_activation[i + 3])).T)

    delete_weights.append(np.dot((np.dot(values[-1],
reverse_weights[-2].T) * dReLU(reverse_relu_layer[-1])).T, x).T)

    delete_bias.append(np.sum(d_cost, axis=0))

    b_val = np.dot((d_cost), reverse_weights[0].T) *
dReLU(reverse_relu_layer[1])
    b_values.append(b_val)
    delete_bias.append(np.sum(b_val, axis=0))

    for i in range(len(weights_vector) - 2):
        b_val = np.dot(b_values[i], reverse_weights[i + 1].T) *
dReLU(reverse_relu_layer[i + 2])
        b_values.append(b_val)
        delete_bias.append(np.sum(b_values[i + 1], axis=0))

    for i in range(len(weights_vector)):
        assert delete_weights[i].shape == reverse_weights[i].shape
        reverse_weights[i] = reverse_weights[i] - learning_rate *
delete_weights[i]
        assert delete_bias[i].shape == reverse_bias[i].shape
        reverse_bias[i] = reverse_bias[i] - learning_rate *
delete_bias[i]

    reverse_weights.reverse()
    reverse_bias.reverse()

    return reverse_weights, reverse_bias

def train_mnist(train_features, expected, weights_vector, biases,
batch_size, epochs, learning_rate):
    loss_list = []
    accuracy_list = []
```

```
    for j in range(epochs):
        l = 0
        accuracy_value = 0
        train_features, expected = shuffle(train_features, expected)

        for i in range(train_features.shape[0] // batch_size - 1):
            start = i * batch_size
            end = (i + 1) * batch_size
            x = train_features[start:end]
            y = expected[start:end]
            error, activation_layer, layer_output, weights_vector,
biases, x, y = feed_forward_mnist(x, y, weights_vector, biases)
            weights_vector, biases =
back_propagation_mnist(activation_layer, layer_output, weights_vector,
biases, batch_size, error, learning_rate, x)
            l += np.mean(error ** 2)
            accuracy_value +=
np.count_nonzero(np.argmax(activation_layer[-1], axis=1) ==
np.argmax(y, axis=1)) / batch_size

        loss_list.append(l / (train_features.shape[0] // batch_size))
        accuracy_list.append(accuracy_value / (train_features.shape[0]
// batch_size))
    print("Train Accuracy:", np.max(accuracy_list) * 100, "%")
    return weights_vector, biases, loss_list, accuracy_list


def test_mnist(xtest, ytest, weights_vector, biases):
    x = xtest
    y = ytest
    _, activation_layer, _, weights_vector, biases, x, y =
feed_forward_mnist(x, y, weights_vector, biases)
    accuracy_val = np.count_nonzero(np.argmax(activation_layer[-1],
axis=1) == np.argmax(ytest, axis=1)) / xtest.shape[0]
    print("Test Accuracy:", 100 * accuracy_val, "%")

# 1. Test your code for backprop algorithm on this sample dataset.
sample_dataset = [[2.7810836,2.550537003,0],
     [1.465489372,2.362125076,0],
     [3.396561688,4.400293529,0],
     [1.38807019,1.850220317,0],
     [3.06407232,3.005305973,0],
     [7.627531214,2.759262235,1],
     [5.332441248,2.088626775,1],
     [6.922596716,1.77106367,1],
     [8.675418651,-0.242068655,1],
     [7.673756466,3.508563011,1]]

n_inputs = len(sample_dataset[0]) - 1
n_outputs = len(set([sample[-1] for sample in sample_dataset]))
```

```python
network = initialize_network(n_inputs, 2, n_outputs)
error = train(network, sample_dataset, l_rate=0.5, n_epoch=1000,
n_outputs=n_outputs)
for layer in network:
    print(layer)

# Plot error vs epoch
plt.plot(error)
plt.title('loss vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.show()
```

```
>epoch=0, lrate=0.500, error=6.504
>epoch=1, lrate=0.500, error=5.657
>epoch=2, lrate=0.500, error=5.269
>epoch=3, lrate=0.500, error=4.998
>epoch=4, lrate=0.500, error=4.741
>epoch=5, lrate=0.500, error=4.481
>epoch=6, lrate=0.500, error=4.190
>epoch=7, lrate=0.500, error=3.876
>epoch=8, lrate=0.500, error=3.557
>epoch=9, lrate=0.500, error=3.245
>epoch=10, lrate=0.500, error=2.951
>epoch=11, lrate=0.500, error=2.680
>epoch=12, lrate=0.500, error=2.433
>epoch=13, lrate=0.500, error=2.211
>epoch=14, lrate=0.500, error=2.012
>epoch=15, lrate=0.500, error=1.836
>epoch=16, lrate=0.500, error=1.679
>epoch=17, lrate=0.500, error=1.541
>epoch=18, lrate=0.500, error=1.418
>epoch=19, lrate=0.500, error=1.309
>epoch=20, lrate=0.500, error=1.212
>epoch=21, lrate=0.500, error=1.126
>epoch=22, lrate=0.500, error=1.050
>epoch=23, lrate=0.500, error=0.981
>epoch=24, lrate=0.500, error=0.920
>epoch=25, lrate=0.500, error=0.864
>epoch=26, lrate=0.500, error=0.814
>epoch=27, lrate=0.500, error=0.769
>epoch=28, lrate=0.500, error=0.728
>epoch=29, lrate=0.500, error=0.690
>epoch=30, lrate=0.500, error=0.656
>epoch=31, lrate=0.500, error=0.624
>epoch=32, lrate=0.500, error=0.595
>epoch=33, lrate=0.500, error=0.569
>epoch=34, lrate=0.500, error=0.544
>epoch=35, lrate=0.500, error=0.521
>epoch=36, lrate=0.500, error=0.500
```

```
>epoch=37, lrate=0.500, error=0.480
>epoch=38, lrate=0.500, error=0.461
>epoch=39, lrate=0.500, error=0.444
>epoch=40, lrate=0.500, error=0.428
>epoch=41, lrate=0.500, error=0.412
>epoch=42, lrate=0.500, error=0.398
>epoch=43, lrate=0.500, error=0.385
>epoch=44, lrate=0.500, error=0.372
>epoch=45, lrate=0.500, error=0.360
>epoch=46, lrate=0.500, error=0.348
>epoch=47, lrate=0.500, error=0.338
>epoch=48, lrate=0.500, error=0.327
>epoch=49, lrate=0.500, error=0.318
>epoch=50, lrate=0.500, error=0.308
>epoch=51, lrate=0.500, error=0.299
>epoch=52, lrate=0.500, error=0.291
>epoch=53, lrate=0.500, error=0.283
>epoch=54, lrate=0.500, error=0.275
>epoch=55, lrate=0.500, error=0.268
>epoch=56, lrate=0.500, error=0.261
>epoch=57, lrate=0.500, error=0.254
>epoch=58, lrate=0.500, error=0.247
>epoch=59, lrate=0.500, error=0.241
>epoch=60, lrate=0.500, error=0.235
>epoch=61, lrate=0.500, error=0.229
>epoch=62, lrate=0.500, error=0.223
>epoch=63, lrate=0.500, error=0.218
>epoch=64, lrate=0.500, error=0.212
>epoch=65, lrate=0.500, error=0.207
>epoch=66, lrate=0.500, error=0.202
>epoch=67, lrate=0.500, error=0.198
>epoch=68, lrate=0.500, error=0.193
>epoch=69, lrate=0.500, error=0.188
>epoch=70, lrate=0.500, error=0.184
>epoch=71, lrate=0.500, error=0.180
>epoch=72, lrate=0.500, error=0.176
>epoch=73, lrate=0.500, error=0.171
>epoch=74, lrate=0.500, error=0.168
>epoch=75, lrate=0.500, error=0.164
>epoch=76, lrate=0.500, error=0.160
>epoch=77, lrate=0.500, error=0.157
>epoch=78, lrate=0.500, error=0.153
>epoch=79, lrate=0.500, error=0.150
>epoch=80, lrate=0.500, error=0.146
>epoch=81, lrate=0.500, error=0.143
>epoch=82, lrate=0.500, error=0.140
>epoch=83, lrate=0.500, error=0.137
>epoch=84, lrate=0.500, error=0.134
>epoch=85, lrate=0.500, error=0.131
>epoch=86, lrate=0.500, error=0.129
```

```
>epoch=87, lrate=0.500, error=0.126
>epoch=88, lrate=0.500, error=0.123
>epoch=89, lrate=0.500, error=0.121
>epoch=90, lrate=0.500, error=0.118
>epoch=91, lrate=0.500, error=0.116
>epoch=92, lrate=0.500, error=0.114
>epoch=93, lrate=0.500, error=0.112
>epoch=94, lrate=0.500, error=0.109
>epoch=95, lrate=0.500, error=0.107
>epoch=96, lrate=0.500, error=0.105
>epoch=97, lrate=0.500, error=0.103
>epoch=98, lrate=0.500, error=0.101
>epoch=99, lrate=0.500, error=0.100
>epoch=100, lrate=0.500, error=0.098
>epoch=101, lrate=0.500, error=0.096
>epoch=102, lrate=0.500, error=0.094
>epoch=103, lrate=0.500, error=0.093
>epoch=104, lrate=0.500, error=0.091
>epoch=105, lrate=0.500, error=0.090
>epoch=106, lrate=0.500, error=0.088
>epoch=107, lrate=0.500, error=0.087
>epoch=108, lrate=0.500, error=0.085
>epoch=109, lrate=0.500, error=0.084
>epoch=110, lrate=0.500, error=0.083
>epoch=111, lrate=0.500, error=0.081
>epoch=112, lrate=0.500, error=0.080
>epoch=113, lrate=0.500, error=0.079
>epoch=114, lrate=0.500, error=0.077
>epoch=115, lrate=0.500, error=0.076
>epoch=116, lrate=0.500, error=0.075
>epoch=117, lrate=0.500, error=0.074
>epoch=118, lrate=0.500, error=0.073
>epoch=119, lrate=0.500, error=0.072
>epoch=120, lrate=0.500, error=0.071
>epoch=121, lrate=0.500, error=0.070
>epoch=122, lrate=0.500, error=0.069
>epoch=123, lrate=0.500, error=0.068
>epoch=124, lrate=0.500, error=0.067
>epoch=125, lrate=0.500, error=0.066
>epoch=126, lrate=0.500, error=0.065
>epoch=127, lrate=0.500, error=0.064
>epoch=128, lrate=0.500, error=0.064
>epoch=129, lrate=0.500, error=0.063
>epoch=130, lrate=0.500, error=0.062
>epoch=131, lrate=0.500, error=0.061
>epoch=132, lrate=0.500, error=0.060
>epoch=133, lrate=0.500, error=0.060
>epoch=134, lrate=0.500, error=0.059
>epoch=135, lrate=0.500, error=0.058
>epoch=136, lrate=0.500, error=0.057
```

```
>epoch=137, lrate=0.500, error=0.057
>epoch=138, lrate=0.500, error=0.056
>epoch=139, lrate=0.500, error=0.055
>epoch=140, lrate=0.500, error=0.055
>epoch=141, lrate=0.500, error=0.054
>epoch=142, lrate=0.500, error=0.054
>epoch=143, lrate=0.500, error=0.053
>epoch=144, lrate=0.500, error=0.052
>epoch=145, lrate=0.500, error=0.052
>epoch=146, lrate=0.500, error=0.051
>epoch=147, lrate=0.500, error=0.051
>epoch=148, lrate=0.500, error=0.050
>epoch=149, lrate=0.500, error=0.050
>epoch=150, lrate=0.500, error=0.049
>epoch=151, lrate=0.500, error=0.049
>epoch=152, lrate=0.500, error=0.048
>epoch=153, lrate=0.500, error=0.048
>epoch=154, lrate=0.500, error=0.047
>epoch=155, lrate=0.500, error=0.047
>epoch=156, lrate=0.500, error=0.046
>epoch=157, lrate=0.500, error=0.046
>epoch=158, lrate=0.500, error=0.045
>epoch=159, lrate=0.500, error=0.045
>epoch=160, lrate=0.500, error=0.044
>epoch=161, lrate=0.500, error=0.044
>epoch=162, lrate=0.500, error=0.043
>epoch=163, lrate=0.500, error=0.043
>epoch=164, lrate=0.500, error=0.043
>epoch=165, lrate=0.500, error=0.042
>epoch=166, lrate=0.500, error=0.042
>epoch=167, lrate=0.500, error=0.042
>epoch=168, lrate=0.500, error=0.041
>epoch=169, lrate=0.500, error=0.041
>epoch=170, lrate=0.500, error=0.040
>epoch=171, lrate=0.500, error=0.040
>epoch=172, lrate=0.500, error=0.040
>epoch=173, lrate=0.500, error=0.039
>epoch=174, lrate=0.500, error=0.039
>epoch=175, lrate=0.500, error=0.039
>epoch=176, lrate=0.500, error=0.038
>epoch=177, lrate=0.500, error=0.038
>epoch=178, lrate=0.500, error=0.038
>epoch=179, lrate=0.500, error=0.037
>epoch=180, lrate=0.500, error=0.037
>epoch=181, lrate=0.500, error=0.037
>epoch=182, lrate=0.500, error=0.036
>epoch=183, lrate=0.500, error=0.036
>epoch=184, lrate=0.500, error=0.036
>epoch=185, lrate=0.500, error=0.036
>epoch=186, lrate=0.500, error=0.035
```

```
>epoch=187, lrate=0.500, error=0.035
>epoch=188, lrate=0.500, error=0.035
>epoch=189, lrate=0.500, error=0.035
>epoch=190, lrate=0.500, error=0.034
>epoch=191, lrate=0.500, error=0.034
>epoch=192, lrate=0.500, error=0.034
>epoch=193, lrate=0.500, error=0.033
>epoch=194, lrate=0.500, error=0.033
>epoch=195, lrate=0.500, error=0.033
>epoch=196, lrate=0.500, error=0.033
>epoch=197, lrate=0.500, error=0.033
>epoch=198, lrate=0.500, error=0.032
>epoch=199, lrate=0.500, error=0.032
>epoch=200, lrate=0.500, error=0.032
>epoch=201, lrate=0.500, error=0.032
>epoch=202, lrate=0.500, error=0.031
>epoch=203, lrate=0.500, error=0.031
>epoch=204, lrate=0.500, error=0.031
>epoch=205, lrate=0.500, error=0.031
>epoch=206, lrate=0.500, error=0.030
>epoch=207, lrate=0.500, error=0.030
>epoch=208, lrate=0.500, error=0.030
>epoch=209, lrate=0.500, error=0.030
>epoch=210, lrate=0.500, error=0.030
>epoch=211, lrate=0.500, error=0.029
>epoch=212, lrate=0.500, error=0.029
>epoch=213, lrate=0.500, error=0.029
>epoch=214, lrate=0.500, error=0.029
>epoch=215, lrate=0.500, error=0.029
>epoch=216, lrate=0.500, error=0.029
>epoch=217, lrate=0.500, error=0.028
>epoch=218, lrate=0.500, error=0.028
>epoch=219, lrate=0.500, error=0.028
>epoch=220, lrate=0.500, error=0.028
>epoch=221, lrate=0.500, error=0.028
>epoch=222, lrate=0.500, error=0.027
>epoch=223, lrate=0.500, error=0.027
>epoch=224, lrate=0.500, error=0.027
>epoch=225, lrate=0.500, error=0.027
>epoch=226, lrate=0.500, error=0.027
>epoch=227, lrate=0.500, error=0.027
>epoch=228, lrate=0.500, error=0.026
>epoch=229, lrate=0.500, error=0.026
>epoch=230, lrate=0.500, error=0.026
>epoch=231, lrate=0.500, error=0.026
>epoch=232, lrate=0.500, error=0.026
>epoch=233, lrate=0.500, error=0.026
>epoch=234, lrate=0.500, error=0.026
>epoch=235, lrate=0.500, error=0.025
>epoch=236, lrate=0.500, error=0.025
```

```
>epoch=237, lrate=0.500, error=0.025
>epoch=238, lrate=0.500, error=0.025
>epoch=239, lrate=0.500, error=0.025
>epoch=240, lrate=0.500, error=0.025
>epoch=241, lrate=0.500, error=0.025
>epoch=242, lrate=0.500, error=0.024
>epoch=243, lrate=0.500, error=0.024
>epoch=244, lrate=0.500, error=0.024
>epoch=245, lrate=0.500, error=0.024
>epoch=246, lrate=0.500, error=0.024
>epoch=247, lrate=0.500, error=0.024
>epoch=248, lrate=0.500, error=0.024
>epoch=249, lrate=0.500, error=0.023
>epoch=250, lrate=0.500, error=0.023
>epoch=251, lrate=0.500, error=0.023
>epoch=252, lrate=0.500, error=0.023
>epoch=253, lrate=0.500, error=0.023
>epoch=254, lrate=0.500, error=0.023
>epoch=255, lrate=0.500, error=0.023
>epoch=256, lrate=0.500, error=0.023
>epoch=257, lrate=0.500, error=0.022
>epoch=258, lrate=0.500, error=0.022
>epoch=259, lrate=0.500, error=0.022
>epoch=260, lrate=0.500, error=0.022
>epoch=261, lrate=0.500, error=0.022
>epoch=262, lrate=0.500, error=0.022
>epoch=263, lrate=0.500, error=0.022
>epoch=264, lrate=0.500, error=0.022
>epoch=265, lrate=0.500, error=0.022
>epoch=266, lrate=0.500, error=0.021
>epoch=267, lrate=0.500, error=0.021
>epoch=268, lrate=0.500, error=0.021
>epoch=269, lrate=0.500, error=0.021
>epoch=270, lrate=0.500, error=0.021
>epoch=271, lrate=0.500, error=0.021
>epoch=272, lrate=0.500, error=0.021
>epoch=273, lrate=0.500, error=0.021
>epoch=274, lrate=0.500, error=0.021
>epoch=275, lrate=0.500, error=0.021
>epoch=276, lrate=0.500, error=0.020
>epoch=277, lrate=0.500, error=0.020
>epoch=278, lrate=0.500, error=0.020
>epoch=279, lrate=0.500, error=0.020
>epoch=280, lrate=0.500, error=0.020
>epoch=281, lrate=0.500, error=0.020
>epoch=282, lrate=0.500, error=0.020
>epoch=283, lrate=0.500, error=0.020
>epoch=284, lrate=0.500, error=0.020
>epoch=285, lrate=0.500, error=0.020
>epoch=286, lrate=0.500, error=0.020
```

```
>epoch=287, lrate=0.500, error=0.019
>epoch=288, lrate=0.500, error=0.019
>epoch=289, lrate=0.500, error=0.019
>epoch=290, lrate=0.500, error=0.019
>epoch=291, lrate=0.500, error=0.019
>epoch=292, lrate=0.500, error=0.019
>epoch=293, lrate=0.500, error=0.019
>epoch=294, lrate=0.500, error=0.019
>epoch=295, lrate=0.500, error=0.019
>epoch=296, lrate=0.500, error=0.019
>epoch=297, lrate=0.500, error=0.019
>epoch=298, lrate=0.500, error=0.019
>epoch=299, lrate=0.500, error=0.018
>epoch=300, lrate=0.500, error=0.018
>epoch=301, lrate=0.500, error=0.018
>epoch=302, lrate=0.500, error=0.018
>epoch=303, lrate=0.500, error=0.018
>epoch=304, lrate=0.500, error=0.018
>epoch=305, lrate=0.500, error=0.018
>epoch=306, lrate=0.500, error=0.018
>epoch=307, lrate=0.500, error=0.018
>epoch=308, lrate=0.500, error=0.018
>epoch=309, lrate=0.500, error=0.018
>epoch=310, lrate=0.500, error=0.018
>epoch=311, lrate=0.500, error=0.018
>epoch=312, lrate=0.500, error=0.017
>epoch=313, lrate=0.500, error=0.017
>epoch=314, lrate=0.500, error=0.017
>epoch=315, lrate=0.500, error=0.017
>epoch=316, lrate=0.500, error=0.017
>epoch=317, lrate=0.500, error=0.017
>epoch=318, lrate=0.500, error=0.017
>epoch=319, lrate=0.500, error=0.017
>epoch=320, lrate=0.500, error=0.017
>epoch=321, lrate=0.500, error=0.017
>epoch=322, lrate=0.500, error=0.017
>epoch=323, lrate=0.500, error=0.017
>epoch=324, lrate=0.500, error=0.017
>epoch=325, lrate=0.500, error=0.017
>epoch=326, lrate=0.500, error=0.017
>epoch=327, lrate=0.500, error=0.016
>epoch=328, lrate=0.500, error=0.016
>epoch=329, lrate=0.500, error=0.016
>epoch=330, lrate=0.500, error=0.016
>epoch=331, lrate=0.500, error=0.016
>epoch=332, lrate=0.500, error=0.016
>epoch=333, lrate=0.500, error=0.016
>epoch=334, lrate=0.500, error=0.016
>epoch=335, lrate=0.500, error=0.016
>epoch=336, lrate=0.500, error=0.016
```

```
>epoch=337, lrate=0.500, error=0.016
>epoch=338, lrate=0.500, error=0.016
>epoch=339, lrate=0.500, error=0.016
>epoch=340, lrate=0.500, error=0.016
>epoch=341, lrate=0.500, error=0.016
>epoch=342, lrate=0.500, error=0.016
>epoch=343, lrate=0.500, error=0.016
>epoch=344, lrate=0.500, error=0.015
>epoch=345, lrate=0.500, error=0.015
>epoch=346, lrate=0.500, error=0.015
>epoch=347, lrate=0.500, error=0.015
>epoch=348, lrate=0.500, error=0.015
>epoch=349, lrate=0.500, error=0.015
>epoch=350, lrate=0.500, error=0.015
>epoch=351, lrate=0.500, error=0.015
>epoch=352, lrate=0.500, error=0.015
>epoch=353, lrate=0.500, error=0.015
>epoch=354, lrate=0.500, error=0.015
>epoch=355, lrate=0.500, error=0.015
>epoch=356, lrate=0.500, error=0.015
>epoch=357, lrate=0.500, error=0.015
>epoch=358, lrate=0.500, error=0.015
>epoch=359, lrate=0.500, error=0.015
>epoch=360, lrate=0.500, error=0.015
>epoch=361, lrate=0.500, error=0.015
>epoch=362, lrate=0.500, error=0.014
>epoch=363, lrate=0.500, error=0.014
>epoch=364, lrate=0.500, error=0.014
>epoch=365, lrate=0.500, error=0.014
>epoch=366, lrate=0.500, error=0.014
>epoch=367, lrate=0.500, error=0.014
>epoch=368, lrate=0.500, error=0.014
>epoch=369, lrate=0.500, error=0.014
>epoch=370, lrate=0.500, error=0.014
>epoch=371, lrate=0.500, error=0.014
>epoch=372, lrate=0.500, error=0.014
>epoch=373, lrate=0.500, error=0.014
>epoch=374, lrate=0.500, error=0.014
>epoch=375, lrate=0.500, error=0.014
>epoch=376, lrate=0.500, error=0.014
>epoch=377, lrate=0.500, error=0.014
>epoch=378, lrate=0.500, error=0.014
>epoch=379, lrate=0.500, error=0.014
>epoch=380, lrate=0.500, error=0.014
>epoch=381, lrate=0.500, error=0.014
>epoch=382, lrate=0.500, error=0.014
>epoch=383, lrate=0.500, error=0.014
>epoch=384, lrate=0.500, error=0.013
>epoch=385, lrate=0.500, error=0.013
>epoch=386, lrate=0.500, error=0.013
```

```
>epoch=387, lrate=0.500, error=0.013
>epoch=388, lrate=0.500, error=0.013
>epoch=389, lrate=0.500, error=0.013
>epoch=390, lrate=0.500, error=0.013
>epoch=391, lrate=0.500, error=0.013
>epoch=392, lrate=0.500, error=0.013
>epoch=393, lrate=0.500, error=0.013
>epoch=394, lrate=0.500, error=0.013
>epoch=395, lrate=0.500, error=0.013
>epoch=396, lrate=0.500, error=0.013
>epoch=397, lrate=0.500, error=0.013
>epoch=398, lrate=0.500, error=0.013
>epoch=399, lrate=0.500, error=0.013
>epoch=400, lrate=0.500, error=0.013
>epoch=401, lrate=0.500, error=0.013
>epoch=402, lrate=0.500, error=0.013
>epoch=403, lrate=0.500, error=0.013
>epoch=404, lrate=0.500, error=0.013
>epoch=405, lrate=0.500, error=0.013
>epoch=406, lrate=0.500, error=0.013
>epoch=407, lrate=0.500, error=0.013
>epoch=408, lrate=0.500, error=0.013
>epoch=409, lrate=0.500, error=0.012
>epoch=410, lrate=0.500, error=0.012
>epoch=411, lrate=0.500, error=0.012
>epoch=412, lrate=0.500, error=0.012
>epoch=413, lrate=0.500, error=0.012
>epoch=414, lrate=0.500, error=0.012
>epoch=415, lrate=0.500, error=0.012
>epoch=416, lrate=0.500, error=0.012
>epoch=417, lrate=0.500, error=0.012
>epoch=418, lrate=0.500, error=0.012
>epoch=419, lrate=0.500, error=0.012
>epoch=420, lrate=0.500, error=0.012
>epoch=421, lrate=0.500, error=0.012
>epoch=422, lrate=0.500, error=0.012
>epoch=423, lrate=0.500, error=0.012
>epoch=424, lrate=0.500, error=0.012
>epoch=425, lrate=0.500, error=0.012
>epoch=426, lrate=0.500, error=0.012
>epoch=427, lrate=0.500, error=0.012
>epoch=428, lrate=0.500, error=0.012
>epoch=429, lrate=0.500, error=0.012
>epoch=430, lrate=0.500, error=0.012
>epoch=431, lrate=0.500, error=0.012
>epoch=432, lrate=0.500, error=0.012
>epoch=433, lrate=0.500, error=0.012
>epoch=434, lrate=0.500, error=0.012
>epoch=435, lrate=0.500, error=0.012
>epoch=436, lrate=0.500, error=0.012
```

```
>epoch=437, lrate=0.500, error=0.012
>epoch=438, lrate=0.500, error=0.011
>epoch=439, lrate=0.500, error=0.011
>epoch=440, lrate=0.500, error=0.011
>epoch=441, lrate=0.500, error=0.011
>epoch=442, lrate=0.500, error=0.011
>epoch=443, lrate=0.500, error=0.011
>epoch=444, lrate=0.500, error=0.011
>epoch=445, lrate=0.500, error=0.011
>epoch=446, lrate=0.500, error=0.011
>epoch=447, lrate=0.500, error=0.011
>epoch=448, lrate=0.500, error=0.011
>epoch=449, lrate=0.500, error=0.011
>epoch=450, lrate=0.500, error=0.011
>epoch=451, lrate=0.500, error=0.011
>epoch=452, lrate=0.500, error=0.011
>epoch=453, lrate=0.500, error=0.011
>epoch=454, lrate=0.500, error=0.011
>epoch=455, lrate=0.500, error=0.011
>epoch=456, lrate=0.500, error=0.011
>epoch=457, lrate=0.500, error=0.011
>epoch=458, lrate=0.500, error=0.011
>epoch=459, lrate=0.500, error=0.011
>epoch=460, lrate=0.500, error=0.011
>epoch=461, lrate=0.500, error=0.011
>epoch=462, lrate=0.500, error=0.011
>epoch=463, lrate=0.500, error=0.011
>epoch=464, lrate=0.500, error=0.011
>epoch=465, lrate=0.500, error=0.011
>epoch=466, lrate=0.500, error=0.011
>epoch=467, lrate=0.500, error=0.011
>epoch=468, lrate=0.500, error=0.011
>epoch=469, lrate=0.500, error=0.011
>epoch=470, lrate=0.500, error=0.011
>epoch=471, lrate=0.500, error=0.011
>epoch=472, lrate=0.500, error=0.011
>epoch=473, lrate=0.500, error=0.010
>epoch=474, lrate=0.500, error=0.010
>epoch=475, lrate=0.500, error=0.010
>epoch=476, lrate=0.500, error=0.010
>epoch=477, lrate=0.500, error=0.010
>epoch=478, lrate=0.500, error=0.010
>epoch=479, lrate=0.500, error=0.010
>epoch=480, lrate=0.500, error=0.010
>epoch=481, lrate=0.500, error=0.010
>epoch=482, lrate=0.500, error=0.010
>epoch=483, lrate=0.500, error=0.010
>epoch=484, lrate=0.500, error=0.010
>epoch=485, lrate=0.500, error=0.010
>epoch=486, lrate=0.500, error=0.010
```

```
>epoch=487, lrate=0.500, error=0.010
>epoch=488, lrate=0.500, error=0.010
>epoch=489, lrate=0.500, error=0.010
>epoch=490, lrate=0.500, error=0.010
>epoch=491, lrate=0.500, error=0.010
>epoch=492, lrate=0.500, error=0.010
>epoch=493, lrate=0.500, error=0.010
>epoch=494, lrate=0.500, error=0.010
>epoch=495, lrate=0.500, error=0.010
>epoch=496, lrate=0.500, error=0.010
>epoch=497, lrate=0.500, error=0.010
>epoch=498, lrate=0.500, error=0.010
>epoch=499, lrate=0.500, error=0.010
>epoch=500, lrate=0.500, error=0.010
>epoch=501, lrate=0.500, error=0.010
>epoch=502, lrate=0.500, error=0.010
>epoch=503, lrate=0.500, error=0.010
>epoch=504, lrate=0.500, error=0.010
>epoch=505, lrate=0.500, error=0.010
>epoch=506, lrate=0.500, error=0.010
>epoch=507, lrate=0.500, error=0.010
>epoch=508, lrate=0.500, error=0.010
>epoch=509, lrate=0.500, error=0.010
>epoch=510, lrate=0.500, error=0.010
>epoch=511, lrate=0.500, error=0.010
>epoch=512, lrate=0.500, error=0.010
>epoch=513, lrate=0.500, error=0.010
>epoch=514, lrate=0.500, error=0.010
>epoch=515, lrate=0.500, error=0.009
>epoch=516, lrate=0.500, error=0.009
>epoch=517, lrate=0.500, error=0.009
>epoch=518, lrate=0.500, error=0.009
>epoch=519, lrate=0.500, error=0.009
>epoch=520, lrate=0.500, error=0.009
>epoch=521, lrate=0.500, error=0.009
>epoch=522, lrate=0.500, error=0.009
>epoch=523, lrate=0.500, error=0.009
>epoch=524, lrate=0.500, error=0.009
>epoch=525, lrate=0.500, error=0.009
>epoch=526, lrate=0.500, error=0.009
>epoch=527, lrate=0.500, error=0.009
>epoch=528, lrate=0.500, error=0.009
>epoch=529, lrate=0.500, error=0.009
>epoch=530, lrate=0.500, error=0.009
>epoch=531, lrate=0.500, error=0.009
>epoch=532, lrate=0.500, error=0.009
>epoch=533, lrate=0.500, error=0.009
>epoch=534, lrate=0.500, error=0.009
>epoch=535, lrate=0.500, error=0.009
>epoch=536, lrate=0.500, error=0.009
```

```
>epoch=537, lrate=0.500, error=0.009
>epoch=538, lrate=0.500, error=0.009
>epoch=539, lrate=0.500, error=0.009
>epoch=540, lrate=0.500, error=0.009
>epoch=541, lrate=0.500, error=0.009
>epoch=542, lrate=0.500, error=0.009
>epoch=543, lrate=0.500, error=0.009
>epoch=544, lrate=0.500, error=0.009
>epoch=545, lrate=0.500, error=0.009
>epoch=546, lrate=0.500, error=0.009
>epoch=547, lrate=0.500, error=0.009
>epoch=548, lrate=0.500, error=0.009
>epoch=549, lrate=0.500, error=0.009
>epoch=550, lrate=0.500, error=0.009
>epoch=551, lrate=0.500, error=0.009
>epoch=552, lrate=0.500, error=0.009
>epoch=553, lrate=0.500, error=0.009
>epoch=554, lrate=0.500, error=0.009
>epoch=555, lrate=0.500, error=0.009
>epoch=556, lrate=0.500, error=0.009
>epoch=557, lrate=0.500, error=0.009
>epoch=558, lrate=0.500, error=0.009
>epoch=559, lrate=0.500, error=0.009
>epoch=560, lrate=0.500, error=0.009
>epoch=561, lrate=0.500, error=0.009
>epoch=562, lrate=0.500, error=0.009
>epoch=563, lrate=0.500, error=0.009
>epoch=564, lrate=0.500, error=0.009
>epoch=565, lrate=0.500, error=0.009
>epoch=566, lrate=0.500, error=0.009
>epoch=567, lrate=0.500, error=0.008
>epoch=568, lrate=0.500, error=0.008
>epoch=569, lrate=0.500, error=0.008
>epoch=570, lrate=0.500, error=0.008
>epoch=571, lrate=0.500, error=0.008
>epoch=572, lrate=0.500, error=0.008
>epoch=573, lrate=0.500, error=0.008
>epoch=574, lrate=0.500, error=0.008
>epoch=575, lrate=0.500, error=0.008
>epoch=576, lrate=0.500, error=0.008
>epoch=577, lrate=0.500, error=0.008
>epoch=578, lrate=0.500, error=0.008
>epoch=579, lrate=0.500, error=0.008
>epoch=580, lrate=0.500, error=0.008
>epoch=581, lrate=0.500, error=0.008
>epoch=582, lrate=0.500, error=0.008
>epoch=583, lrate=0.500, error=0.008
>epoch=584, lrate=0.500, error=0.008
>epoch=585, lrate=0.500, error=0.008
>epoch=586, lrate=0.500, error=0.008
```

```
>epoch=587, lrate=0.500, error=0.008
>epoch=588, lrate=0.500, error=0.008
>epoch=589, lrate=0.500, error=0.008
>epoch=590, lrate=0.500, error=0.008
>epoch=591, lrate=0.500, error=0.008
>epoch=592, lrate=0.500, error=0.008
>epoch=593, lrate=0.500, error=0.008
>epoch=594, lrate=0.500, error=0.008
>epoch=595, lrate=0.500, error=0.008
>epoch=596, lrate=0.500, error=0.008
>epoch=597, lrate=0.500, error=0.008
>epoch=598, lrate=0.500, error=0.008
>epoch=599, lrate=0.500, error=0.008
>epoch=600, lrate=0.500, error=0.008
>epoch=601, lrate=0.500, error=0.008
>epoch=602, lrate=0.500, error=0.008
>epoch=603, lrate=0.500, error=0.008
>epoch=604, lrate=0.500, error=0.008
>epoch=605, lrate=0.500, error=0.008
>epoch=606, lrate=0.500, error=0.008
>epoch=607, lrate=0.500, error=0.008
>epoch=608, lrate=0.500, error=0.008
>epoch=609, lrate=0.500, error=0.008
>epoch=610, lrate=0.500, error=0.008
>epoch=611, lrate=0.500, error=0.008
>epoch=612, lrate=0.500, error=0.008
>epoch=613, lrate=0.500, error=0.008
>epoch=614, lrate=0.500, error=0.008
>epoch=615, lrate=0.500, error=0.008
>epoch=616, lrate=0.500, error=0.008
>epoch=617, lrate=0.500, error=0.008
>epoch=618, lrate=0.500, error=0.008
>epoch=619, lrate=0.500, error=0.008
>epoch=620, lrate=0.500, error=0.008
>epoch=621, lrate=0.500, error=0.008
>epoch=622, lrate=0.500, error=0.008
>epoch=623, lrate=0.500, error=0.008
>epoch=624, lrate=0.500, error=0.008
>epoch=625, lrate=0.500, error=0.008
>epoch=626, lrate=0.500, error=0.008
>epoch=627, lrate=0.500, error=0.008
>epoch=628, lrate=0.500, error=0.008
>epoch=629, lrate=0.500, error=0.008
>epoch=630, lrate=0.500, error=0.008
>epoch=631, lrate=0.500, error=0.008
>epoch=632, lrate=0.500, error=0.008
>epoch=633, lrate=0.500, error=0.007
>epoch=634, lrate=0.500, error=0.007
>epoch=635, lrate=0.500, error=0.007
>epoch=636, lrate=0.500, error=0.007
```

```
>epoch=637, lrate=0.500, error=0.007
>epoch=638, lrate=0.500, error=0.007
>epoch=639, lrate=0.500, error=0.007
>epoch=640, lrate=0.500, error=0.007
>epoch=641, lrate=0.500, error=0.007
>epoch=642, lrate=0.500, error=0.007
>epoch=643, lrate=0.500, error=0.007
>epoch=644, lrate=0.500, error=0.007
>epoch=645, lrate=0.500, error=0.007
>epoch=646, lrate=0.500, error=0.007
>epoch=647, lrate=0.500, error=0.007
>epoch=648, lrate=0.500, error=0.007
>epoch=649, lrate=0.500, error=0.007
>epoch=650, lrate=0.500, error=0.007
>epoch=651, lrate=0.500, error=0.007
>epoch=652, lrate=0.500, error=0.007
>epoch=653, lrate=0.500, error=0.007
>epoch=654, lrate=0.500, error=0.007
>epoch=655, lrate=0.500, error=0.007
>epoch=656, lrate=0.500, error=0.007
>epoch=657, lrate=0.500, error=0.007
>epoch=658, lrate=0.500, error=0.007
>epoch=659, lrate=0.500, error=0.007
>epoch=660, lrate=0.500, error=0.007
>epoch=661, lrate=0.500, error=0.007
>epoch=662, lrate=0.500, error=0.007
>epoch=663, lrate=0.500, error=0.007
>epoch=664, lrate=0.500, error=0.007
>epoch=665, lrate=0.500, error=0.007
>epoch=666, lrate=0.500, error=0.007
>epoch=667, lrate=0.500, error=0.007
>epoch=668, lrate=0.500, error=0.007
>epoch=669, lrate=0.500, error=0.007
>epoch=670, lrate=0.500, error=0.007
>epoch=671, lrate=0.500, error=0.007
>epoch=672, lrate=0.500, error=0.007
>epoch=673, lrate=0.500, error=0.007
>epoch=674, lrate=0.500, error=0.007
>epoch=675, lrate=0.500, error=0.007
>epoch=676, lrate=0.500, error=0.007
>epoch=677, lrate=0.500, error=0.007
>epoch=678, lrate=0.500, error=0.007
>epoch=679, lrate=0.500, error=0.007
>epoch=680, lrate=0.500, error=0.007
>epoch=681, lrate=0.500, error=0.007
>epoch=682, lrate=0.500, error=0.007
>epoch=683, lrate=0.500, error=0.007
>epoch=684, lrate=0.500, error=0.007
>epoch=685, lrate=0.500, error=0.007
>epoch=686, lrate=0.500, error=0.007
```

```
>epoch=687, lrate=0.500, error=0.007
>epoch=688, lrate=0.500, error=0.007
>epoch=689, lrate=0.500, error=0.007
>epoch=690, lrate=0.500, error=0.007
>epoch=691, lrate=0.500, error=0.007
>epoch=692, lrate=0.500, error=0.007
>epoch=693, lrate=0.500, error=0.007
>epoch=694, lrate=0.500, error=0.007
>epoch=695, lrate=0.500, error=0.007
>epoch=696, lrate=0.500, error=0.007
>epoch=697, lrate=0.500, error=0.007
>epoch=698, lrate=0.500, error=0.007
>epoch=699, lrate=0.500, error=0.007
>epoch=700, lrate=0.500, error=0.007
>epoch=701, lrate=0.500, error=0.007
>epoch=702, lrate=0.500, error=0.007
>epoch=703, lrate=0.500, error=0.007
>epoch=704, lrate=0.500, error=0.007
>epoch=705, lrate=0.500, error=0.007
>epoch=706, lrate=0.500, error=0.007
>epoch=707, lrate=0.500, error=0.007
>epoch=708, lrate=0.500, error=0.007
>epoch=709, lrate=0.500, error=0.007
>epoch=710, lrate=0.500, error=0.007
>epoch=711, lrate=0.500, error=0.007
>epoch=712, lrate=0.500, error=0.007
>epoch=713, lrate=0.500, error=0.007
>epoch=714, lrate=0.500, error=0.007
>epoch=715, lrate=0.500, error=0.007
>epoch=716, lrate=0.500, error=0.007
>epoch=717, lrate=0.500, error=0.007
>epoch=718, lrate=0.500, error=0.006
>epoch=719, lrate=0.500, error=0.006
>epoch=720, lrate=0.500, error=0.006
>epoch=721, lrate=0.500, error=0.006
>epoch=722, lrate=0.500, error=0.006
>epoch=723, lrate=0.500, error=0.006
>epoch=724, lrate=0.500, error=0.006
>epoch=725, lrate=0.500, error=0.006
>epoch=726, lrate=0.500, error=0.006
>epoch=727, lrate=0.500, error=0.006
>epoch=728, lrate=0.500, error=0.006
>epoch=729, lrate=0.500, error=0.006
>epoch=730, lrate=0.500, error=0.006
>epoch=731, lrate=0.500, error=0.006
>epoch=732, lrate=0.500, error=0.006
>epoch=733, lrate=0.500, error=0.006
>epoch=734, lrate=0.500, error=0.006
>epoch=735, lrate=0.500, error=0.006
>epoch=736, lrate=0.500, error=0.006
```

```
>epoch=737, lrate=0.500, error=0.006
>epoch=738, lrate=0.500, error=0.006
>epoch=739, lrate=0.500, error=0.006
>epoch=740, lrate=0.500, error=0.006
>epoch=741, lrate=0.500, error=0.006
>epoch=742, lrate=0.500, error=0.006
>epoch=743, lrate=0.500, error=0.006
>epoch=744, lrate=0.500, error=0.006
>epoch=745, lrate=0.500, error=0.006
>epoch=746, lrate=0.500, error=0.006
>epoch=747, lrate=0.500, error=0.006
>epoch=748, lrate=0.500, error=0.006
>epoch=749, lrate=0.500, error=0.006
>epoch=750, lrate=0.500, error=0.006
>epoch=751, lrate=0.500, error=0.006
>epoch=752, lrate=0.500, error=0.006
>epoch=753, lrate=0.500, error=0.006
>epoch=754, lrate=0.500, error=0.006
>epoch=755, lrate=0.500, error=0.006
>epoch=756, lrate=0.500, error=0.006
>epoch=757, lrate=0.500, error=0.006
>epoch=758, lrate=0.500, error=0.006
>epoch=759, lrate=0.500, error=0.006
>epoch=760, lrate=0.500, error=0.006
>epoch=761, lrate=0.500, error=0.006
>epoch=762, lrate=0.500, error=0.006
>epoch=763, lrate=0.500, error=0.006
>epoch=764, lrate=0.500, error=0.006
>epoch=765, lrate=0.500, error=0.006
>epoch=766, lrate=0.500, error=0.006
>epoch=767, lrate=0.500, error=0.006
>epoch=768, lrate=0.500, error=0.006
>epoch=769, lrate=0.500, error=0.006
>epoch=770, lrate=0.500, error=0.006
>epoch=771, lrate=0.500, error=0.006
>epoch=772, lrate=0.500, error=0.006
>epoch=773, lrate=0.500, error=0.006
>epoch=774, lrate=0.500, error=0.006
>epoch=775, lrate=0.500, error=0.006
>epoch=776, lrate=0.500, error=0.006
>epoch=777, lrate=0.500, error=0.006
>epoch=778, lrate=0.500, error=0.006
>epoch=779, lrate=0.500, error=0.006
>epoch=780, lrate=0.500, error=0.006
>epoch=781, lrate=0.500, error=0.006
>epoch=782, lrate=0.500, error=0.006
>epoch=783, lrate=0.500, error=0.006
>epoch=784, lrate=0.500, error=0.006
>epoch=785, lrate=0.500, error=0.006
>epoch=786, lrate=0.500, error=0.006
```

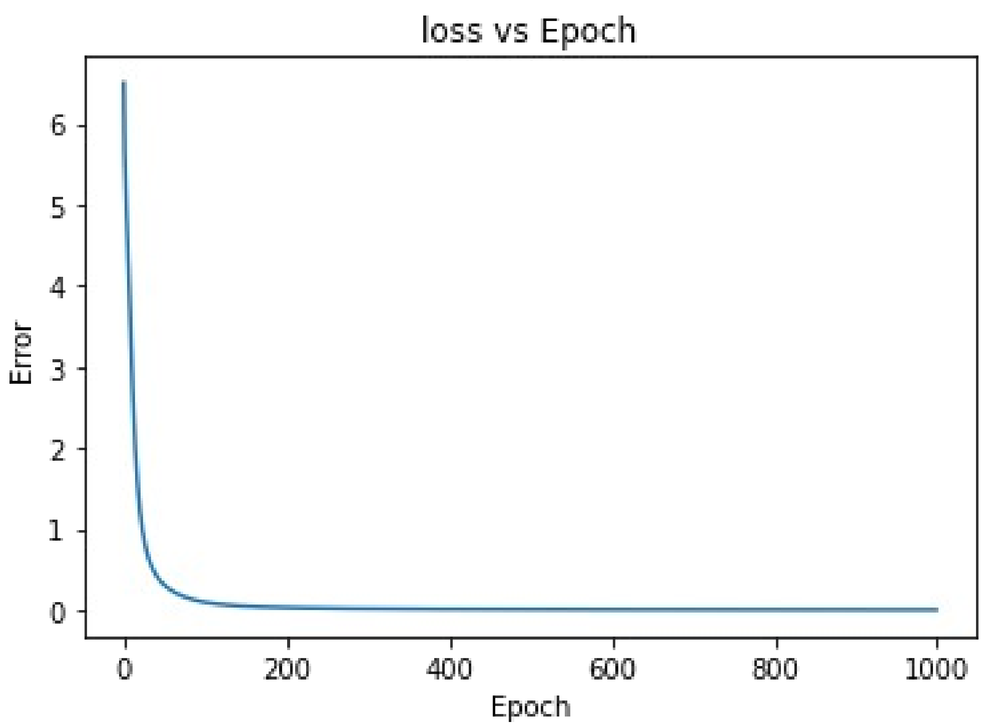
```
>epoch=787, lrate=0.500, error=0.006
>epoch=788, lrate=0.500, error=0.006
>epoch=789, lrate=0.500, error=0.006
>epoch=790, lrate=0.500, error=0.006
>epoch=791, lrate=0.500, error=0.006
>epoch=792, lrate=0.500, error=0.006
>epoch=793, lrate=0.500, error=0.006
>epoch=794, lrate=0.500, error=0.006
>epoch=795, lrate=0.500, error=0.006
>epoch=796, lrate=0.500, error=0.006
>epoch=797, lrate=0.500, error=0.006
>epoch=798, lrate=0.500, error=0.006
>epoch=799, lrate=0.500, error=0.006
>epoch=800, lrate=0.500, error=0.006
>epoch=801, lrate=0.500, error=0.006
>epoch=802, lrate=0.500, error=0.006
>epoch=803, lrate=0.500, error=0.006
>epoch=804, lrate=0.500, error=0.006
>epoch=805, lrate=0.500, error=0.006
>epoch=806, lrate=0.500, error=0.006
>epoch=807, lrate=0.500, error=0.006
>epoch=808, lrate=0.500, error=0.006
>epoch=809, lrate=0.500, error=0.006
>epoch=810, lrate=0.500, error=0.006
>epoch=811, lrate=0.500, error=0.006
>epoch=812, lrate=0.500, error=0.006
>epoch=813, lrate=0.500, error=0.006
>epoch=814, lrate=0.500, error=0.006
>epoch=815, lrate=0.500, error=0.006
>epoch=816, lrate=0.500, error=0.006
>epoch=817, lrate=0.500, error=0.006
>epoch=818, lrate=0.500, error=0.006
>epoch=819, lrate=0.500, error=0.006
>epoch=820, lrate=0.500, error=0.006
>epoch=821, lrate=0.500, error=0.006
>epoch=822, lrate=0.500, error=0.006
>epoch=823, lrate=0.500, error=0.006
>epoch=824, lrate=0.500, error=0.006
>epoch=825, lrate=0.500, error=0.006
>epoch=826, lrate=0.500, error=0.006
>epoch=827, lrate=0.500, error=0.006
>epoch=828, lrate=0.500, error=0.006
>epoch=829, lrate=0.500, error=0.006
>epoch=830, lrate=0.500, error=0.006
>epoch=831, lrate=0.500, error=0.006
>epoch=832, lrate=0.500, error=0.006
>epoch=833, lrate=0.500, error=0.006
>epoch=834, lrate=0.500, error=0.005
>epoch=835, lrate=0.500, error=0.005
>epoch=836, lrate=0.500, error=0.005
```

```
>epoch=837, lrate=0.500, error=0.005
>epoch=838, lrate=0.500, error=0.005
>epoch=839, lrate=0.500, error=0.005
>epoch=840, lrate=0.500, error=0.005
>epoch=841, lrate=0.500, error=0.005
>epoch=842, lrate=0.500, error=0.005
>epoch=843, lrate=0.500, error=0.005
>epoch=844, lrate=0.500, error=0.005
>epoch=845, lrate=0.500, error=0.005
>epoch=846, lrate=0.500, error=0.005
>epoch=847, lrate=0.500, error=0.005
>epoch=848, lrate=0.500, error=0.005
>epoch=849, lrate=0.500, error=0.005
>epoch=850, lrate=0.500, error=0.005
>epoch=851, lrate=0.500, error=0.005
>epoch=852, lrate=0.500, error=0.005
>epoch=853, lrate=0.500, error=0.005
>epoch=854, lrate=0.500, error=0.005
>epoch=855, lrate=0.500, error=0.005
>epoch=856, lrate=0.500, error=0.005
>epoch=857, lrate=0.500, error=0.005
>epoch=858, lrate=0.500, error=0.005
>epoch=859, lrate=0.500, error=0.005
>epoch=860, lrate=0.500, error=0.005
>epoch=861, lrate=0.500, error=0.005
>epoch=862, lrate=0.500, error=0.005
>epoch=863, lrate=0.500, error=0.005
>epoch=864, lrate=0.500, error=0.005
>epoch=865, lrate=0.500, error=0.005
>epoch=866, lrate=0.500, error=0.005
>epoch=867, lrate=0.500, error=0.005
>epoch=868, lrate=0.500, error=0.005
>epoch=869, lrate=0.500, error=0.005
>epoch=870, lrate=0.500, error=0.005
>epoch=871, lrate=0.500, error=0.005
>epoch=872, lrate=0.500, error=0.005
>epoch=873, lrate=0.500, error=0.005
>epoch=874, lrate=0.500, error=0.005
>epoch=875, lrate=0.500, error=0.005
>epoch=876, lrate=0.500, error=0.005
>epoch=877, lrate=0.500, error=0.005
>epoch=878, lrate=0.500, error=0.005
>epoch=879, lrate=0.500, error=0.005
>epoch=880, lrate=0.500, error=0.005
>epoch=881, lrate=0.500, error=0.005
>epoch=882, lrate=0.500, error=0.005
>epoch=883, lrate=0.500, error=0.005
>epoch=884, lrate=0.500, error=0.005
>epoch=885, lrate=0.500, error=0.005
>epoch=886, lrate=0.500, error=0.005
```

```
>epoch=887, lrate=0.500, error=0.005
>epoch=888, lrate=0.500, error=0.005
>epoch=889, lrate=0.500, error=0.005
>epoch=890, lrate=0.500, error=0.005
>epoch=891, lrate=0.500, error=0.005
>epoch=892, lrate=0.500, error=0.005
>epoch=893, lrate=0.500, error=0.005
>epoch=894, lrate=0.500, error=0.005
>epoch=895, lrate=0.500, error=0.005
>epoch=896, lrate=0.500, error=0.005
>epoch=897, lrate=0.500, error=0.005
>epoch=898, lrate=0.500, error=0.005
>epoch=899, lrate=0.500, error=0.005
>epoch=900, lrate=0.500, error=0.005
>epoch=901, lrate=0.500, error=0.005
>epoch=902, lrate=0.500, error=0.005
>epoch=903, lrate=0.500, error=0.005
>epoch=904, lrate=0.500, error=0.005
>epoch=905, lrate=0.500, error=0.005
>epoch=906, lrate=0.500, error=0.005
>epoch=907, lrate=0.500, error=0.005
>epoch=908, lrate=0.500, error=0.005
>epoch=909, lrate=0.500, error=0.005
>epoch=910, lrate=0.500, error=0.005
>epoch=911, lrate=0.500, error=0.005
>epoch=912, lrate=0.500, error=0.005
>epoch=913, lrate=0.500, error=0.005
>epoch=914, lrate=0.500, error=0.005
>epoch=915, lrate=0.500, error=0.005
>epoch=916, lrate=0.500, error=0.005
>epoch=917, lrate=0.500, error=0.005
>epoch=918, lrate=0.500, error=0.005
>epoch=919, lrate=0.500, error=0.005
>epoch=920, lrate=0.500, error=0.005
>epoch=921, lrate=0.500, error=0.005
>epoch=922, lrate=0.500, error=0.005
>epoch=923, lrate=0.500, error=0.005
>epoch=924, lrate=0.500, error=0.005
>epoch=925, lrate=0.500, error=0.005
>epoch=926, lrate=0.500, error=0.005
>epoch=927, lrate=0.500, error=0.005
>epoch=928, lrate=0.500, error=0.005
>epoch=929, lrate=0.500, error=0.005
>epoch=930, lrate=0.500, error=0.005
>epoch=931, lrate=0.500, error=0.005
>epoch=932, lrate=0.500, error=0.005
>epoch=933, lrate=0.500, error=0.005
>epoch=934, lrate=0.500, error=0.005
>epoch=935, lrate=0.500, error=0.005
>epoch=936, lrate=0.500, error=0.005
```

```
>epoch=937, lrate=0.500, error=0.005
>epoch=938, lrate=0.500, error=0.005
>epoch=939, lrate=0.500, error=0.005
>epoch=940, lrate=0.500, error=0.005
>epoch=941, lrate=0.500, error=0.005
>epoch=942, lrate=0.500, error=0.005
>epoch=943, lrate=0.500, error=0.005
>epoch=944, lrate=0.500, error=0.005
>epoch=945, lrate=0.500, error=0.005
>epoch=946, lrate=0.500, error=0.005
>epoch=947, lrate=0.500, error=0.005
>epoch=948, lrate=0.500, error=0.005
>epoch=949, lrate=0.500, error=0.005
>epoch=950, lrate=0.500, error=0.005
>epoch=951, lrate=0.500, error=0.005
>epoch=952, lrate=0.500, error=0.005
>epoch=953, lrate=0.500, error=0.005
>epoch=954, lrate=0.500, error=0.005
>epoch=955, lrate=0.500, error=0.005
>epoch=956, lrate=0.500, error=0.005
>epoch=957, lrate=0.500, error=0.005
>epoch=958, lrate=0.500, error=0.005
>epoch=959, lrate=0.500, error=0.005
>epoch=960, lrate=0.500, error=0.005
>epoch=961, lrate=0.500, error=0.005
>epoch=962, lrate=0.500, error=0.005
>epoch=963, lrate=0.500, error=0.005
>epoch=964, lrate=0.500, error=0.005
>epoch=965, lrate=0.500, error=0.005
>epoch=966, lrate=0.500, error=0.005
>epoch=967, lrate=0.500, error=0.005
>epoch=968, lrate=0.500, error=0.005
>epoch=969, lrate=0.500, error=0.005
>epoch=970, lrate=0.500, error=0.005
>epoch=971, lrate=0.500, error=0.005
>epoch=972, lrate=0.500, error=0.005
>epoch=973, lrate=0.500, error=0.005
>epoch=974, lrate=0.500, error=0.005
>epoch=975, lrate=0.500, error=0.005
>epoch=976, lrate=0.500, error=0.005
>epoch=977, lrate=0.500, error=0.005
>epoch=978, lrate=0.500, error=0.005
>epoch=979, lrate=0.500, error=0.005
>epoch=980, lrate=0.500, error=0.005
>epoch=981, lrate=0.500, error=0.005
>epoch=982, lrate=0.500, error=0.005
>epoch=983, lrate=0.500, error=0.005
>epoch=984, lrate=0.500, error=0.005
>epoch=985, lrate=0.500, error=0.005
>epoch=986, lrate=0.500, error=0.005
```

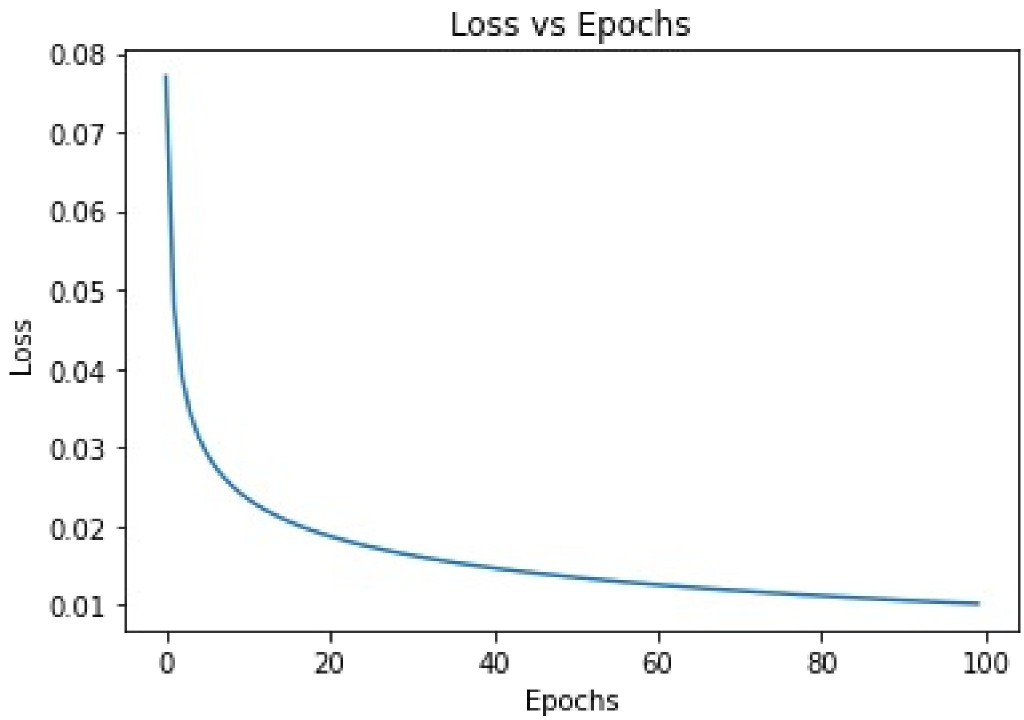
```
>epoch=987, lrate=0.500, error=0.005
>epoch=988, lrate=0.500, error=0.005
>epoch=989, lrate=0.500, error=0.005
>epoch=990, lrate=0.500, error=0.005
>epoch=991, lrate=0.500, error=0.005
>epoch=992, lrate=0.500, error=0.005
>epoch=993, lrate=0.500, error=0.005
>epoch=994, lrate=0.500, error=0.005
>epoch=995, lrate=0.500, error=0.005
>epoch=996, lrate=0.500, error=0.005
>epoch=997, lrate=0.500, error=0.005
>epoch=998, lrate=0.500, error=0.005
>epoch=999, lrate=0.500, error=0.005
[{'weights_vector': [1.1739001268700677, -1.7555553312213197, -
0.3262722929280680115], 'output': 0.9254478213519465, 'delta':
0.00010239853271766191}, {'weights_vector': [-2.259389346141733,
3.1105471980807393, 1.7287382085736973], 'output':
0.009061700941492763, 'delta': -3.1002191017455026e-05}]
[{'weights_vector': [-2.651396276884352, 6.269001182718462, -
1.672352275848774], 'output': 0.016806907550381226, 'delta': -
0.00027772465824087476}, {'weights_vector': [2.720751940173647, -
6.2262583594778995, 1.613061420485804], 'output': 0.9832801875841655,
'delta': 0.00027487806809311713}]
```

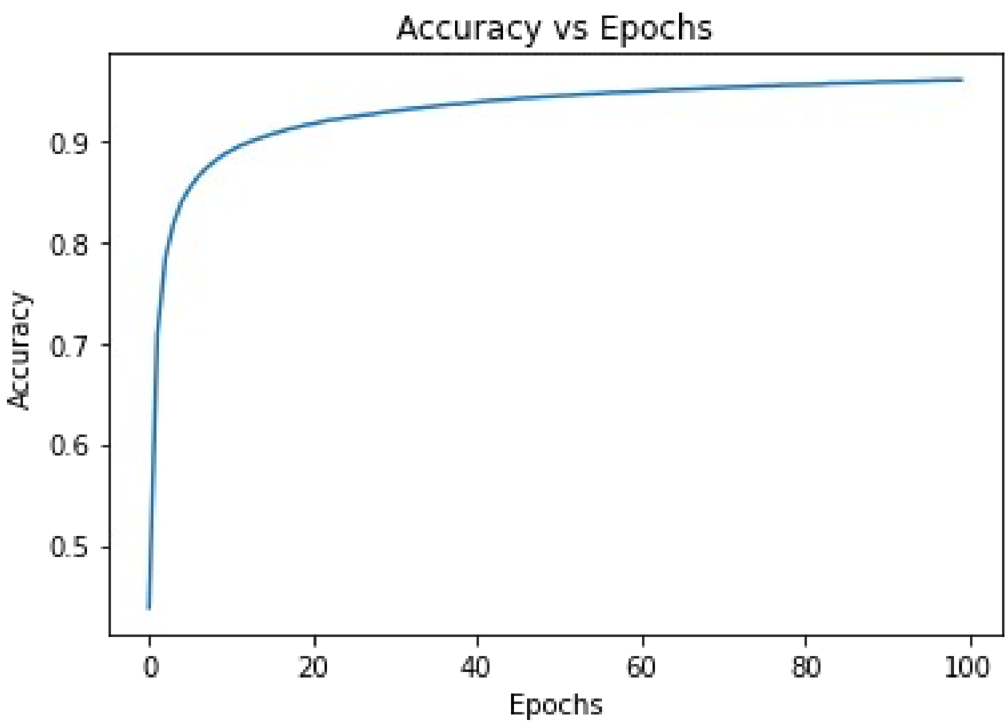



```python
def visualize_loss_epochs(val, ylab):
    plt.plot(val)
    plt.xlabel("Epochs")
    plt.ylabel(ylab)
```

```
    plt.title(ylab + ' vs Epochs')
    plt.show()

# Read MNIST data
train_images, train_labels, test_images, test_labels =
read_mnist(path='/home/sumedh/Desktop/CMSC_733_HW5/MNISTArchive')

# train_features and expected output
train_features = train_images
train_labels_ = train_labels

# Hyperparameters
batch = 64
learning_rate = 1e-3
epochs = 100

# Train and test
x = train_features[:batch]
y = train_labels_[:batch]

# Initialize weights and biases
weights_vector, biases = initialize_network_mnist(train_features, y,
2, [1000, 500, 50])


# Train and test
weights_vector, biases, loss, accuracy = train_mnist(train_features,
train_labels_, weights_vector, biases, batch, epochs, learning_rate)
test_mnist(test_images, test_labels, weights_vector, biases)

# Visualize loss and accuracy
visualize_loss_epochs(loss, 'Loss')
visualize_loss_epochs(accuracy, 'Accuracy')
```

```
Train Accuracy: 95.95951173959445 %
Test Accuracy: 95.39 %
```





Write-up (10 pts)

1. You are required to report a) train error w.r.t epoch, b) train and test accuracy numbers on MNIST dataset at the end of training.

2. Experiment with different number of a) hidden layers b) training epochs and report the ablation study.

1. You are required to report a) train error w.r.t epoch, b) train and test accuracy numbers on MNIST dataset at the end of training.

a. Please see the above plots for train error w.r.t epochs AKA Loss vs Epochs

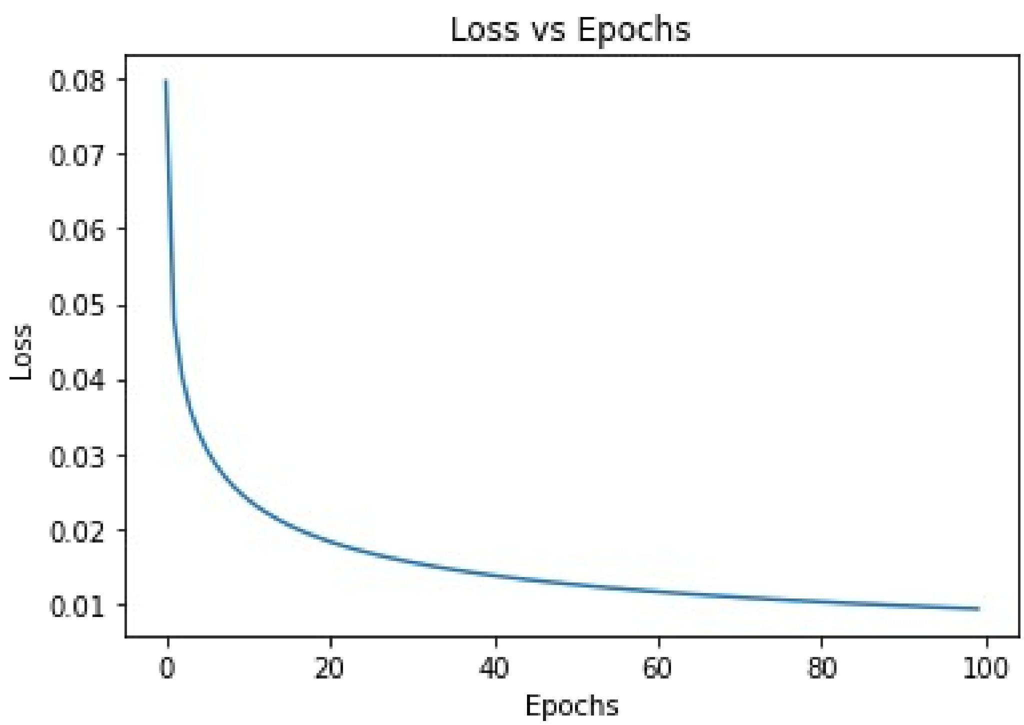
b. Train and Test accuracy numbers on MNIST dataset at the end of training:

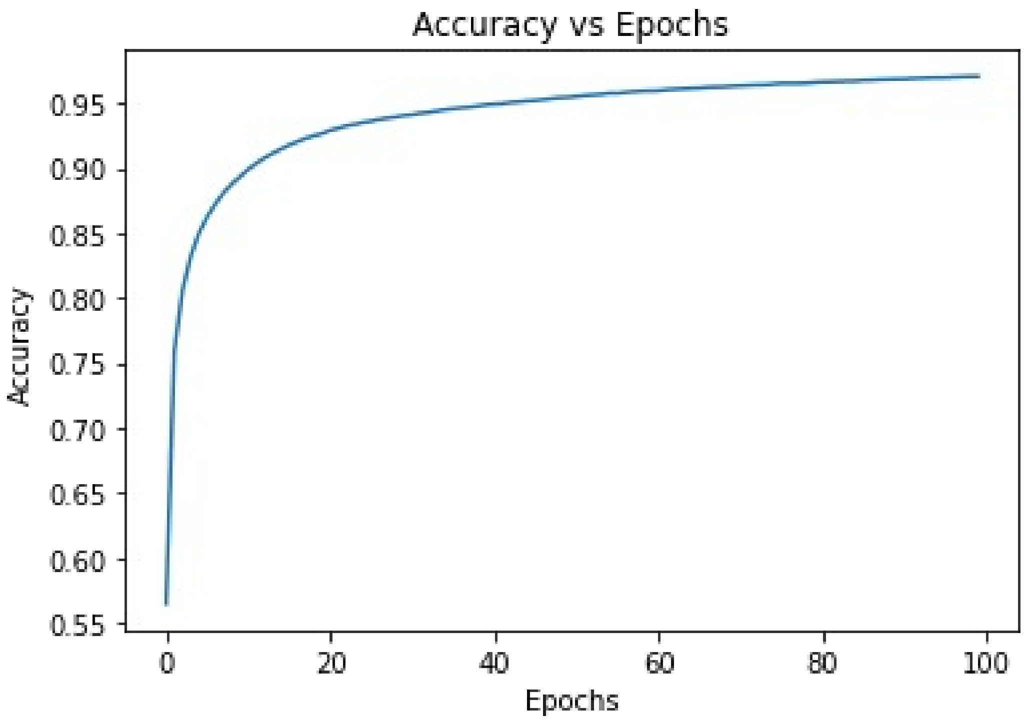
```
Train Accuracy MNIST dataset: 95.96 %
Test Accuracy MNIST dataset: 95.39 %
```

2. Experiment with different number of a) hidden layers b) training epochs and report the ablation study.

```
# Experiment 1 - Varying number of hidden layers

# 1 hidden layer
weights_vector, biases = initialize_network_mnist(train_features, y,
1, [1000, 500, 50])
weights_vector, biases, loss, accuracy = train_mnist(train_features,
train_labels_, weights_vector, biases, batch, epochs, learning_rate)
visualize_loss_epochs(loss, 'Loss')
visualize_loss_epochs(accuracy, 'Accuracy')
```

```
Train Accuracy: 97.05176093916755 %
```



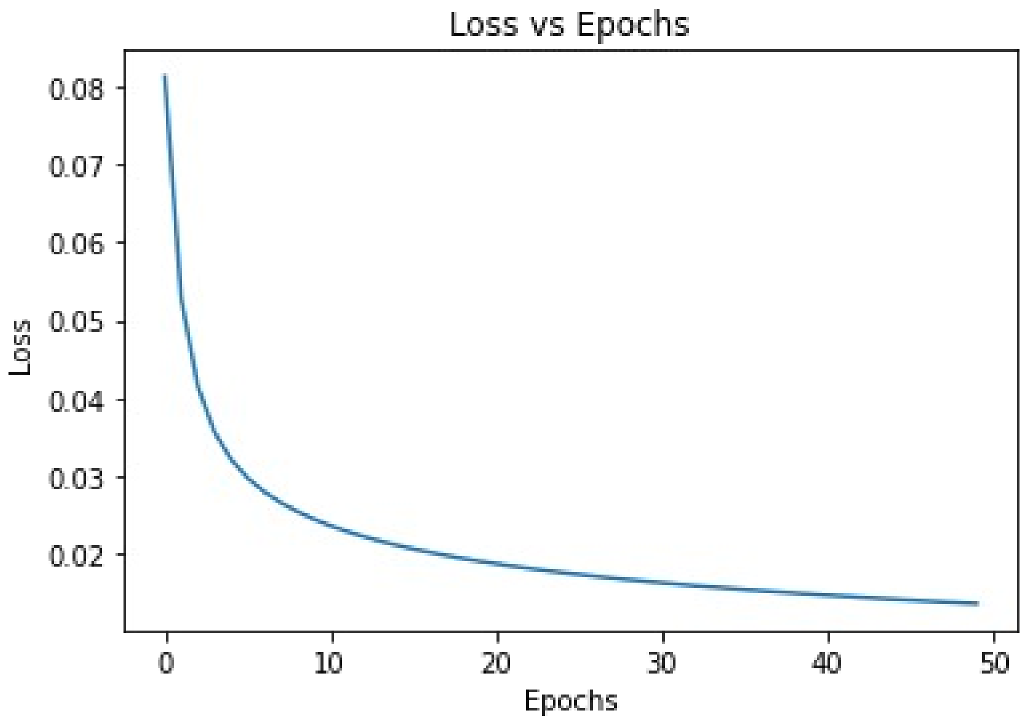


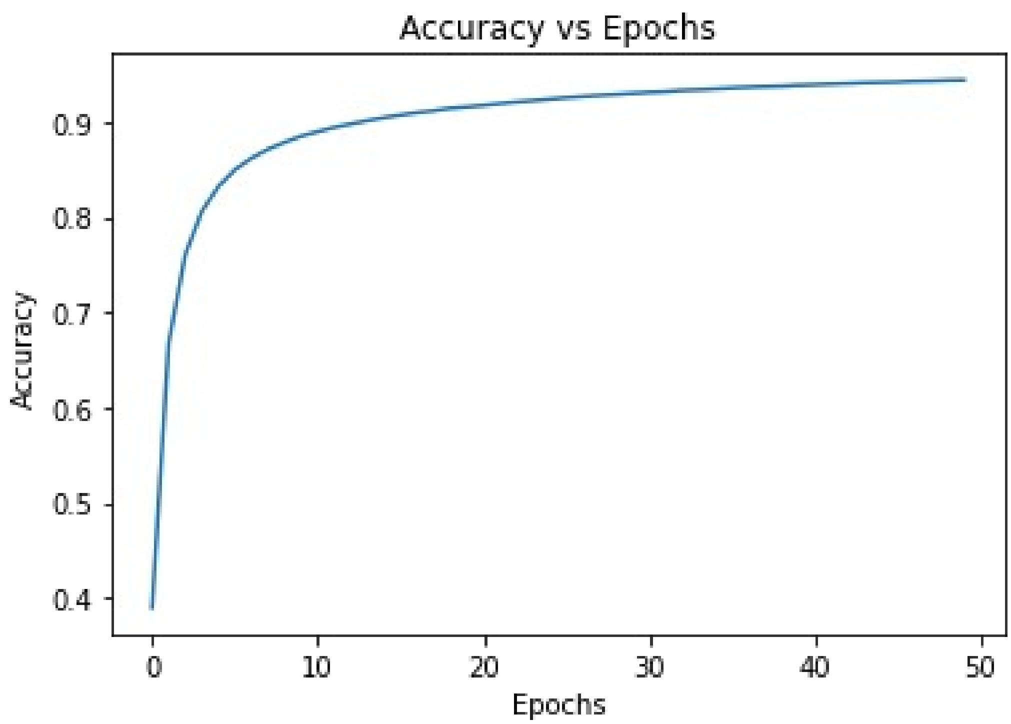

```
test_mnist(test_images, test_labels, weights_vector, biases)
```

```
Test Accuracy: 94.13 %
```

```
# Different number of epochs
epochs = 50
weights_vector, biases = initialize_network_mnist(train_features, y,
2, [1000, 500, 50])
weights_vector, biases, loss, accuracy = train_mnist(train_features,
train_labels_, weights_vector, biases, batch, epochs, learning_rate)
visualize_loss_epochs(loss, 'Loss')
visualize_loss_epochs(accuracy, 'Accuracy')
```

```
Train Accuracy: 94.3920090715048 %
```





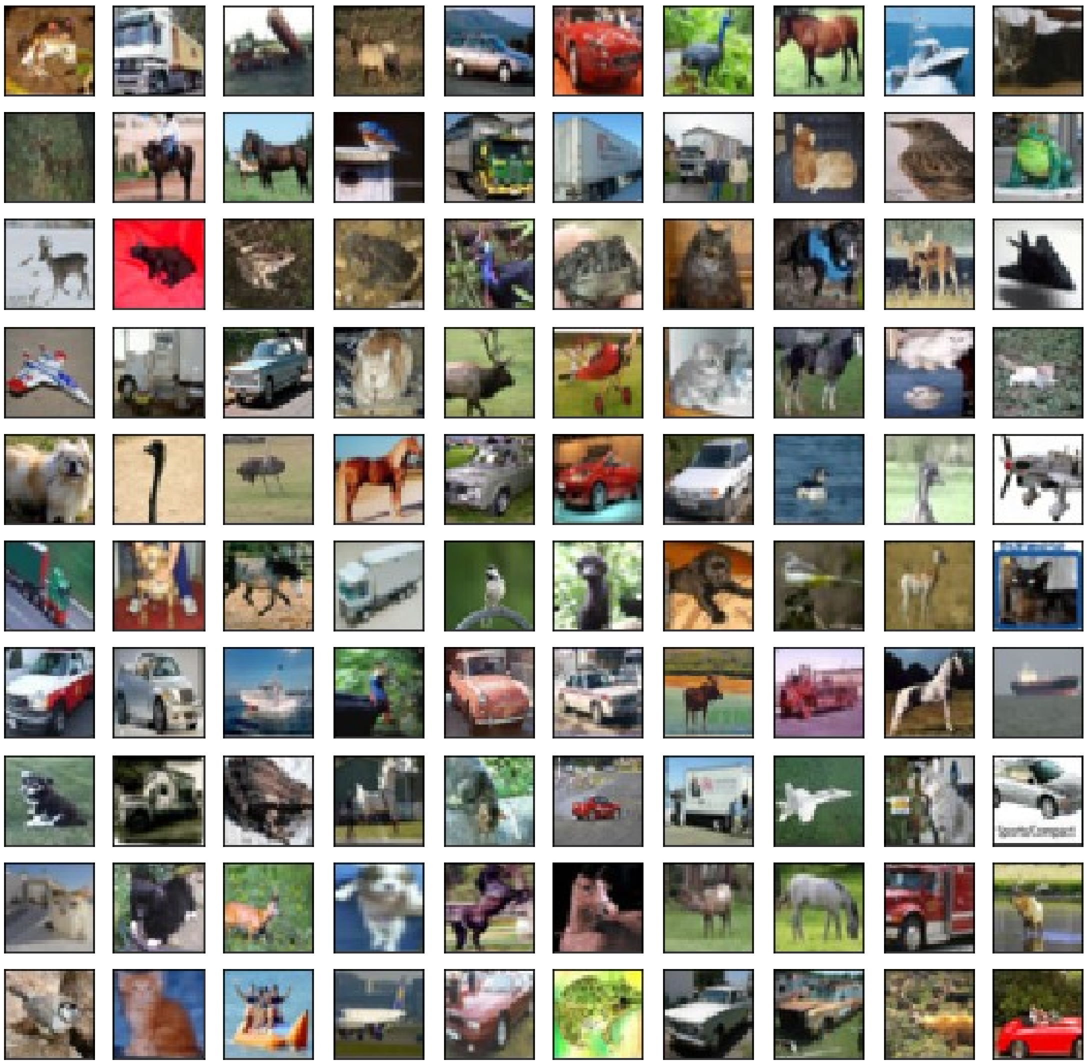

```
test_mnist(test_images, test_labels, weights_vector, biases)
Test Accuracy: 94.13 %
```

Abalation Study:

- More the Dense the network and the Epochs, more is the accuracy of the model

Part 2: Training an Image Classifier

##Overview CIFAR10 dataset will be used to train an image classifier.



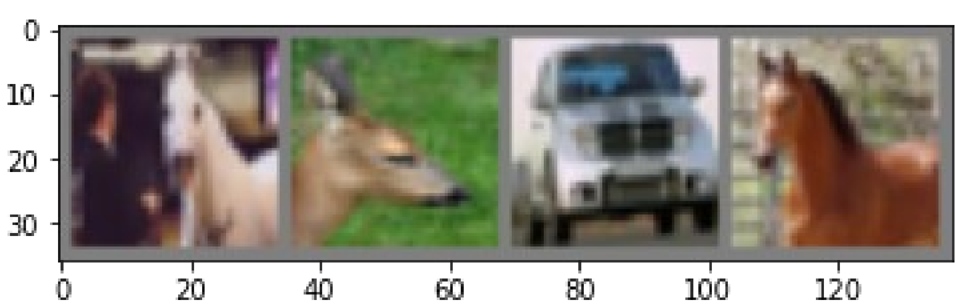
##Data Using `torchvision`, it's extremely easy to load `CIFAR10`.

```
## The output of torchvision datasets are PILImage images of range [0,
1]. We transform them to Tensors of normalized range [-1, 1].
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4
```

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True,
transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,
batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True,
transform=transform)
testloader = torch.utils.data.DataLoader(testset,
batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

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

```
## Let us show some of the training images, for fun.

# functions to show an image
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```




```
horse deer  car   horse
```

##Code (20 pts)

###Define a Convolutional Neural Network (10 pt)

Create a neural network that take 3-channel images. It should go as Conv2d --> ReLU --> MaxPool2d --> Conv2d --> ReLU --> MaxPool2d --> Flatten --> Linear --> ReLU --> Linear --> ReLU --> Linear

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        ## TODO: Add layers to your neural net
        # Create a neural network that take 3-channel images
        # It should go as Conv2d --> ReLU --> MaxPool2d --> Conv2d -->
ReLU --> MaxPool2d --> Flatten --> Linear --> ReLU --> Linear --> ReLU
--> Linear
        self.conv1 = nn.Conv2d(3,32,3,padding=1)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(32,16,3,padding=1)
        self.dp= nn.Dropout(p=0.25)
        self.fc1 = nn.Linear(16 * 8 * 8,128)
        self.fc2 = nn.Linear(128,64)
        self.fc3 = nn.Linear(64,32)
        self.fc4 = nn.Linear(32,16)
        self.fc5 = nn.Linear(16,10)



    def forward(self, x):
        ## TODO: run forward pass as mentioned above.
        # Conv2d --> ReLU --> MaxPool2d --> Conv2d --> ReLU -->
MaxPool2d --> Flatten --> Linear --> ReLU --> Linear --> ReLU -->
Linear
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x=  self.dp(x)
        x = x.view(-1,16 * 8 * 8)
        # increase the number of neurons in the hidden layers
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)

        return x


net = Net()
```

###Define a Loss function and optimizer (5 pt)

Let’s use a Classification Cross-Entropy loss and SGD with momentum. (Feel free to experiment with other loss functions and optimizers to observe differences)

```
# Let’s use a Classification Cross-Entropy loss and SGD with momentum.
(Feel free to experiment with other loss functions and optimizers to
observe differences)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), learning_rate=0.001,
momentum=0.9)
```

###Train the network (5 pts)

This is when things start to get interesting. We simply have to loop over our data iterator, and feed the inputs to the network and optimize.

```
epochs = 10 ## define number of epochs to train

epoch_loss = {}
train_losses = []
train_accuracy = []
for epoch in range(epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    correct_values = 0.0
    total = 0.0
    for i, data in enumerate(trainloader, 0):

        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # TODO: add line to zero the parameter gradients below
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        _, predictions = torch.max(outputs, 1)
        total += labels.size(0)

        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        correct_values += (predictions == labels).float().sum()
        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss /
2000:.3f}')
            running_loss = 0.0
    train_accuracy.append((100 * correct_values) / total)
    train_losses.append(running_loss/(i+1))
```

```
    # print(f'Accuracy for{epoch}: {(100 * correct_values) / total}')
print('Finished Training')

## Let’s quickly save our trained model:

PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

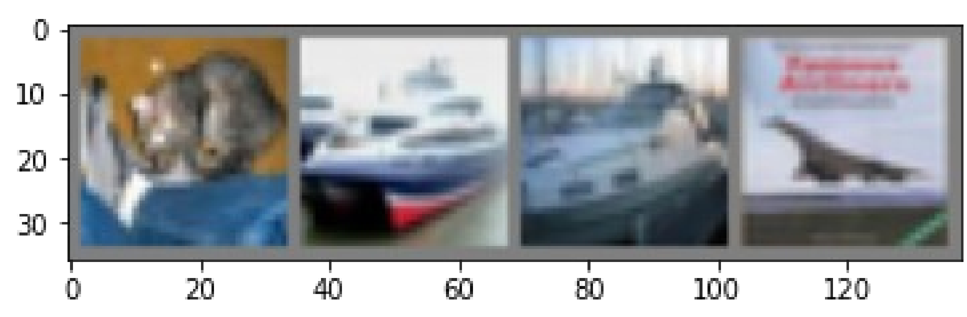
```
[1,  2000] loss: 2.304
[1,  4000] loss: 2.284
[1,  6000] loss: 2.064
[1,  8000] loss: 1.861
[1, 10000] loss: 1.747
[1, 12000] loss: 1.655
[2,  2000] loss: 1.531
[2,  4000] loss: 1.471
[2,  6000] loss: 1.428
[2,  8000] loss: 1.378
[2, 10000] loss: 1.354
[2, 12000] loss: 1.305
[3,  2000] loss: 1.255
[3,  4000] loss: 1.212
[3,  6000] loss: 1.210
[3,  8000] loss: 1.179
[3, 10000] loss: 1.180
[3, 12000] loss: 1.181
[4,  2000] loss: 1.094
[4,  4000] loss: 1.101
[4,  6000] loss: 1.090
[4,  8000] loss: 1.057
[4, 10000] loss: 1.055
[4, 12000] loss: 1.067
[5,  2000] loss: 0.988
[5,  4000] loss: 0.985
[5,  6000] loss: 0.989
[5,  8000] loss: 1.002
[5, 10000] loss: 0.965
[5, 12000] loss: 0.978
[6,  2000] loss: 0.907
[6,  4000] loss: 0.925
[6,  6000] loss: 0.927
[6,  8000] loss: 0.928
[6, 10000] loss: 0.931
[6, 12000] loss: 0.917
[7,  2000] loss: 0.842
[7,  4000] loss: 0.885
[7,  6000] loss: 0.860
[7,  8000] loss: 0.896
[7, 10000] loss: 0.866
```

```
[7, 12000] loss: 0.859
[8,  2000] loss: 0.800
[8,  4000] loss: 0.801
[8,  6000] loss: 0.840
[8,  8000] loss: 0.844
[8, 10000] loss: 0.852
[8, 12000] loss: 0.826
[9,  2000] loss: 0.776
[9,  4000] loss: 0.780
[9,  6000] loss: 0.779
[9,  8000] loss: 0.787
[9, 10000] loss: 0.822
[9, 12000] loss: 0.807
[10,  2000] loss: 0.743
[10,  4000] loss: 0.730
[10,  6000] loss: 0.752
[10,  8000] loss: 0.766
[10, 10000] loss: 0.749
[10, 12000] loss: 0.785
Finished Training
```

###Test the network on the test data We have trained the network over the training dataset. But we need to check if the network has learnt anything at all.

We will check this by predicting the class label that the neural network outputs, and checking it against the ground-truth. If the prediction is correct, we add the sample to the list of correct predictions.

```
dataiter = iter(testloader)
images, labels = dataiter.next()

imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in
range(4)))
```




```
GroundTruth:  cat   ship  ship  plane
```

```
net = Net()
net.load_state_dict(torch.load(PATH))
outputs = net(images)
_, predicted = torch.max(outputs, 1)
```

```python
print('Predicted: ', ' '.join(f'{classes[predicted[j]]:5s}'
                              for j in range(4)))
```

```
Predicted:  cat   ship  plane ship
```

```python
# prepare to count predictions for each class
correct = 0
total = 0

with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as
prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 *
correct // total} %')

correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# again no gradients needed
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1

# print accuracy for each class
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

```
Accuracy of the network on the 10000 test images: 68 %
Accuracy for class: plane is 76.8 %
Accuracy for class: car   is 76.1 %
Accuracy for class: bird  is 48.9 %
Accuracy for class: cat   is 56.1 %
Accuracy for class: deer  is 66.6 %
```

```
Accuracy for class: dog   is 53.9 %
Accuracy for class: frog  is 75.8 %
Accuracy for class: horse is 74.4 %
Accuracy for class: ship  is 76.3 %
Accuracy for class: truck is 77.6 %
```

Write-up (5 pt)

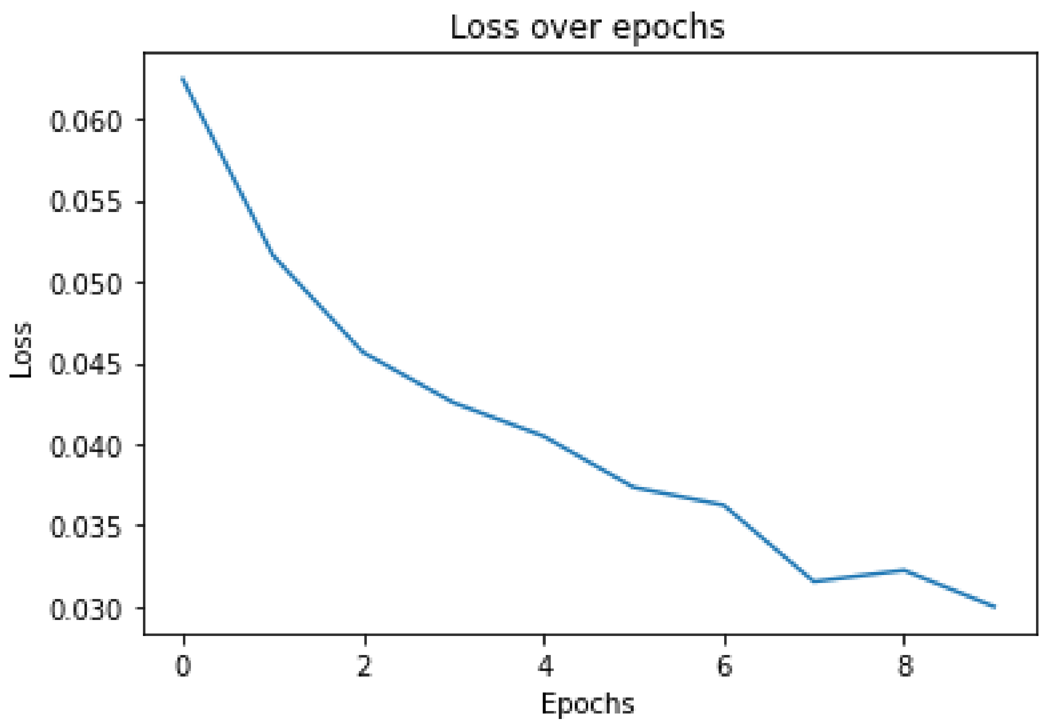
(1 pt) Show plot for loss over epochs.

(1 pt) Show plot for accuracy over epochs.

(3 pt) Show confusion matrix on test data.

(1 pt) Show plot for loss over epochs.

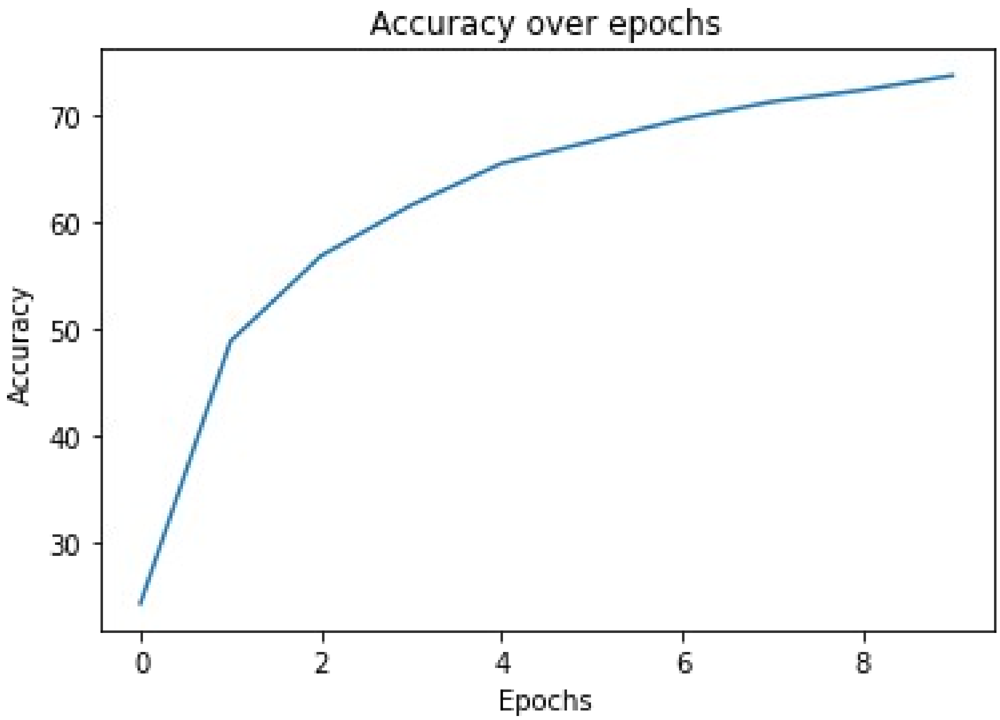
```
# Plot for loss over epochs
plt.plot(train_losses)
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



(1 pt) Show plot for accuracy over epochs.

```
# Plot for accuracy over epochs
plt.plot(train_accuracy)
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
```

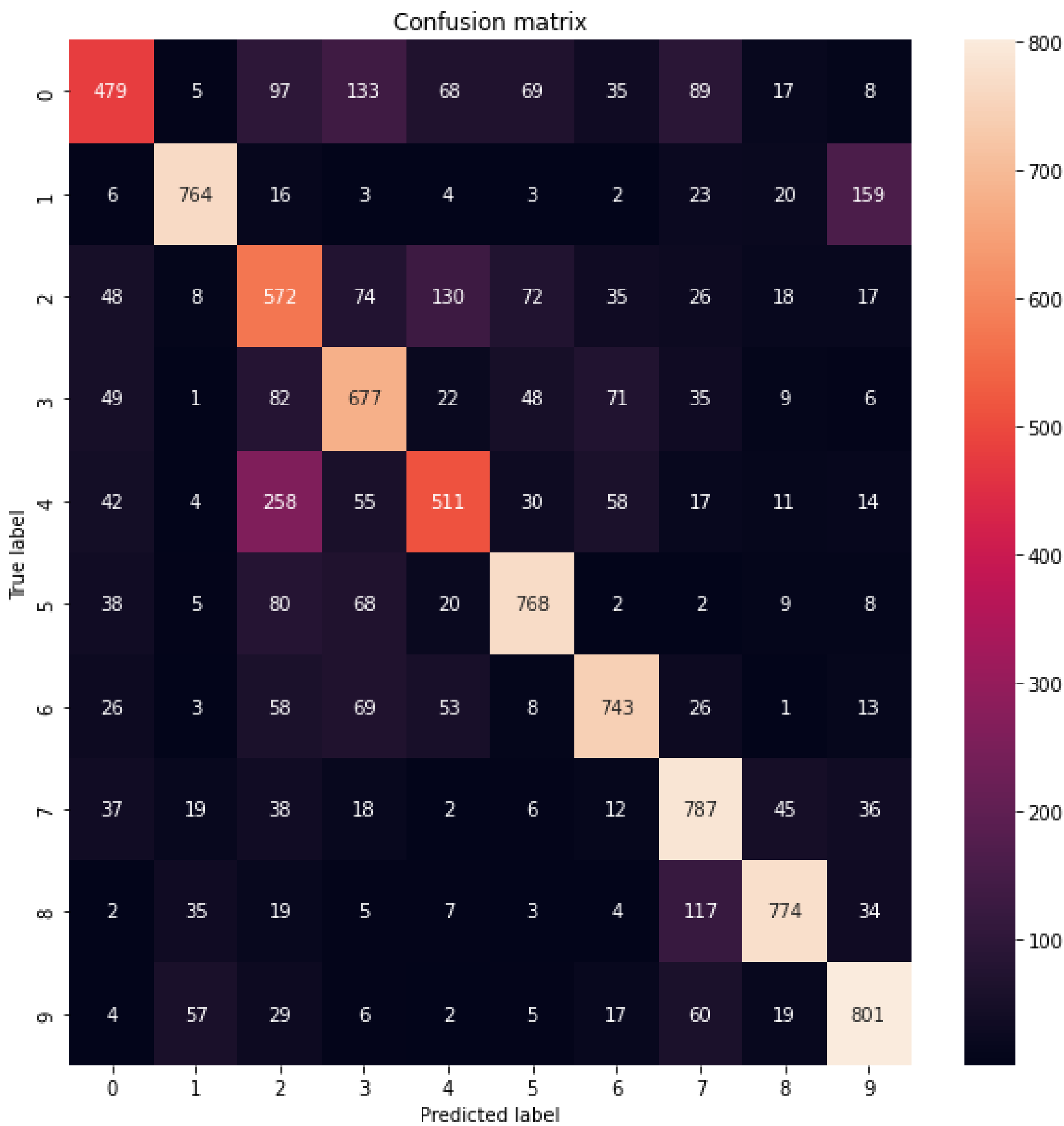
```
plt.ylabel('Accuracy')
plt.show()
```



(3 pt) Show confusion matrix on test data.

```
# Plot for confusion matrix using sklearn
from sklearn.metrics import confusion_matrix
import seaborn as sns

# get all predictions in an array and plot them as a confusion matrix
with torch.no_grad():
    n_classes = 10
    n_images = len(testloader.dataset)
    class_correct = list(0. for i in range(n_classes))
    class_total = list(0. for i in range(n_classes))
    y_true = []
    y_pred = []
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        c = (predictions == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1
            y_true.append(classes[label])
            y_pred.append(classes[predictions[i]])
```

```
cm = confusion_matrix(y_true, y_pred)

# Plot Confusion Matrix
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt="d")
plt.title("Confusion matrix")
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



Extra Credits (5 pt)

Run VGG with pre-trained weights in this colab. Test any two images of your choice to your model and to VGG model and show accuracy (images must include objects from CIFAR10 classes). Discuss which model performs better and why.

Part 3: Semantic Segmentation

Overview

Semantic Segmentation is an image analysis task in which we classify each pixel in the image into a class. So, let's say we have the following image.

And then given the above image its semantically segmentated image would be the following

As you can see, that each pixel in the image is classified to its respective class.

Data

WARNING: Colab deletes all files everytime runtime is disconnected. Make sure to re-download the inputs when it happens.

```
import os
import tarfile
import shutil
import urllib.request

url='http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar'
path='VOC'
def get_archive(path,url):
  try:
    os.mkdir(path)
  except:
    path=path

  filename='devkit'
  urllib.request.urlretrieve(url,f"{path}/{filename}.tar")

get_archive(path,url)
def extract(path):
  tar_file=tarfile.open(f"{path}/devkit.tar")
  tar_file.extractall('./')
  tar_file.close()
  shutil.rmtree(path)
```

```
extract(path)
```

Helper Functions

```
from PIL import Image
import matplotlib.pyplot as plt
import torch
from torchvision import models
import torchvision.transforms as T
import numpy as np
import cv2

"""Various RGB palettes for coloring segmentation labels."""
VOC_CLASSES = [
    "background",
    "aeroplane",
    "bicycle",
    "bird",
    "boat",
    "bottle",
    "bus",
    "car",
    "cat",
    "chair",
    "cow",
    "diningtable",
    "dog",
    "horse",
    "motorbike",
    "person",
    "potted plant",
    "sheep",
    "sofa",
    "train",
    "tv/monitor",
]


VOC_COLORMAP = [
    [0, 0, 0],
    [128, 0, 0],
    [0, 128, 0],
    [128, 128, 0],
    [0, 0, 128],
    [128, 0, 128],
    [0, 128, 128],
    [128, 128, 128],
    [64, 0, 0],
    [192, 0, 0],
    [64, 128, 0],
```

```
    [192, 128, 0],
    [64, 0, 128],
    [192, 0, 128],
    [64, 128, 128],
    [192, 128, 128],
    [0, 64, 0],
    [128, 64, 0],
    [0, 192, 0],
    [128, 192, 0],
    [0, 64, 128],
]

if torch.cuda.is_available():
  device=torch.device('cuda:0')
  print('Cuda')
else:
  device=torch.device('cpu')
  print('cpu')
```

```
Cuda
```

Code (25 pt)

1. Implement Data Loader for training and validation (5 pt)

```
import os
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import cv2

# You can modify this class
class VocDataset(Dataset):
  def __init__(self, dir, color_map):
    self.root=os.path.join(dir,'VOCdevkit/VOC2007')
    self.target_dir=os.path.join(self.root,'SegmentationClass')
    self.images_dir=os.path.join(self.root,'JPEGImages')
    file_list =
os.path.join(self.root,'ImageSets/Segmentation/trainval.txt')
    self.files = [line.rstrip() for line in tuple(open(file_list,
"r"))]
    self.color_map=color_map

  def convert_to_segmentation_mask(self,mask):
    # This function converts color channels of semgentation masks to
number of classes
    # Semantic Segmentation requires a segmentation mask to be a NumPy
array with the shape
    # This part is implemented for displaying colorized results in
subpart 3
    # YOUR CODE HERE:
```

```
    height, width = mask.shape[:2]
    segmentation_mask = np.zeros((height, width, len(self.color_map)),
dtype=np.float32)
    for label_index, label in enumerate(self.color_map):
          segmentation_mask[:, :, label_index] = np.all(mask == label,
axis=-1).astype(float)
    return segmentation_mask

  def __getitem__(self,index):
    # YOUR CODE HERE:
    image_id=self.files[index]
    image_path=os.path.join(self.images_dir,f"{image_id}.jpg")
    label_path=os.path.join(self.target_dir,f"{image_id}.png")
    image=cv2.imread(image_path)
    image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    image=cv2.resize(image,(256,256))
    image=torch.tensor(image).float()
    label=cv2.imread(label_path)
    label=cv2.cvtColor(label,cv2.COLOR_BGR2RGB)
    label=cv2.resize(label,(256,256))
    label = self.convert_to_segmentation_mask(label)
    label=torch.tensor(label).float()

    return image,label

  def __len__(self):
    return len(self.files)

# Load the dataset
dataset = VocDataset('',VOC_COLORMAP)
# Create a dataloader
dataloader = DataLoader(dataset, batch_size=4, shuffle=True,
num_workers=0)

# Get a batch of training data
images, labels = next(iter(dataloader))
print(images.shape) # batch_size x 3 x 256 x 256
print(labels.shape) # batch_size x 21 x 256 x 256

dataset.__len__()
```

```
torch.Size([4, 256, 256, 3])
torch.Size([4, 256, 256, 21])
```

```
422
```

```
# Train set and validation set
train_set,val_set=torch.utils.data.random_split(dataset,
[int(len(dataset)*0.9),round(len(dataset)*0.1)+1])
train_loader = DataLoader(train_set, batch_size=10, shuffle=True,
```

```
num_workers=0)
val_loader = DataLoader(val_set, batch_size=10, shuffle=True,
num_workers=0)
test_loader = DataLoader(dataset, batch_size=10, shuffle=True,
num_workers=0)

# print the length of train set and validation set
print(len(train_set))
print(len(val_set))
```

```
379
43
```

###2. Define model and training code (15 pt) Implement FCN-32 model. You can use encoder as pretrained model provided by torchvision.

```
import torch
class FCN32(torch.nn.Module):
  def __init__(self, n_classes, pretrained_model):
    # YOUR CODE HERE:
    super(FCN32, self).__init__()
    self.pretrained_model=pretrained_model
    # encoder
    self.encoder =
torch.nn.Sequential(*list(pretrained_model.features.children()))

    self.encoder_classifier = torch.nn.Sequential(
        torch.nn.Conv2d(512, 4096, kernel_size=1),
        torch.nn.ReLU(inplace=True),
        torch.nn.Dropout(),
        torch.nn.Conv2d(4096, 4096, kernel_size=1),
        torch.nn.ReLU(inplace=True),
        torch.nn.Dropout()
    )

    # decoder
    self.decoder = torch.nn.Sequential(
        torch.nn.ConvTranspose2d(4096, 512, kernel_size=3, stride=2,
padding=1, output_padding=1),
        torch.nn.BatchNorm2d(512),
        torch.nn.ReLU(inplace=True),
        torch.nn.ConvTranspose2d(512, 256, kernel_size=3, stride=2,
padding=1, output_padding=1),
        torch.nn.BatchNorm2d(256),
        torch.nn.ReLU(inplace=True),
        torch.nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2,
padding=1, output_padding=1),
        torch.nn.BatchNorm2d(128),
        torch.nn.ReLU(inplace=True),
```

```
        torch.nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2,
padding=1, output_padding=1),
        torch.nn.BatchNorm2d(64),
        torch.nn.ReLU(inplace=True),
        torch.nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2,
padding=1, output_padding=1),
        torch.nn.BatchNorm2d(32),
        torch.nn.ReLU(inplace=True),
        torch.nn.Conv2d(32, n_classes, kernel_size=1)
    )

  # forward function
  def forward(self, x):
    # apply encoder
    output = self.encoder(x)
    output = self.encoder_classifier(output)

    # apply decoder
    output = self.decoder(output)

    # return the predicted label image
    return output
```

Training code for the semantic segmentation model. Implment both training and validation parts.

```
import torchvision
from torch.utils.data import Dataset, DataLoader, random_split
import tqdm
import sklearn.metrics

def metrics(y_pred,y_true):
  y_pred=torch.argmax(y_pred,dim=1)
  y_true=torch.argmax(y_true,dim=1)

iou=sklearn.metrics.jaccard_score(y_true.flatten(),y_pred.flatten(),av
erage='weighted')
  return iou

def train(model,optim,loss_f,epochs,scheduler,path_for_models):
  try:
    os.mkdir(path_for_models)
  except:
    path_for_models=path_for_models

  min_iou=0.3
  device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
```

```
  for epoch in (range(epochs)):
    for (X_train,y_train) in train_loader:

#X_train,y_train=X_train.to(device),y_train.to(device,dtype=torch.int6
4)
      X_train = X_train.permute(0, 3, 1, 2)
      y_train = y_train.permute(0, 3, 1, 2)
      y_pred=model(X_train)
      loss=loss_f(y_pred,y_train)
      optim.zero_grad()
      loss.backward()
      optim.step()
    ious=[]
    val_losses=[]
    with torch.no_grad():
      for b,(X_test,y_test) in enumerate(val_loader):
        #X_test,y_test=X_test.to(device),y_test.to(device)
        X_test = X_test.permute(0, 3, 1, 2)
        y_test = y_test.permute(0, 3, 1, 2)
        y_val=model(X_test)
        val_loss=loss_f(y_val,y_test)
        val_losses.append(val_loss)
        iou_= metrics(y_val,y_test)
        ious.append(iou_)
      ious=torch.tensor(ious)
      val_losses=torch.tensor(val_losses)
      scheduler.step(val_losses.mean())
      if ious.mean() > min_iou:
        min_iou=ious.mean()

torch.save(model.state_dict(),f"{path_for_models}/fc32model.pth")
    print(f"epoch : {epoch:2} train_loss: {loss:10.4} , val_loss :
{val_losses.mean()} val_iou: {ious.mean()}")


# YOUR CODE HERE:
# Load the pretrained model
pretrained_net = torchvision.models.vgg16(pretrained=True)

# Create the model
model = FCN32(n_classes=21, pretrained_model=pretrained_net)

# Define the loss function
criterion = torch.nn.BCEWithLogitsLoss()

# Define the optimizer
optimizer = torch.optim.Adam(model.parameters(), learning_rate=0.0001)

# Define the learning rate scheduler
```

```
scheduler=torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,patienc
e=3,verbose=True)

# Define the number of epochs
num_epochs = 50

# Training
train(model,optimizer,criterion,50,scheduler,'models')
```

```
epoch :  0 train_loss:     0.6648 , val_loss : 0.6636990904808044
val_iou: 0.06645422412390647
epoch :  1 train_loss:     0.6364 , val_loss : 0.6362327337265015
val_iou: 0.08778575214069836
epoch :  2 train_loss:     0.6141 , val_loss : 0.6133363246917725
val_iou: 0.09312329576172612
epoch :  3 train_loss:     0.5907 , val_loss : 0.5913186073303223
val_iou: 0.09464990766848036
epoch :  4 train_loss:     0.5656 , val_loss : 0.5679311156272888
val_iou: 0.2324325057114301
epoch :  5 train_loss:     0.5409 , val_loss : 0.5452107787132263
val_iou: 0.41707641293873376
epoch :  6 train_loss:     0.5188 , val_loss : 0.5199036598205566
val_iou: 0.4579303562816353
epoch :  7 train_loss:      0.492 , val_loss : 0.4977358281612396
val_iou: 0.4847278200530288
epoch :  8 train_loss:     0.4763 , val_loss : 0.47471779584884644
val_iou: 0.5118984880514561
epoch :  9 train_loss:     0.4523 , val_loss : 0.45439058542251587
val_iou: 0.5209477738132241
epoch : 10 train_loss:     0.4322 , val_loss : 0.43684038519859314
val_iou: 0.4643655911862023
epoch : 11 train_loss:     0.4136 , val_loss : 0.41474658250808716
val_iou: 0.5127492233099312
epoch : 12 train_loss:     0.4064 , val_loss : 0.39939576387405396
val_iou: 0.5001176605563764
epoch : 13 train_loss:     0.3741 , val_loss : 0.3752719759941101
val_iou: 0.522937654484529
epoch : 14 train_loss:     0.3573 , val_loss : 0.3595080077648163
val_iou: 0.48293371758004644
epoch : 15 train_loss:     0.3274 , val_loss : 0.34308212995529175
val_iou: 0.4604205207269535
epoch : 16 train_loss:     0.3141 , val_loss : 0.32389575242996216
val_iou: 0.5082112142136836
epoch : 17 train_loss:     0.3005 , val_loss : 0.3092826306819916
val_iou: 0.4967499853737859
epoch : 18 train_loss:     0.2919 , val_loss : 0.3004592955112457
val_iou: 0.45425104732319915
epoch : 19 train_loss:     0.2796 , val_loss : 0.28386443853378296
val_iou: 0.4848243542509671
epoch : 20 train_loss:     0.2603 , val_loss : 0.2734500765800476
```

```
val_iou: 0.4625630128900521
epoch : 21 train_loss:     0.2498 , val_loss : 0.2596544921398163
val_iou: 0.49339914879378755
epoch : 22 train_loss:     0.2281 , val_loss : 0.2495810091495514
val_iou: 0.47001007134394923
epoch : 23 train_loss:      0.232 , val_loss : 0.2385433167219162
val_iou: 0.4921154370066295
epoch : 24 train_loss:     0.2253 , val_loss : 0.23071618378162384
val_iou: 0.47666870035236286
epoch : 25 train_loss:     0.2179 , val_loss : 0.21743515133857727
val_iou: 0.5281408654875122
epoch : 26 train_loss:     0.1951 , val_loss : 0.2124149352312088
val_iou: 0.4918044469313593
epoch : 27 train_loss:     0.1942 , val_loss : 0.20237627625465393
val_iou: 0.5135075499190845
epoch : 28 train_loss:     0.1816 , val_loss : 0.19977621734142303
val_iou: 0.4718099421178922
epoch : 29 train_loss:     0.1873 , val_loss : 0.1904156506061554
val_iou: 0.4872751439392774
epoch : 30 train_loss:     0.1755 , val_loss : 0.18541817367076874
val_iou: 0.4743586532203481
epoch : 31 train_loss:     0.1729 , val_loss : 0.17950211465358734
val_iou: 0.47317039454764587
epoch : 32 train_loss:     0.1733 , val_loss : 0.17246535420417786
val_iou: 0.4978187478283007
epoch : 33 train_loss:     0.1529 , val_loss : 0.16348038613796234
val_iou: 0.5557358747506514
epoch : 34 train_loss:     0.1606 , val_loss : 0.16116449236869812
val_iou: 0.5090593370927705
epoch : 35 train_loss:     0.1388 , val_loss : 0.1532069444656372
val_iou: 0.5487954078731645
epoch : 36 train_loss:      0.137 , val_loss : 0.15702210366725922
val_iou: 0.4644225886492058
epoch : 37 train_loss:     0.1339 , val_loss : 0.14986129105091095
val_iou: 0.4977137157242331
epoch : 38 train_loss:     0.1423 , val_loss : 0.14701567590236664
val_iou: 0.49407479803750504
epoch : 39 train_loss:     0.1297 , val_loss : 0.1460372507572174
val_iou: 0.47051191287931593
epoch : 40 train_loss:     0.1304 , val_loss : 0.14011716842651367
val_iou: 0.4849578439702487
epoch : 41 train_loss:     0.1282 , val_loss : 0.13591761887073517
val_iou: 0.49490342485657574
epoch : 42 train_loss:     0.1186 , val_loss : 0.13616472482681274
val_iou: 0.48368507896896257
epoch : 43 train_loss:     0.1242 , val_loss : 0.13051514327526093
val_iou: 0.500175378949139
epoch : 44 train_loss:     0.1225 , val_loss : 0.13564935326576233
val_iou: 0.45883721497583296
epoch : 45 train_loss:     0.1078 , val_loss : 0.12426996231079102
```

```
val_iou: 0.5095538133009494
epoch : 46 train_loss:     0.1167 , val_loss : 0.12910175323486328
val_iou: 0.47383339967311766
epoch : 47 train_loss:     0.1112 , val_loss : 0.11978821456432343
val_iou: 0.5207056062378072
epoch : 48 train_loss:     0.1146 , val_loss : 0.1271398812532425
val_iou: 0.45826667805208154
epoch : 49 train_loss:    0.09299 , val_loss : 0.12176956236362457
val_iou: 0.47477227814603784
```

```
model.load_state_dict(torch.load('./models/fc32model.pth'))

model.eval()
```

```
FCN32(
  (pretrained_model): VGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (3): ReLU(inplace=True)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 
ceil_mode=False)
      (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (6): ReLU(inplace=True)
      (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (8): ReLU(inplace=True)
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 
ceil_mode=False)
      (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (11): ReLU(inplace=True)
      (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (13): ReLU(inplace=True)
      (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (15): ReLU(inplace=True)
      (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 
ceil_mode=False)
      (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (18): ReLU(inplace=True)
      (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
      (20): ReLU(inplace=True)
```

```
      (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (22): ReLU(inplace=True)
      (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (25): ReLU(inplace=True)
      (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (27): ReLU(inplace=True)
      (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (29): ReLU(inplace=True)
      (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
    (classifier): Sequential(
      (0): Linear(in_features=25088, out_features=4096, bias=True)
      (1): ReLU(inplace=True)
      (2): Dropout(p=0.5, inplace=False)
      (3): Linear(in_features=4096, out_features=4096, bias=True)
      (4): ReLU(inplace=True)
      (5): Dropout(p=0.5, inplace=False)
      (6): Linear(in_features=4096, out_features=1000, bias=True)
    )
  )
  (encoder): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
```

```
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (encoder_classifier): Sequential(
    (0): Conv2d(512, 4096, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
  )
  (decoder): Sequential(
    (0): ConvTranspose2d(4096, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), output_padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), output_padding=(1, 1))
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), output_padding=(1, 1))
```

```
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), output_padding=(1, 1))
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), output_padding=(1, 1))
    (13): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (14): ReLU(inplace=True)
    (15): Conv2d(32, 21, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

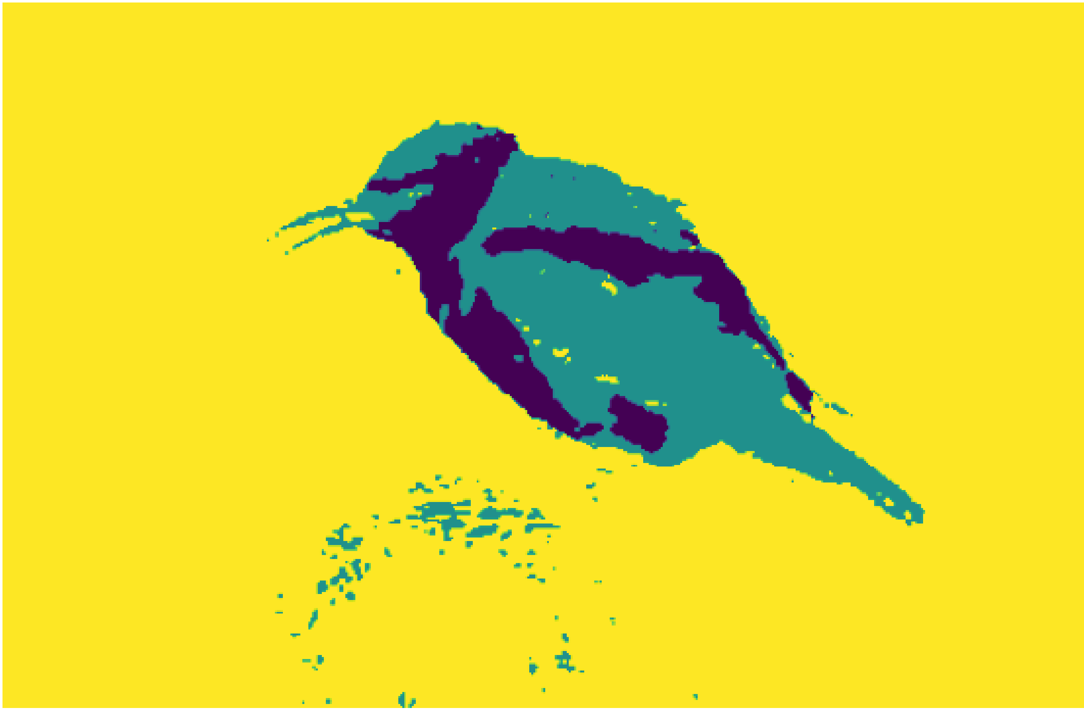
3. Inference for semantic segmentation (5 pt)

Implement the inference code for semantic segmentation. Display the visualization results. Plot the image and colorized image (similar to the results in overview).

```
import imageio

# YOUR CODE HERE:

def decode_segmap(image,colors,nc=21):
  r = np.zeros_like(image).astype(np.uint8)
  g = np.zeros_like(image).astype(np.uint8)
  b = np.zeros_like(image).astype(np.uint8)
  # convert colors to list
  for l in range(0, nc):
    idx = image == l
    r[idx] = colors[l][0]
    g[idx] = colors[l][1]
    b[idx] = colors[l][2]
  rgb = np.stack([r, g, b], axis=2)
  return rgb

def image(img_path):
  img=cv2.imread(img_path,cv2.IMREAD_COLOR)
  img= torch.tensor(img)
  image = torch.argmax(img.squeeze(), dim=2).detach().cpu().numpy()
  plt.figure(figsize=(10, 10))
  plt.imshow(image)
  plt.axis('off')
  return image

# Plot original image 000395.jpg
plt.figure(figsize=(10, 10))
```

```
plt.imshow(imageio.imread('./VOCdevkit/VOC2007/JPEGImages/000395.jpg')
)
plt.axis('off')
plt.show()

rgb =
decode_segmap(image('./VOCdevkit/VOC2007/JPEGImages/000395.jpg'),VOC_C
OLORMAP)
plt.figure(figsize=(10, 10))
plt.imshow(rgb)
plt.axis('off')
plt.show()
```

```
/tmp/ipykernel_28229/3732159665.py:29: DeprecationWarning: Starting
with ImageIO v3 the behavior of this function will switch to that of
iio.v3.imread. To keep the current behavior (and make this warning
disappear) use `import imageio.v2 as imageio` or call
`imageio.v2.imread` directly.

plt.imshow(imageio.imread('./VOCdevkit/VOC2007/JPEGImages/000395.jpg')
)
```



Write-up (5 pt)

- Describe the properties of segmentation model
- Describe the evaluation metric (IoU) for segmentation model

Describe the properties of segmentation model

Image segmentation is the process of dividing an image into multiple segments or regions, each of which corresponds to a different object or background. The FCN32 model is a type of convolutional neural network that is commonly used for image segmentation tasks.

- a. One of the key properties of the FCN32 model is that it is fully convolutional, meaning that it contains only convolutional layers and does not have any fully connected layers. This allows the model to take train_features images of any size and produce corresponding segmentation maps of the same size.
- a. Another important property of the FCN32 model is that it uses skip connections, which allow the model to retain spatial information from earlier layers in the network. This can help the model to make more fine-grained predictions and improve its overall accuracy.
- a. Additionally, the FCN32 model uses a technique called upsampling to increase the resolution of the segmentation maps it produces. This allows the model to make more detailed predictions and produce segmentation maps with higher spatial resolution.

Overall, the FCN32 model is a powerful tool for image segmentation tasks, and its fully convolutional and upsampling properties make it well-suited for a variety of applications.

Intersection-Over-Union [IOU] (Jaccard Index)

For good reason, the Intersection-Over-Union (IoU), also known as the Jaccard Index, is one of the most often employed metrics in semantic segmentation. The IoU is a relatively simple yet incredibly useful statistic. IoU is the predicted segmentation's area of overlap divided by the predicted segmentation's area of union divided by the predicted segmentation's area of union divided by the predicted segmentation's area of union divided by the predicted segmentation's area of union divided.

```
def metrics(y_pred,y_true):
  y_pred=torch.argmax(y_pred,dim=1)
  y_true=torch.argmax(y_true,dim=1)

iou=sklearn.metrics.jaccard_score(y_true.flatten(),y_pred.flatten(),av
erage='weighted')
  return iou
```

I have used jaccard score from sklearn on y_true and y_pred by flattening the data by weighted average.

Hint

- Refer to original paper FCNet : https://arxiv.org/abs/1411.4038

- Figures for FCNet Structure: https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1
- PyTorch Tutorial for Image semgnetation: https://towardsdatascience.com/train-neural-net-for-semantic-segmentation-with-pytorch-in-50-lines-of-code-830c71a6544f

Part 4: Text2Img Generation (10 Points)

We have provided link to 'DALL.E' mini model to generate images from a text prompt in an interactive way.

https://colab.research.google.com/github/borisdayma/dalle-mini/blob/main/tools/inference/inference_pipeline.ipynb#scrollTo=118UKH5bWCGa

Write-up (10 pts)

1. Try different prompts (as per your understanding) to reveal biases encoded by model (for example, birds always exist in the similar surroundings like trees).
2. By inputting creative text prompts, you should report the failure cases in your writeup i.e. when model doesn't quite understand the semantics of text prompt (for example, in case of long and complex sentences).

Revealation of the biases encoded by the model

Promt 1: A stoned monkey playing DJ in Amazon forest

The DALL.E mini model was able to encode the given promt. In the given promt, I was expecting Monkey, forest, DJ and facial expression of the monkey(stoned expression). The model was able to correctly encode and generate the images

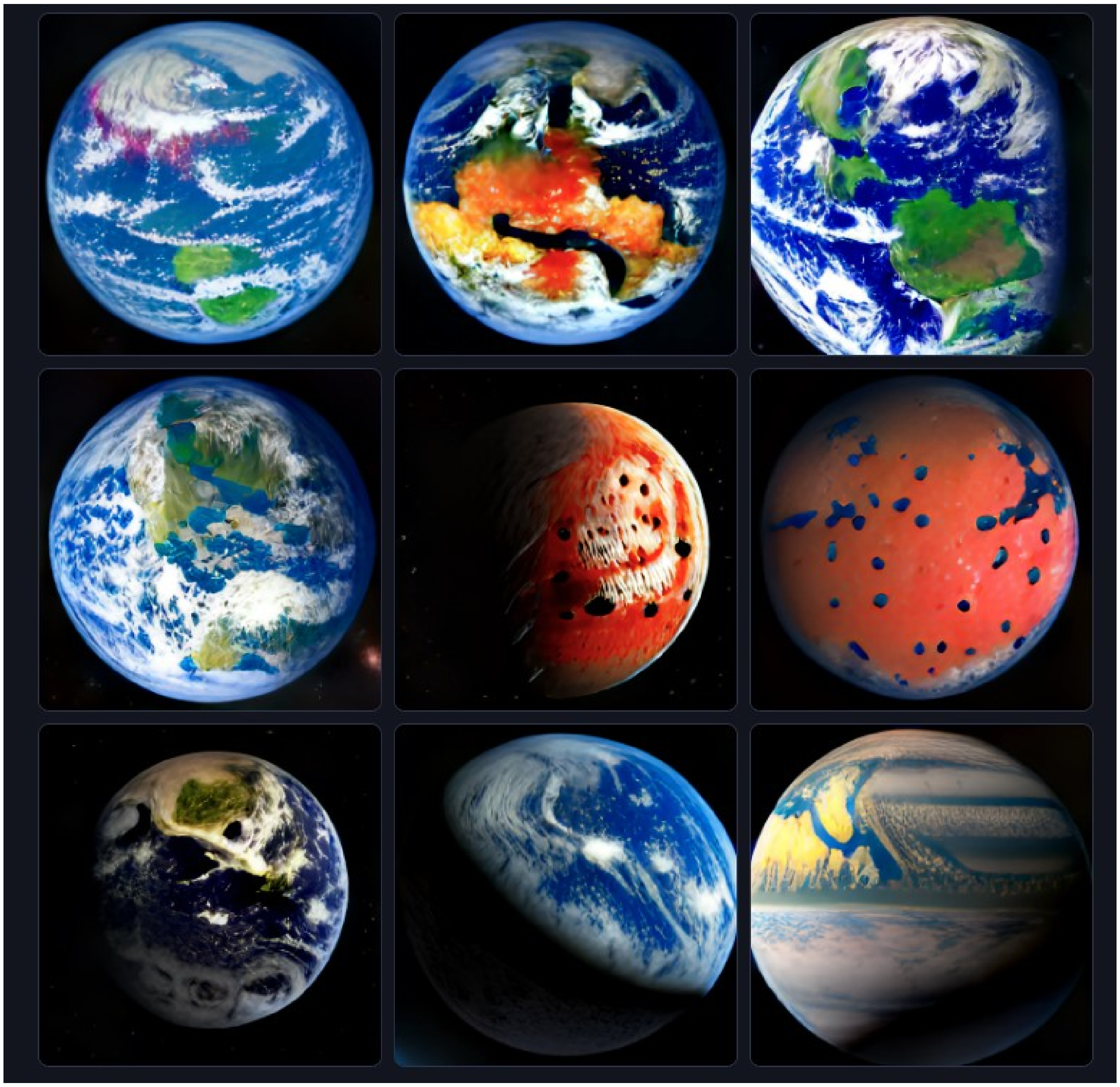
Promt 2: Santa having vacation on planet Saturn with his family

The DALL.E mini model was able to encode the given promt. In the given promt, I was expecting a santa, vaction environment, planet saturn and his family. The model was able to correctly encode and generate the images

Failure Cases

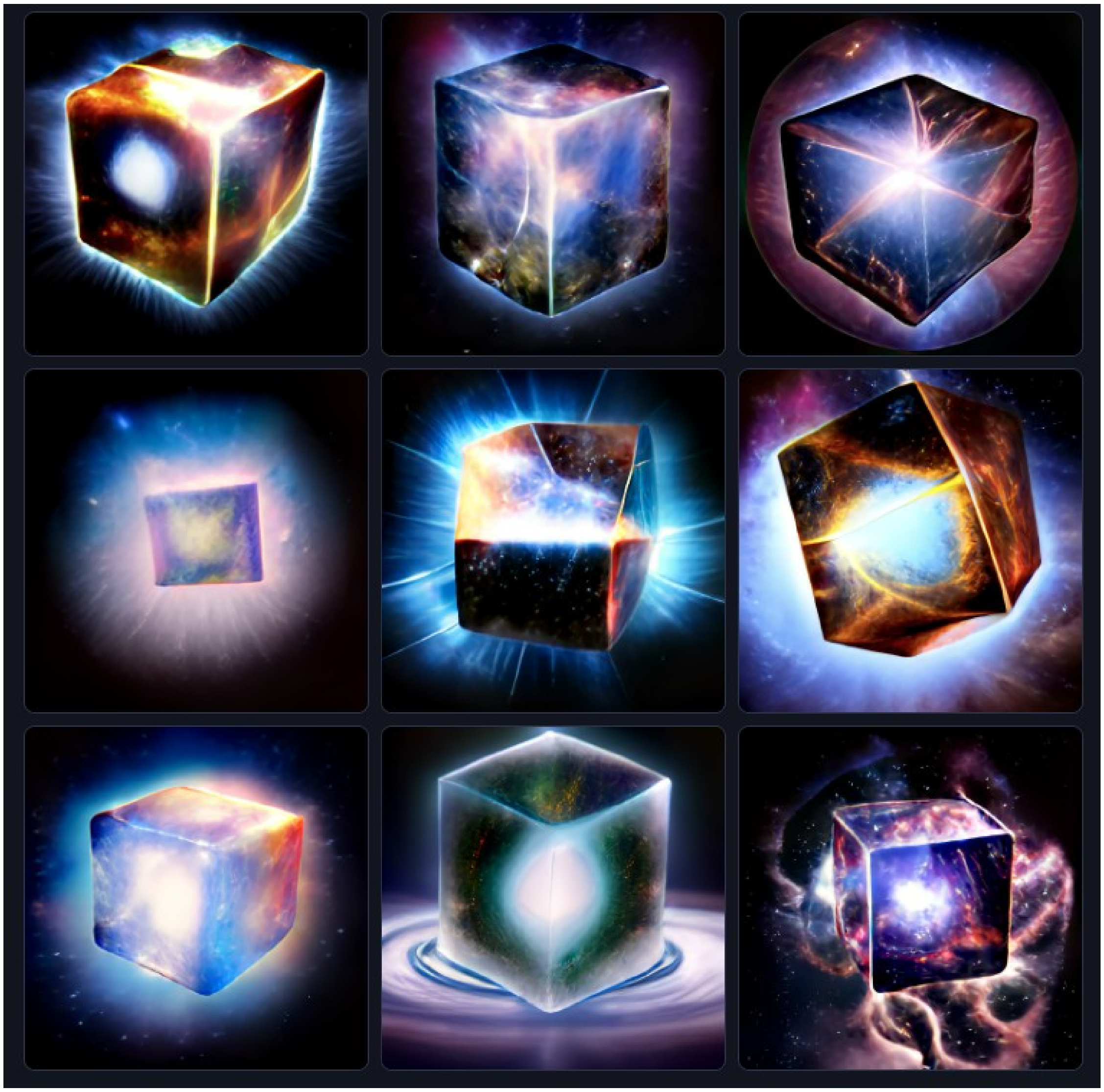
Promt 1: Satilliete view of a bacteria on a planet which is 10 light years away

The DALL.E mini model failed to encode the given promt. In the given promt, I was expecting a image of bacteria. How so far the logic behind the promt, model should generate the bacteria image. But it failed to view show bacteria



Promt 2: Microscopic view of Cosmic Cube from Iron man 2

The DALL.E mini model failed to encode the given promt. In the given promt, I was expecting a microscopic view but it gave me macroscopic view.



Extra Credit (15 pts)

In this part, you would compare the results of two recent text-to-image generation models: DALL E (https://www.craiyon.com) v/s Stable Diffusion (https://huggingface.co/spaces/stabilityai/stable-diffusion).

1. You can compare the results of two models in terms of: image quality, diversity of background, grounding in the text prompt and so on.

2. Similar to the main write-up, you are required to report 2 biases and 2 failure cases: i) where these models are unfairly biased, and ii) cases where one model is able to

rectify the mistakes (of not understanding the semantics of text prompt) made by other one.

Note: You shouldn't copy/past examples from internet, and any event of exact matching for any of the text prompts would be penalized.

2 Biases:

Promt 1: A glass turtle floating in space with oxygen

The DALL.E mini model

Model was able to encode biases for the given promt unfairly but it was accurate to 70 percent. In the given promt, I was expecting a glass turtle, space, floating with oxygen visualization. The unfair bias here is that oxygen was missing and no proper visualization on whether turtle is glass or not

Stable Diffusion

Model was able to encode biases for the given promt with atmost accuracy to 85 percent. In the given promt, I was expecting a glass turtle, space, floating with oxygen visualization. All the biases are visually presented, I can say there is a huge improvement in image clarity and biases in diffusion model.

Promt 2: A man with beard watching a movie along with baby Yoda

The DALL.E mini model

Model was able to encode biases for the given promt with atmost accuracy to 80 percent. In the given promt, I was expecting a man with beard, movie environment, baby yoda. All the biases are visually presented but the clarity of the image is poor and blurry

Stable Diffusion

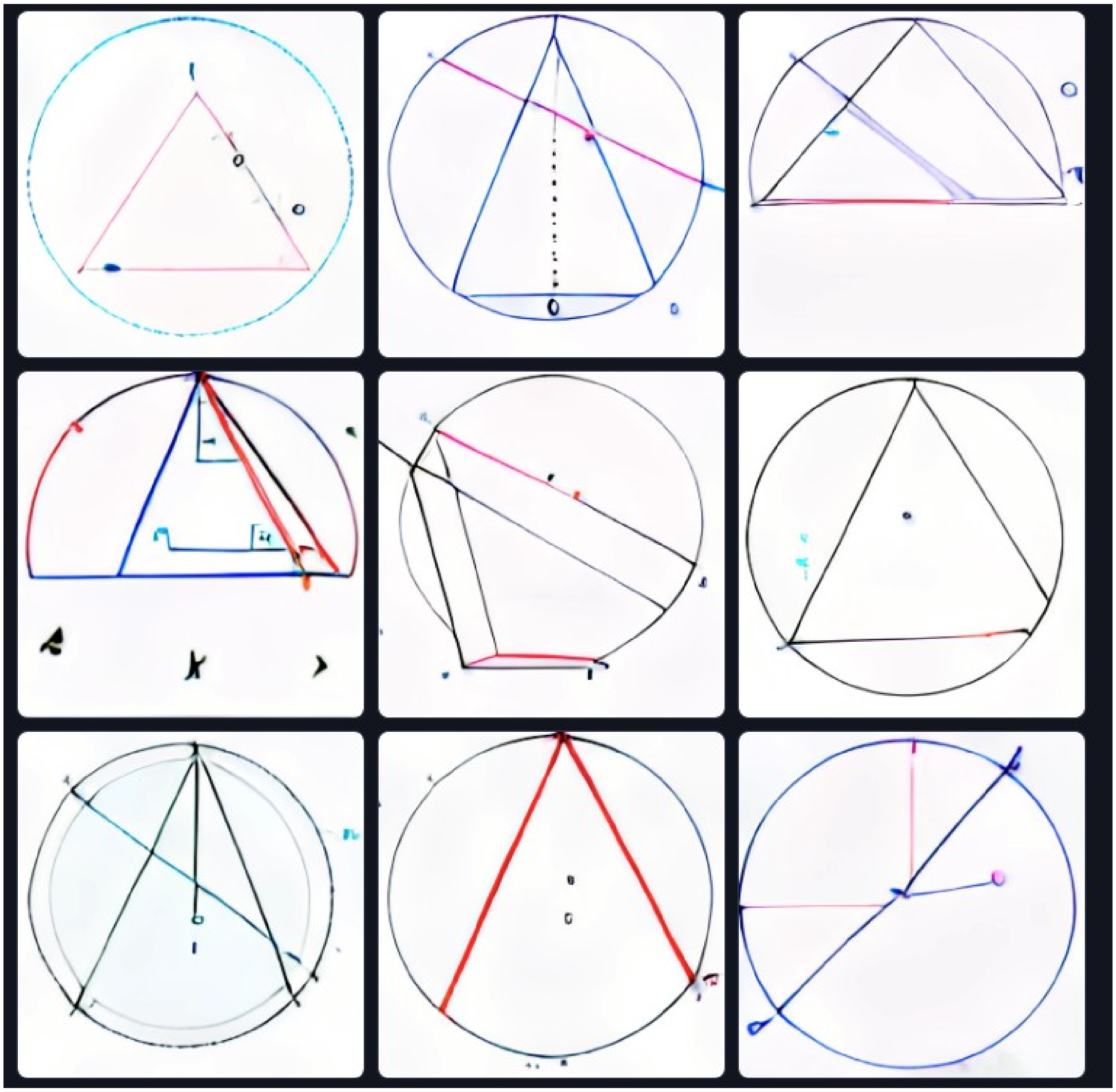
Model was able to encode biases for the given promt unfairly but it was accurate to 60 percent. In the given promt, I was expecting expecting a man with beard, movie environment, baby yoda. Not all the biases are visually presented, In few images there arent any humans and the movie environment was missing. I can say there is a huge improvement in image clarity and biases in diffusion model, but the biases encoded are poor when comapared with DALLIE.

2 Failure Cases

Promt 1: Convert a triangle to a circle

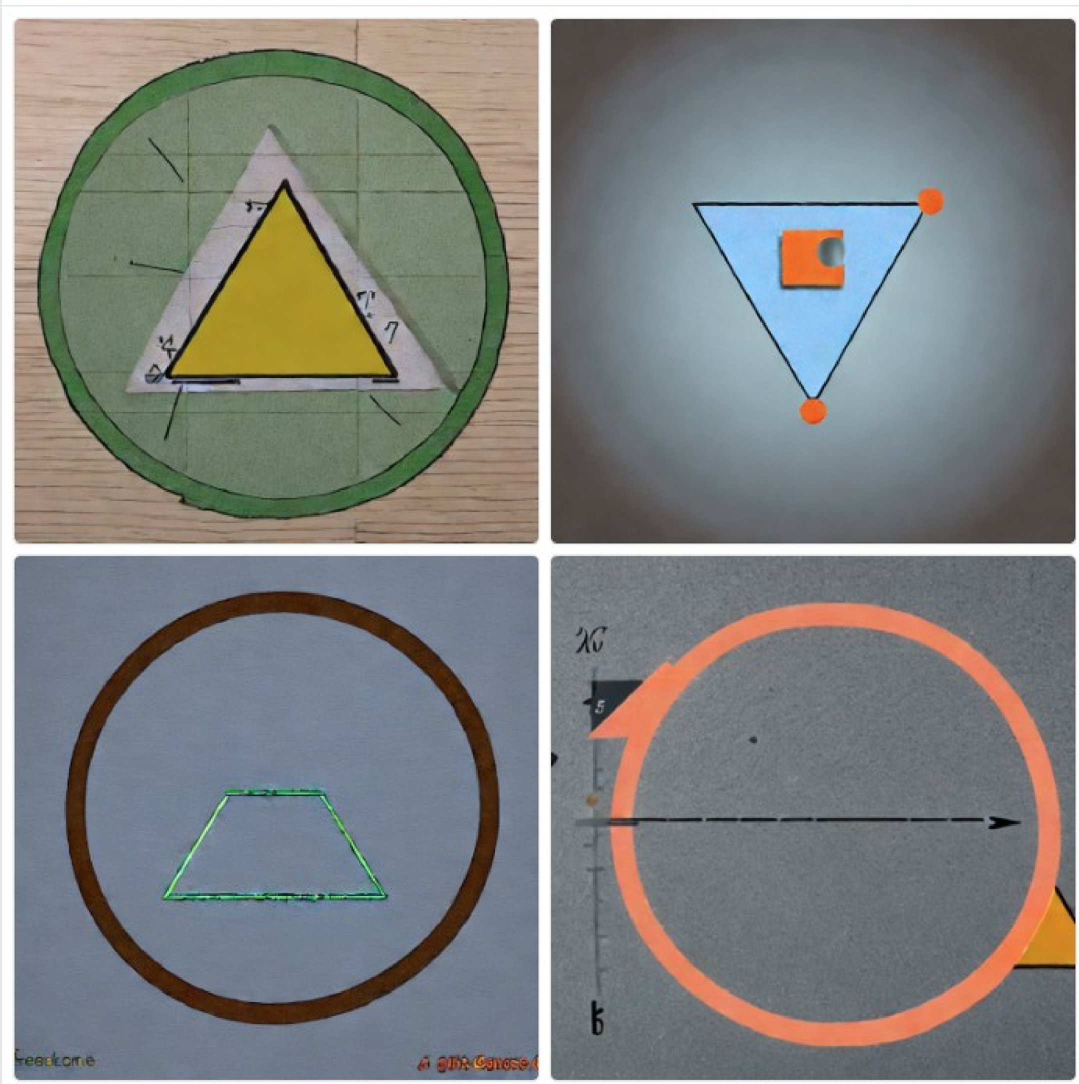
The DALL.E mini model

Model was unable to generate or encode the given prompt. In the given promt, I was expecting a new shape that doesnt have 3 corners. However, model was generting images with indvidual circles and triangles overlapped.



Stable Diffusion

Model was unable to generate or encode the given prompt. In the given promt, I was expecting a new shape that doesnt have 3 corners. However, model was generting images with indvidual circles and triangles overlapped. However, the image clarity is good and pictures are more colorful



Promt 2: A human with multiple skin tones and genders

The DALL.E mini model

Model was unable to generate or encode the given prompt. In the given promt, I was expecting a new kind of human being with different color tones like pink, orange or any other color with a different gender other than male, female and trans. Rather, it was generating images that are inclind toward more brown and multiple eyes. Dallie is comparitively better than diffusion as it was able to modify features of humans with different facial elements

Stable Diffusion

Model was unable to generate or encode the given prompt. In the given promt, I was expecting a new kind of human being with different color tones like pink, orange or any other color with a different gender other than male, female and trans. Model generate weired pitures of multiple humans in single image and a bunch of cartoon. Diffusion model was completely wrong in this case.