REZOLVAREA SISTEMELOR DE ECUAȚII LINIARE FOLOSIND TEHNICI DIN INTELIGENȚA ARTIFICIALĂ

STUDENT: ADRIAN PAL

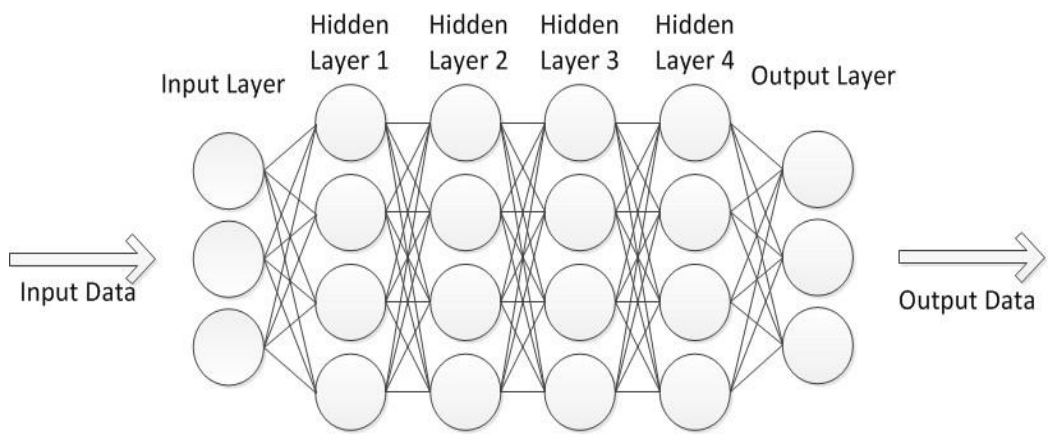
PROF. COORDONATOR: LECT. DR. MAFTEIU-SCAI LIVIU OCTAVIAN

Sisteme de Ecuații – Rețele Neuronale

$$\begin{cases} 7x + 3y + 2z = 1 \\ 3x + y + 5z = 15 \\ 12x + 2y + 8z = 3 \end{cases} \longrightarrow \left[\begin{array}{ccc|c} 7 & 3 & 2 & 1 \\ 3 & 1 & 5 & 15 \\ 12 & 2 & 8 & 3 \end{array}\right]$$

Sistem pătratic de ecuatii liniare. [3x3]

Figura 1



Rețea neuronală de tip feed forward

Figura 2

- Rezolvarea sistemelor de ecuații liniare pătratice, folosind tehnici de inteligența artificială
- Aflarea necunoscutelor unui sistem de ecuații liniare. (*Ex dat: X,Y,Z)* -> folosind rețele neuronale

Lucrări Asemănătoare

- Algoritmi genetici
- Particle Swarm Optimization
- Glowworm swarm optimization (GSO)
- Algoritmi mematici (AMs)

- Rețele neuronale recurente pentru rezolvarea ecuației Sylvester cu coeficienții unei matrici care variază în timp se pot găsi în lucrarea “A recurrent neural network for solving sylvester equation with time-varying coefficients”-Yunong Zhang, Danchi Jiang, Jun Wang

- Rețele neuronale recurente pentru rezolvarea ecuațiilor algebrice simultane se pot găsi în lucrarea: “Solving simultaneous linear equations using recurrent neural networks” - Jun Wang, Hua Li

Contribuții Proprii

- Implementare rezolvare sistemelor de ecuații liniare pătratice folosind rețele neuronle cu arhitectura de tip feedforward în Python.
- O comparație între două abordări diferite de a stoca sistemele în setul de date ⇒
  - Toate sistemele sunt în același set de date, indiferent de dimensiune
  - Vor exista mai multe seturi de date pentru fiecare dimensiune de sistem

Sisteme de Ecuații Liniare

- Sisteme liniare pătratice, deterministe, cu soluție unică ⇒ determinant matrice sist. ≠ 0
- Dimensiuni testate [2x2 – 100x100]
- Coeficienții și soluțiile sistemelor - de tip float cu 18 decimale
- Algoritmul este bazat pe învațare supravegheată
- Pentru rezolvarea sistemelor ⇒ Cramer, Gauss, Gauss-Jordan ETC sau metode iterative precum metoda Jacobi, Gauss-Seidel ETC

Setul de Date

- Setul de date: coeficienții matricei sistemului, coeficienții matricei libere ai sistemului și necunoscutele acestuia.

- Un algoritm este antrenat pe date de intrare care au fost etichetate pentru o anumită ieșire.

Setul de date în format bytecode

	system	...	output
0	b'\xdc\xa8\x83}|\x99\xc3?\xe0\xbfq\xe6\xd4\xef...	...	b'\xec,x3\x12<\xdb?\xecA\xf89nE\xc6?'
1	b'6\x87\xfb\x1a2\xb8\xe0?!\xf7\x97~\x05K\xed?x...	...	b'\xa4\x12p\xcf\xdd^\xd1?\xc5\xf9\x82\xec\xb12...
2	b'NL\xbd\xae\x1a\x7f\xe0?D9\x1e\x90}\xe2\xeb?1...	...	b'\xc8:\x87\x10\xfa\x91\xd8?\xda\xc71s\xe4T\xda?'
3	b'\xaa\xdd9(\x9a\x8e\xd8?4 \xe2\x96\xd2Y\xe5?\...	...	b'\xd9 z\x83>\xed\xc4?F\x0e\x1ezy\xc6\xe8?'
4	b'M\xfc\xb9\x82I<\xe6?_OU8-\xfa\xdc?\xe8\x08\x...	...	b'\xfa\xf6(5[A\xc6?\xea\x8f\x9a>\x8aA\xe3?'

Setul de date după transformare în liste

```
[array([0.15312153, 0.02337582, 0.42959237, 0.36818639]), array([0.06922681, 0.24687153]), array([0.42554145, 0.17399385])]
[array([0.52248483, 0.91540789, 0.61348552, 0.01039379]), array([0.21045886, 0.16728874]), array([0.27141519, 0.07499229])]
[array([0.51551565, 0.87139776, 0.96620892, 0.12468585]), array([0.55643189, 0.42223667]), array([0.38390972, 0.41143142])]
[array([0.38370375, 0.66721467, 0.68746006, 0.22297942]), array([0.57930795, 0.2850298 ]), array([0.16349012, 0.77422785])]
[array([0.69485927, 0.45276957, 0.12190631, 0.86471741]), array([0.39326916, 0.54153992]), array([0.17386952, 0.60175049])]
```

Generarea sistemelor: Se vor stoca in bytecode, dupa care se vor transforma in liste.

Reprezentarea Datelor

- Pentru metoda în care se vor stoca sisteme de diferite dimensiuni, se va folosi conceptul de ragged tensors pentru a aduce sistemele la aceeași dimensiune.
- Setul de date este împărțit în două părți:
  - 80% dedicat antrenării modelului.
  - 20% dedicat testării.

```
[array([0.15312153, 0.02337582, 0.42959237, 0.36818639]), array([0.06922681, 0.24687153]), array([0.42554145, 0.17399385])]
[array([0.52248483, 0.91540789, 0.61348552, 0.01039379]), array([0.21045886, 0.16728874]), array([0.27141519, 0.07499229])]
[array([0.51551565, 0.87139776, 0.96620892, 0.12468585]), array([0.55643189, 0.42223667]), array([0.38390972, 0.41143142])]
[array([0.38370375, 0.66721467, 0.68746006, 0.22297942]), array([0.57930795, 0.2850298 ]), array([0.16349012, 0.77422785])]
[array([0.69485927, 0.45276957, 0.12190631, 0.86471741]), array([0.39326916, 0.54153992]), array([0.17386952, 0.60175049])]
```

Primul rând din setul de date este scris astfel:

$$\begin{cases} 0.15312153 \cdot X + 0.02337582 \cdot Y = 0.06922681 \\ 0.42959237 \cdot X + 0.36818639 \cdot Y = 0.24687153 \end{cases}$$

$$Unde: X = 0.42554145, Y = 0.17399385.$$

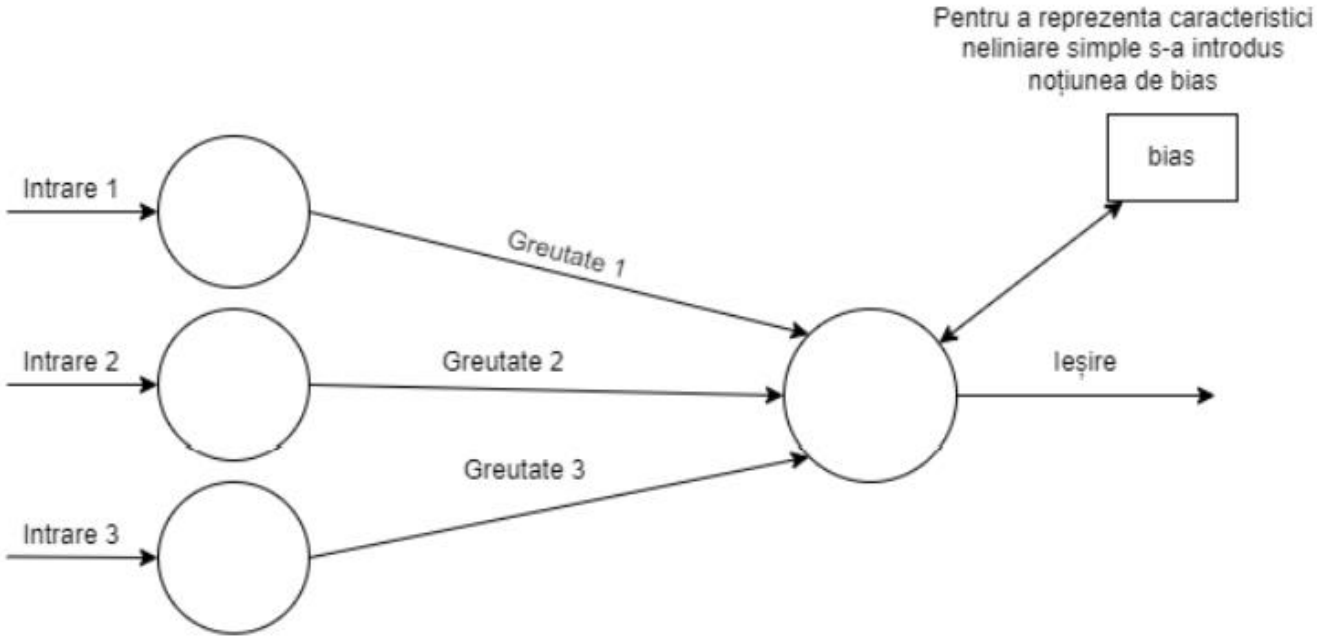
- Coloana I si Coloana II → variabile independente
- Coloana III → variabile dependente.

Perceptronul (1950 – Frank Rosenblatt)

- Un singur strat de noduri de ieșire; intrările conduc direct la ieșiri printr-un număr de greutăți sau, altfel numite, ponderi.

- Pentru fiecare nod se calculează suma produselor ponderilor (weights) și intrărilor (inputs) la care se mai adaugă un bias.

- Ieșirea trebuie sa treaca de o valoare prag pentru a intra în urmatorul neuron.



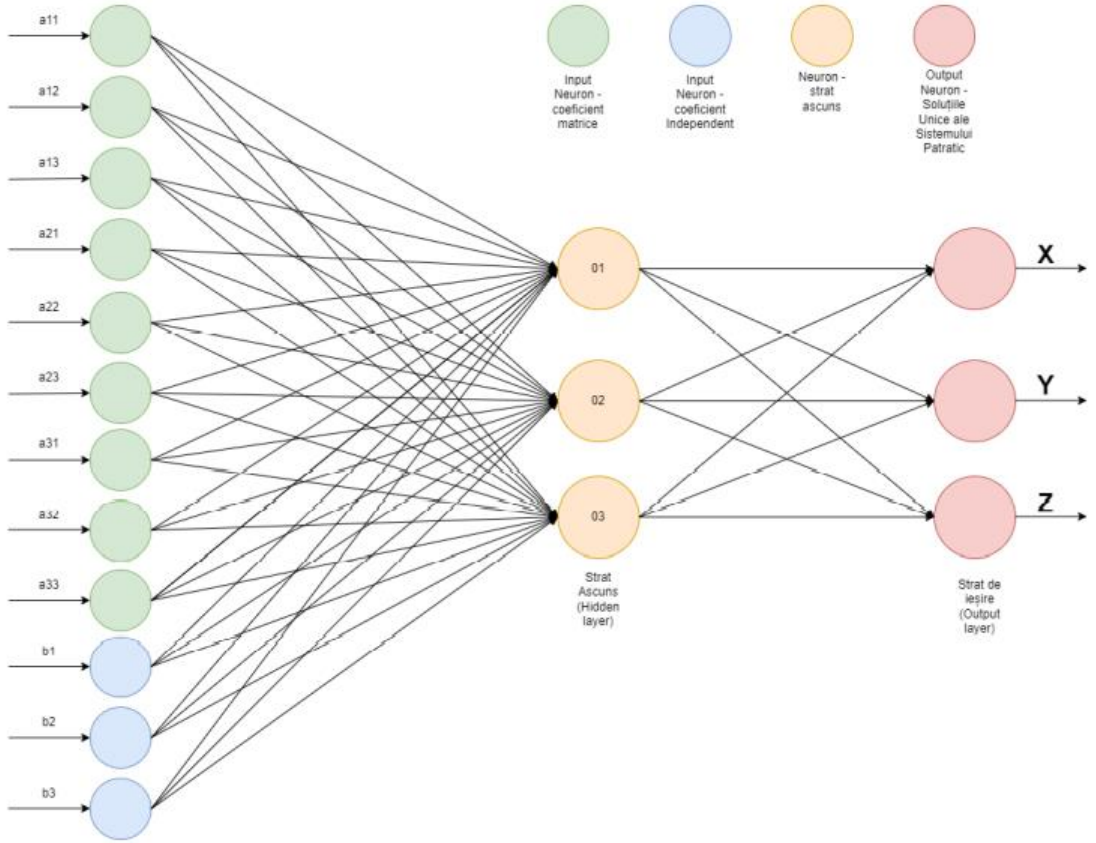
Ieșire = Intrare1 * Greutate 1 + Intrare 2 * Greutare 2 + Intrare 3 * Greutate 3 + bias

Rețeaua Neuronală

- O rețea neuronală este o serie de algoritmi care încearcă să recunoască relațiile care stau la baza unui set de date.
- Datele se iau în loturi (batch-uri) și se calculează relația dintre valorile dependente si cele independente.

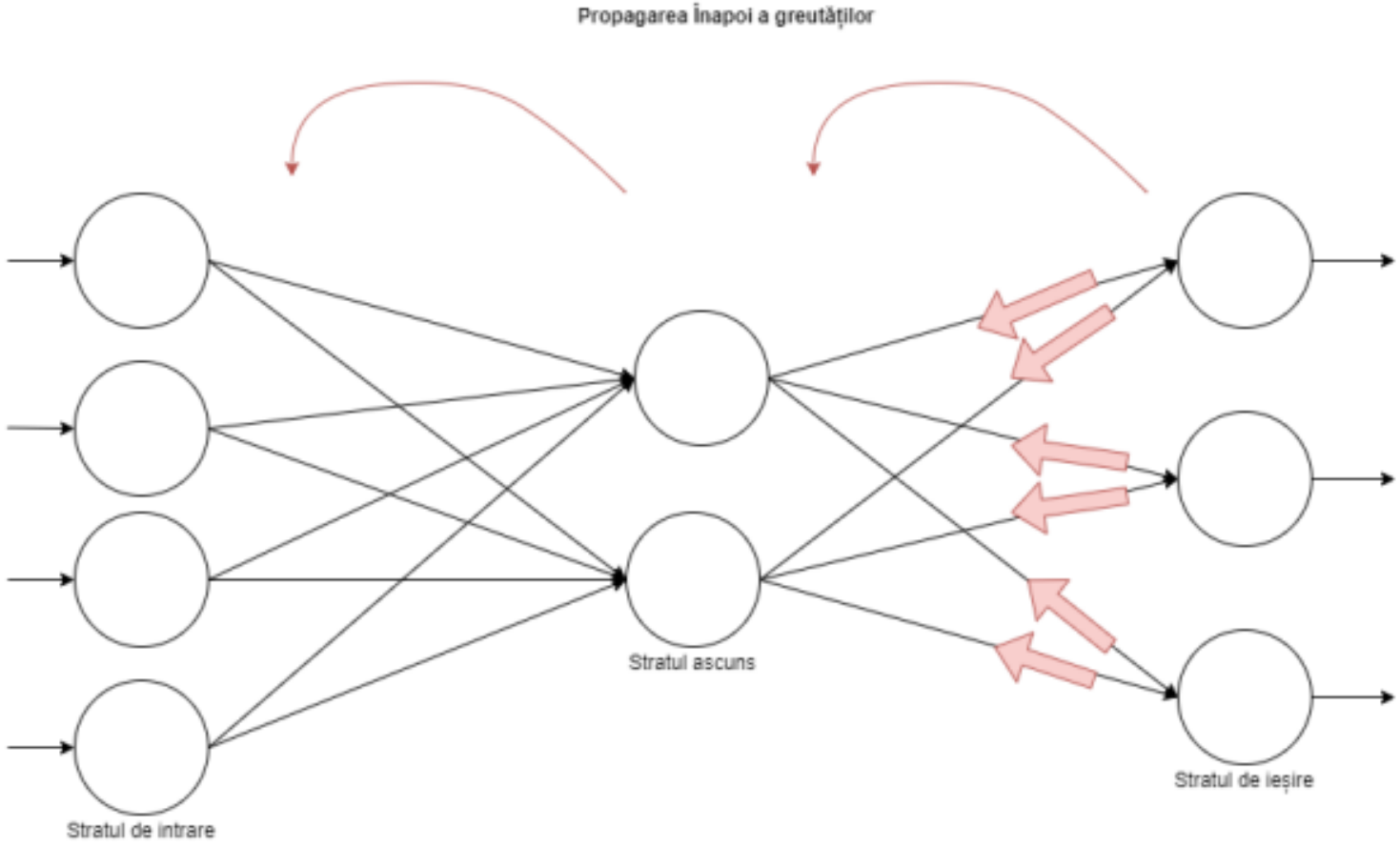
Fie urmatorul sistem patratic, deterministic, cu solutie unica:

$$\begin{cases} 3X + 5Y + Z = 23 \\ 7X - 2Y + 4Z = 8 \\ -6X + 3Y + 2Z = 15 \end{cases}$$



Procesul de Învațare al Rețelei Neuronale

- Algoritmul rețelei neuronale este conceput să itereze prin setul de date de mai multe ori (iterațiile sunt numite epochi).
- Este metoda prin care greutățile rețelei neuronale se reglează pe baza ratei de eroare obținute în epoca precedentă.



Rețea Neuronală

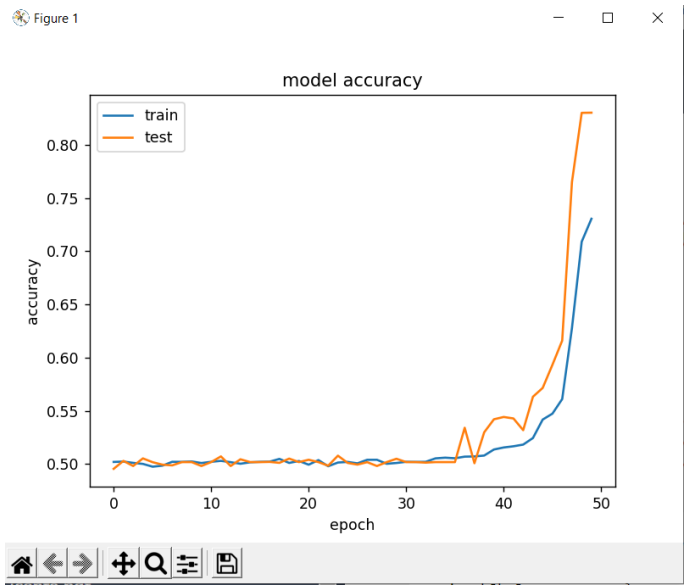
Implementată în Python cu ajutorul librăriei Keras.

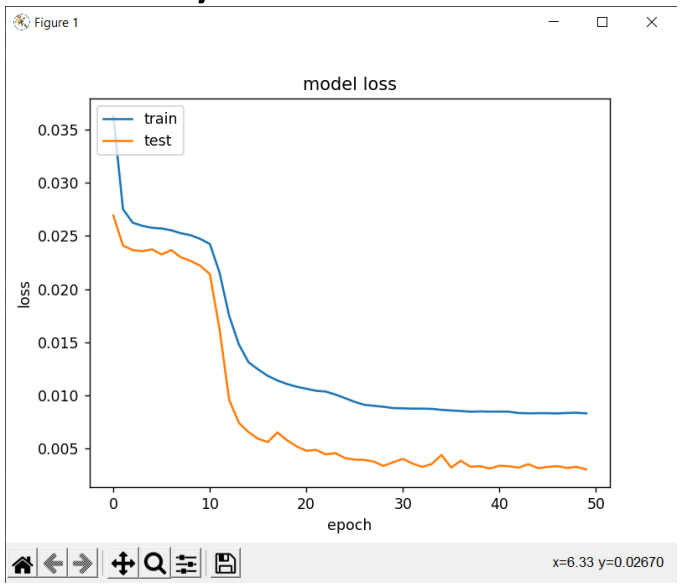
```
# Model folosit din Libraia Keras, indica faptul ca se va folosi o retea neuronala.
model = Sequential()

# Setul de date este normalizat, coeficientii fiind in intervalul [0,1].
# Se initializeaza greutatile cu valori intre 0.1 si 0.9.
initializer = tf.keras.initializers.RandomUniform(minval=0.1, maxval=0.9)
# Se aplica conceptul de Flattening discutat si se introduce forma coeficientilor.
# In cazul de fata, pentru sisteme de rangul 2 vor fi 6 coeficienti.
# Dupa reformatare doua randuri si trei valori pe fiecare rand.
model.add(Flatten(input_shape=(2, 3)))

# Se adauga un "Dense" de 32, adica un strat ascuns cu 32 de neuroni
# Functia de activare: Rectified Linear Unit si setarea unor hiper-paramaterii.
# Pentru a nu face "overfitting" se regularieaza ponderile cu valori intre 10^(-7) si 10^(-4)
model.add(Dense(32, activation='relu', kernel_initializer=initializer,
                kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
                bias_regularizer=regularizers.L2(1e-7),
                activity_regularizer=regularizers.L2(1e-7), use_bias=True))
# Se anuleaza 10% din neuroni din strat pentru a nu face "overfit"
model.add(Dropout(0.1))
# Se adauga inca un strat ascuns cu 16 de neuroni
model.add(Dense(16, activation='relu'))
# Se anuleaza 10% din neuroni din strat pentru a nu face "overfit"
model.add(Dropout(0.1))
```

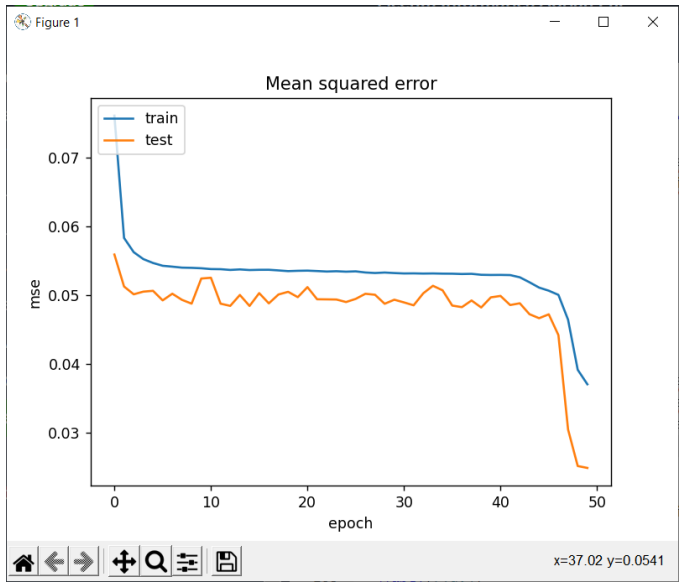
Implementarea este pentru sisteme în dimensiunea 2.

```
25      # Se adauga inca un strat ascuns cu 8 de neuroni
26      model.add(Dense(8, activation='relu'))
27      # Se anuleaza 20% din neuroni din strat pentru a nu face "overfit"
28      model.add(Dropout(0.2))
29      # Ultimul strat este de iesire si va avea 2 neuroni, fiind cele doua necunoscute
30      model.add(Dense(2, activation='relu'))
31
32      # In cazul de fata se foloseste optimizatorul Adam inloc de Stocastic gradient descent
33      adam = tf.keras.optimizers.Adam(learning_rate=0.01, beta_1=0.1,
34                                      beta_2=0.9, epsilon=1e-09, amsgrad=True,
35                                      name="Adam")
36      # Functia de pierdere msle, setarea optimizatorului si metricile.
37      model.compile(loss='mean_squared_logarithmic_error',
38                    optimizer=adam, metrics=["accuracy", "mse", "mae"])
39      # Inceperea procesului de antranare a retelei, 100 de iteratii, lotul fiind de 128.
40      # Impartirea setului de date este de 8/10 antrenare si 2/10 testare.
41      history = model.fit(X_train, Y_train, epochs=100,
42                          batch_size=128, verbose='auto', validation_split=0.2,
43                          shuffle=True)
```

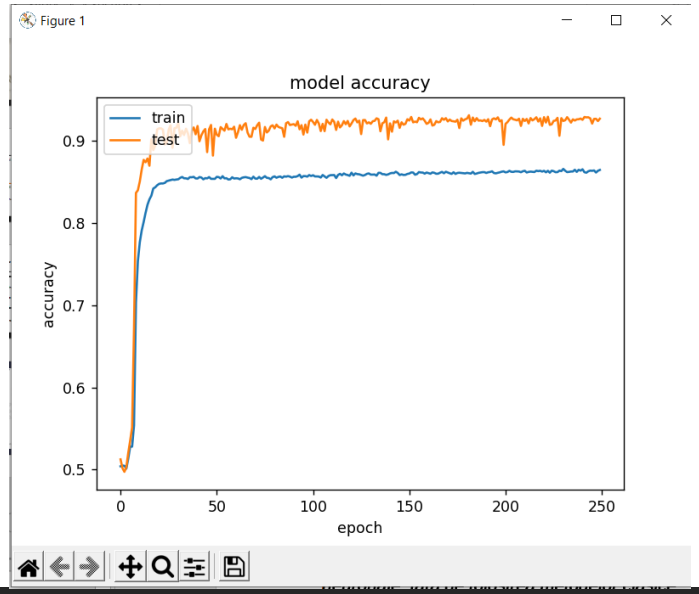
50 De Iterații Prin Setul De Date

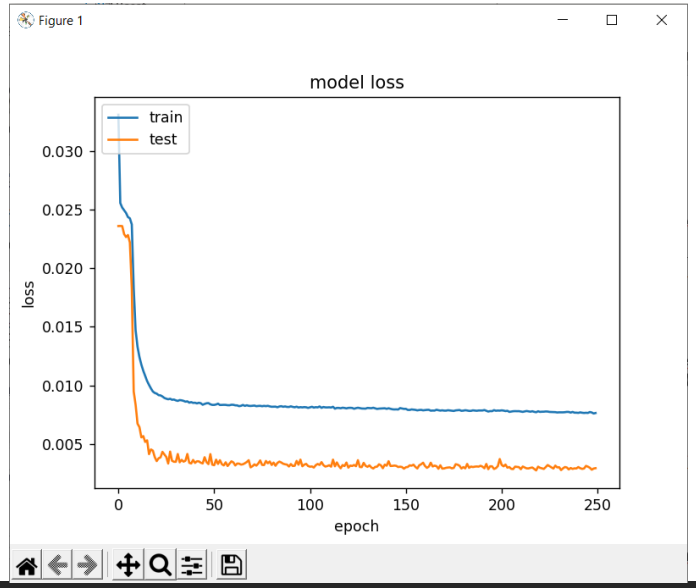


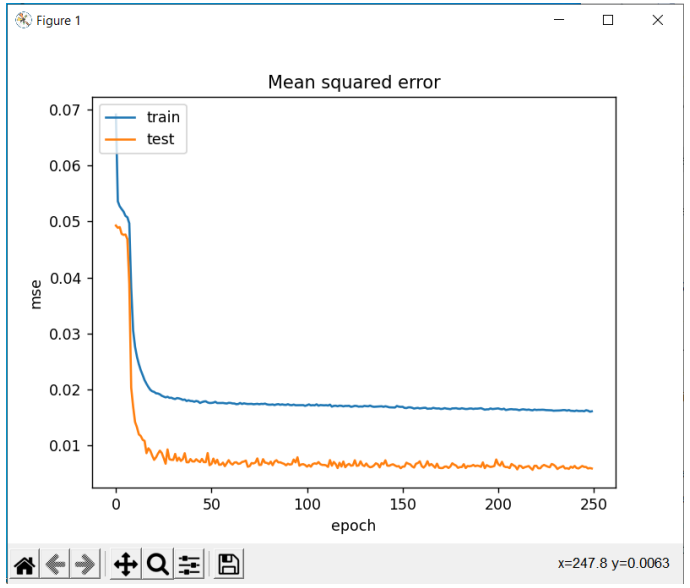




250 De Iterații Prin Setul De Date







Rularea Programului

```
Epoch 19/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0113 - accuracy: 0.8151 - mse: 0.0235 - mae: 0.1158 - val_loss: 0.0054 - val_accuracy: 0.8894 - val_mse: 0.0114
  - val_mae: 0.0775
Epoch 20/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0109 - accuracy: 0.8220 - mse: 0.0227 - mae: 0.1135 - val_loss: 0.0054 - val_accuracy: 0.8957 - val_mse: 0.0112
  - val_mae: 0.0766
Epoch 21/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0107 - accuracy: 0.8254 - mse: 0.0223 - mae: 0.1122 - val_loss: 0.0055 - val_accuracy: 0.9008 - val_mse: 0.0109
  - val_mae: 0.0770
Epoch 22/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0105 - accuracy: 0.8282 - mse: 0.0219 - mae: 0.1112 - val_loss: 0.0053 - val_accuracy: 0.8836 - val_mse: 0.0109
  - val_mae: 0.0762
Epoch 23/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0104 - accuracy: 0.8281 - mse: 0.0217 - mae: 0.1105 - val_loss: 0.0054 - val_accuracy: 0.8876 - val_mse: 0.0112
  - val_mae: 0.0784
Epoch 24/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0102 - accuracy: 0.8335 - mse: 0.0214 - mae: 0.1095 - val_loss: 0.0047 - val_accuracy: 0.9080 - val_mse: 0.0101
  - val_mae: 0.0725
Epoch 25/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0101 - accuracy: 0.8346 - mse: 0.0212 - mae: 0.1089 - val_loss: 0.0044 - val_accuracy: 0.9083 - val_mse: 0.0097
  - val_mae: 0.0707
Epoch 26/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0100 - accuracy: 0.8345 - mse: 0.0210 - mae: 0.1085 - val_loss: 0.0045 - val_accuracy: 0.9113 - val_mse: 0.0099
  - val_mae: 0.0711
Epoch 27/100
638/638 [==============================] - 6s 10ms/step - loss: 0.0099 - accuracy: 0.8362 - mse: 0.0208 - mae: 0.1078 - val_loss: 0.0046 - val_accuracy: 0.8937 - val_mse: 0.0096
  - val_mae: 0.0715
```

Avantajele Utilizării Rețelelor Neuronale

- Complexitatea timpului pentru antrenarea unei rețele neuronale care are 4 straturi cu i, j, k, s, respectiv i noduri, cu t exemple de antrenament si n epoci.

$$O(n \cdot t(i \cdot j + j \cdot + k \cdot l) = O(n) \rightarrow$$ Timp de execuție liniar

- Cramer : $O(n! \cdot n)$, unde n → Dimensiune matricei
- Eliminarea Gauss-Jordan: $O(n^4 \log_2 ||A||)$, unde n → Dimensiune matrice, ||A|| → Determinant matricei.
- Metodele iterative: Jacobi, Gauss Seidel - $O(n)$ (există cazuri în care nu converg).

Pentru sisteme de rang 2, sunt afișate soluțiile oferite de rețeaua neuronală pe diferite epochi, adică iterații. Cu cât crește numărul de iterații, cu atat soluțiile prezise sunt mai aproape de soluțiile reale.

a1	*b1*	*CI.I*	*a2*	*b2*	*CI.II*	*X*	*Y*
0.09164798	0.5580665	0.10545199	0.8613886	0.2382838	0.6613243	**0.7495206**	**0.0658702**
0.14868072	0.8305696	0.6490381	0.81800604	0.01250122	0.7370453	**0.8915233**	**0.62184525**
0.99819815	0.28540322	0.46820056	0.864885	0.7295135	0.7923411	**0.23978394**	**0.80184335**
0.9839497	0.8719645	0.8122804	0.8366895	0.02552434	0.05261341	**0.03569331**	**0.89127475**
0.8972369	0.06865425	0.534667	0.89386237	0.84690475	0.958732	**0.5540261**	**0.54729754**
0.14637789	0.9570755	0.76131064	0.51966166	0.17714322	0.278714	**0.279767**	**0.7526668**

Tabela 5.1: *Soluțiile reale pentru 6 sisteme.*

a1	*b1*	*CI.I*	*a2*	*b2*	*CI.II*	*X*	*Y*
0.09164798	0.5580665	0.10545199	0.8613886	0.2382838	0.6613243	**0.701601**	**0.09142095**
0.14868072	0.8305696	0.6490381	0.81800604	0.01250122	0.7370453	**0.80360806**	**0.56892455**
0.99819815	0.28540322	0.46820056	0.864885	0.7295135	0.7923411	**0.26409557**	**0.7420951**
0.9839497	0.8719645	0.8122804	0.8366895	0.02552434	0.05261341	**0.03093034**	**0.8059547**
0.8972369	0.06865425	0.534667	0.89386237	0.84690475	0.958732	**0.5059705**	**0.50059485**
0.14637789	0.9570755	0.76131064	0.51966166	0.17714322	0.278714	**0.29913455**	**0.66831315**

Tabela 5.2: *Soluțiile prezise de rețea pentru 6 sisteme - 20 de iterații*

a1	*b1*	*CI.I*	*a2*	*b2*	*CI.II*	*X*	*Y*
0.09164798	0.5580665	0.10545199	0.8613886	0.2382838	0.6613243	**0.74965434**	**0.0652753**
0.14868072	0.8305696	0.6490381	0.81800604	0.01250122	0.7370453	**0.8911243**	**0.62134653**
0.99819815	0.28540322	0.46820056	0.864885	0.7295135	0.7923411	**0.23952546**	**0.80121432**
0.9839497	0.8719645	0.8122804	0.8366895	0.02552434	0.05261341	**0.03568743**	**0.89122156**
0.8972369	0.06865425	0.534667	0.89386237	0.84690475	0.958732	**0.5541543**	**0.54728543**
0.14637789	0.9570755	0.76131064	0.51966166	0.17714322	0.278714	**0.279684**	**0.7528954**

Tabela 5.3: *Soluțiile prezise de rețea pentru 6 sisteme - 100 de iteratii*

Se notează sistemele sub următoarea formă:

$$\begin{cases} a1 \cdot X + b1 \cdot Y = CI.I \\ a2 \cdot X + b2 \cdot Y = CI.II \end{cases} \tag{5.2}$$

Concluzii și Direcții Viitoare

- Rețelele neuronale artificiale sunt eficiente chiar și în rezolvarea sistemelor de ecuații.
- Folosirea metodei care stocheaza doar sisteme de același rang in setul de date este incurajată.
- Modelul neuronal se antrenează pe un sistem de calcul performant.
- Folosirea rețelelor neuronale recurente sau convoluționale. (*layers.LSTM / layers.Conv2D, layers.MaxPooling2D*).
- Realizarea unei aplicații interactive care folosește modelul neuronal pentru a prezice soluțiile unor sisteme.