Neural Network Classifier – Part 1

Dr. Kalidas Y.,

IIT Tirupati

By the end of this lecture, you will understand the principles of a neural network based classification algorithm

Brief timelines

- Biological experiments
  - Alexander Bain 1873 - Memory
  - William James 1890 – Memory and Actions
  - Sherrington 1898 – Experiments on rats
  - Donald Hebb 1940 – Neural learning
- Computational experiments
  - McCulloth and Pitts 1943 – Theoretical model
  - Farley and Clark 1954 – Experiment on electrical circuits
- Algorithms
  - Rosenblatt 1958 – Linear model
  - Minsky and Pepert 1969 – Non linear issues
  - Rumelhart and McClelland 1986 – Texts
- Deep networks
  - Rina Dechter 1986
  - Igor Aizenberg 2000
  - Geoff Hinton 2006…

84) key phrase… “Neural Network”

Binary Logistic Regression

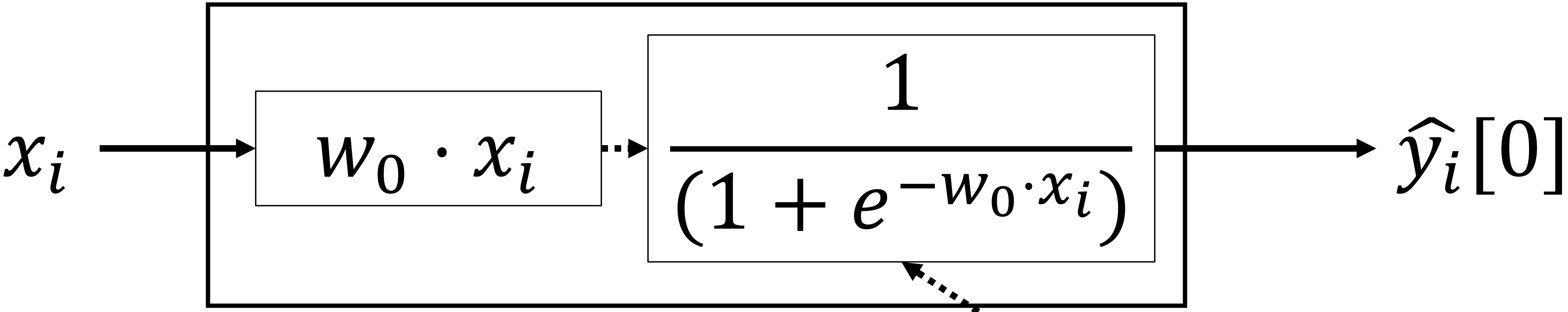
- $y_i \in \{+1, -1\}$
- $\mathrm{F}(w) = \frac{1}{1+e^{-y_i (w \cdot x_i)}}$

- In general if we can define ***Loss Function Per Point***, that would do!

Summarizing Multi Class Logistic Regression…

1. Input xi is m-dimensional data point
2. Output yi is k-dimensional data point
    1. k-class classification problem
    2. One hot encoded representation
3. Model, $f(x) = softmax(W \times x)$
    1. $W_{k \times m}$ is a kxm matrix (that needs to be learnt)
4. Data set, $D = \{(x_1, y_1), \dots (x_N, y_N)\}$
5. Loss function, $L(W) = \sum_{i=1}^{i=N} l_i$
    - l_i is the choice of sub-loss function between two arrays of numbers
    - Squared Error, $l_i = \sum_{j=1}^{j=k} (y_i[j] - \hat{y}_i[j])^2$
    - Absolute Error, $l_i = \sum_{j=1}^{j=k} |y_i[j] - \hat{y}_i[j]|$
    - Cross Entropy Loss, $l_i = -\sum_{j=1}^{j=k} y_i[j] * \log(\hat{y}_i[j])$ (Popular choice!)

Neuron *Transformation Function*

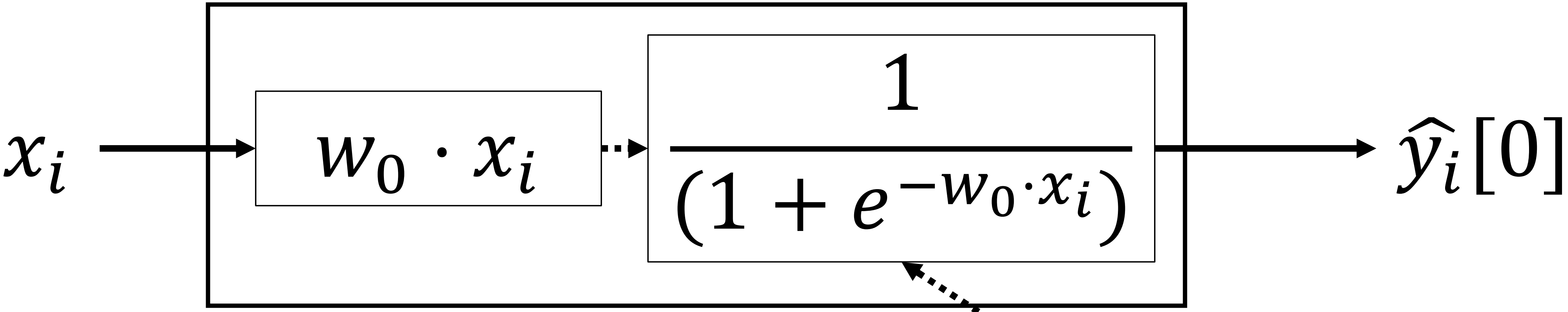


Schematic of a Neuron

Called activation function, *a(x)*

xi is m dimensional vector
w0 is a m dimensional vector

85) key phrase… “Sigmoid Function”



Schematic of a Neuron

Called activation function, *a(x)*

xi is m dimensional vector
w0 is a m dimensional vector

86) key phrase… “Rectified Linear Unit”

$$x_i \longrightarrow \boxed{w_0 \cdot x_i} \dashrightarrow \boxed{\max(0, w_0 \cdot x_i)} \longrightarrow \widehat{y_i}[0]$$

Schematic of a Neuron

Called activation function, *a(x)*

xi is m dimensional vector
w0 is a m dimensional vector

Several Activation Functions…

- Identity, $a(z) = z$
- Binary step, $a(z) = \begin{cases} 0 \ (\forall z < 0) \\ 1 \ (\forall z \geq 1) \end{cases}$ (for example, approx. $a(z) = \frac{1}{1+e^{-2000*z}}$)
- Sigmoid, $a(z) = \frac{1}{1+e^{-z}}$
- Tanh, $a(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
- Rectified Linear Unit (ReLU), $a(z) = \max(0, z)$
- Leaky ReLU, $a(z) = \begin{cases} 0.01 * z \ (\forall z < 0) \\ z \ (\forall z \geq 0) \end{cases}$
- Parametric ReLu, $a(z) = \begin{cases} \alpha * z \ (\forall z < 0) \\ z \ (\forall z \geq 0) \end{cases}$
- Several other…

Which one to take??? It's a <u>hyper parameter</u> search or common norm or intuition based…

*On a funny note, you will find people talking a lot about these, don't worry.. it's usual b**t! ☺*

87) key phrase… "Neural Network"

Neuron 0

$$x_i \rightarrow w_0 \cdot x_i \rightarrow \frac{1}{(1 + e^{-w_0 \cdot x_i})} \rightarrow \hat{y}_i[0]$$

Neuron 1

$$x_i \rightarrow w_1 \cdot x_i \rightarrow \frac{1}{(1 + e^{-w_1 \cdot x_i})} \rightarrow \hat{y}_i[1]$$

xi is m dimensional vector

w0, w1 are a m dimensional vectors

Can we have k Neurons???

Combining several *"Neural Transformations"*

1. $y_i[0] = a(w_0 \cdot x_i)$
2. $y_i[1] = a(w_1 \cdot x_i)$
3. $y_i[2] = a(w_2 \cdot x_i)$
4. ...
5. $y_i[k-1] = a(w_{k-1} \cdot x_i)$

xi is m dimensional vector
w0,…,wk-1 are a m dimensional vectors

Combining several *"Neural Transformations"*

1. $y_i[0] = a(w_0 \cdot x_i)$
2. $y_i[1] = a(w_1 \cdot x_i)$
3. $y_i[2] = a(w_2 \cdot x_i)$
4. …
5. $y_i[k-1] = a(w_{k-1} \cdot x_i)$

Let, $W_{k\times m} = \begin{bmatrix} w_{0,0}, \dots, w_{0,m-1} \\ w_{1,0}, \dots, w_{1,m-1} \\ \dots \\ w_{k-1,0}, \dots, w_{k-1,m-1} \end{bmatrix}$

xi is m dimensional vector

w0,…,wk-1 are a m dimensional vectors

Combining several *"Neural Transformations"*

1. $y_i[0] = a(w_0 \cdot x_i)$
2. $y_i[1] = a(w_1 \cdot x_i)$
3. $y_i[2] = a(w_2 \cdot x_i)$
4. ...
5. $y_i[k-1] = a(w_{k-1} \cdot x_i)$

Let, $W_{k \times m} = \begin{bmatrix} w_{0,0}, \dots, w_{0,m-1} \\ w_{1,0}, \dots, w_{1,m-1} \\ \dots \\ w_{k-1,0}, \dots, w_{k-1,m-1} \end{bmatrix}$

Let, $x_i' = W \times x_i$
Let, $x_i'' = a(x_i')$
Then,

$\hat{y_i} = x_i''$ (Regression) OR...
$\hat{y_i} = softmax(x_i'')$ (Classification)

xi is m dimensional vector
w0,...,wk-1 are a m dimensional vectors

Combining several *"Neural Transformations"*

1. $y_i[0] = a(w_0 \cdot x_i)$
2. $y_i[1] = a(w_1 \cdot x_i)$
3. $y_i[2] = a(w_2 \cdot x_i)$
4. ...
5. $y_i[k-1] = a(w_{k-1} \cdot x_i)$

Let, $W_{k \times m} = \begin{bmatrix} w_{0,0}, \dots, w_{0,m-1} \\ w_{1,0}, \dots, w_{1,m-1} \\ \dots \\ w_{k-1,0}, \dots, w_{k-1,m-1} \end{bmatrix}$

Let, $x_i' = W \times x_i$

Let, $x_i'' = a(x_i')$

Then,

$\hat{y_i} = x_i''$ OR...

$\hat{y_i} = softmax(x_i'')$

Squared Error Loss function,

$$L(W) = \frac{1}{2}(||\hat{y_i} - y_i||)^2$$

xi is m dimensional vector

w0,…,wk-1 are a m dimensional vectors

88) key phrase… “Layers in a neural network”

- A layer is an array of numbers

89) key phrase… “Input layer”

- Input xi m-dimensional data

90) key phrase… “Output layer”

- Output $\widehat{y_i}$ k-dimensional data

91) key phrase… “Hidden layer”

- An array of numbers after some vector operations or *transformations before*
- This array of numbers will under go *transforms after* as well

Two Layer Neural Network Regressor

- xi is m-dimensional data point (input layer – input layer)
- yi is k-dimensional data point (output layer – output layer)
  - k-class classification problem
  - one hot encoding

[There is no hidden layer]

simplified representation without bias

- Data set, D={(xi,yi)} i=1..N is given
- *Model,* $f(x_i) = a(W \times x_i)$
- *Loss function,* $L(W) = \frac{(||f(x_i) - y_i||)^2}{2}$

Three Layer Neural Network Regressor

- xi is m-dimensional data point
- yi is k-dimensional data point
  - k-class classification problem
  - one hot encoding
- ***There is 1 hidden layer (h-dimensional)***
- Data set, D={(xi,yi)} i=1..N is given
- *Model,*
  - $x_i' = a_1(W_1 \times x_i)$ *Note that* W_1 *is* $h \times m$ *matrix*
  - $f(x_i) = a_2(W_2 \times x_i')$ *Note that* W_2 *is* $k \times h$ *matrix*
- *Loss function,* $L(W_1, W_2) = \frac{(||f(x_i) - y_i||)^2}{2}$

simplified representation without bias

Four Layer Neural Network Regressor

- xi is m-dimensional data point
- yi is k-dimensional data point
  - k-class classification problem
  - one hot encoding
- ***There are 2 hidden layers (h_1-dimensions and h_2-dimensions)***
- Data set, D={(xi,yi)} i=1..N is given

simplified representation without bias

- *Model,*
  - $x_i^{(1)} = a_1(W_1 \times x_i)$ *Note that* W_1 *is* $h_1 \times m$ *matrix*
  - $x_i^{(2)} = a_2\left(W_2 \times x_i^{(1)}\right)$ *Note that* W_2 *is* $h_2 \times h_1$ *matrix*
  - $f(x_i) = a_3\left(W_3 \times x_i^{(2)}\right)$ *Note that* W_3 *is* $k \times h_2$ *matrix*
- *Loss function,* $L(W_1, W_2, W_3) = \frac{(||f(x_i) - y_i||)^2}{2}$

92) key phrase…"Multi Layer Neural Network Regressor"

- L+1 layer neural network
- xi is m-dimensional data point $x_i^{(0)}$ //0th layer (input layer $h_0 = m$)
- yi is k-dimensional data point $x_i^{(L)}$ //Lth layer (output layer $h_L = k$)
  - k-class classification problem
  - one hot encoding
- ***There are L-1 hidden layers (h_i dimensions in ith layer)***
- Data set, D={(xi,yi)} i=1..N is given
- *Model,*
  - $x_i^{(1)} = a_1\left(W_1 \times x_i^{(0)}\right)$ *Note that W_1 is $h_1 \times h_0$ matrix*
  - …
  - $x_i^{(l)} = a_l\left(W_l \times x_i^{(l-1)}\right)$ *Note that W_l is $h_l \times h_{l-1}$ matrix*
  - …
  - $f(x_i) = x_i^{(L)} = a_L\left(W_L \times x_i^{(L-1)}\right)$
- *Loss function,* $L([W_1, \dots, W_L]) = \frac{(||f(x_i) - y_i||)^2}{2}$

simplified representation without bias

Multi Layer Neural Network Regressor (*with bias*)

- L+1 layer neural network
- xi is m-dimensional data point $x_i^{(0)}$ //0th layer (input layer $h_0 = m$)
- yi is k-dimensional data point $x_i^{(L)}$ //Lth layer (output layer $h_L = k$)
  - [Optional]
    - k-class classification problem
    - one hot encoding
  - Its just a numeric vector in case of regression
- ***There are L-1 hidden layers (h_i dimensions in ith layer)***
- Data set, D={(xi,yi)} i=1..N is given
- *Model,*
  - $x_i^{(0)} = x_i^{(0)} \odot 1$
  - $x_i^{(1)} = a_1\left(W_1 \times x_i^{(0)}\right) \odot 1$ *Note that* W_1 *is* $h_1 \times (h_0 + 1)$ *matrix*
  - …
  - $x_i^{(l)} = a_i\left(W_l \times x_i^{(l-1)}\right) \odot 1$ *Note that* W_l *is* $h_l \times (h_{l-1} + 1)$ *matrix*
  - …
  - $f(x_i) = x_i^{(L)} = a_L\left(W_L \times x_i^{(L-1)}\right)$
- *Loss function,* $L([W_1, \dots, W_L]) = \frac{(f(x_i) - y_i)^2}{2}$

Define concatenation term:

$$\begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \mathbf{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{bmatrix} \odot \mathbf{1}$$

93) key phrase…"Multi Layer Neural Network Classifier"

$$\hat{y}_i = softmax(x_i^{(L)})$$

$$Cross\ Entropy\ Loss,\ l_i = -\sum_{j=1}^{j=k} y_i[j] * log(\hat{y}_i[j])$$

+ additionally regularization

Weight update equation

$$W_{(new)} = W_{(old)} - \eta \nabla L(W) \Big|_{W=W_{(old)}}$$

Weigh update for…
a three Layer Neural Network Regressor

- xi is m-dimensional data point
- yi is k-dimensional data point
  - [Optional]
    - k-class classification problem
    - one hot encoding
  - Its just a numeric vector in case of regression
- ***There is 1 hidden layer (h-dimensional)***

simplified representation without bias

- Data set, D={(xi,yi)} i=1..N is given
- *Model,*
  - $x_i' = a_1(W_1 \times x_i)$ *Note that* W_1 *is* $h \times m$ *matrix*
  - $f(x_i) = a_2(W_2 \times x_i')$ *Note that* W_2 *is* $m \times h$ *matrix*
- *Loss function,* $L(W_1, W_2) = \frac{(||f(x_i) - y_i||)^2}{2}$

Example of…
weight update for a 3 layer neural network

- $L([W_1, W_2]) = \frac{1}{2} * (f(x_i) - y_i)^2$
- $f(x_i') = a_2(W_2 \times x_i')$
- $a_2(W_2 \times x_i') = \frac{1}{1+e^{-W_2 \times x_i'}}$
- $x_i' = a_1(W_1 \times x_i)$
- $a_1(W_1 \times x_i) = \frac{1}{1+e^{-W_1 \times x_i}}$
- $L([W_1, W_2]) = \frac{1}{2} * \left(a_2\left(W_2 \times \left(a_1(W_1 \times x_i)\right)\right) - y_i\right)^2$
- $\frac{\partial L}{\partial W_1[1,2]} = \frac{\partial L}{\partial a_2} \times \frac{\partial a_2}{\partial a_1} \times \frac{\partial a_1}{\partial (W_1 \times x_i)} \times \frac{\partial (W_1 \times x_i)}{\partial W_1[1,2]}$

Similarly, you have to differentiate ..for all elements of all W's!

$$\underbrace{= \frac{1}{2} * 2 * \left(a_2\left(W_2 \times \left(a_1(W_1 \times x_i)\right)\right) - y_i\right)^1}_{\frac{\partial L}{\partial a_2}} * \underbrace{a_2\left(W_2 \times a_1(W_1 \times x_i)\right) * \left(1 - a_2\left(W_2 \times a_1(W_1 \times x_i)\right)\right)}_{\frac{\partial a_2}{\partial a_1}} * \underbrace{a_1(W_1 \times x_i) * \left(1 - a_1(W_1 \times x_i)\right)}_{\frac{\partial a_1}{\partial (W_1 \times x)}} * \underbrace{x_{1,2}}_{\frac{\partial (W_1 \times x)}{\partial W_1[1,2]}}$$

94) key phrase… "Vanishing gradients"

95) key phrase… “Exploding gradients”

ReLU activation function solves this problem…

96) key phrase… “Automatic Differentiation”