PSO-NeuralNet Document

Nafis Neehal

6 August 2018

1   Network Description

Artificial Neural Network: Dataset- MNIST, Training Data - 42,000, Test Data-
10,000

Layer	Number of Nodes	Layer Dim	Weight Dim	Bias Dim	Activation
Input	784	784x1	None	None	None
Hidden 1	16	16x1	16x784	16x1	ReLU
Hidden 2	16	16x1	16x16	16x1	ReLU
Output	10	10x1	10x16	10x1	Sigmoid

Table 1: Artificial Neural Network Structure

2   Forward Propagation

Algorithm 1 Forward Propagation in Neural Network

1: **procedure** FORWARDPROPAGATION(X)   ▷ Forward Propagation
2:        $Z^{[1]} = W^{[1]}.X + b^{[1]}$
3:        $A^{[1]} = \text{ReLU}(Z^{[1]})$
4:        $Z^{[2]} = W^{[2]}.A^{[1]} + b^{[2]}$
5:        $A^{[2]} = \text{ReLU}(Z^{[2]})$
6:        $Z^{[3]} = W^{[3]}.A^{[2]} + b^{[3]}$
7:        $A^{[3]} = \text{Sigmoid}(Z^{[3]})$
8: **end procedure**

The notations that have been used in forward propagation is described below:

- $Z^{[i]}$ = Output of ith Layer of Neural Network

- $A^{[i]}$ = Activation of ith Layer of Neural Network

- X = input

- $W^{[i]}$ = Weight from $(i-1)^{th}$ to i^{th} Layer of Neural Network

- $b^{[i]}$ = Biases for i^{th} Layer of Neural Network

3   Back Propagation

Algorithm 2 Back Propagation in Neural Network

1: **procedure** BACKPROPAGATION(X, Y)
2:     $dZ^{[3]} = A^{[3]}$ - Y   ▷ Derivatives Calculation
3:     $dW^{[3]} = (1/m)\ dZ^{[3]}.A^{[2].T}$
4:     $dB^{[3]} = (1/m)\ np.sum(dZ^{[3]}, axis=1, Keepdims = True)$
5:     $dZ^{[2]} = W^{[3].T}.dZ^{[3]} * DReLU(Z^{[2]})$
6:     $dW^{[2]} = (1/m)\ dZ^{[2]}.A^{[1].T}$
7:     $dB^{[2]} = (1/m)\ np.sum(dZ^{[2]}, axis=1, Keepdims = True)$
8:     $dZ^{[1]} = W^{[2].T}.dZ^{[2]} * DReLU(Z^{[1]})$
9:     $dW^{[1]} = (1/m)\ dZ^{[1]}.X^{T}$
10:     $dB^{[1]} = (1/m)\ np.sum(dZ^{[1]}, axis=1, Keepdims = True)$
11:
12:     $W^{[1]} = W^{[1]}$ - 0.01 * $dW^{[1]}$ ▷ Parameter Update
13:     $b^{[1]} = b^{[1]}$ - 0.001 * $dB^{[1]}$
14:     $W^{[2]} = W^{[1]}$ - 0.01 * $dW^{[2]}$
15:     $b^{[2]} = b^{[1]}$ - 0.001 * $dB^{[2]}$
16:     $W^{[3]} = W^{[1]}$ - 0.01 * $dW^{[3]}$
17:     $b^{[3]} = b^{[1]}$ - 0.001 * $dB^{[3]}$
18: **end procedure**

The notations that have been used in back propagation is described below:

- $dZ^{[i]}$ = Derivative of Z of ith Layer of Neural Network

- $A^{[i].T}$ = Transpose of $A^{[i]}$

- $dW^{[i]}$ = Derivative of W of ith Layer of Neural Network

- $dB^{[i]}$ = Derivative of B of ith Layer of Neural Network

4   Loss Function for Neural Network
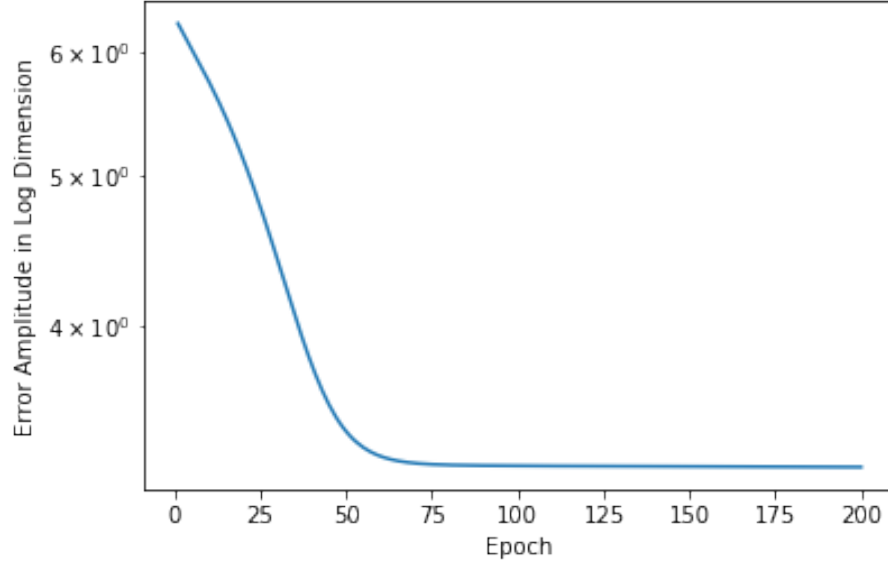

Algorithm 3 Loss Function in Neural Network

1: **procedure** LOSSFUNCTION(Y)
2:     RETURN $\frac{1}{m}\sum_{i=1}^{m}(-Y^{(i)}\log(\hat{Y^{(i)}}) - (1 - Y^{(i)})\log(1 - \hat{Y^{(i)}}))$
3: **end procedure**
---

5 Performance of Back Propagation

6 Particle Swarm Optimization as a replacement of Backpropagation in Neural Network

Main Equation of Particle Swarm Optimization -

$WV_{ij}(t)$ = Intertia Component
$r_1c_1(P_{ij}(t) - x_{ij}(t))$ = Cognitive Component
$r_2c_2(g_j(t) - x_{ij}(t))$ = Social Component
c_1, c_2 = Acceleration Coefficients
W = Inertia Coefficient
t, t(t+1) = timestamps
$V_{ij}(t)$ = j^{th} velocity component of ith particle
$P_{ij}(t)$ = j^{th} component of ith particle's best position
$x_{ij}(t)$ = j^{th} component of ith particle's current position
r_1, r_2 = $N(\mu, \sigma^2)$

For determining the value of W, c_1 and c_2, I've used Constriction coefficients proposed by Clerc and Kennedy, 2002.

Let,
$$\phi = \phi_1 + \phi_2 \geq 4$$
$$0 \leq \kappa \leq 1$$

3

$$\chi = \frac{2\kappa}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}$$

Commonly, $\kappa = 1$, $\phi_1 = 2.05$, $\phi_2 = 2.05$
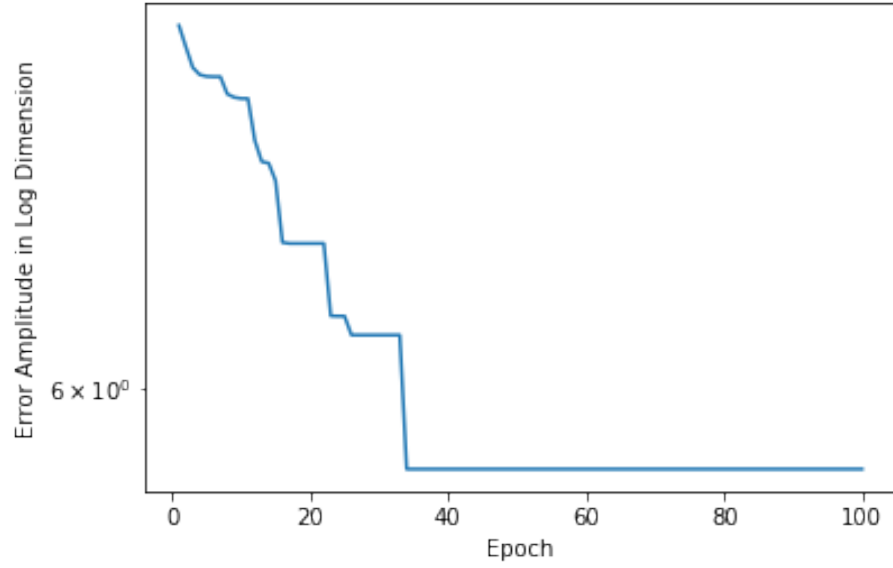
So, Finally,
W $= \chi$, $C_1 = \kappa\phi_1$, $C_2 = \kappa\phi_2$

And for Velocity and Position Update,

$$V_{ij}(t+1) = WV_{ij}(t) + r_1 c_1 (P_{ij}(t) - x_{ij}(t)) + r_2 c_2 (g_j(t) - x_{ij}(t))$$

$$x_{ij}(t+1) = x_{ij}(t) + V_{ij}(t+1)$$

Performance of PSO is shown in the following graph. Swarm Population Size $= 50$, Total Epochs $= 100$.

4

Algorithm 4 Particle Swarm Optimization in Neural Network

1: **procedure** INITIALIZEPARTICLE(nVar, nPop)
2:      globalBestPosition $\leftarrow \vec{0}$ of dimension (nVar x 1)
3:      globalBestCost $\leftarrow$ None
4:      **for** each particle$_{(i)}$ in range(1, nPop) **do**
5:          particle$_{(i)}$.position $\leftarrow N(\mu, \sigma^2)$ of dimension (nVar x 1)
6:          particle$_{(i)}$.velocity $\leftarrow \vec{0}$ of dimension (nVar x 1)
7:          particle$_{(i)}$.cost $\leftarrow$ None
8:          particle$_{(i)}$.bestPosition $\leftarrow$ particle$_{(i)}$.position
9:          particle$_{(i)}$.bestCost $\leftarrow$ particle$_{(i)}$.cost
10:     **end for**
11: **end procedure**
12: **procedure** INITIALIZEPARAMETERS($NeuralNetwork, X, Y, maxIt, nPop$)
13:     nVar $\leftarrow$ NeuralNetwork.numWeights + NeuralNetwork.numBiases
14:     $\kappa \leftarrow 1$
15:     $\phi_1 \leftarrow 2.05$
16:     $\phi_2 \leftarrow 2.05$
17:     $\phi \leftarrow \phi_1 + \phi_2$
18:     $\chi \leftarrow \dfrac{2\kappa}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}$
19:     maxIt $\leftarrow 100$
20:     nPop $\leftarrow 50$
21:     w $\leftarrow \chi$
22:     wdamp $\leftarrow 0.99$
23:     c$_1 \leftarrow \chi.\phi_1$
24:     c$_2 \leftarrow \chi.\phi_2$
25: **end procedure**
26: **procedure** COSTPSO($NeuralNetwork, particle$)
27:     NeuralNetwork.weights *gets* particle.position.slice
28:     NeuralNetwork.biases *gets* particle.position.slice
29:     NeuralNetwork.ForwardPropagation(X)
30:     RETURN NeuralNetwork.Loss(Y)
31: **end procedure**
32: **procedure** INITIALIZESWARM($NeuralNetwork$)
33:     **for** each particle$_{(i)}$ in range(1, nPop) **do**
34:         particle$_{(i)}$.cost = COSTPSO(NeuralNetwork, particle$_{(i)}$)
35:         **if** particle$_{(i)}$.bestCost $< globalBestCost$ **then**
36:             globalBestPosition = particle$_{(i)}$.bestPosition
37:             globalBestCost = particle$_{(i)}$.bestCost
38:         **end if**
39:     **end for**
40: **end procedure**

Algorithm 5 Particle Swarm Optimization in Neural Network

1: **procedure** PSOLOOP($NeuralNetwork$)
2:     **for** each epoch$_{(i)}$ in range(1, maxIt) **do**
3:       **for** each particle$_{(j)}$ in range(1, nPop) **do**
4:         $V_j(t+1) \leftarrow WV_j(t) + r_1c_1(P_j(t) - x_j(t)) + r_2c_2(g(t) - x_j(t))$
5:         $x_j(t+1) \leftarrow x_j(t) + V_j(t+1)$
6:         particle$_{(j)}$.cost $\leftarrow$ COSTPSO(NeuralNetwork,particle$_j$)
7:         **if** particle$_{(j)}$.cost <particle$_{(j)}$.bestCost **then**
8:           particle$_{(j)}$.bestPoition $\leftarrow$ particle$_{(j)}$.position
9:           particle$_{(j)}$.besCost $\leftarrow$ particle$_{(j)}$.cost
10:           **if** particle$_{(j)}$.bestCost <globalBestCost **then**
11:             globalBestPosition $\leftarrow$ particle$_{(j)}$.bestPosition
12:             globalBestCost $\leftarrow$ particle$_{(j)}$.cost.bestCost
13:           **end if**
14:         **end if**
15:       **end for**
16:     **end for**
17: **end procedure**