HIGH PERFORMANCE PARALLEL PROGRAMMING (CS61064)

Soumyajit Dey
CSE, IIT Kharagpur

Machine Learning

- Learning Rules/Functions from Examples
- Deep Learning (a branch of ML) has gained significant success over the past few years.
- Advancements in Computer Architecture and GPU Programming have allowed ideas developed in the 80s to be fully realized.

Examples

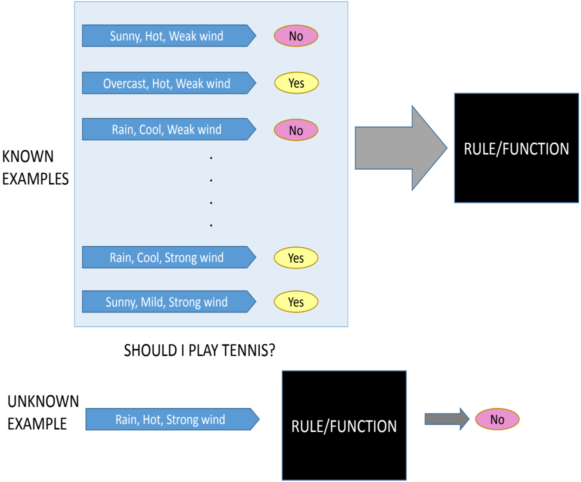


Figure: Decision Making

Examples

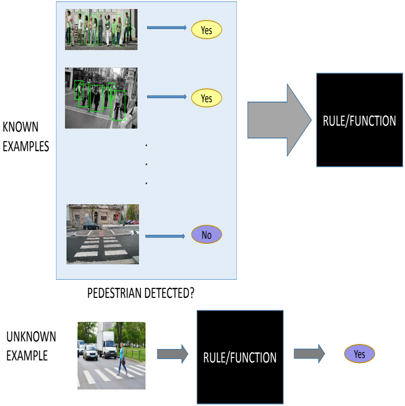


Figure: Pedestrian Detection

ML Problem Characteristics

- Characterize each example by a vector of real or boolean values (feature vector)
- Associate a label/target with each example (Supervised Learning).
- Labels may be discrete values (Classification) or continuous values (Regression)
- A set of feature vectors and labels constitute a training data set.
- Using the training dataset and some supervised learning algorithm a predictive model is trained.

Supervised Learning Workflow

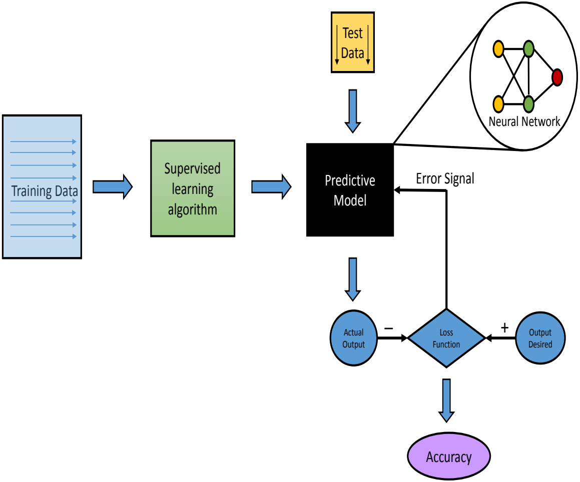


Figure: Pedestrian Detection

Supervised Learning Algorithm

- Characterize the model to be learned by some parameter θ
- Define a loss function between predicted output and actual output. (function of θ and inputs)
- Update θ so that the loss function is minimized.
- The more the loss function is closer to the minima, the more closer is the predicted output to the actual output

Neural Networks

- Neural networks have gained popularity as parametric learning methodologies, due to advancement of Deep Learning.
- The methodology is derived directly from the working model of the human brain.
- Can be depicted as a network of weights associating input values to output values.

Supervised Learning Workflow

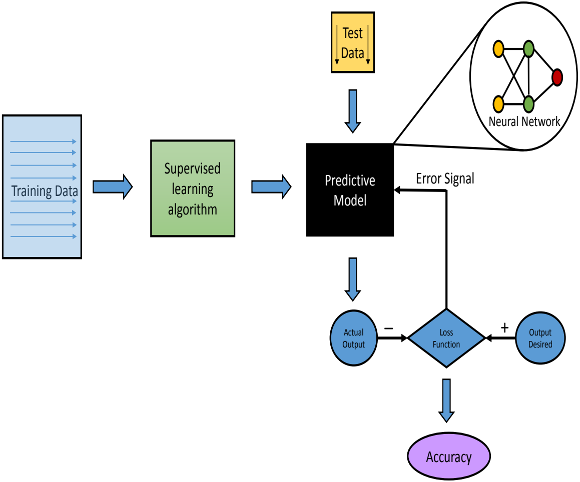


Figure: Pedestrian Detection

Building Block of a NN

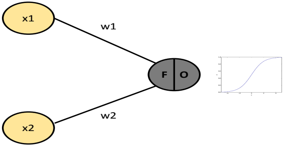
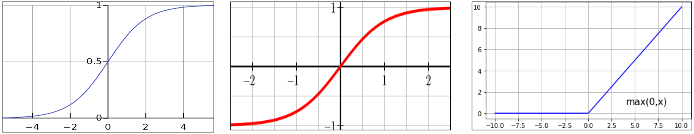


Figure: Perceptron

- The yellow nodes are inputs while the grey node is a neuron.
- The edges(synapses) have weights)
- The incoming value to the neuron is $f = \sum w_i x_i$
- The outgoing value is a nonlinear function of f i.e. $o = \sigma(f)$

Activation Functions

SIGMOID | HYPERBOLIC TANGENT | RECTIFIED LINEAR UNIT



$$y = \frac{1}{1+e^{-x}}$$

$$y = \frac{1-e^{-2x}}{1+e^{-2x}}$$

$$y = \max(0, x)$$

Figure: Linear/Non-linear functions

Multilayer Perceptron/ FeedForward Network

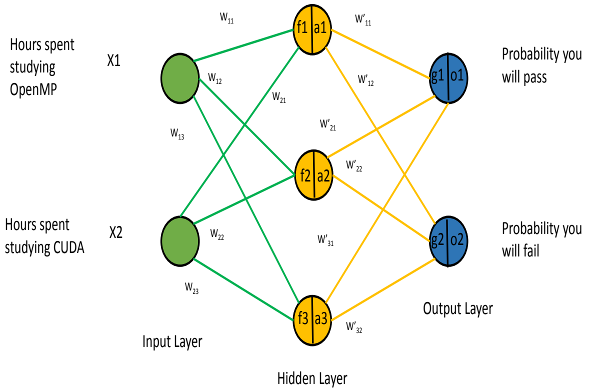


Figure: Classification

Multilayer Perceptron/ FeedForward Network

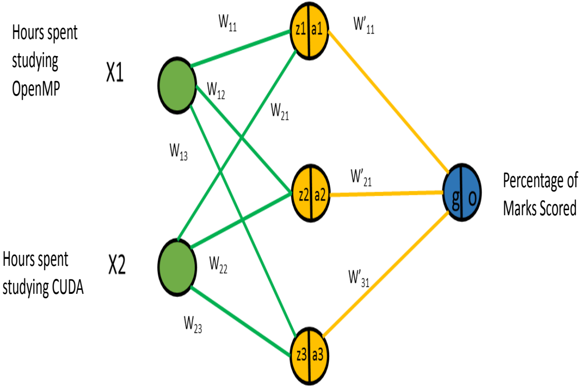


Figure: Regression

Note

- The objective of this course is to get you acquainted with the computation involved while training and testing a neural network.
- We shall not discuss core ML principles which should be followed while designing neural networks for various problem domains.

Neural Networks

- Each neuron of a layer accumulates a weighted sum of the inputs from the previous layer.
- Each neuron applies an activation function to its input and propagates the output to a neuron of the next layer..
- This results in a series of linear and non-linear transformations from the input layer to the output layer.
- The predicted output value for every input example is a function of the input feature values and the weights and activation in the network

Feedforward NN Forward Propagation

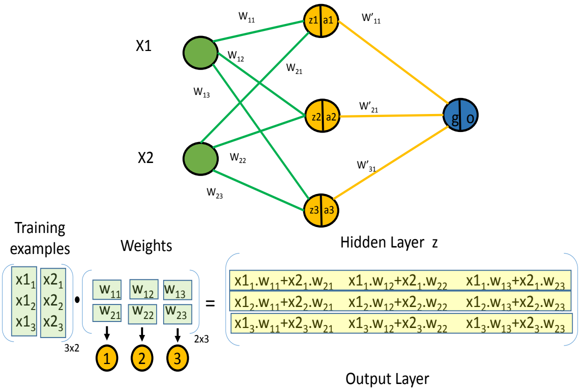


Figure: Feedforward propagation

Feedforward NN Forward Propagation

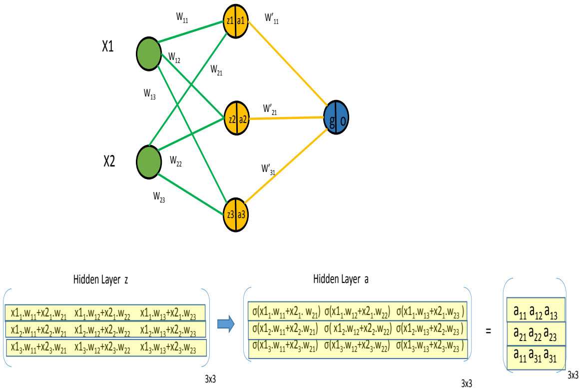


Figure: Feedforward propagation

Feedforward NN Forward Propagation

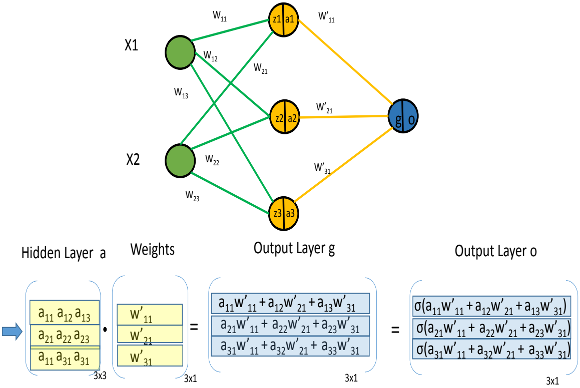


Figure: Feedforward propagation

Neural Networks

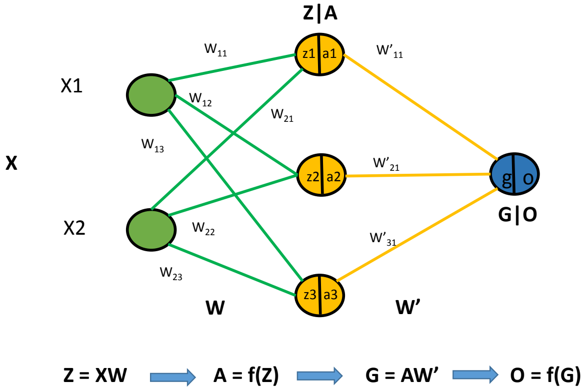


Figure: Feedforward propagation

The feedforward propagation can therefore be expressed as a series of matrix computations and element-wise non-linear transformations which involves scope for GPU parallelization.

Neural Networks

- Given the structure and weights of a neural network, we now know how to compute the predicted output value for an input example.
- The structure of the network i.e. the number of layers and number of neurons per layer are referred as hyperparameters (to be decided by the user).
- The weights are the actual parameters (θ) which will be learned during the course of training.
- Training involves the minimization of a cost/loss function.

Loss Function

- Define a loss function $J(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - o_i)^2$ where y_i and o_i are the actual outputs and predicted outputs of input example i respectively.
- Recall the feedforward propagation equations. $\mathbf{Z} = \mathbf{XW}, \mathbf{A} = f(\mathbf{Z}), \mathbf{G} = \mathbf{AW}', \mathbf{O} = f(\mathbf{G})$
- Therefore, $J = \sum \frac{1}{2}(\mathbf{Y} - f(f(\mathbf{XW})\mathbf{W'})^2)$ where the summation operation is over the elements of the column vector obtained from the loss function.
-

Minimizing Loss Function

- Find values of **W** and **W'** so that J(w) is minimized.
- Compute $\frac{\partial J}{\partial \mathbf{W}}$ and $\frac{\partial J}{\partial \mathbf{W'}}$
- Instead of setting the partial derivatives to zero and finding a solution, we perform numerical gradient descent.
- Update the weights **W** and **W'** in the direction of the steepest gradient descent

Gradient Descent

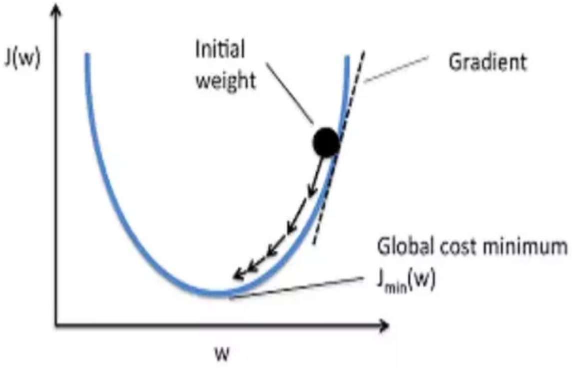


Figure: Obtaining minimum J

Gradient Descent

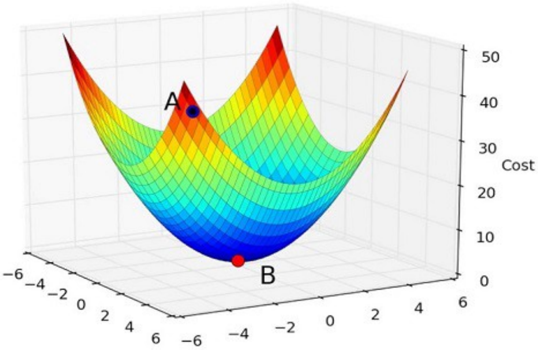


Figure: Obtaining minimum J

Training Using Backpropagation

- Perform feedforward propagation to obtain predicted output values for each input example.
- Compute loss function $J(w)$
- Compute $\frac{\partial J}{\partial \mathbf{W}}$ for each weight matrix
- Update weights of every weight matrix W with the gradient.
- Repeat steps 1-3 until there is no change in gradient.

Computing Partial Derivatives w.r.t a Matrix

$$\frac{\partial J}{\partial W} = \begin{bmatrix} \frac{\partial J}{w_{11}} & \frac{\partial J}{w_{12}} & \frac{\partial J}{w_{13}} & \cdots & \frac{\partial J}{w_{1n}} \\ \frac{\partial J}{w_{21}} & \frac{\partial J}{w_{22}} & \frac{\partial J}{w_{23}} & \cdots & \frac{\partial J}{w_{2n}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J}{w_{d1}} & \frac{\partial J}{w_{d2}} & \frac{\partial J}{w_{d3}} & \cdots & \frac{\partial J}{w_{dn}} \end{bmatrix}$$

The dimensions of the weight matrix and its gradients will be the same.

Neural Networks

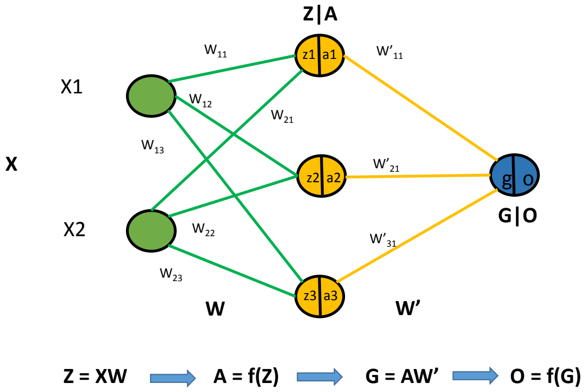


Figure: Feedforward propagation

We first compute $\frac{\partial J}{\partial \mathbf{W'}}$

Chain Rule

$$\frac{\partial J}{\partial \mathbf{W'}} = \frac{\partial}{\partial \mathbf{W'}} \sum \frac{1}{2}(\mathbf{Y} - \mathbf{O})^2 = -\sum (Y - O) \frac{\partial O}{\partial \mathbf{G}} \frac{\partial G}{\partial \mathbf{W'}}$$

$$= -\sum (Y - O) f'(G) \frac{\partial G}{\partial \mathbf{W'}}$$

Consider input example 1 and one weight say w'_{11}
The derivative is
$-(y_1 - o_1) f'(g1) \frac{\partial}{\partial w'_{11}} (w'_{11} a_{11} + w'_{21} a_{12} + w'_{31} a_{13})$
$= \delta_1^1 a_{11}$
For input example 2, the gradient would be $= \delta_2^1 a_{21}$
For input example 3, the gradient would be $= \delta_3^1 a_{31}$

Computing Partial Derivatives w.r.t a Matrix

$$\frac{\partial J}{\partial W'} = \begin{bmatrix} \frac{\partial J}{w'_{11}} \\ \\ \frac{\partial J}{w'_{21}} \\ \\ \frac{\partial J}{w'_{31}} \end{bmatrix} = \begin{bmatrix} \delta^1_1 a_{11} + \delta^1_2 a_{21} + \delta^1_3 a_{31} \\ \\ \delta^1_1 a_{12} + \delta^1_2 a_{22} + \delta^1_3 a_{32} \\ \\ \delta^1_1 a_{13} + \delta^1_2 a_{23} + \delta^1_3 a_{33} \end{bmatrix}$$

This can be expressed as $\mathbf{A}^T \delta^1$

Computing gradient w.r.t W

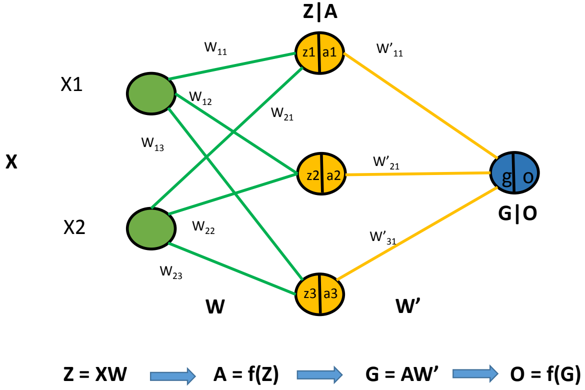


Figure: Feedforward propagation

In a similar fashion we compute $\frac{\partial J}{\partial \mathbf{W}}$

Chain Rule

$$\frac{\partial J}{\partial \mathbf{W}} = -\sum (Y - O)\frac{\partial O}{\partial \mathbf{G}}\frac{\partial G}{\partial \mathbf{W}}$$

$$= -\sum (Y - O) f'(G)\frac{\partial G}{\partial \mathbf{A}}\frac{\partial A}{\partial \mathbf{W}}$$

$$= \sum \delta^1 \mathbf{W'}^T \frac{\partial A}{\partial \mathbf{W}}$$

$$= \sum \delta^1 \mathbf{W'}^T \frac{\partial A}{\partial \mathbf{Z}}\frac{\partial Z}{\partial \mathbf{W}}$$

$$= \sum \delta^1 \mathbf{W'}^T f'(\mathbf{Z})\frac{\partial Z}{\partial \mathbf{W}}$$

$$= \mathbf{X}^T \delta^1 \mathbf{W'}^T f'(\mathbf{Z}) \text{ (Derive!)}$$

$$= \mathbf{X}^T \delta^2$$

Backward Propagation Delta Rule

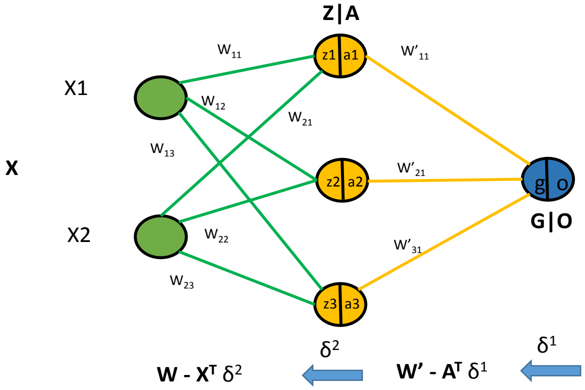


Figure: Backward propagation

Summary

- Feedforward propagation can be performed by a series of linear and non linear transformations involving matrix operations starting from the input layer.
- Backpropagation also involves a series of linear and non linear transformations involving matrix operations starting from the output layer.
- Each operation and transformation exhibits parallelism and scope for optimizations using a GPU.

Building a DL Library

A DL Library should have support for the following

- Provide constructs for specifying a network.
- Provide efficient routines for feedforward, backpropagation and gradient computation.
- Provide routines for training and testing.
- Should support parallel and distributed processing for the computation passes.

DL Libraries like Tensorflow and Theano use a Computational Graph Abstraction for encoding a neural network.

Computational Graph

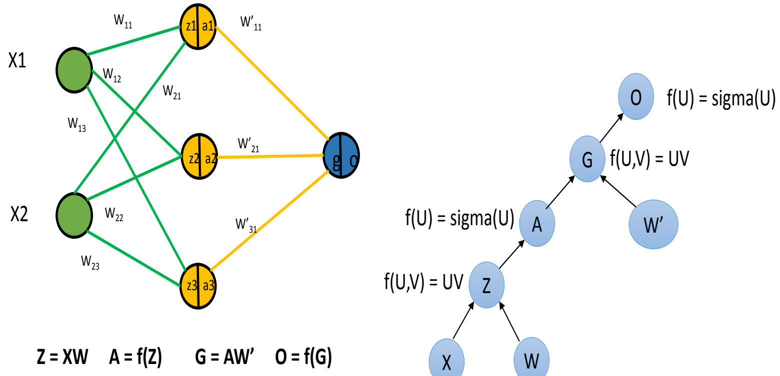


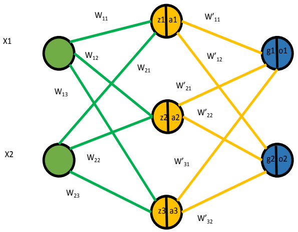
Figure: MLP vs CG

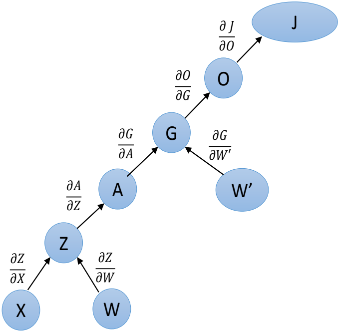
Computational Graph

A graph that denotes the functional description of the required computation.

- A node with no incoming edge is a tensor, matrix, vector or scalar value.
- A node with an incoming edge is a function of the edge's tail node. computation.
- An edge represents a data dependency between nodes.
- A node knows how to compute its value and the value of its derivative w.r.t each incoming edges's tail node.

Computational Graph





J is the loss function

Z = XW A = f(Z) G = AW' O = f(G)

Figure: MLP vs CG

Computations for a CG

- **Forward Computation:** Loop over each node in topological order and compute the value of the node given its inputs.
- **Backward Computation** Loop over each node in reverse topological order and compute the derivative of the final goal node with respect to each incoming edge's tail node.

Backpropagation: Gradient w.r.t W'

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial O}\frac{\partial O}{\partial G}\frac{\partial G}{\partial A}\frac{\partial A}{\partial Z}\frac{\partial Z}{\partial W}$$

J is the loss function

Z = XW A = f(Z) G = AW' O = f(G)

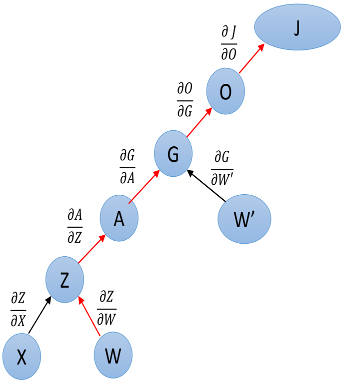


Figure: Computing gradients on CG

Backpropagation: Gradient w.r.t W

$$\frac{\partial J}{\partial W'} = \frac{\partial J}{\partial O}\frac{\partial O}{\partial G}\frac{\partial G}{\partial W'}$$

J is the loss function

Z = XW A = f(Z) G = AW' O = f(G)

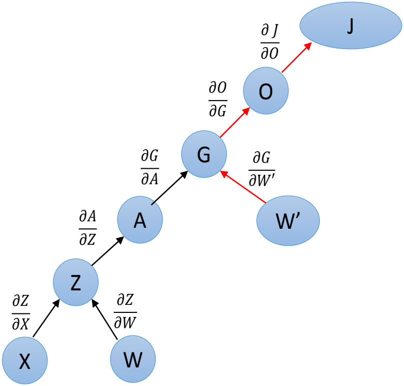


Figure: Computing gradients on CG