MLP Implementation In CUDA

MLPs are inspired by neurons in brain [1]. Multilayer Perceptrons are universal function approximators [2] [3]. This feature makes them applicable and attractive for the problem domains in which a mapping needs to be constructed between given input data to output data. They can be thought as a black box which maps given input to output data. Multi-layer perceptron consists of one input, one or more hidden layers and one output. The information is supplied from input and then flows through hidden layers and finally reaches to output layer. Therefore, this type of architecture is classified as feed-forward neural network architecture. The hidden layer number, node number's in layer's are up to the designer. The output layer node number depends on the problem to be solved. If the mapped function is so complicated the hidden layer number could be high. The hidden layers provide information to be extracted hierarchically. First hidden layer extracts useful knowledge from inputted information. Second hidden layer extracts knowledge from first hidden layer and so on. The knowledge is extracted from knowledge. At the output highest level information is obtained. Another important feature of the MLPs is the ability of generalization. It can estimate the target values of samples that are not shown previously in learning algorithm.

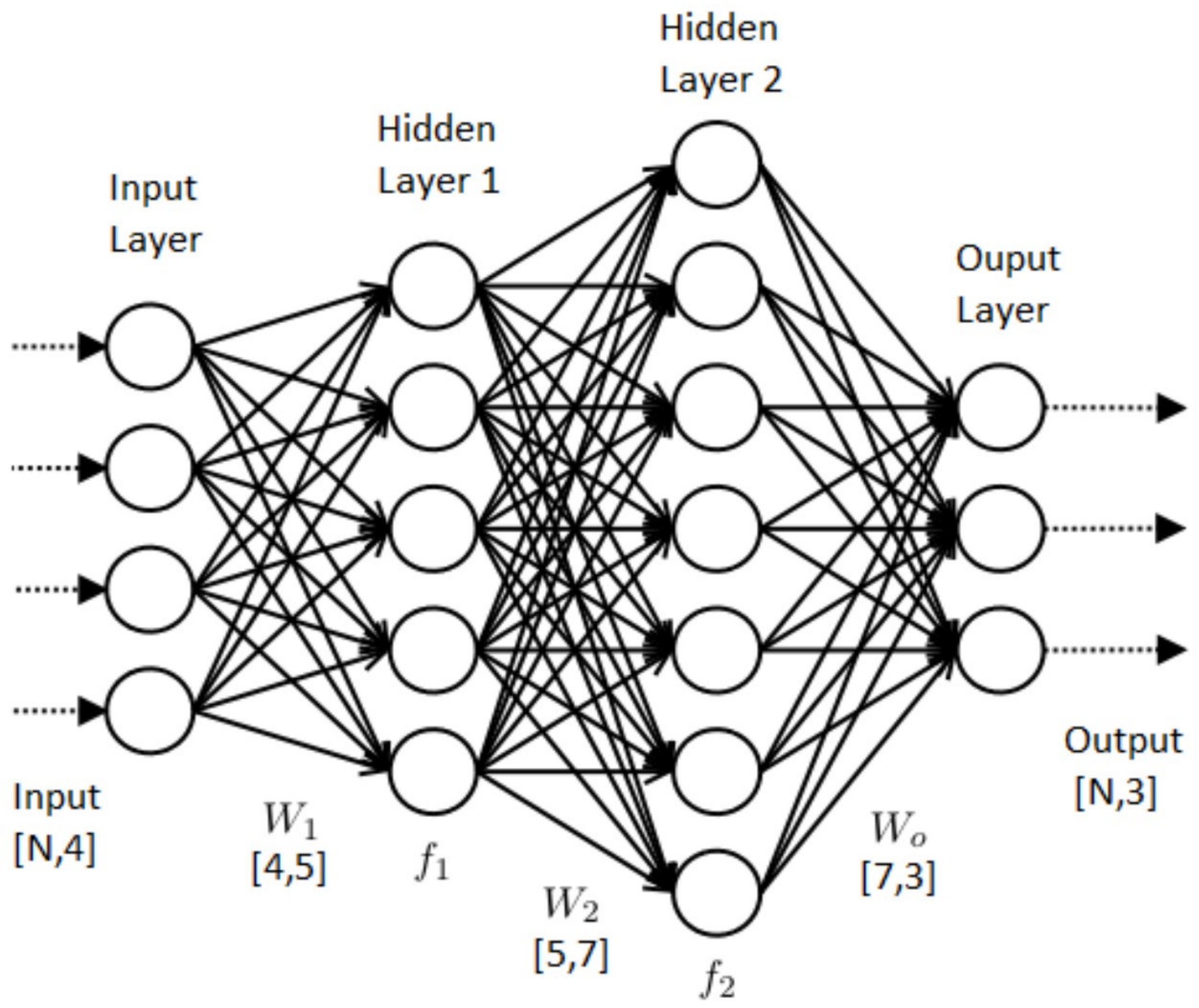
Hidden layers' nodes accumulate information from previous layer and applies a nonlinearity to this information. This is what enables MLPs to learn nonlinear mappings. There are several nonlinearity functions that could be used for this purpose. Main characteristic of the activation is being differentiable. Differentiability is critical in training algorithms. The activation function selection is also up to designer. By considering the features of the problem he/she must decide which function to use. Different activation functions possess different advantages and disadvantages.

MLPs are used in supervised learning tasks in which the target values are provided by the designer. These tasks are classified as regression and classification. MLP tries to match each given data point to given target. In regression problems network tries to learn continuous numerical values. When the information is provided to inputs, the network outputs one or more numerical values. In classification tasks network learns to separate one or more classes from each other. When the data is supplied to network, it outputs the probability of given data belonging to each class. In unsupervised learning no target values is given [4]. The aim of the MLP here is to obtain the structure of data such as cluster, hierarchies lyies in data. In reinforcement learning problems, value functions or policy functions are approximated. In this domain MLPs are used as function approximators [5]. They approximate the value function or policy function of the problem.

Learning in MLP means that mapping capability of the MLP between inputs and target is getting better and better progressively. A cost function is defined how well MLP performs the mapping task. A set of learning algorithms exists. They try minimize the error(cost) function by tweaking parameters of the network. Learning is realized in hidden layers. Learning simply involves two phases. In the first one, the data is supplied to network and the output is observed and we have the correct target values. We obtain an error metric using the target values and the network output. In second step we modify the weight vectors so that the new values of the weight will produce less error. This modification part could be performed by using different number of samples. The update of weights could be modified by using only one sample. The error is calculated for this sample and the weights are updated by using this error. Learning with this technique could be quite slow because using individual sample, learning algorithm might converge with oscillation. Another way is to modify the weights using the whole data set. The problem with this technique is that if we have thousands of samples, only one weight updating will take so long. There is also another technique in between these two extremes. A predefined number of samples are used to train the network in each step.

Our project purposes a simple Multilayer Perceptron implementation in CUDA for classification and regression tasks in machine learning. In serial programming, the outputs of layers in MLP are produced sequentially. A matrix of inputs is given, first layer output is calculated by performing required weight multiplications and nonlinear transformation for each sample in order. By using parallel computing, for individual sample the outputs are computed parallelly. All MLP operations can be deduced to matrix operations. GPUs are capable of running thousands of concurrent threads. Matrix operations can be performed on GPU quite fast. So, implementing a MLP for GPU will result in huge performance gains.

<u>MLP Operations</u>



(Figure 1: Example of a Multilayer Perceptron. Input, output and layer weights' sizes are shown. [6])
All related operations are shown below. Hidden layers first calculate linear combination of previous layers' then applies a nonlinear function. The output layer, if the problem is a regression problem, just performs weighted summation of last hidden layer. If classification problem, it performs again weighted summation and applies softmax function. Softmax function turns this summation into probability distribution.

$$InputData = X^{(0)} = \begin{vmatrix} X_1^{(0)0} & X_2^{(0)0} & X_3^{(0)0} \\ X_1^{(0)1} & X_2^{(0)1} & X_3^{(0)2} \\ X_1^{(0)2} & X_2^{(0)2} & X_3^{(0)2} \\ X_1^{(0)3} & X_2^{(0)3} & X_3^{(0)3} \\ X_1^{(0)4} & X_2^{(0)4} & X_3^{(0)4} \end{vmatrix} \qquad TargetData = R = \begin{vmatrix} R_1^0 & R_2^0 & R_3^0 \\ R_1^1 & R_2^1 & R_3^1 \\ R_1^2 & R_2^2 & R_3^2 \\ R_1^3 & R_2^3 & R_3^3 \\ R_1^4 & R_2^4 & R_3^4 \end{vmatrix}$$

$$X_{input\ node\ index}^{(0)sample\ index} \qquad R_{output\ node\ index}^{sample\ index}$$

Each sample is represented as row vector

$$First\ sample\ is\ X^{(0)0} = \begin{vmatrix} X_1^{(0)0} & X_2^{(0)0} & X_3^{(0)0} \end{vmatrix}$$

$$Second\ sample\ is\ X^{(0)1} = \begin{vmatrix} X_1^{(0)1} & X_2^{(0)1} & X_3^{(0)1} \end{vmatrix}$$

$$First\ sample's\ target\ is\ R^0 = \begin{vmatrix} R_1^0 & R_2^0 & R_3^0 \end{vmatrix}$$

$$Second\ sample's\ target\ is\ R^1 = \begin{vmatrix} R_1^0 & R_2^0 & R_3^0 \end{vmatrix}$$

$$W^{(1)} = \begin{vmatrix} W_{1,1}^{(1)} & W_{1,2}^{(1)} & W_{1,3}^{(1)} & W_{1,4}^{(1)} & W_{1,5}^{(1)} \\ W_{2,1}^{(1)} & W_{2,2}^{(1)} & W_{2,3}^{(1)} & W_{2,4}^{(1)} & W_{2,5}^{(1)} \\ W_{3,1}^{(1)} & W_{3,2}^{(1)} & W_{3,3}^{(1)} & W_{3,4}^{(1)} & W_{3,5}^{(1)} \\ W_{4,1}^{(1)} & W_{4,2}^{(1)} & W_{4,3}^{(1)} & W_{4,4}^{(1)} & W_{4,5}^{(1)} \end{vmatrix} \qquad W^{(2)} = \begin{vmatrix} W_{1,1}^{(2)} & W_{1,2}^{(2)} & W_{1,3}^{(2)} & W_{1,4}^{(2)} & W_{1,5}^{(2)} & W_{1,6}^{(2)} & W_{1,7}^{(2)} \\ W_{2,1}^{(2)} & W_{2,2}^{(2)} & W_{2,3}^{(2)} & W_{2,4}^{(2)} & W_{2,5}^{(2)} & W_{2,6}^{(2)} & W_{2,7}^{(2)} \\ W_{3,1}^{(2)} & W_{3,2}^{(2)} & W_{3,3}^{(2)} & W_{3,4}^{(2)} & W_{3,5}^{(2)} & W_{3,6}^{(2)} & W_{3,7}^{(2)} \\ W_{4,1}^{(2)} & W_{4,2}^{(2)} & W_{4,3}^{(2)} & W_{4,4}^{(2)} & W_{4,5}^{(2)} & W_{4,6}^{(2)} & W_{4,7}^{(2)} \\ W_{5,1}^{(2)} & W_{5,2}^{(2)} & W_{5,3}^{(2)} & W_{5,4}^{(2)} & W_{5,5}^{(2)} & W_{5,6}^{(2)} & W_{5,7}^{(2)} \end{vmatrix}$$

$$B^{(1)} = \begin{vmatrix} B_1^{(1)} & B_2^{(1)} & B_3^{(1)} & B_4^{(1)} & B_5^{(1)} \end{vmatrix}$$

$$B^{(2)} = \begin{vmatrix} B_1^{(2)} & B_2^{(2)} & B_3^{(2)} & B_4^{(2)} & B_5^{(2)} & B_6^{(2)} & B_7^{(2)} \end{vmatrix}$$

$$B^{(3)} = \begin{vmatrix} B_1^{(3)} & B_2^{(3)} & B_3^{(3)} \end{vmatrix}$$

$$X_0^{(0)} = X_0^{(1)} = X_0^{(2)} = +1$$

$$W_o = W^{(3)} = \begin{vmatrix} W_{1,1}^{(3)} & W_{1,2}^{(3)} & W_{1,3}^{(3)} \\ W_{2,1}^{(3)} & W_{2,2}^{(3)} & W_{2,3}^{(3)} \\ W_{3,1}^{(3)} & W_{3,2}^{(3)} & W_{3,3}^{(3)} \\ W_{4,1}^{(3)} & W_{4,2}^{(3)} & W_{4,3}^{(3)} \\ W_{5,1}^{(3)} & W_{5,2}^{(3)} & W_{5,3}^{(3)} \\ W_{6,1}^{(3)} & W_{6,2}^{(3)} & W_{6,3}^{(3)} \\ W_{7,1}^{(3)} & W_{7,2}^{(3)} & W_{7,3}^{(3)} \end{vmatrix}$$

$$Softmax_i(a) = \frac{\exp a_i}{\sum \exp a_i} \qquad \begin{aligned} O^{(l)} &= X^{(l-1)}.\,W^{(l)} + B^{(l)} \\ O^{(1)} &= X^{(0)}.\,W^{(1)} + B^{(1)} \\ O^{(2)} &= X^{(1)}.\,W^{(2)} + B^{(2)} \\ O^{(3)} &= X^{(2)}.\,W^{(3)} + B^{(3)} \end{aligned} \qquad \begin{aligned} X^{(l)} &= \sigma(O^{(l)}) \\ X^{(1)} &= \sigma(O^{(1)}) \\ X^{(2)} &= \sigma(O^{(2)}) \end{aligned} \qquad \sigma(x) = \frac{1}{1+e^{-x}}$$

$$if\ regression\ Output = O^{(3)}\ ifclassification\ Output = Softmax(O^{(3)})$$

Learning Algorithm (Backpropagation)

Learning algorithm adjusts network’s weights to minimize error (cost) function. While training the network we choose to use backpropagation algorithm. Backpropagation algorithm adjusts weights of network by following gradient direction in which the error is reduced on error surface. It is a gradient based algorithm. It calculates a gradient on surface of error function. Then it modifies the weights by adding the calculated gradient in reverse direction to weights.

$$\Delta W = -\eta . \frac{\partial E}{\partial W} \quad \frac{\partial E}{\partial W} \text{ is the gradient in which error increases.}$$

$$\eta \text{ controls magnitude of the gradient. Inverse gradient reduces the error.}$$

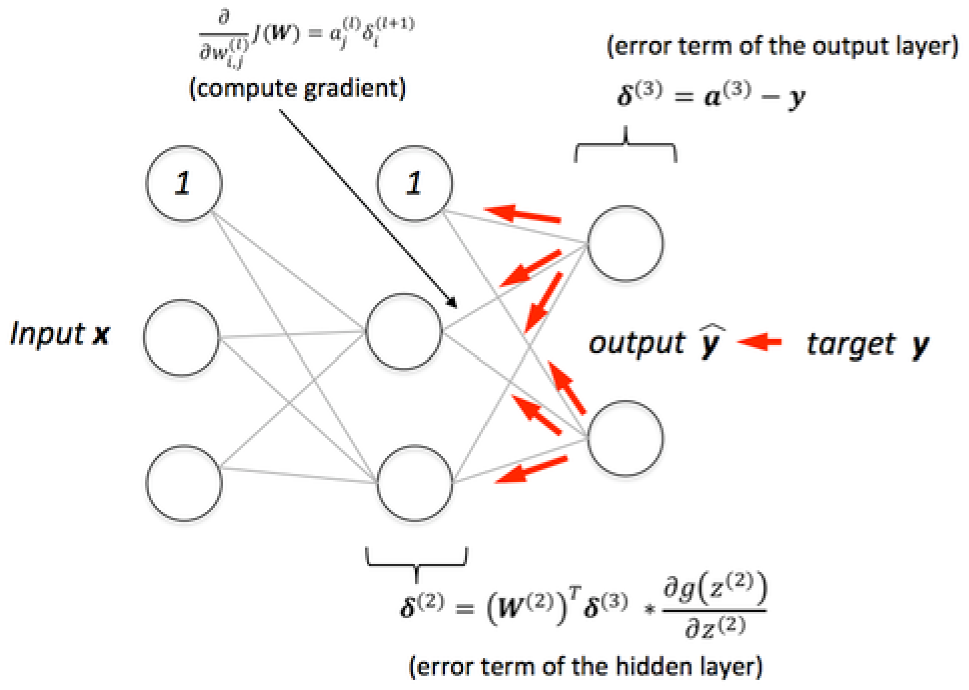
In classification tasks cross entropy function, in regression tasks summed squared error function is used.

$$\text{if regression Summed Squared Loss } E = \frac{1}{2} \sum_{t(\text{sample index})=0}^{n} \sum_{m(\text{output node index})=1}^{M} (R_m^t - O_m^{(3)t})^2$$

$$Error = R - Output$$

$$\text{if classification Cross Entropy Loss} = E = \sum_{t(\text{sample index})=0}^{n} \sum_{m(\text{output node index})=1}^{M} (-R_m^t * log(O_m^{(3)t}))$$

Backpropagation algorithm relies on chain rule. By starting from output is distributes error from output layer to input layer. Each weight is changed depending on how much it contributes to error.



$\frac{\partial}{\partial w_{i,j}^{(l)}} J(\boldsymbol{W}) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\boldsymbol{\delta}^{(3)} = \boldsymbol{a}^{(3)} - \boldsymbol{y}$
1
1
Input **x**
output $\hat{\boldsymbol{y}}$
target **y**
$\boldsymbol{\delta}^{(2)} = (\boldsymbol{W}^{(2)})^T \boldsymbol{\delta}^{(3)} * \frac{\partial g(z^{(2)})}{\partial z^{(2)}}$
(error term of the hidden layer)

(Figure 2: It shows how backpropagation distributes error originated at output to inner layers. [7])

$$\Delta W^{(3)}_{(k,m)} = \eta.\sum_t \sum_m (R^t_m - O^{(3)t}_m).\,X^{(2)t}$$

$$\Delta W^{(2)}_{(j,k)} = \eta.\sum_t \sum_m (R^t_m - O^{(3)t}_m).\,W^{(3)}_{(k,m)}.\,\sigma(O^{(2)t}_k).\,(1-\sigma(O^{(2)t}_k)).\,X^{(1)t}_j$$

$$\Delta W^{(1)}_{(i,j)} = \eta.\sum_t \sum_m (R^t_m - O^{(3)t}_m).\sum_{k=1}.\,W^{(3)}_{(k,m)}.\,\sigma(O^{(2)t}_k).\,(1-\sigma(O^{(2)t}_k).\,W^{(2)}_{(j,k)}.\,\sigma(O^{(1)t}_j).\,(1-\sigma(O^{(1)t}_j).\,X^{(0)t}_i$$

$\odot$ *is Hadamard multiplication*

$$\begin{gathered}
\textit{In matrix operations form:}\\
\textit{Output Error} = \textit{Error} = R - \textit{Output}\\
\Delta W^{(3)} = \eta.\,(X^{(2)})^T.\,\textit{Output Error}\\
\Delta W^{(2)} = \eta.\,(X^{(1)})^T.\,((\textit{Output Error}.\,(W^{(3)})^T)\odot(\sigma(O^{(2)}).\,(1-\sigma(O^{(2)})))\\
\Delta W^{(1)} = \eta.\,(X^{(0)})^T.\,((((\textit{Output Error}.\,(W^{(3)})^T)\odot(\sigma(O^{(2)}).\,(1-\sigma(O^{(2)})))).\,(W^{(2)})^T)\odot(\sigma(O^{(1)}).\,(1-\sigma(O^{(1)}))))\\
\textit{Hidden Layer 2 Error} = (\textit{Output Error}.\,(W^{(3)})^T)\odot(\sigma(O^{(2)}).\,(1-\sigma(O^{(2)}))\\
\textit{Hidden Layer 1 Error} = (\textit{Hidden Layer 2 Error}.\,(W^{(2)})^T)\odot(\sigma(O^{(1)}).\,(1-\sigma(O^{(1)}))\\
\Delta W^{(3)} = \eta.\,(X^{(2)})^T.\,\textit{Output Error}\\
\Delta W^{(2)} = \eta.\,(X^{(1)})^T.\,\textit{Hidden Layer 2 Error}\\
\Delta W^{(1)} = \eta.\,(X^{(0)})^T.\,\textit{Hidden Layer 1 Error}\\
\textit{LayerError}(l) = (\textit{LayerError}(l+1).\,(W^{(l+1)})^T)\odot(\sigma(O^{(l)}).\,(1-\sigma(O^{(l)})))
\end{gathered}$$

$$\Delta B^{(3)} = \begin{vmatrix} +1\\ +1\\ +1\\ \vdots \end{vmatrix}.\,\textit{Output Layer Error} \qquad \Delta B^{(2)} = \begin{vmatrix} +1\\ +1\\ +1\\ \vdots \end{vmatrix}.\,\textit{Hidden Layer 2 Error}$$

$$\Delta B^{(1)} = \begin{vmatrix} +1\\ +1\\ +1\\ \vdots \end{vmatrix}.\,\textit{Hidden Layer 1 Error}$$

$$\textit{Dimension of column vector of } +1s \textit{ is } (\textit{sample number}, 1)$$

$$\frac{\partial(\frac{1}{2}\Sigma_t\Sigma_m(R^t_m - O^{(3)t}_m)^2)}{\partial w} = \sum_t\sum_m (R^t_m - O^{(3)t}_m).\,-1.\,\frac{\partial O^{(3)t}_m}{\partial w}$$

$$\frac{\partial(\Sigma_t\Sigma_m(-R^t_m * log(O^{(3)t}_m)))}{\partial w} = \sum_t\sum_m (R^t_m - O^{(3)t}_m).\,-1.\,\frac{\partial O^{(3)t}_m}{\partial w}$$

For classification and regression cost functions we obtain same gradient formulas.

All these learning operations can be represented with matrices operations. Matrix multiplication, Hadamard multiplication, transpose operation, matrix subtraction and addition, scalar-matrix multiplication.

[1] Warren S.McCulloch and Walter H. Pitt’s 1943 paper, ‘‘A Logical Calculus of the Ideas Immanent in Nervous Activity,’’

[2] Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", *Mathematics of Control, Signals, and Systems*, 2(4), 303-314.

[3] Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", *Neural Networks*, 4(2), 251–257

[4] Hinton, G. E.; Salakhutdinov, R.R. (28 July 2006). "Reducing the Dimensionality of Data with Neural Networks". *Science*. 313 (5786): 504–507.

[5] Tesauro, Gerald (March 1995). "Temporal Difference Learning and TD-Gammon"

[6] https://techblog.viasat.com/wp-content/uploads/2017/07/ANN-Diagram.png

[7] https://sebastianraschka.com/images/faq/visual-backpropagation/backpropagation.png