036049: Applied ML4SE, Winter 2023/2024
Homework 3

Due: February 27, 2024

This homework is related to material starting in Chapter 3-6 related to neural networks.

Theory

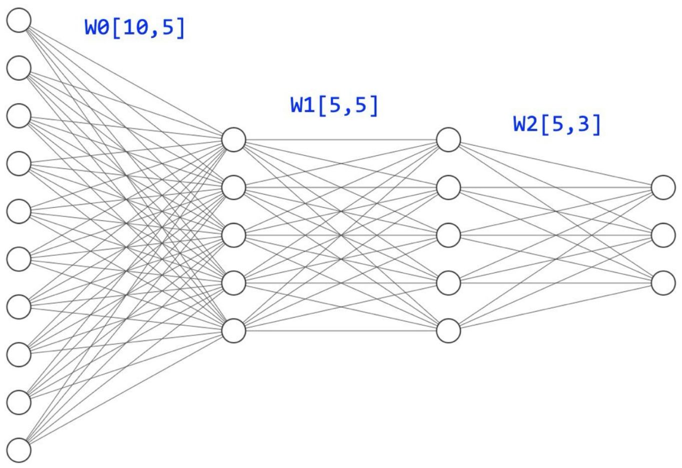
In this section you will consider very basic questions related to machine learning. Please answer the following multiple choice questions:

1. Please do Problem 3.1 in UDL.
2. Please do Problem 3.5 in UDL.
3. Please do Problem 3.10 in UDL.
4. Please do Problem 4.4 in UDL.
5. Please do Problem 4.5 in UDL.
6. Please do Problem 4.6 in UDL.

Remember to provide the correct answers along with explanations for each question.

Computation

1. In this exercise, you will implement a **fully connected neural network (NN)** using `Python` and `NumPy` as in the figure below.



Specifically, your task is to implement the following functions:

- `init_weights(n_inputs, n_hidden, n_output)`
  This function should randomly initialize the NN's weights using the `np.random.randn()` function in `NumPy`. It should return the weight matrices `W0`,`W1`, and `W2` for the input-to-hidden, hidden-to-hidden, and hidden-to-output layers, respectively. The number of input, hidden, and output units should be passed as arguments to the function.
- `feedforward(x,W0,W1,W2)`
  This function should implement the feedforward operation of the neural network. It should take as input an example `x` and the weight matrices `W0,W1,W2` and return the pre-activations `z0,z1,z2` and activations `a0,a1,a2` for the input, hidden, and output layers, respectively. The feedfoward operation should consist of matrix multiplications and pointwise application of the non-linear function f, which can be chosen as the sigmoid function.
- `predict(x,W0,W1,W2)`
  This function should take as input an example `x` and the weight matrices `W0,W1,W2` and return the prediction of the NN for that example. This can be done by passing the example `x` through the feedfoward operation and returning the network's output.

You can test your implementation using the following code snippet:

```
X_train = np.random.randn(1000,10)
Y_train = np.random.randn(1000,3)

n_inputs = 10
n_hidden = 5
n_output = 3
W0, W1, W2 = init_weights(n_inputs, n_hidden, n_output)

Y_predicted = predict(X_train,W0,W1,W2)
```

2. **Forward propagation**:
   - Define a function `forward_propagation(W,A,f)` that takes in the following parameters:
     - `W`: a matrix of weights for a layer
     - `A`: the activation matrix for the previous layer
     - `f`: the activation function for the current layer
   - The function should first augment the weight matrix `W` by appending the bias vector as the last column and update the `A` matrix by appending a column of ones.
   - Next, compute the matrix product of the augmented `W` and `A` matrices and save the result in a variable `Z`.
   - Apply the activation function `f` element-wise on `Z` and save the result in a variable A_{out}
   - Return the A_{out} matrix
   - Test your function by initializing a weight matrix `W` $= [[1,2],[3,4]]$ and an activation matrix `A`$[[1,2],[3,4],[1,1]]$ and an activation function $f = lambda\, x : x**2$. Your function should return a matrix $[[7,10],[15,22]]$.

3. **Non-linear activation functions**. Here you will implement the sigmoid activation function in Python as follows:

- Define a function named sigmoid that takes in a variable z
- Inside the function, calculate the value of $1/(1 + e * *(-z))$ and store it in a variable named `result`.
- Return the value of `result`.
- Test the function by calling it with input $z = 0$ and print the result. The output should be 0.5
- Test the function again with input $z = 2$ and print the result. The output should be close to 0.88
- Test the function one more time with input $z = -2$ and print the result. The output should be close to 0.12

4. **Implement the ReLU activation function in Python.**

   - Define a function $relu(z)$ that takes in a scalar or array-like input z and returns ReLU activation by applying the ReLU function to each element of the input.
   - Use the function to apply the ReLU activation to a scalar, a list, and a numpy array, and print the results.

5. **PyTorch textbook example:** Type in and run the code in Fig. 7.8 on page 112 in your book which trains a two-layer network on random data. Modify the code to output graphs showing (i) loss convergence, (ii) prediction vs. original output values after training. Experiment with hidden layer dimension and learning rate.