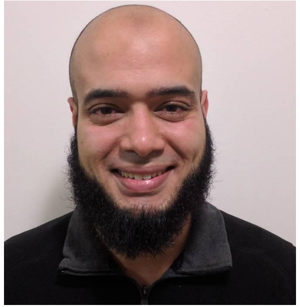
Machine Learning

Homework 2 - Backpropagation

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Problem #1: Sklearn

- You will study the API function for NN in sklearn
  - sklearn.neural_network.MLPRegressor
- Apply NN on the previous dataset: data2_200x30.csv
- Find a good network that has low validation error

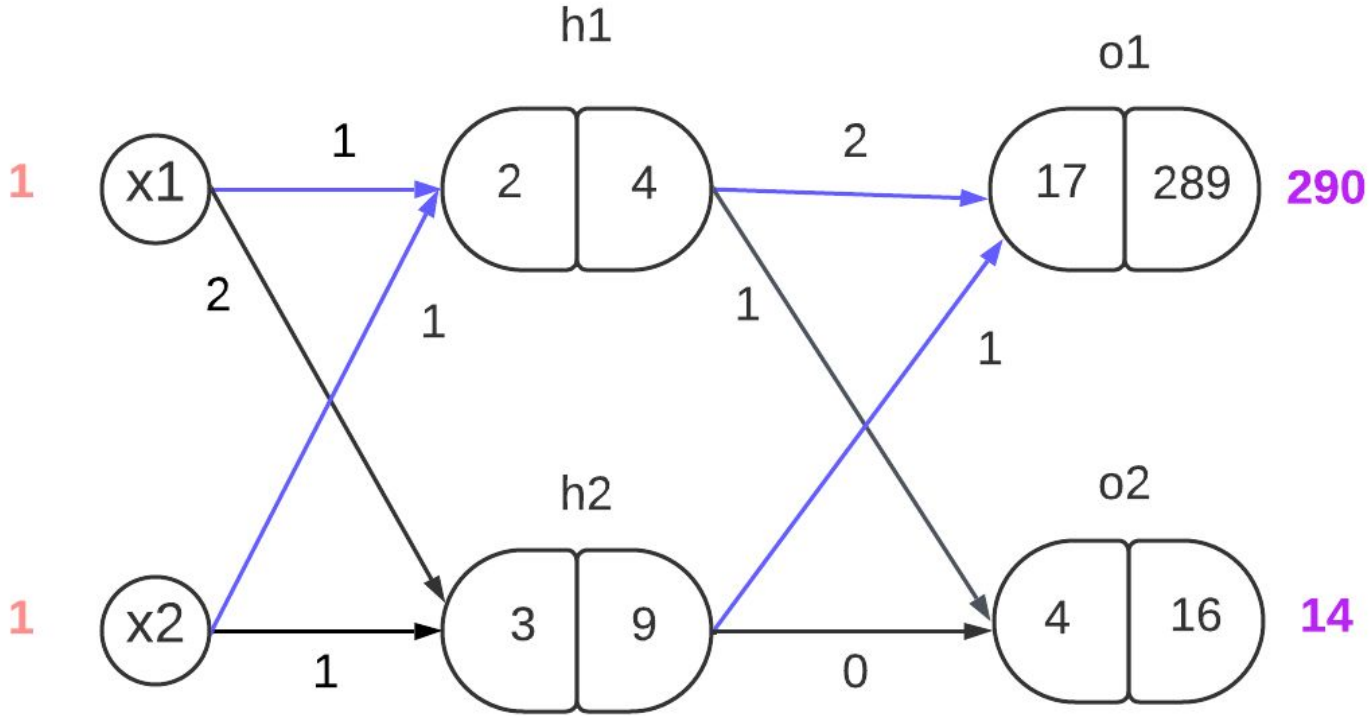
```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Regressors Homework')

    parser.add_argument('--dataset', type=str, default='data2_200x30.csv')
    parser.add_argument('--preprocessing', type=int, default=1,
                        help='0 for no processing, 1 for min/max scaling and 2 for standrizing')
```

Problem #2: Backpropagation

- In this homework, we would like to implement a neural network
- Specifically, we would like to do a single backpropagation step
    - Then training is just to keep repeating it
- We will focus only with 3 layers nn: input, hidden, output. **No bias**
- Implement the network and validate it using the lecture example
    - Then extend it to also allow sigmoid activation function
    - Use activation on all nodes, as we did in the lecture example
- We will feed a specific input/weights/outputs to compare and judge correctness
- The project requires project building skills
    - And also preferred OOP skills

Review by paper and pencil



One way

- This is just one way suggested for you
- Create the following classes
- **NeuralNetwork, NeuronLayer, Neuron**
  - NeuralNetwork class has 2 NeuronLayers: hidden and output
  - Each layer has its Neuron nodes
  - Each neuron is connected with weights to the previous layer
    - Matrices are usually backward: from a layer to previous one
    - E.g. if input is 5 values and hidden is 10, then the weight matrix is 10x5 NOT 5x10

Classes Methods

- **NeuralNetwork**
  - feed_forward
  - compute_delta (for output and hidden)
  - update_weights (for output and hidden)
  - **train_step**: it just calls the previous 3 methods
- **NeuronLayer**
  - feed_forward over its nodes
- **Neuron**
  - compute activation: supports {polynomial, sigmoid and identity}
  - compute activation derivative
  - Calc_net_out: compute the input net and apply activation ont

```python
def poly():     # 2 x 2 x 2
    hidden_layer_weights = np.array([[1, 1],
                                     [2, 1]])
    output_layer_weights = np.array([[2, 1],
                                     [1, 0]])

    nn = NeuralNetwork(hidden_layer_weights, output_layer_weights, 'poly')

    nn.train_step([1, 1], [290, 14])
```

```
network output: [289, 16]
Delta o[0]: -34.0
Delta o[1]: 16.0
Delta h[0]: -208.0
Delta h[1]: -204.0
node o: 0 - w_ho: 0: Delata -136.0 => new w = 70.0
node o: 0 - w_ho: 1: Delata -306.0 => new w = 154.0
node o: 1 - w_ho: 0: Delata 64.0 => new w = -31.0
node o: 1 - w_ho: 1: Delata 144.0 => new w = -72.0
node h: 0 - w_ih: 0: Delata -208.0 => new w = 105.0
node h: 0 - w_ih: 1: Delata -208.0 => new w = 105.0
node h: 1 - w_ih: 0: Delata -204.0 => new w = 104.0
node h: 1 - w_ih: 1: Delata -204.0 => new w = 103.0
```

```python
def sigm():     # 2 4 3
    hidden_layer_weights = np.array([[0.1, 0.1],      # 4x2 NOT 2x4
                                     [0.2, 0.1],
                                     [0.1, 0.3],
                                     [0.5, 0.01]])

    output_layer_weights = np.array([[0.1, 0.2, 0.1, 0.2],
                                     [0.1, 0.1, 0.1, 0.5],
                                     [0.1, 0.4, 0.3, 0.2]])

    nn = NeuralNetwork(hidden_layer_weights, output_layer_weights, 'sigmoid')

    nn.train_step([1, 2], [0.4, 0.7, 0.6])
```

```
network output: [0.5913212667539777, 0.6219200057374265, 0.6508562785102494]
Delta o[0]: 0.04623477887224621
Delta o[1]: -0.01835937944358026
Delta o[2]: 0.011556701931083076
Delta h[0]: 0.000963950492482261
Delta h[1]: 0.0028912254002713203
Delta h[2]: 0.0013867143674З1997
Delta h[3]: 0.000556197739142091
node o: 0 - w_ho: 0: Delata 0.026559222739603632 => new w = 0.0867203886301982
node o: 0 - w_ho: 1: Delata 0.027680191578841717 => new w = 0.18615990421057915
node o: 0 - w_ho: 2: Delata 0.030893513891333994 => new w = 0.08455324305433301
node o: 0 - w_ho: 3: Delata 0.028996038295713737 => new w = 0.18550198085214314
node o: 1 - w_ho: 0: Delata -0.010546408134670482 => new w = 0.10527320406733524
node o: 1 - w_ho: 1: Delata -0.010991533920193718 => new w = 0.10549576696009687
node o: 1 - w_ho: 2: Delata -0.01226751284879592 => new w = 0.1061337564243979
node o: 1 - w_ho: 3: Delata -0.01151404380893776 => new w = 0.505757021904689
node o: 2 - w_ho: 0: Delata 0.006638660943333523 => new w = 0.09668066952833325
node o: 2 - w_ho: 1: Delata 0.006918854837737182 => new w = 0.39654057258113146
node o: 2 - w_ho: 2: Delata 0.00772204691694 1944 => new w = 0.2961389765415290 4
node o: 2 - w_ho: 3: Delata 0.007247759802026145 => new w = 0.1963761200989869 4
```

Code Namings

- You may use good variables **naming** that match the rules
- dE_dO_net for $\partial E/\partial o\_net$
- dE_dO_out for $\partial E/\partial o\_out$
- dE_dH_net for $\partial E/\partial h\_net$
- dE_dH_out for $\partial E/\partial h\_out$
- d_net_d_out for $\partial o\_net/\partial o\_out$ and $\partial h\_net/\partial h\_out$
- d_out_d_net for $\partial o\_out/\partial o\_net$ and $\partial h\_out/\partial h\_net$
- dE_dW for $\partial E/\partial w$

Problem #3: Derivatives

- Compute the **derivative** of the following functions:
  - Sigmoid
  - Tanh
  - Reul
  - See here or google
- Show that tanh(net) = 2 x **sigmoid**(2 x net) - 1

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."