Neural networks

Edited by
Matteo Bertoia

Contents

Software needs

- Anaconda3:

- useful platform for data science, big data, machine learning;
- it allows an easy and user friendly (through GUI) installation for libraries/modules (ex. PyTorch, Numpy…);

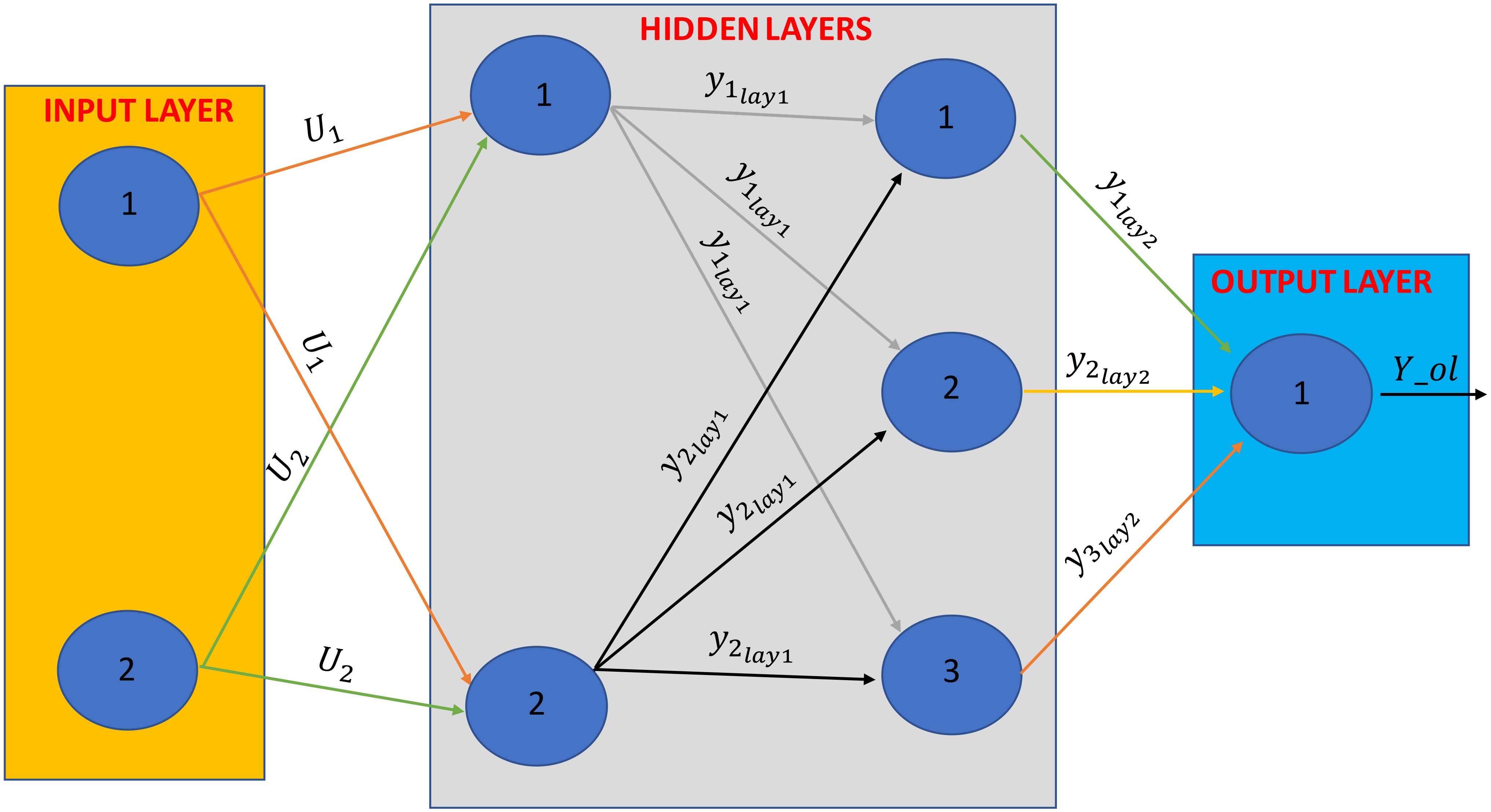
- Python 3.9.13:

- high level language with many good libraries for data science

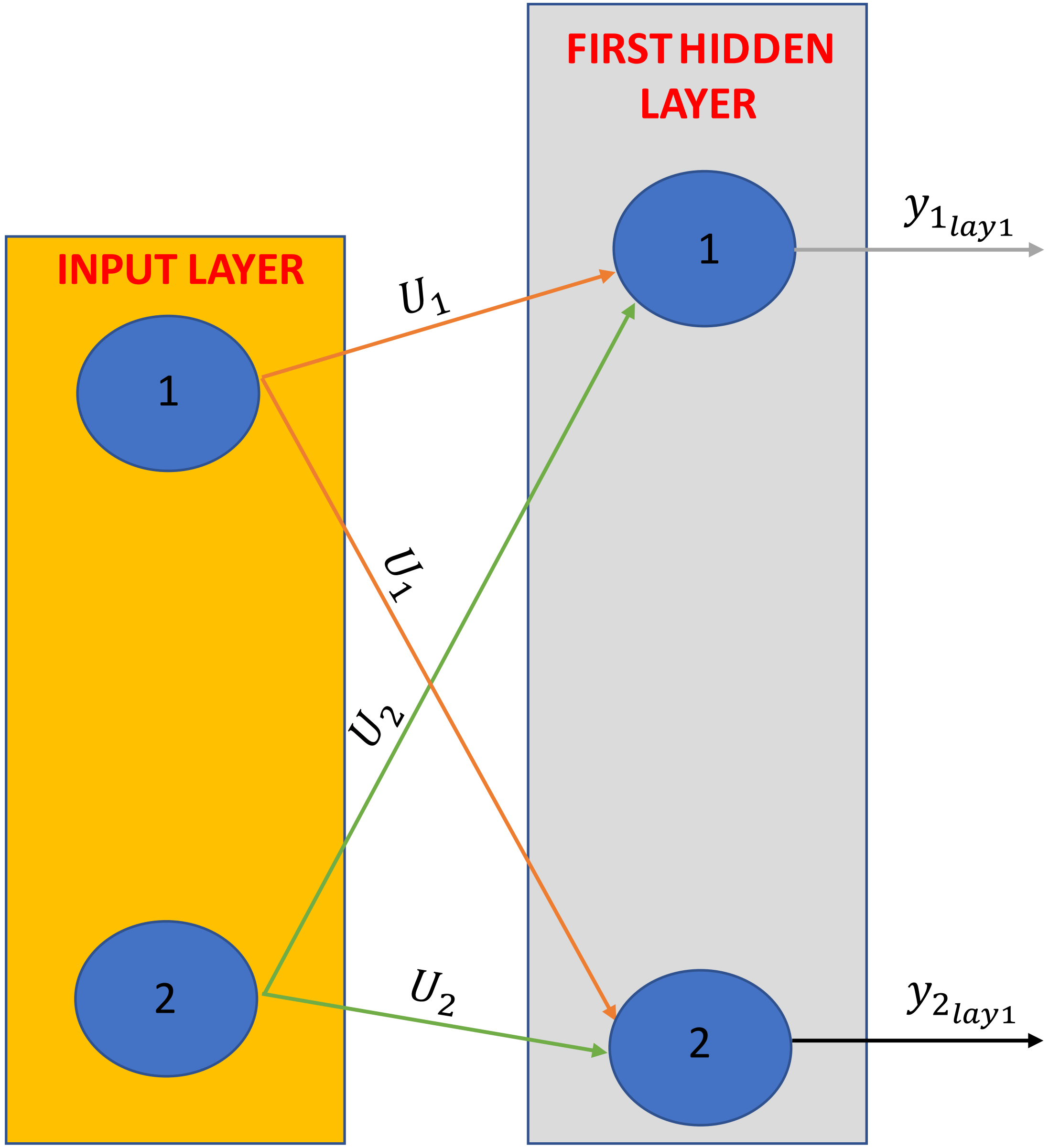
- Jupyter lab:

- useful code editor for data science; it is a good candidate to implement NNs

NN architecture (example feed-forward case)



The output from the first hidden layer => $Y_{lay1} = \left(1 - \sigma\left(A_{lay1}U_{lay1} + B_{lay1}\right)\right) f\left(U_{lay1}\right) + \sigma\left(A_{lay1}U_{lay1} + B_{lay1}\right) g\left(U_{lay1}\right)$



where:

$$A_{lay1}(weight) = \begin{bmatrix} w_{11_{lay1}} & w_{12_{lay1}} \\ w_{21_{lay1}} & w_{22_{lay1}} \end{bmatrix}$$

$$B_{lay1}(bias) = \begin{bmatrix} b_{1_{lay1}} \\ b_{2_{lay1}} \end{bmatrix}$$

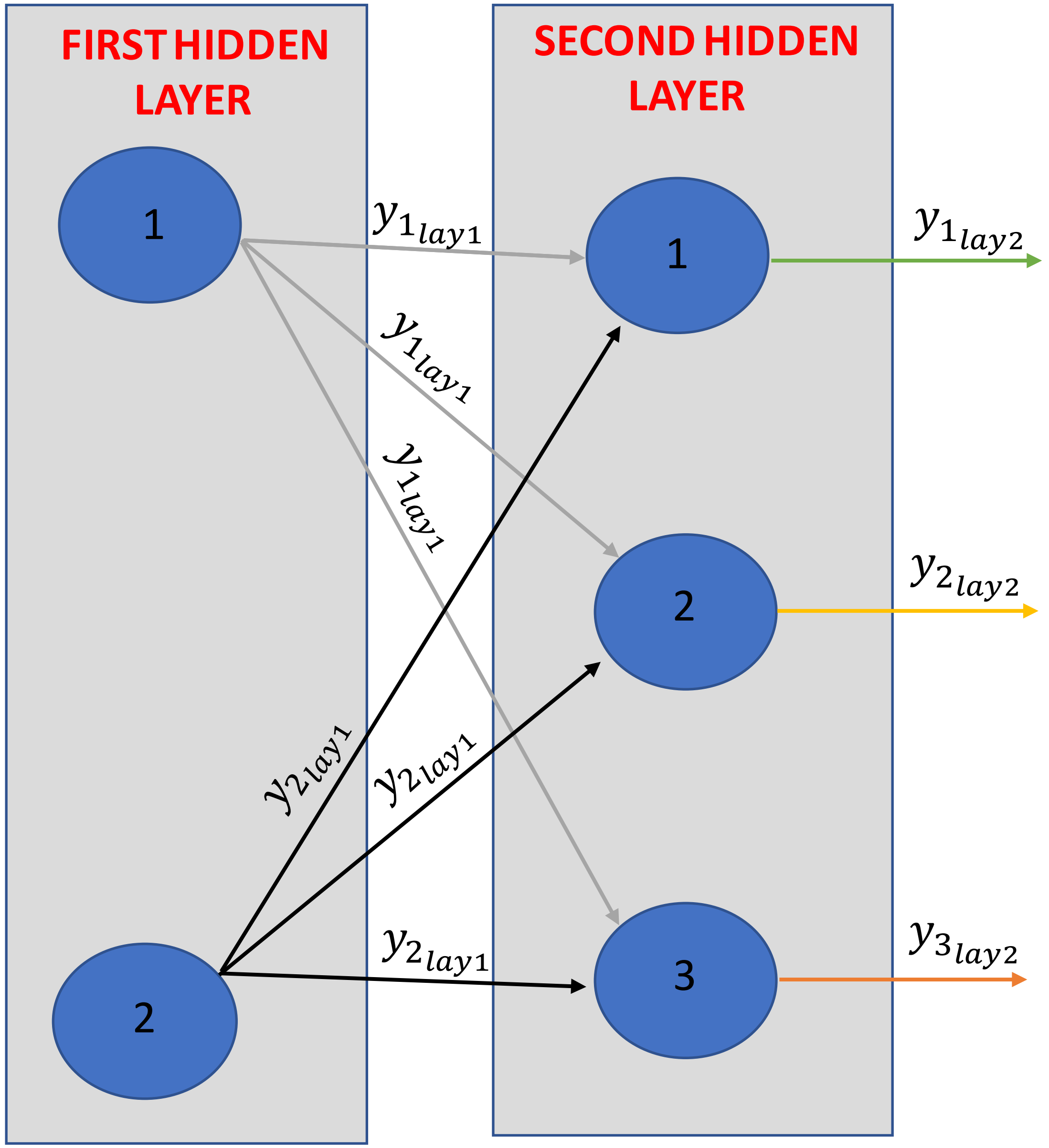
$$Y_{lay1}(output) = \begin{bmatrix} y_{1_{lay1}} \\ y_{2_{lay1}} \end{bmatrix}$$

$$U_{lay1}(input) = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$$

$$\sigma(u) = \begin{cases} 0 & if\ u < 0 \\ u & otherwise \end{cases}$$

f and g are smooth functions

The output from the second hidden layer => $Y_{lay2} = \left(1 - \sigma\left(A_{lay2}U_{lay2} + B_{lay2}\right)\right) f\left(U_{lay2}\right) + \sigma\left(A_{lay2}U_{lay2} + B_{lay2}\right) g\left(U_{lay2}\right)$



where:

$$A_{lay2}(weight) = \begin{bmatrix} w_{11_{lay2}} & w_{12_{lay2}} \\ w_{21_{lay2}} & w_{22_{lay2}} \\ w_{31_{lay2}} & w_{32_{lay2}} \end{bmatrix}$$

$$B_{lay2}(bias) = \begin{bmatrix} b_{1_{lay2}} \\ b_{2_{lay2}} \\ b_{3_{lay2}} \end{bmatrix}$$

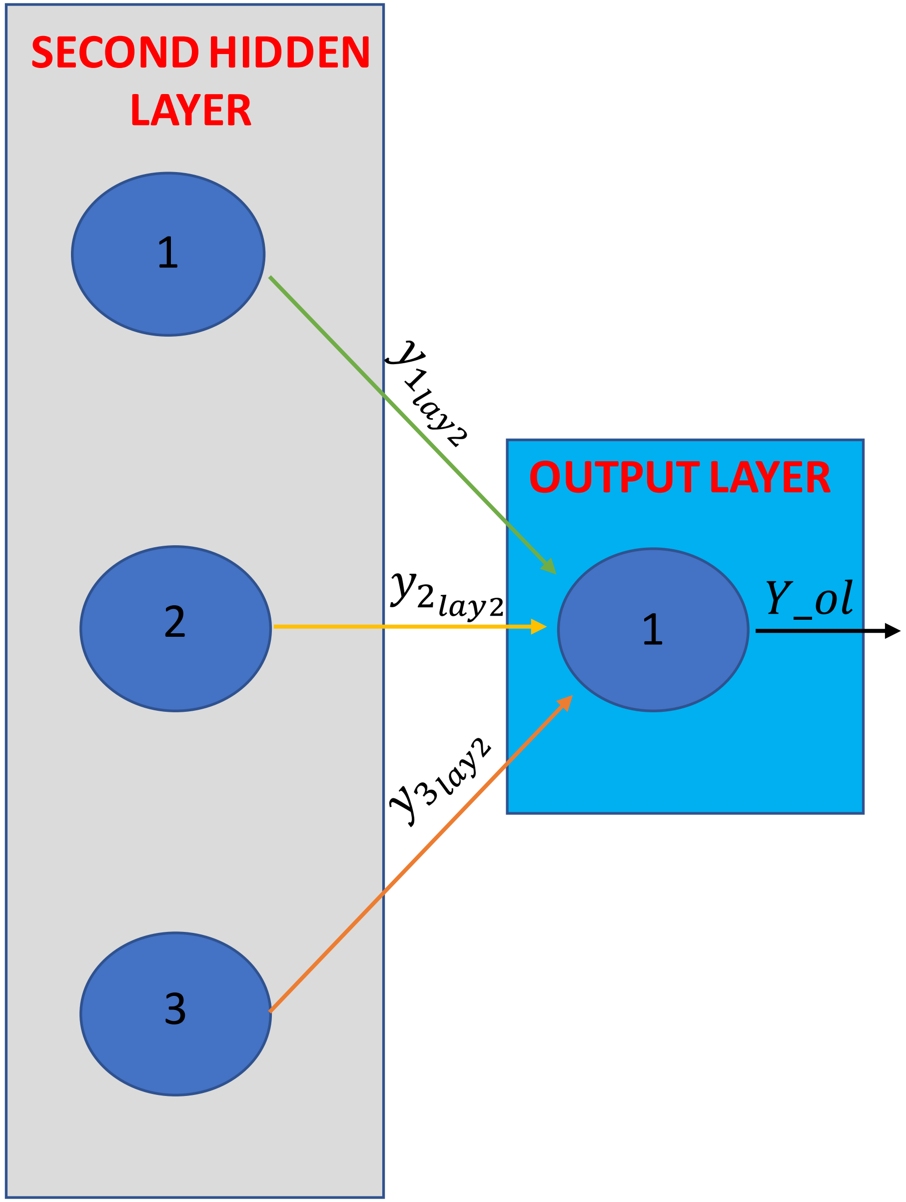
$$Y_{lay2}(output) = \begin{bmatrix} y_{1_{lay2}} \\ y_{2_{lay2}} \\ y_{3_{lay2}} \end{bmatrix}$$

$$U_{lay2}(input) = \begin{bmatrix} y_{1_{lay1}} \\ y_{2_{lay1}} \end{bmatrix}$$

$$\sigma(u) = \begin{cases} 0 & if\ u < 0 \\ u & otherwise \end{cases}$$

$f\ and\ g\ are\ smooth\ functions$

The output from the output layer => $Y_{ol} = \left(1 - \sigma(A_{ol}U_{ol} + B_{ol})\right)f(U_{ol}) + \sigma(A_{ol}U_{ol} + B_{ol})\,g(U_{ol})$



where:

$$A_{ol}(weight) = \begin{pmatrix} w_{11_{ol}} & w_{12_{ol}} & w_{13_{ol}} \end{pmatrix}$$

$$Y_{ol}(output) = y\_ol$$

$$B_{ol}(bias) = b\_ol$$

$$U_{ol}(input) = \begin{bmatrix} y_{1_{lay2}} \\ y_{2_{lay2}} \\ y_{3_{lay2}} \end{bmatrix}$$

$$\sigma(u) = \begin{cases} 0 & if\ u < 0 \\ u & otherwise \end{cases}$$

f *and* g *are smooth functions*

Clarifications

- Giving the matrix A (weight), vector B (bias), u (input), the output for each Layer is made in 3 steps:

1) $Y_0 = Au + B$ (output from linear function)

2) $Y_1 = \sigma(Y_0)$ (output from nonlinear function $ReLU$)

3) $Y_2 = (1 - Y_1)f(u) + Y_1 g(u)$ (output from convex combination of f and g «modern machine learning»)

- The compact form to descrive the output for each Layer is:

$$Y_2 = (1 - \sigma(Au + B))f(u) + \sigma(Au + B)g(u)$$

Coding

PROJECT FILES:

- **main.py**
- **layer_linear_fun.py**
- **layer_ReLU_fun.py**
- **layer_convex_comb_fun.py**
- **trainingNN_fun.py**
- **layer_class.py**

Create the NN

1 Run main.py

2 The User set (from standard input) the dimension of the input/output for Layer 0

3 A tensor of random values (var «**data**») is created as an input for Layer 0

4 Then **apply_linear_layer** is called and a linear function $y_0 = (Au + B)$ is created with random values for A and B, then $y_0(u)$ is calculated with $u := data$

5 Then **apply_ReLU** is called and is calculated $\sigma(y_0) = y_1$

6 Then **apply_convex_comb** is called and a convex combination of f and g is calculated: $y_2 = (1 - y_1)f + y_1 g$

7 Then all the information about Layer 1 are printed to standard output such as A, B, y_0, y_1, y_2

8 Ask the User if wants to add a new Layer (it would be Layer 1). If yes, proceed to 9

YES

9a Repeat the step from 2 to 8 (not 3) for Layer X but this time with $u = y_2$ (the output from the last Layer added).
Note that the input dimension of the current Layer must be the output dimension of the previous one

NO

9b The NN is saved into array (var «**layer_list**»); each element is an object of the Layer class

Training of the NN

1 Run TrainingNN (**layer_list**)

2 Set input and output data (measurements) of the NN

3 Ask User if wants to tune A e B by considering y_0 (or y_1 or y_2) for each Layer

4 Run the training algorithm to find a good approximation for A and B with k steps.
For each step:
- Calculate the output of the NN (seeing NN as a black box)
- Calculate the error between the output of the NN and the measurements (MSE)
- Calculate the partial derivatives with respect the components of A_{ij} and B_{ij} (for each Layer)
- Update A and B (for each Layer)
- Print the error every 500 steps to have an idea about the performances

Clarification

If the NN is what shown in slide 3, the y_{nn} (output of the NN) is calculated as follows:

1) First Initialize y_nn:

$$y_{nn} = U$$

2) For i in range(0, num_layer)

$$Y_0 = A_i \, y_{nn} + B_i$$
$$\mathrm{Y}_1 = ReLU \, (Y_0)$$
$$y_{nn} = (1 - Y_1) f(y_{nn}) + Y_1 g(y_{nn})$$

To calculate the error use MSE (Mean Square Error):

$$Loss^{(n)}\left(A_{i,j}, B_{i,j}\right) = \sum_{k=1}^{M} \frac{\left(ynn_k^{(n)} - ytarget_k\right)^2}{N}$$

Then update the matrixes:

$$A_{i,j}(n+1) = A_{i,j}(n) - l_r \frac{\partial Loss^{(n)}\left(A_{i,j}, B_{i,j}\right)}{\partial A_{i,j}}$$

$$B_{i,j}(n+1) = B_{i,j}(n) - l_r \frac{\partial Loss^{(n)}\left(A_{i,j}, B_{i,j}\right)}{\partial B_{i,j}}$$

$l_r \coloneqq learning\ rate$

Testing

Test 1

Description:

- U is a 1000 x 3 random matrix containing numbers from 0 to 10, so there are 1000 random sources (rows), each one with dimension 3 (columns).
- I create 2 Layers; the first one has 3 input and 4 output, the second one has 4 input and 3 output.
- The training was done with 20000 iterations and $l_r = $ 1e-5
- The measurements from the system are $(y_{nn})_{ij} = e^{u_{ij}}$
- I consider first Y_0 and then Y_1 as the output for each Layer in the NN

Results:

See Test1aLog.txt (output Y_0) and Test1bLog.txt (output Y_1)

Test 2

Description:

- U is a 1000 x 3 random matrix containing numbers from 0 to 10, so there are 1000 random sources (rows), each one with dimension 3 (columns).
- I create 2 Layers; the first one has 3 input and 4 output, the second one has 4 input and 3 output.
- The training was done with 20000 iterations and $l_r = $ 1e-5
- The measurements from the system are $(y_{nn})_{ij} = \sin(u_{ij})$
- I consider first Y_0 and then Y_1 as the output for each Layer in the NN

Results:

See Test2aLog.txt (output Y_0) and Test2bLog.txt (output Y_1)

Test 3

Description:

- U is a 1000 x 3 random matrix containing numbers from 0 to 10, so there are 1000 random sources (rows), each one with dimension 3 (columns).
- I create 2 Layers; the first one has 3 input and 4 output, the second one has 4 input and 3 output.
- The training was done with 20000 iterations and $l_r = $ 1e-5
- The measurements from the system are $(y_{nn})_{ij} = \sum_{k=0}^{j}(u_{ik})$
- I consider first Y_0 and then Y_1 as the output for each Layer in the NN

Results:

See Test3aLog.txt (output Y_0) and Test3bLog.txt (output Y_1)

Test 4

Description:

- U is a 100 x 1 random matrix containing numbers from 0 to 10, so there are 100 random sources (rows), each one with dimension 1 (column).
- I create 2 Layers; the first one has 1 input and 4 output, the second one has 4 input and 1 output.
- The training was done with 20000 iterations and $l_r = $ 1e-5
- The measurements from the system are $(y_{nn})_i = \mathrm{e}^{u_i}$
- I consider first Y_0 and then Y_1 as the output for each Layer in the NN

Results:

See Test4aLog.txt (output Y_0) and Test4aLog.png (Y_0 Vs Target output)

See Test4bLog.txt (output Y_1) and Test4bLog.png (Y_1 Vs Target output)

Test 5

Description:

- U is a 100 x 1 random matrix containing numbers from 0 to 10, so there are 100 random sources (rows), each one with dimension 1 (column).
- I create 2 Layers; the first one has 1 input and 4 output, the second one has 4 input and 1 output.
- The training was done with 300000 iterations and l_r $=$ 1e-5
- The measurements from the system are $(y_{nn})_i = \mathrm{e}^{u_i}$
- I consider first Y_0 and then Y_1 as the output for each Layer in the NN

Results:

See Test5aLog.txt (output Y_0) and Test5aLog.png (Y_0 Vs Target output)
See Test5bLog.txt (output Y_1) and Test5bLog.png (Y_1 Vs Target output)

Test 6

Description:

- U is a 100 x 1 random matrix containing numbers from 0 to 100, so there are 100 random sources (rows), each one with dimension 1 (column).
- I create 2 Layers; the first one has 1 input and 4 output, the second one has 4 input and 1 output.
- The training was done with 20000 iterations and l_r = 1e-5
- The measurements from the system are $(y_{nn})_i = 5u_i$
- I consider first Y_0 and then Y_1 as the output for each Layer in the NN

Results:

See Test6aLog.txt (output Y_0) and Test6aLog.png (Y_0 Vs Target output)
See Test6bLog.txt (output Y_1) and Test6bLog.png (Y_1 Vs Target output)

Test 7

Description:

- U is a 100 x 1 random matrix containing numbers from 0 to 100, so there are 100 random sources (rows), each one with dimension 1 (column).
- I create 2 Layers; the first one has 1 input and 4 output, the second one has 4 input and 1 output.
- The training was done with 300000 iterations and l_r = 1e-5
- The measurements from the system are $(y_{nn})_i = 5u_i$
- I consider first Y_0 and then Y_1 as the output for each Layer in the NN

Results:

See Test7aLog.txt (output Y_0) and Test7aLog.png (Y_0 Vs Target output)
See Test7bLog.txt (output Y_1) and Test7bLog.png (Y_1 Vs Target output)

THANKS!