NNC_Final_Project_Code_Demo

December 16, 2019

[1]:
```
# ------------------------------------------
# CS6673 Neural Network Computing
# Final Project
# ------------------------------------------
# Note (IMPORTANT! Please Read!)
# 1. The results generated by part2&3 will be slightly different from the
# result included in the report due to random initialization of weight&bias
# before training.
# 2. For part 4, we use the weight/bias provided for part1
#
#
print(f"Note (IMPORTANT! Please Read!)\n 1. The results generated by part2&3␣
↪will be slightly different from the result \n included in the report due to␣
↪random initialization of weight&bias for each run.\n 2. For Part 4, we use the␣
↪weight/bias provided for part1")
```

```
Note (IMPORTANT! Please Read!)
 1. The results generated by part2&3 will be slightly different from the result
 included in the report due to random initialization of weight&bias for each
run.
 2. For Part 4, we use the weight/bias provided for part1
```

[2]:
```
import numpy as np
import getopt
import sys

class Model:
    def __init__(self, n0, n1, n2):
        self.n_0 = n0
        self.n_1 = n1
        self.n_2 = n2

        self.reinitialize_weights = True

        # weights and biases initialized as placeholders.
        # reinitialized later
        self.weights_1 = np.zeros((self.n_0, self.n_1), dtype=int)
```

```
        self.weights_2 = np.zeros((self.n_1, self.n_2), dtype=int)
        self.biases_1 = np.zeros((1, self.n_1), dtype=int)
        self.biases_2 = np.zeros((1, self.n_2), dtype=int)

        self.activations_1 = np.zeros((1, self.n_1), dtype=int)
        self.activations_2 = np.zeros((1, self.n_2), dtype=int)
        self.sensitivities_1 = np.zeros((1, self.n_1), dtype=int)
        self.sensitivities_2 = np.zeros((1, self.n_2), dtype=int)

        self.hyper_params = HyperParams()

        self.model_info = Info()


class HyperParams:
    def __init__(self, no_training_steps=800, alpha_sch=2, percentage=0.98):
        self.alpha_list = [0.1, 0.2, 0.3]
        self.zeta_list = [0.5, 1, 1.5]
        self.x0_list = [0.5, 1, 1.5]
        self.max_epochs = 700  # empirically chosen
        self.tolerance = 0.05
        self.training_steps = no_training_steps
        self.learning_rate = self.alpha_list[1]
        self.lr_scheduling_option = alpha_sch  # {0: no_sch, 1: step_sch, 2:␣
 ↪per_sch}
        self.lr_perc_decrease = percentage
        self.zeta = self.zeta_list[1]
        self.x0 = self.x0_list[1]
        self.cost_fn = 0  # {0: quadratic, 1: cross-entropy}

class Info:
    def __init__(self, total_epochs=0, last_epoch_error=0.0, convergence=False):
        self.total_epochs_req = total_epochs
        self.last_epoch_error = last_epoch_error
        self.converged = convergence
```

[3]:
```
import pandas as pd
import matplotlib.pyplot as plt
sheets = {}

def update_sheet(writer, sheet_name, sheet_obj):
    df_obj = {
        'Model architecture': sheet_obj['model_arch_list'],
        'Model weights': sheet_obj['model_weight_list'],
        'Model biases': sheet_obj['model_bias_list'],
        '# Training epochs': sheet_obj['total_epochs_req_list'],
        'Learning Rate': sheet_obj['learning_rate_list'],
```

```
        'Zeta': sheet_obj['zeta_list'],
        'X0': sheet_obj['x0_list'],
        'Cost Function': sheet_obj['cost_fn_list'],
        'Last epoch error': sheet_obj['last_epoch_error_list'],
        'Did converge?': sheet_obj['converged_list'],
    }
    df = pd.DataFrame(df_obj)
    df.to_excel(writer, sheet_name=sheet_name, index=False)

def save_data(sheet_name, model):
    try:
        sheet_obj = sheets[sheet_name]
    except KeyError:
        sheet_obj = {}
    if not sheet_obj:
        sheet_obj = {'model_arch_list': [], 'model_weight_list': [],␣
↪'model_bias_list': [],
                     'total_epochs_req_list': [], 'learning_rate_list': [],␣
↪'zeta_list': [],
                     'x0_list': [], 'cost_fn_list': [], 'last_epoch_error_list':␣
↪[],
                     'converged_list': []}
    sheet_obj['model_arch_list'].append("[ " + str(model.n_0) + ", " + str(model.
↪n_1) +
                                        ", " + str(model.n_2) + "]")
    sheet_obj['model_weight_list'].append("Weights1 = " + str(model.weights_1) +
                                          "\nWeights2 = " + str(model.weights_2))
    sheet_obj['model_bias_list'].append("Biases1 = " + str(model.biases_1) +
                                        "\nBiases2 = " + str(model.biases_2))
    sheet_obj['total_epochs_req_list'].append(model.model_info.total_epochs_req)
    sheet_obj['learning_rate_list'].append(model.hyper_params.learning_rate)
    sheet_obj['zeta_list'].append(model.hyper_params.zeta)
    sheet_obj['x0_list'].append(model.hyper_params.x0)
    sheet_obj['cost_fn_list'].append("Quadratic" if model.hyper_params.cost_fn␣
↪== 0
                                     else "Cross-Entropy")
    sheet_obj['last_epoch_error_list'].append(model.model_info.last_epoch_error)
    sheet_obj['converged_list'].append("Yes" if model.model_info.converged else␣
↪"No")
    sheets[sheet_name] = sheet_obj

def export_data():
    print("Starting export")
    writer = pd.ExcelWriter('Results.xlsx', engine='xlsxwriter')
    if not writer:
        print("Error while opening writer. Exiting.")
```

```
        return
    for sheet_name in sheets:
        sheet_obj = sheets[sheet_name]
        if not sheet_obj:
            print("Skipping sheet - %s " % sheet_name)
            continue
        print("Updating sheet - %s " % sheet_name)
        update_sheet(writer, sheet_name, sheet_obj)
    writer.save()
    print("Data exported")

def print_table_a(sheet_name):
    global sheets
    try:
        sheet_obj = sheets[sheet_name]
        simple_table = {
            'alpha': sheet_obj['learning_rate_list'],
            'zeta': sheet_obj['zeta_list'],
            'X\u2080': sheet_obj['x0_list'],
            'Final Epoch Error': sheet_obj['last_epoch_error_list'],
            'Convergence': sheet_obj['converged_list'],
            '# Training Epochs': sheet_obj['total_epochs_req_list']
        }
        df = pd.DataFrame(simple_table)
        print(df)
    except KeyError:
        print("Sheet doesn't exist")
```

[4]:
```
def transfer_ftn(n_l, x0):
    a_l = np.tanh(n_l / (2 * x0))
    return a_l

# we only save a_l NOT n_l if using bipolar sigmoid transfer function
def derivative_transfer_ftn(a_l, x0):
    derivative = ((1 + a_l) * (1 - a_l)) / (2 * x0)
    return derivative

def init_weights_biases(model):
    if model.reinitialize_weights:
        model.weights_1 = np.random.uniform(-1 * model.hyper_params.zeta,
                                            model.hyper_params.zeta, model.
 ↪weights_1.shape)
        model.weights_2 = np.random.uniform(-1 * model.hyper_params.zeta,
                                            model.hyper_params.zeta, model.
 ↪weights_2.shape)
        model.biases_1 = np.random.uniform(-1 * model.hyper_params.zeta,
```

```
                                                    model.hyper_params.zeta, model.
↪biases_1.shape)
        model.biases_2 = np.random.uniform(-1 * model.hyper_params.zeta,
                                                    model.hyper_params.zeta, model.
↪biases_2.shape)
    return [model.weights_1, model.weights_2], [model.biases_1, model.biases_2]

def train_nn(x_train, y_train, model):
    Q = len(x_train)
    weight_list, bias_list = init_weights_biases(model)
    weight_list_len = len(weight_list)
    for epoch in range(model.hyper_params.max_epochs):
        epoch_error = 0
        for iteration in range(Q):
            x = x_train[iteration]
            y = y_train[iteration]
            x = np.array(x).reshape((1, len(x)))
            y = np.array(y).reshape((1, len(y)))

            # Calculate activations for all layers
            # don't need to save n_l if we are using bipolar sigmoid transfer␣
↪function
            a_l_list = [x]
            for i in range(len(weight_list)):
                n_l = np.matmul(a_l_list[-1], weight_list[i]) + bias_list[i]
                a_l = transfer_ftn(n_l, model.hyper_params.x0)
                a_l_list.append(a_l)

            # calculating the error for this example
            y_hat = a_l_list[-1]  # activation of the last layer
            example_error = np.matmul(y_hat - y, (y_hat - y).T)
            example_error = np.asscalar(example_error)
            epoch_error = epoch_error + example_error

            # Calculate sensitivities for last layer. Performs element-wise␣
↪multiplication.
            # quadratic cost function
            if model.hyper_params.cost_fn == 0:
                s_L = np.multiply((y_hat - y), derivative_transfer_ftn(y_hat,␣
↪model.hyper_params.x0))
            # cross entropy cost function
            elif model.hyper_params.cost_fn == 1:
                s_L = y_hat - y

            # Calculate sensitivites for other layers
            sensitivities_list = [s_L]
```

```
            for l in range(weight_list_len - 1, 0, -1):
                s_l = np.multiply(np.matmul(sensitivities_list[0],␣
↪weight_list[l].T), \
                                  derivative_transfer_ftn(a_l_list[l], model.
↪hyper_params.x0))
                sensitivities_list.insert(0, s_l)

            # Update weights and biases
            for l in range(weight_list_len):
                weight_list[l] = weight_list[l] - \
                                 (model.hyper_params.learning_rate *
                                  np.matmul(a_l_list[l].T,␣
↪sensitivities_list[l]))

                bias_list[l] = bias_list[l] - \
                               (model.hyper_params.learning_rate *␣
↪sensitivities_list[l])

        # epoch error is not normalized (not divided by number of examples)
        if epoch_error < model.hyper_params.tolerance:
            break

    num_training_epochs = epoch + 1
    if num_training_epochs < model.hyper_params.max_epochs:
        convergence = True
    else:
        convergence = False
    update_model_info(model, weight_list, bias_list, num_training_epochs,␣
↪epoch_error, convergence)
    return model

def update_model_info(model, weight_list, bias_list, num_training_epochs,␣
↪epoch_error, convergence):
    model.weights_1 = weight_list[0]
    model.weights_2 = weight_list[1]
    model.biases_1 = bias_list[0]
    model.biases_2 = bias_list[1]
    model.model_info.total_epochs_req = num_training_epochs
    model.model_info.last_epoch_error = epoch_error
    model.model_info.converged = convergence

def extract_model_info(model, sheet_name, verbose=False, export_to_excel=True):
    if verbose:
        ␣
↪print("-----------------------------------------------------------------------------")
        print(f"Learning rate = {model.hyper_params.learning_rate} | "
```

```
            f"Zeta = {model.hyper_params.zeta} | "
            f"x0 = {model.hyper_params.x0}")
        print(f"Convergence = {model.model_info.converged} | "
            f"Training Epochs = {model.model_info.total_epochs_req} | "
            f"Squared Error = {model.model_info.last_epoch_error}")
        ␣
↪print("--------------------------------------------------------------------------------")
    elif export_to_excel:
        save_data(sheet_name, model)

def part_2a(x_train, y_train, model, sheet_name, table=None):
    # TODO
    # Look for patterns when do we get non-convergent results
    # Try all 3X3X3=27 hyper parameter combinations of alpha, zeta and x0
    num_convergence = 0
    for alpha in model.hyper_params.alpha_list:
        for zeta in model.hyper_params.zeta_list:
            for x0 in model.hyper_params.x0_list:
                model.hyper_params.learning_rate = alpha
                model.hyper_params.zeta = zeta
                model.hyper_params.x0 = x0
                model = train_nn(x_train, y_train, model)
                if model.model_info.converged:
                    num_convergence += 1
                # extract_model_info(model, sheet_name, verbose=True)
                extract_model_info(model, sheet_name, verbose=False) # for demo␣
↪purpose
    ␣
↪print("----------------------------------------------------------------------\n")
    print(f"Number of convergent hyper parameter combinations =␣
↪{num_convergence} (out of 27)")
    ␣
↪print("----------------------------------------------------------------------\n")

def part_2b(x_train, y_train, cost_fn, sheet_name, table=None):
    N1_list = [1, 2, 4, 6, 8, 10]
    convergence_list = []
    for i in range(len(N1_list)):
        model = Model(2, N1_list[i], 1)
        model.hyper_params.cost_fn = cost_fn
        num_convergence = 0
        for iters in range(100):
            model = train_nn(x_train, y_train, model)
            if model.model_info.converged:
                num_convergence += 1
            extract_model_info(model, sheet_name, verbose=False)
        convergence_list.append(num_convergence)
```

```
        print(f"Convergence for N1 = %d -> %d" % (N1_list[i], num_convergence))

    print(f"Convergence results for N1 = [1,2,4,6,8,10] (out of 100):␣
↪{convergence_list}")

    # Results mostly converge for N1=4 and above. For N1=2, almost 70% of the␣
↪times,
    # it converges. For N1=1, it doesn't converge at all.
    # This is probably because the XOR problem is not linearly separable and we␣
↪need a higher
    # number of neurons in the hidden layer to approximate the function (see␣
↪universality theorem).

def xor_weight_validation(x_train, y_train, model, sheet_name):
    model.hyper_params.max_epochs = 1

    # Setting initial weights and biases for xor weight validation
    model.weights_1 = np.array([[0.197, 0.3191, -0.1448, 0.3594],
                                [0.3099, 0.1904, -0.0347, -0.4861]]).
↪reshape(model.weights_1.shape)
    model.weights_2 = np.array([0.4919, -0.2913, -0.3979, 0.3581]).reshape(model.
↪weights_2.shape)
    model.biases_1 = np.array([-0.3378, 0.2771, 0.2859, -0.3329]).reshape(model.
↪biases_1.shape)
    model.biases_2 = np.array([-0.1401]).reshape(model.biases_2.shape)
    model.reinitialize_weights = False

    model = train_nn(x_train, y_train, model)

    print("Weights1=", model.weights_1, sep="\n")
    print("Biases1=", model.biases_1, sep="\n")
    print("Weights2=", model.weights_2, sep="\n")
    print("Biases2=", model.biases_2, sep="\n")

def final_verification(x_train, y_train, model):
    model.hyper_params.learning_rate = 0.2
    model.hyper_params.zeta = 1.0
    model.hyper_params.x0 = 1.0
    model.hyper_params.cost_fn = 1
    model.hyper_params.max_epochs = 1
    model.weights_1 = np.array([[0.197, 0.3191, -0.1448, 0.3594],
                                [0.3099, 0.1904, -0.0347, -0.4861]]).
↪reshape(model.weights_1.shape)
    model.weights_2 = np.array([0.4919, -0.2913, -0.3979, 0.3581]).reshape(model.
↪weights_2.shape)
```

```
    model.biases_1 = np.array([-0.3378, 0.2771, 0.2859, -0.3329]).reshape(model.
 ↪biases_1.shape)
    model.biases_2 = np.array([-0.1401]).reshape(model.biases_2.shape)
    model.reinitialize_weights = False
    model = train_nn(x_train, y_train, model)
    print("Weights1=", model.weights_1, sep="\n")
    print("Biases1=", model.biases_1, sep="\n")
    print("Weights2=", model.weights_2, sep="\n")
    print("Biases2=", model.biases_2, sep="\n")
```

[5]:
```
# ---------------------------------------------------
# Results Demo
# ---------------------------------------------------
```

[6]:
```
# Part 1: XOR Weights Validation
x_train = [[1, 1], [1, -1], [-1, 1], [-1, -1]]
y_train = [[-1], [1], [1], [-1]]
model = Model(2, 4, 1)
model.hyper_params = HyperParams()
xor_weight_validation(x_train, y_train, model, sheet_name="XOR weights␣
 ↪validation")
```

```
Weights1=
[[ 0.19347555  0.31675476 -0.144748    0.36374521]
 [ 0.30686735  0.18845288 -0.03301594 -0.48859019]]
Biases1=
[[-0.32243413  0.26504268  0.27330512 -0.32503622]]
Weights2=
[[ 0.47534885]
 [-0.27642811]
 [-0.38395025]
 [ 0.34801327]]
Biases2=
[[-0.08027444]]
```

[7]:
```
# Part 2a: Verying alpha, zeta and x0 (Quadratic Cost Function)
# using quadratic cost function

model = Model(2, 4, 1)
model.hyper_params.cost_fn = 0
part_2a(x_train, y_train, model, sheet_name="A-Z-X0 variations (Quad)")
print_table_a("A-Z-X0 variations (Quad)") # Table 1
```

```
----------------------------------------------------------------------

Number of convergent hyper parameter combinations = 16 (out of 27)
----------------------------------------------------------------------
```

```
   Alpha Zeta  Xo  Final Epoch Error Convergence  # Training Epochs
0    0.1  0.5  0.5           0.049993         Yes                151
1    0.1  0.5  1.0           0.406422          No                700
2    0.1  0.5  1.5           4.049384          No                700
3    0.1  1.0  0.5           0.049455         Yes                110
4    0.1  1.0  1.0           0.049870         Yes                485
5    0.1  1.0  1.5           3.924008          No                700
6    0.1  1.5  0.5           3.996796          No                700
7    0.1  1.5  1.0           0.049971         Yes                588
8    0.1  1.5  1.5           0.062027          No                700
9    0.2  0.5  0.5           0.049036         Yes                 64
10   0.2  0.5  1.0           4.204976          No                700
11   0.2  0.5  1.5           4.093781          No                700
12   0.2  1.0  0.5           0.049674         Yes                 54
13   0.2  1.0  1.0           0.049949         Yes                236
14   0.2  1.0  1.5           0.055472          No                700
15   0.2  1.5  0.5           0.049167         Yes                555
16   0.2  1.5  1.0           0.049913         Yes                248
17   0.2  1.5  1.5           0.049957         Yes                611
18   0.3  0.5  0.5           5.313238          No                700
19   0.3  0.5  1.0           4.310545          No                700
20   0.3  0.5  1.5           4.136845          No                700
21   0.3  1.0  0.5           0.048860         Yes                 29
22   0.3  1.0  1.0           0.049617         Yes                187
23   0.3  1.0  1.5           0.049693         Yes                351
24   0.3  1.5  0.5           0.049865         Yes                 43
25   0.3  1.5  1.0           0.049675         Yes                104
26   0.3  1.5  1.5           0.049973         Yes                442
```

[8]:
```
# part 2b: Varying N1 (hidden layer neurons) (Quadratic Cost Functions)
part_2b(x_train, y_train, cost_fn=0, sheet_name="N1 variations (Quad)")
```

```
Convergence for N1 = 1 -> 0
Convergence for N1 = 2 -> 84
Convergence for N1 = 4 -> 99
Convergence for N1 = 6 -> 100
Convergence for N1 = 8 -> 100
Convergence for N1 = 10 -> 98
Convergence results for N1 = [1,2,4,6,8,10] (out of 100): [0, 84, 99, 100, 100,
98]
```

[9]:
```
# Part 3a: Varying alpha, zeta and x0 (hidden layer neurons) (Cross Entropy Cost␣
↪Function)
# using cross entropy cost ftn
model.hyper_params.cost_fn = 1
part_2a(x_train, y_train, model, sheet_name="A-Z-X0 variations (CrsEnt)")
```

```
print_table_a("A-Z-XO variations (Quad)") # Table 3
```

```
----------------------------------------------------------------------

Number of convergent hyper parameter combinations = 16 (out of 27)
----------------------------------------------------------------------

   Alpha Zeta Xo  Final Epoch Error Convergence  # Training Epochs
0    0.1  0.5  0.5           0.049993         Yes                151
1    0.1  0.5  1.0           0.406422          No                700
2    0.1  0.5  1.5           4.049384          No                700
3    0.1  1.0  0.5           0.049455         Yes                110
4    0.1  1.0  1.0           0.049870         Yes                485
5    0.1  1.0  1.5           3.924008          No                700
6    0.1  1.5  0.5           3.996796          No                700
7    0.1  1.5  1.0           0.049971         Yes                588
8    0.1  1.5  1.5           0.062027          No                700
9    0.2  0.5  0.5           0.049036         Yes                 64
10   0.2  0.5  1.0           4.204976          No                700
11   0.2  0.5  1.5           4.093781          No                700
12   0.2  1.0  0.5           0.049674         Yes                 54
13   0.2  1.0  1.0           0.049949         Yes                236
14   0.2  1.0  1.5           0.055472          No                700
15   0.2  1.5  0.5           0.049167         Yes                555
16   0.2  1.5  1.0           0.049913         Yes                248
17   0.2  1.5  1.5           0.049957         Yes                611
18   0.3  0.5  0.5           5.313238          No                700
19   0.3  0.5  1.0           4.310545          No                700
20   0.3  0.5  1.5           4.136845          No                700
21   0.3  1.0  0.5           0.048860         Yes                 29
22   0.3  1.0  1.0           0.049617         Yes                187
23   0.3  1.0  1.5           0.049693         Yes                351
24   0.3  1.5  0.5           0.049865         Yes                 43
25   0.3  1.5  1.0           0.049675         Yes                104
26   0.3  1.5  1.5           0.049973         Yes                442
```

[10]:
```
# Part 3b: Varying N1 (hidden layer neurons) (Cross Entropy Cost Function)
part_2b(x_train, y_train, cost_fn=1, sheet_name="N1 variations (CrsEnt)")
```

```
Convergence for N1 = 1 -> 0
Convergence for N1 = 2 -> 61
Convergence for N1 = 4 -> 96
Convergence for N1 = 6 -> 98
Convergence for N1 = 8 -> 99
Convergence for N1 = 10 -> 100
Convergence results for N1 = [1,2,4,6,8,10] (out of 100): [0, 61, 96, 98, 99,
100]
```

[11]:
```
# Part 4: Weights and biases for N1 = 4, alpha = 0.2, zeta = 1.0 and x0 = 1.0␣
↪after 1 epoch
final_verification(x_train, y_train, model)
```

```
Weights1=
[[ 0.19383967  0.30895515 -0.14727152  0.36911844]
 [ 0.29881712  0.18811518 -0.02889164 -0.48928638]]
Biases1=
[[-0.30754551  0.24693804  0.25732658 -0.30887916]]
Weights2=
[[ 0.44724602]
 [-0.24360234]
 [-0.35686098]
 [ 0.32501904]]
Biases2=
[[-0.01923564]]
```