Unit-5
Multilayer Perceptrom

By: Arjun Saud, Asst. Prof. CDCSIT,TU

Introduction

- A multilayer feed-forward network consists of an input layer, one or more hidden layers, and an output layer. Computations take place in the hidden and output layers only.
- The input signal propagates through the network in a forward direction layer-by-layer. Such neural networks are called multilayer perceptrons (MLPs).
- They have been successfully applied to many difficult and diverse problems.
- Multilayer perceptrons are typically trained using so-called error backpropagation algorithm. This is a supervised error-correction learning algorithm.

Introduction

Properties of MLPs

- Each neuron has a smooth (differentiable everywhere) nonlinear activation function. This is usually a sigmoidal nonlinearity defined by the logistic function

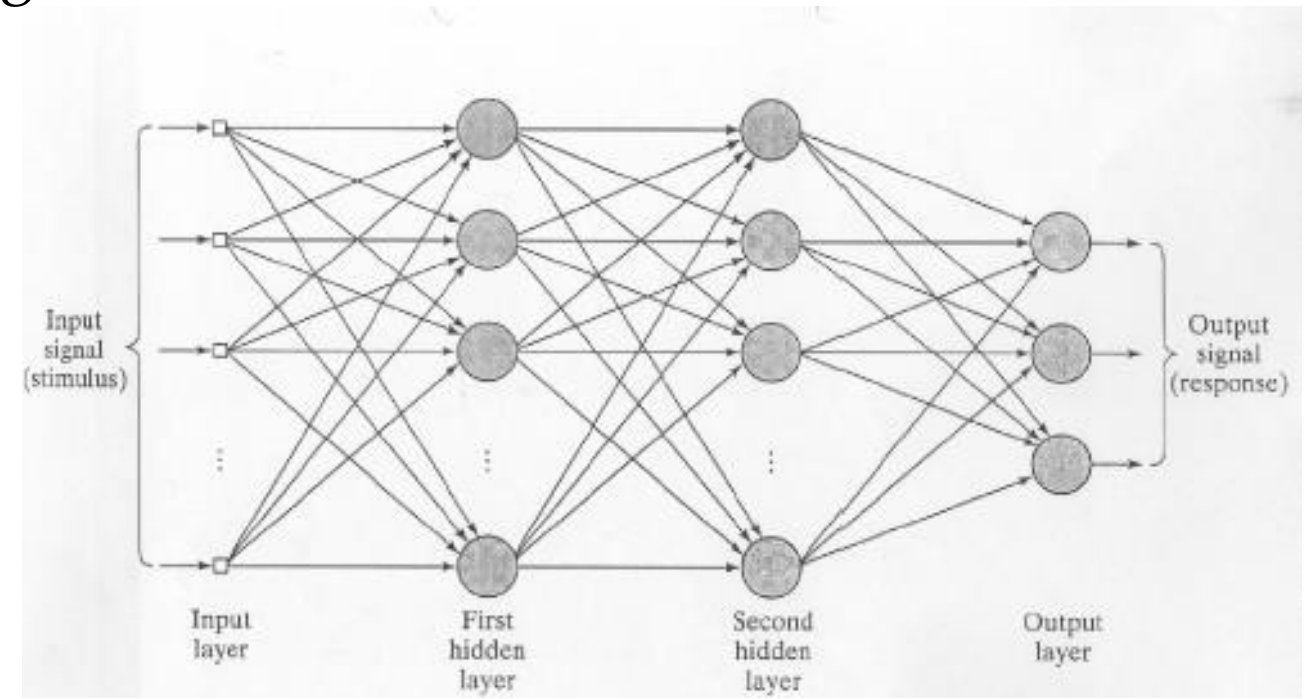
$$y_j = \frac{1}{1+\exp(-v_j)}, \text{ where } v_j \text{ is weighted sum of inputs}$$

  Note: Nonlinearities are important: otherwise the network could be reduced to a linear single-layer perceptron.

- The network contains hidden layer(s), enabling learning complicated tasks and mappings.
- The network has a high connectivity.

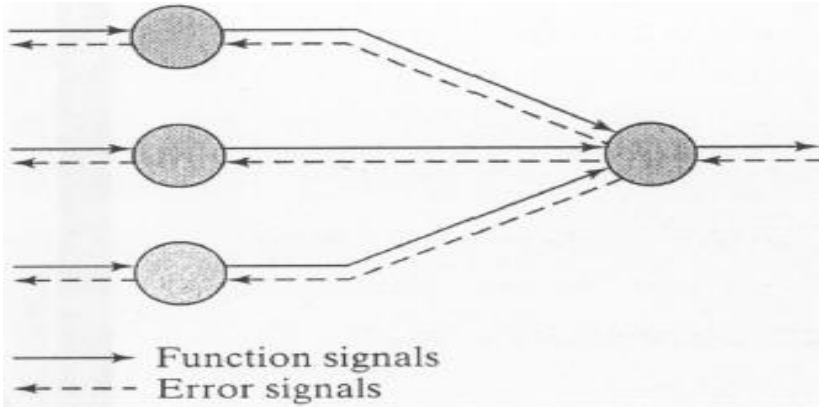
Introduction

- An architectural graph of a fully connected multilayer perceptron with two hidden layers and an output layer is given below.



Introduction

- Two kinds of signals appear in the MLP networks: Function Signal and Error Signal.
  - **Function Signals:** Function signals are the Input signals that propagates in forward direction through the network and produces output signals in the last phase.
  - **Error Signals:** These are the signals originate at output neurons, and propagate layer by layer in backward direction through the network.



Introduction

- Each hidden or output neuron performs two computations:

1. The computation of the function signal appearing at its output. This is a nonlinear function of the input signal and synaptic weights of that neuron.
2. The computation of an estimate of the gradient vector, needed in the backward pass.

Batch and Online Learning

Batch Learning

- In the batch method of supervised learning, adjustments to the synaptic weights of the multilayer perceptron are updated *after the presentation of all the N examples in the* training sample T that constitute one *epoch of training*.
- *In other words, the cost function* for batch learning is defined by the average error energy and adjustments to the synaptic weights of the multilayer perceptron are made on an *epoch-by-epoch basis*.

Batch and Online Learning

Batch Learning

- for each epoch of training, the examples in the training sample t are *randomly shuffled*.
- With the method of gradient descent used to perform the training, the advantages of batch learning include the following:
  - *Accurate estimation of the gradient vector*, **thereby guaranteeing,** convergence of the method of steepest descent to a local minima for non-convex surfaces and at global minima for convex surface;
  - *parallelization of the learning process*.

Batch and Online Learning

Batch Learning

- For large datasets, batch learning is computationally intractable because it is not possible to store entire dataset into memory at once.
- Since it updates weights less frequently, it is computationally faster.

Batch and Online Learning

Online Learning

- In the on-line method of supervised learning, adjustments to the synaptic weights of the multilayer perceptron are performed on an example-by-example basis. The cost function to be minimized is therefore the total instantaneous error energy for one training example.
- Due to noise gradient descent calculated for each of the training examples, online learning jumps here and there which makes it difficult to converge in global minima. Due to this behavior of online learning, it is also called stochastic method.

Batch and Online Learning

Online Learning

- Besides this, parallelization of online learning method is not possible.
- Since it updates weights example by example basis, it is computationally slower than batch learning. But, it is computationally tractable for the large training set because we do not need to store large dataset into memory at once.

Batch and Online Learning

Trade-off Between Batch and Online Learning

- If we look above discussion, we can conclude that Batch learning is better approach. But, it is computationally intractable for large datasets.
- On the other hand, online learning is computationally tractable for large datasets but it is difficult to converge in global minim.
- Thus, the better approach is to use mini-batch learning approach, where entire dataset is divided into batches of size 16 or 32 or 64 or 128 etc., on the basis of need and synaptic weights of the multilayer perceptron are updated *after the presentation of all the N examples in a mini-batch.*

Backpropagation Algorithm

- A popular method for the training of multilayer perceptrons is the back-propagation algorithm. The training proceeds in two phases:
  - **Forward Phase**: In this phase, the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output. Thus, in this phase, changes are confined to the activation potentials and outputs of the neurons in the network.

Backpropagation Algorithm

- **Backward Phase**: In this phase, an error signal is produced by comparing the output of the network with a desired response. The resulting error signal is propagated in backward direction through the network, again layer by layer. In this second phase, successive adjustments are made to the synaptic weights of the network

Backpropagation Algorithm

- The error signal at the output of neuron j at iteration n is given by

$$e_j(n) = d_j(n) - y_j(n)$$

where $d_j(n)$ is desired output and $y_j(n)$ is output produced by MLP

- The total instantaneous error energy E(n) for all the neurons in the output layer is therefore

$$E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

where C is the set of neurons in output layer

Backpropagation Algorithm

- Let N be the total number of training vectors (examples). Then the average squared error is:

$$E_{avg} = \frac{1}{N}\sum_{n=1}^{N} E(n)$$

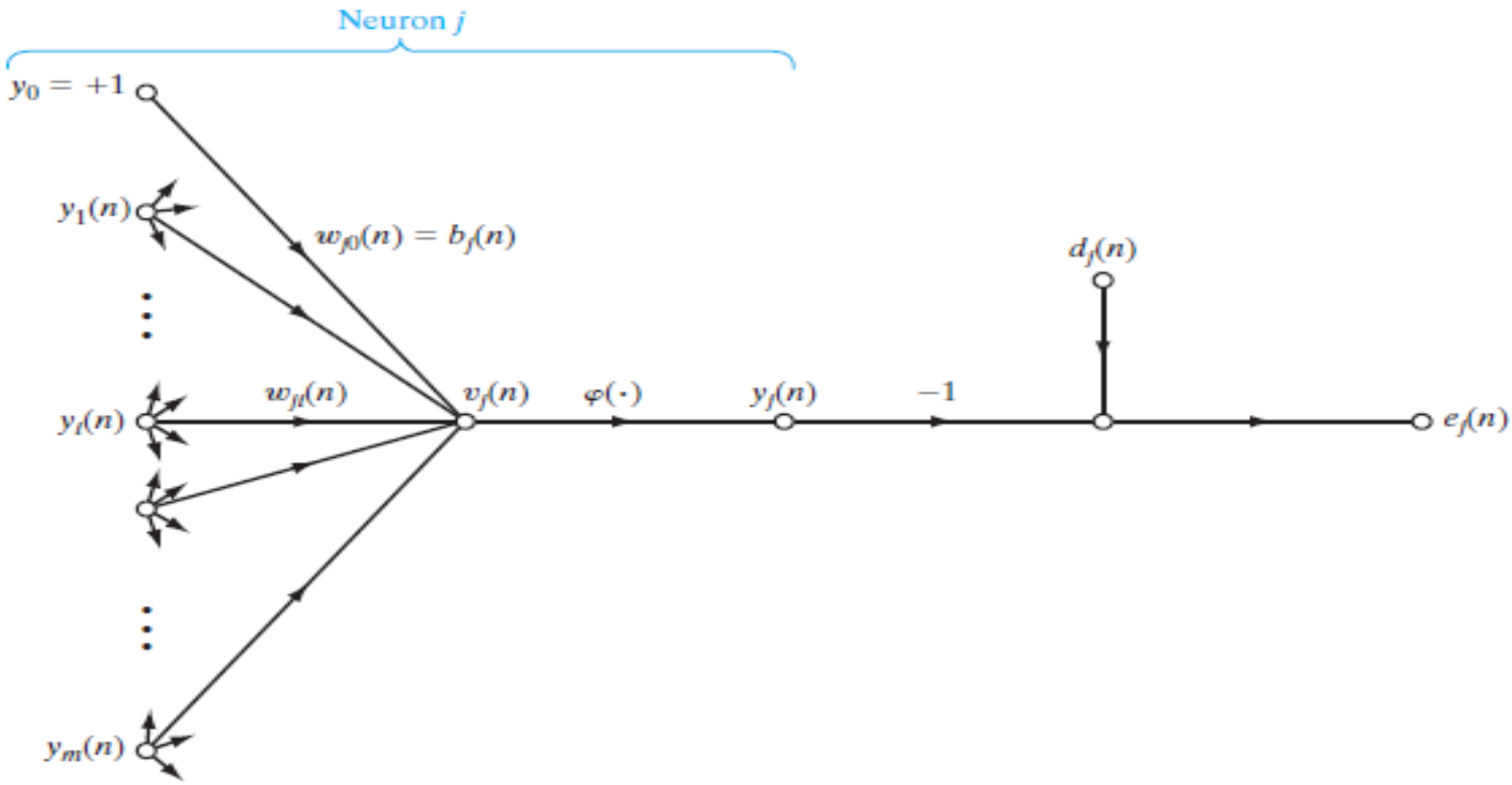
- Consider the neuron j as given in next slide. The local field $v_j(n)$ and output $y_j(n)$ of neuron j is given by:

$$v_j = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

$$y_j(n) = \varphi_j(v_j(n))$$

where y_i *is output of neuron i and* w_{ji} *is weight of link fron neuron i to j*

Backpropagation Algorithm



Backpropagation Algorithm

- The correction $\Delta w_{ji}(n)$ made to the weight is proportional to the partial derivative $\partial \mathrm{E(n)}/\partial w_{ji}(n)$ of instantaneous error
- Using the chain rule of calculus, this gradient can be expressed as follows:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad (1)$$

- We can get following partial derivatives

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \qquad \frac{\partial e_j(n)}{\partial y_j(n)} = -1 \qquad \frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n))$$

Backpropagation Algorithm

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

- Putting all partial derivatives in equation 1, we get

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \qquad (2)$$

- The correction applied to the weight is defined by

$$\Delta w_{ji} = -\alpha \frac{\partial E(n)}{\partial w_{ji}(n)} \qquad (3)$$

Backpropagation Algorithm

- Using equation (2), equation (3) can be written as

$$\Delta w_{ji} = \alpha e_j(n)\varphi_j'(v_j(n))y_i(n) \qquad (4)$$

- This can be written as

$$\Delta w_{ji} = \alpha \delta_j(n) y_i(n)$$

$$\text{where } \delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \qquad (5)$$

- The error term in equation 4 and 5 depends upon location of neuron in the MLP. There can be two possible scenarios.

Backpropagation Algorithm

Case I: Neuron j is output layer neuron

- The desired response $d_j(n)$ for the neuron j is directly available. Computation of the error $e_j(n)$ is straightforward in this case. We can use equation 4 or 5 to calculate weight update term.

Case II: Neuron j is hidden layer neuron

- There is no desired response available for neuron j. The error signal for a hidden neuron must be determined recursively in terms of the error signals of all neurons connected to it as below:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \qquad (6)$$

Backpropagation Algorithm

Algorithm

1. Initialize all weights and biases in *network*
2. While terminating condition is not satisfied
3. for each training tuple **X** in *D*
4. for each input layer unit *j:* $y_j = x_j$; // output of an input unit is its actual input value
5. for each hidden or output layer unit *j*

   compute the net input of unit j with respect to the previous layer, $v_j = \sum_i w_{ji} y_i$

   compute the output of each unit j

$$y_j = \frac{1}{1+e^{-v_j}}$$

Learning in ANN

Algorithm

6. for each unit j in the output layer

$$\delta_j = e_j \varphi_j'(v_j) = \varphi_j'(v_j)(d_j - y_j)$$

7. for each unit j in the hidden layers, from the last to the first hidden layer

$$\delta_j = \varphi_j'(v_j) \sum_k \delta_k w_{kj}$$

8. for each weight w_{ji} in *network*

$$\Delta w_{ji} = \alpha \delta_j y_i$$

$$w_{ji} = w_{ji} + \Delta w_{ji} \qquad \textit{//Weight Update}$$

Backpropagation Algorithm

- Backpropagation algorithm needs to compute $\varphi_j^{'}(v_j)$ to compute δ_j . This depends upon choice of activation function.

Case I: Logistic Activation Function

$$\varphi(x) = y = \frac{1}{1+e^{-x}} \Rightarrow \varphi'(x) = y(1-y)$$

If we consider general sigmoid function, then

$$\varphi(x) = y = \frac{1}{1+e^{-ax}} \Rightarrow \varphi'(x) = ay(1-y)$$

Note: Compute Derivatives of Logistic activation function.

Backpropagation Algorithm

Case II: Hyperbolic Tangent Function

$$\varphi(x) = y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \Rightarrow \varphi'(x) = (1+y)(1-y)$$

If we consider general Tanh function, then

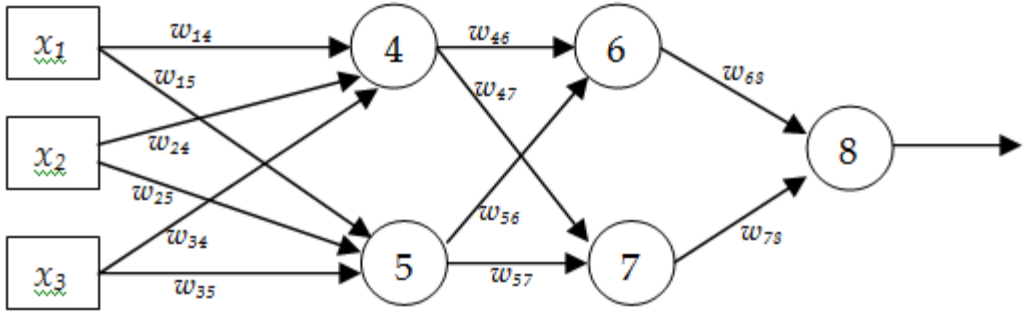
$$\varphi(x) = y = a\tanh(bx) \Rightarrow \varphi'(x) = \frac{b}{a}(a+y)(a-y)$$

Note: Compute Derivatives of Hyperbolic Function.

Backpropagation Algorithm

Example

- Consider a MLP given below. Let the learning rate be 1. The initial weights of the network are given in the table below. Assume that first training tuple is (1, 0, 1) and its target output is 1. Calculate weight updates by using back-propagation algorithm. Assume $\varphi(x) = \frac{1}{1+e^{-x}}$.



w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{47}	w_{56}	w_{57}	w_{68}	w_{78}
0.6	0.4	0.2	-0.3	0.7	-0.6	0.4	0.7	0.1	0.8	0.2	0.5

Backpropagation Algorithm

Solution

Forward Pass

$v_4=1*0.6+0*0.2+1*0.7=1.3$ $y_4=1/1+e^{-1.3}=0.786$

$v_5=1*0.4+0*(-0.3)+1*(-0.6)=-0.2$ $y_5=1/1+e^{0.2}=0.45$

$v_6=0.786*0.4+0.45*0.1=0.36$ $y_6=1/1+e^{-0.36}=0.59$

$v_7=0.786*0.7+0.45*0.8=0.91$ $y_7=1/1+e^{-0.91}=0.71$

$v_8=0.59*0.2+0.71*0.5=0.47$ $y_8=1/1+e^{-0.47}=0.61$

Backpropagation Algorithm

Solution

Backward Pass

$$\delta_8 = y_8(1-y_8)(d_8-y_8) = 0.61*(1-0.61)*(1-0.61) = 0.093$$

$$\delta_7 = y_7(1-y_7)\delta_8 w_{87} = 0.71*(1-0.71)*0.093*0.5 = 0.0096$$

$$\delta_6 = y_6(1-y_6)\delta_8 w_{86} = 0.59*(1-0.59)*0.093*0.2 = 0.0045$$

$$\delta_5 = y_5(1-y_5)(\delta_7 w_{75} + \delta_6 w_{65})$$

$$= 0.45*(1-0.45)*(0.0096*0.8 + 0.0045*0.1) = 0.002$$

$$\delta_4 = y_4(1-y_4)(\delta_7 w_{74} + \delta_6 w_{64})$$

$$= 0.786*(1-0.786)*(0.0096*0.7 + 0.0045*0.4) = 0.0014$$

Backpropagation Algorithm

Solution

Update Weights

$$w_{14} = w_{14} + 1 * \delta_4 * y_1 = 0.6 + 1 * 0.0014 * 1 = ?$$

$$w_{15} = w_{15} + 1 * \delta_5 * y_1 = ? \qquad w_{24} = w_{24} + 1 * \delta_4 * y_2 = ?$$

$$w_{25} = w_{25} + 1 * \delta_5 * y_2 = ? \qquad w_{34} = w_{34} + 1 * \delta_4 * y_3 = ?$$

$$w_{35} = w_{35} + 1 * \delta_5 * y_3 = ? \qquad w_{46} = w_{46} + 1 * \delta_6 * y_4 = ?$$

$$w_{47} = w_{47} + 1 * \delta_7 * y_4 = ? \qquad w_{56} = w_{56} + 1 * \delta_6 * y_5 = ?$$

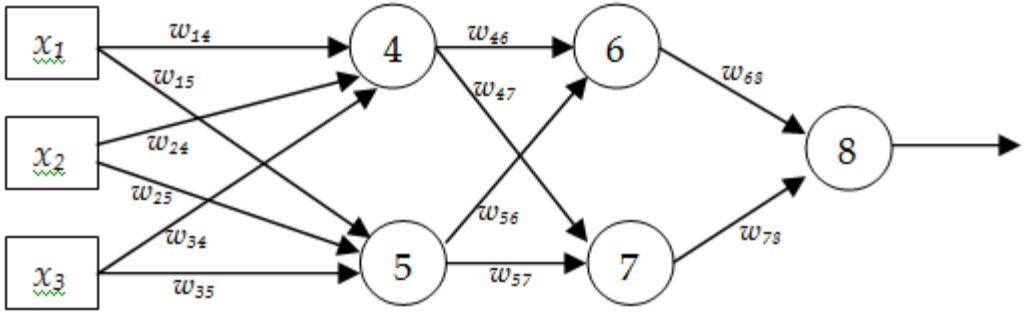
$$w_{57} = w_{57} + 1 * \delta_7 * y_5 = ? \qquad w_{68} = w_{68} + 1 * \delta_8 * y_6 = ?$$

$$w_{78} = w_{78} + 1 * \delta_8 * y_7 = ?$$

Backpropagation Algorithm

HW

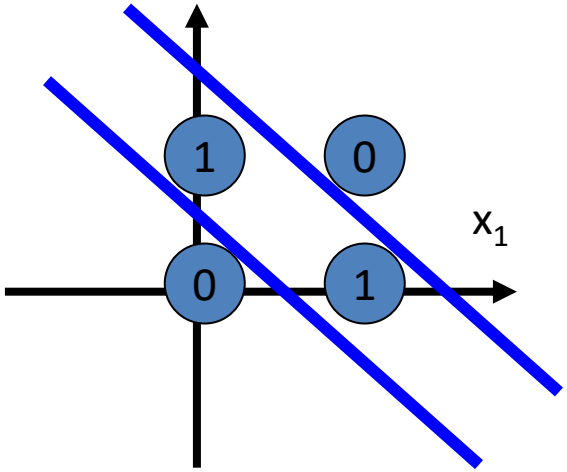
- Consider a MLP given below. Let the learning rate be 1. The initial weights of the network are given in the table below. Assume that first training tuple is (1, 0, 1) and its target output is 1. Calculate weight updates by using back-propagation algorithm. Assume $\varphi(x) = y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$



w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{47}	w_{56}	w_{57}	w_{68}	w_{78}
0.6	0.4	0.2	-0.3	0.7	-0.6	0.4	0.7	0.1	0.8	0.2	0.5

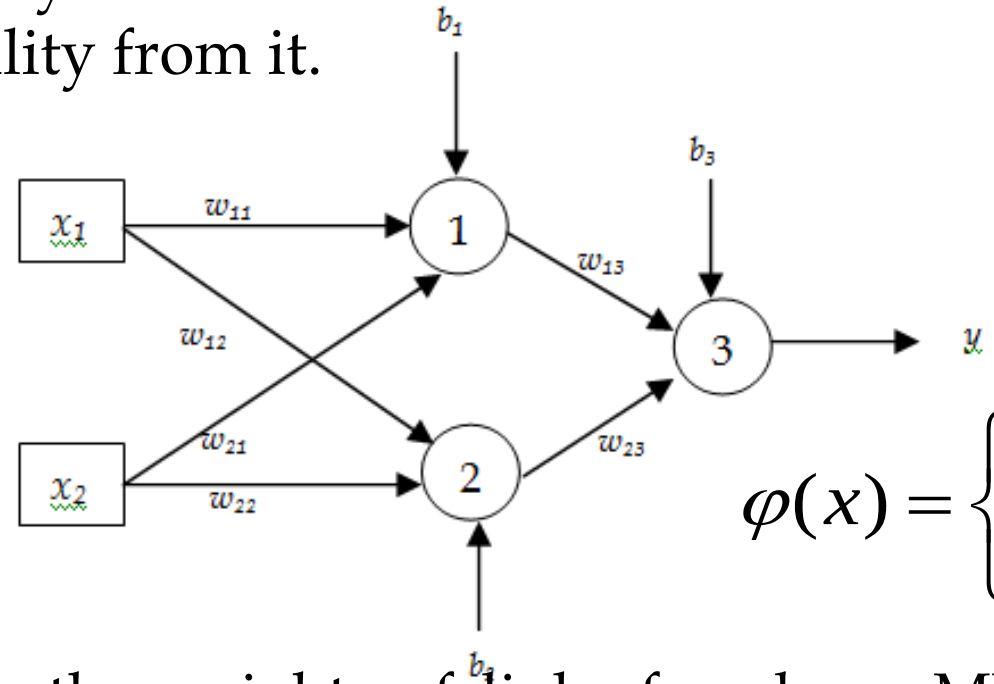
XOR Problem

- We know that single layer perceptron can be trained to classify patterns that is linearly separable.
- However, XOR function is non-linearly separable as shown in the figure given below. Thus, we can not train single layer perceptron for XOR function. This, problem is called XOR Problem.



XOR Problem

- We need to use multilayer perceptron that contains one hidden layer with two neurons to achieve XOR functionality from it.



$$\varphi(x) = \begin{cases} 1 & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases}$$

- Determine the weights of links for above MLP such that MLP functions as XOR. Assume that threshold activation function (as shown above) is used in each of the neuron.

XOR Problem

- We can have following realization of AND, OR, and NOR gates using perceptron.

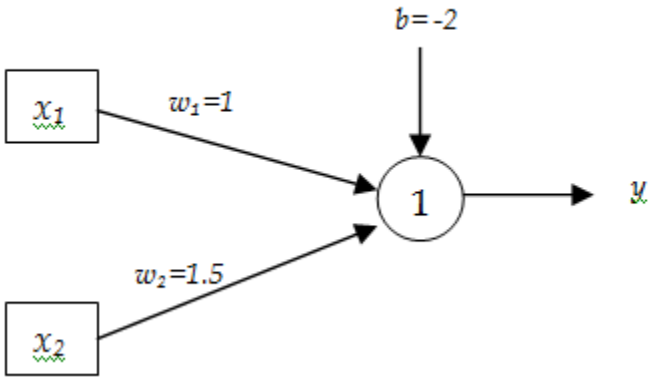


Fig: AND Function

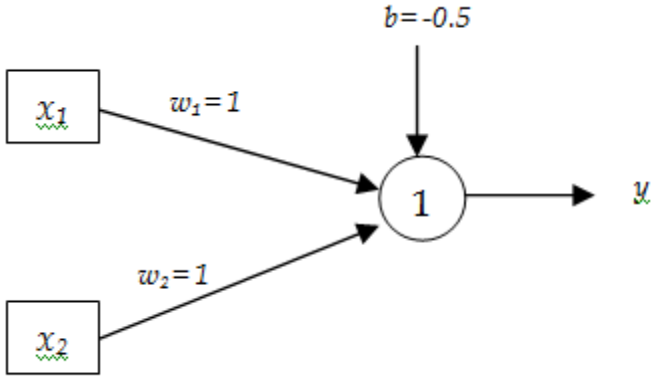


Fig: OR Function

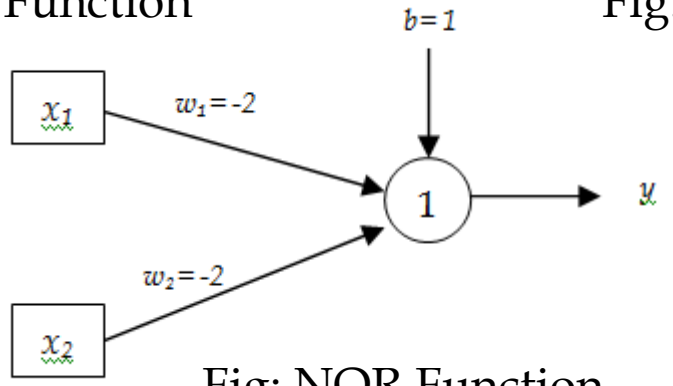


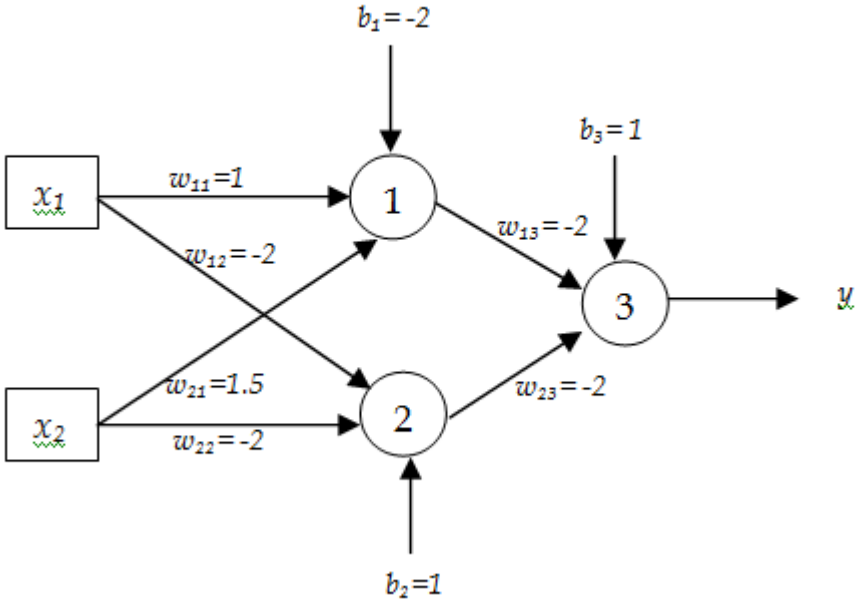
Fig: NOR Function

XOR Problem

- XOR function can be represented in terms of function of AND and NOR functions as below.

  $NOR(AND(x_1,x_2),NOR(x_1,x_2))$

- Thus, Weights of MLP that acts as XOR function can be set as below



XOR Problem

CW

- Realize NOT and NAND function using perceptron.
- Can we realize XNOR function using perceptron? If yes, realize it using perceptron. Otherwise, realize XNOR function using MLP.

Heuristics For making The Backpropagation Algorithm Perform Better

- There are methods that significantly improves the backpropagation algorithm's performance, as described below.

1. **Stochastic vs Batch Update:** As mentioned previously, the stochastic (sequential or online) mode of back-propagation learning is computationally faster than the batch mode for large training datasets. However, there can be difficulty in convergence in case of stochastic update. Therefore, better choice is to use mini-batch update strategy.

Heuristics For making The Backpropagation Algorithm Perform Better

2. **Maximizing Information Content:** Two ways of maximizing information content are:
   - Use an example that results in the largest training error.
   - Use an example that is radically different from all those previously used.

- In pattern-classification tasks using sequential back-propagation learning, a simple and commonly used technique is to randomize (i.e., shuffle) the order in which the examples are presented to the multilayer perceptron from one epoch to the next.
- Ideally, the randomization ensure that successive examples in an epoch presented to the network rarely belong to the same class.

Heuristics For making The Backpropagation Algorithm Perform Better

3. **Activation Function:** Sigmoid function and hyperbolic tangent function should be choice for activation function in MLP.

- Sigmoid activation function has following properties:

$$\varphi(-x) = \frac{1}{1+e^{x}} = \frac{e^{-x}}{1+e^{-x}} = \frac{1+e^{-x}-1}{1+e^{-x}} = 1 - \frac{1}{1+e^{-x}} = 1 - \varphi(x)$$

$$\Rightarrow x \to \varphi(x) \qquad \varphi(-x) = -\varphi(x)$$

- The function that holds above relation is called odd function. Thus, sigmoid functions are odd functions.

Heuristics For making The Backpropagation Algorithm Perform Better

- We know that general form of hyperbolic tangent function is.

$$\varphi(x) = a\tanh(bx) = a * \frac{e^{bx} - e^{-bx}}{e^{bx} + e^{-bx}}$$

- According to LeCun (1993), a=1.7159 and b=2/3=0.67 suitable values for the above activation function.
- Thus hyperbolic tangent function has following useful properties:

$\varphi(1) = 1$ $\varphi(-1) = -1$ $\varphi(0) = \text{ab}$

Heuristics For making The Backpropagation Algorithm Perform Better

4. **Target Values:** It is important that the target values (desired response) be chosen within the range of the sigmoid activation function. For the limiting value *a and -a, we set:*

$$d_j = a - \varepsilon \qquad\qquad d_j = -a + \varepsilon$$

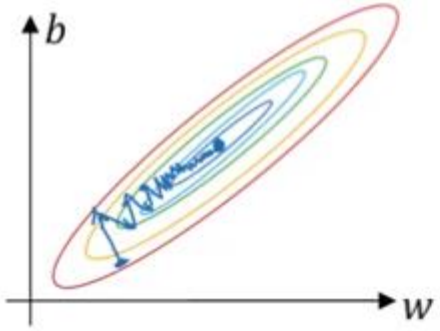
Heuristics For making The Backpropagation Algorithm Perform Better

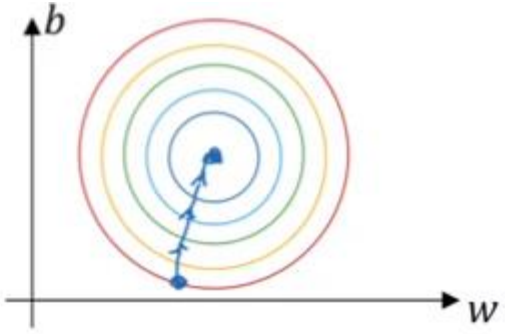
5. **Normalizing the Inputs:** Each input variable should be *preprocessed so that its* mean value is close to zero. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values. It is required only when features have different ranges.

- For example, consider a data set containing two features, x1 and x2.Where x1 ranges from 0–100, while x2 ranges from 10,000–200,000. While making prediction, the attribute x2 will intrinsically influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor.

Heuristics For making The Backpropagation Algorithm Perform Better

- Input features are of different scale (Unnormalized) leads to elongated loss function but normalized input features leads to more symmetric loss function.





- Thus with normalized input features gradient descent converges much faster.

Heuristics For making The Backpropagation Algorithm Perform Better

Ways of Normalization

- Min-max Normalization/Scaling

$$x = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- Mean Normalization/Scaling

$$x = \frac{x - \mu}{\max(x) - \min(x)}$$

- Standard Normalization/Scaling

$$x = \frac{x - \mu}{\sigma} \qquad \textit{where } \mu \textit{ is mean}$$

Heuristics For making The Backpropagation Algorithm Perform Better

Example

- Normalize the following data using Min-Max, mean, and Standard Normalization.

Salary	Age
45000	42
32000	26
58000	48
37000	32

Heuristics For making The Backpropagation Algorithm Perform Better

Solution

Min-Max Normalization

Salary	Age
0.50	0.73
0.00	0.00
1.00	1.00
0.19	0.27

Mean Normalization

Salary	Age
0.08	0.23
-0.42	-0.50
0.58	0.50
-0.23	-0.23

Standard Normalization

Salary	Age
0.20	0.59
-1.12	-1.29
1.53	1.29
-0.61	-0.59

Note:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}},$$

where μhemean and N is number of data points

Heuristics For making The Backpropagation Algorithm Perform Better

6. **Weight Initialization: G**ood choice for the initial values of the synaptic weights and thresholds of the network can be of tremendous help in a successful network design.

- If weights are initialized with very high values the activation field(weighted sum of inputs) becomes significantly higher and if an activation function like sigmoid is applied, the function maps its value near to 1 where the slope of gradient changes slowly and learning takes a lot of time. This problem is often referred to as the vanishing gradient.

Heuristics For making The Backpropagation Algorithm Perform Better

- If weights are initialized with low values activation field gets mapped near to 0. In such case, the back-propagation algorithm may operate on a very flat area around the origin of the error surface.
- For these reasons, the use of both large and small values for initializing the synaptic weights should be avoided.
- Thus, better weight initialization method is to initialize weights randomly such that weights have uniform distribution with zero mean and a variance equal to the reciprocal of the number of synaptic connections of a neuron. i.e.

$$\sigma_w = m^{-1/2} = \sqrt{\frac{1}{m}}$$

where m is number of synnaptic connections or links

Heuristics For making The Backpropagation Algorithm Perform Better

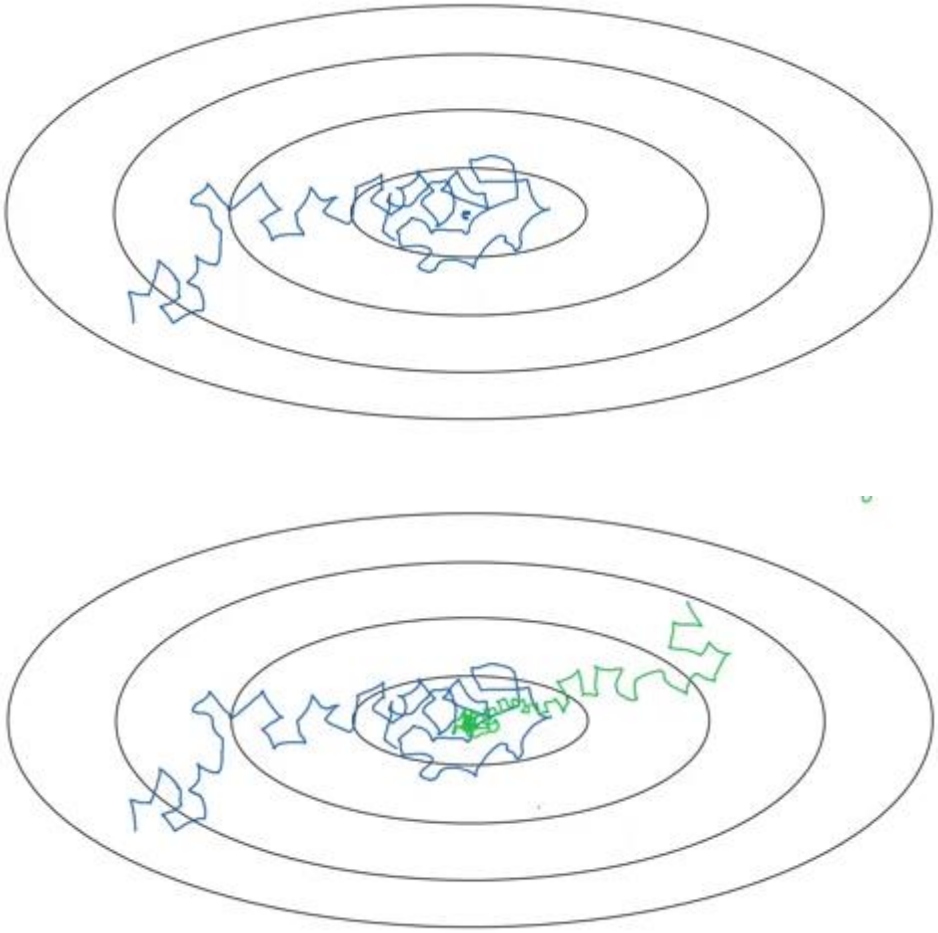
7. **Learning Rate:** Learning rate should be between 0 and 1. It must be sufficiently small. Large value of learning rate may cause difficulty is convergence of algorithm.

- The last layers usually have larger local gradients than the layers at the front end of the network. Hence, the learning-rate parameter should be assigned a smaller value in the last layers than in the front layers of the multilayer perceptron.
- Neurons with many inputs should have a smaller learning-rate parameter than neurons with few inputs so as to maintain a similar learning time for all neurons in the network.

Heuristics For making The Backpropagation Algorithm Perform Better

- The learning rate needs to be reduced over time. The reason for this is that in the beginning, the bigger steps can be taken but as the cost function reaches to its minimum, learning rate needs to be decreased.
- If the learning rate is same or large, the algorithm may never converge and will always oscillate around the minima (BLUE). If learning rate is decreased over time or sufficiently small, it will converge to minima in the later epochs (GREEN).

Heuristics For making The Backpropagation Algorithm Perform Better



Heuristics For making The Backpropagation Algorithm Perform Better

8. **Learning From Hints:** Learning from examples is the process of taking input-output examples of an unknown function f and infering an implementation of f. Learning from hints allows for general information about f to be used instead of just input-output examples.

Backpropagation and Differentiation

- The **Backpropagation** (BP) is an algorithm used to train the Neural Network. The goal of BP is to optimize the loss function by finding good/optimal weights values.
- The BP works by calculating the **gradient** of the loss function and propagated back to all weights. The BP uses **Automatic differentiation (AD)** to calculate the gradient automatically.
- AD is a technique to numerically evaluate the derivative of a function using an algorithm.

Backpropagation and Differentiation

- Suppose that we have to compute derivative of following function *at x_1=1.5 and x_2=0.5.*

$$y = \left[\sin(x_1 / x_2) + x_1 / x_2 - \exp(x_2)\right] \times \left[x_1 / x_2 - \exp(x_2)\right]$$

- Computation of functional value

$$v_a = x_1 = 1.5 \qquad v_b = x_2 = 0.5$$

$$v_1 = v_a / v_b = 3.0 \qquad v_2 = \sin(v_1) = 0.1411$$

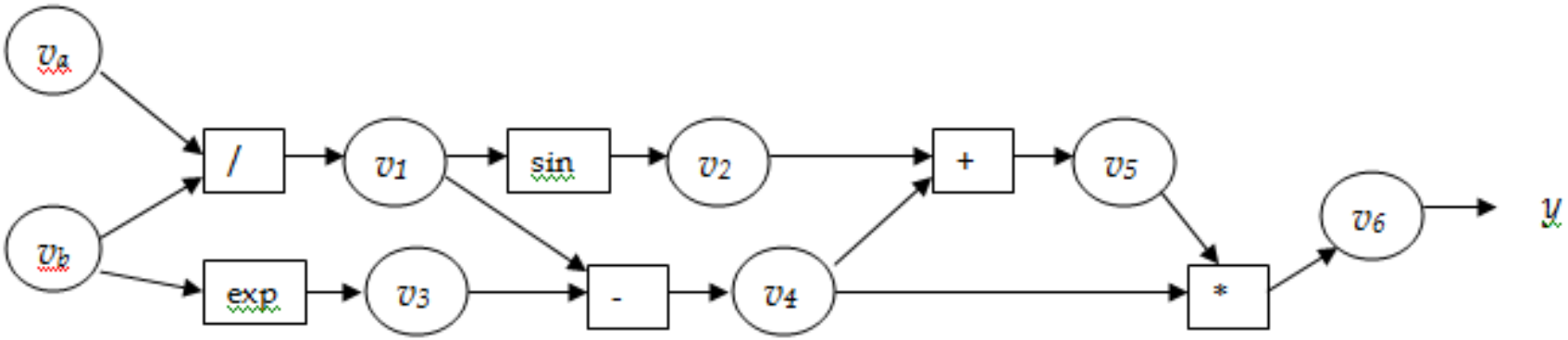
$$v_3 = \exp(v_b) = 1.6487 \qquad v_4 = v_1 - v_3 = 1.4513$$

$$v_5 = v_2 + v_4 = 1.4924 \qquad v_6 = v_5 \times v_4 = 2.0167$$

$$y = v_6 = 2.0167$$

Backpropagation and Differentiation

- We have following computation graph of above computation.



Backpropagation and Differentiation

- In case of forward mode AD we can compute derivatives as we go. Suppose we have to compute partial derivative w.r.t. x_1.

$$v_a = x_1 = 1.5 \qquad v_a' = 1$$

$$v_b = x_2 = 0.5 \qquad v_b' = 0$$

$$v_1 = v_a / v_b = 3.0 \qquad v_1' = (v_b v_a' - v_a v_b') / v_b^2 = 2.0$$

$$v_2 = \sin(v_1) = 0.1411 \qquad v_2' = \cos(v_1) v_1' = -1.98$$

$$v_3 = \exp(v_b) = 1.6487 \qquad v_3' = \exp(v_b).v_b' = 0$$

$$v_4 = v_1 - v_3 = 1.4513 \qquad v_4' = v_1' - v_3' = 2.0$$

$$v_5 = v_2 + v_4 = 1.4924 \qquad v_5' = v_2' + v_4' = 0.02$$

$$v_6 = v_5 \times v_4 = 2.0167 \qquad v_6' = y' = v_5 v_4' + v_4 v_5' = 3.0118$$

Backpropagation and Differentiation

- Reverse mode AD walks backwards through the computation graph and computes derivatives of the output with respect to the local variable. This quantity is sometimes called an adjoint variable $v_i' = \frac{\partial y}{\partial v_i}$
- Mathematically, the adjoint is computed by looking at adjoints of the children of vertex v_i on the graph:

$$v_i' = \sum_{j:Children\ of\ i} v_j' \frac{\partial v_j}{\partial v_i}$$

Backpropagation and Differentiation

- In case of reverse AD we can compute derivatives by moving backward from output to input layer of computation graph. This procedure is also followed by BP to compute gradient.

$$v_6 = v_5 \times v_4 = 2.0167 \qquad v_6' = y' = 1$$

$$v_5 = v_2 + v_4 = 1.4924 \qquad v_5' = v_6' \frac{\partial v_6}{\partial v_5} = v_6' v_4 = 1.3513$$

$$v_4 = v_1 - v_3 = 1.4513 \qquad v_4' = v_5' \frac{\partial v_5}{\partial v_4} + v_6' \frac{\partial v_6}{\partial v_4} = v_5' + v_6' v_5 = 2.8437$$

$$v_3 = \exp(v_b) = 1.6487 \qquad v_3' = v_4' \frac{\partial v_4}{\partial v_3} = -v_4' = -2.8437$$

$$v_2 = \sin(v_1) = 0.1411 \qquad v_2' = v_5' \frac{\partial v_5}{\partial v_2} = v_5' = 1.3513$$

$$v_1 = v_a / v_b = 3.0 \qquad v_1' = v_2' \frac{\partial v_2}{\partial v_1} + v_4' \frac{\partial v_4}{\partial v_1} = v_2' \cos(v_1) + v_4' = 1.5059$$

$$v_b = x_2 = 0.5 \qquad v_b' = v_1' \frac{\partial v_1}{\partial v_b} + v_3' \frac{\partial v_3}{\partial v_b} = v_3' v_3 - v_1' v_1 / v_b = -13.7239$$

$$v_a = x_1 = 1.5 \qquad v_a' = v_1' \frac{\partial v_1}{\partial v_a} = v_1' / v_b = 3.0118$$

Jacobian and Hessian

- The Jacobian is a matrix of all the first-order partial derivatives of a vector-valued function. In the neural network case, it is an N-by-W matrix, where N is the number of entries in our training set and W is the total number of parameters (weights) of our network.
- It can be generated by taking the partial derivatives of each output with respect to each weight.
- Each row of the Jacobian corresponds to a particular example in the training sample.

Jacobian and Hessian

$$J = \begin{bmatrix} \frac{\partial F(x_1, w)}{\partial w_1} & \cdots & \frac{\partial F(x_1, w)}{\partial w_W} \\ \vdots & \ddots & \vdots \\ \frac{\partial F(x_N, w)}{\partial w_1} & \cdots & \frac{\partial F(x_N, w)}{\partial w_W} \end{bmatrix}.$$

- While it is a good exercise to compute the gradient of a neural network with respect to a single parameter (e.g., a single weight), in practice this tends to be quite slow. This approach can't take advantage of vectorization. Thus, for vectorized implementation of backpropagation algorithm , it is necessary to use Jacobian.

Jacobian and Hessian

- Hessian of a cost function is square matrix whose elements are second order partial derivative of the cost function with respect to each weight in the MLP.

$$H = \frac{\partial^2 f}{\partial w^2} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 w_2} & \cdots\cdots & \frac{\partial^2 f}{\partial w_1 w_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial w_n w_1} & \frac{\partial^2 f}{\partial w_n w_2} & \cdots\cdots & \frac{\partial^2 f}{\partial w_n^2} \end{bmatrix}$$

Role of Hessian in Online Learning

- The Hessian plays an important role in the study of neural networks:
  - The eigenvalues of the Hessian have a profound influence on the dynamics of back-propagation learning.
  - The inverse of the Hessian provides a basis for pruning insignificant synaptic weights from a MLP.
  - The Hessian is basic to the formulation of second-order optimization methods as an alternative to back-propagation learning.

Role of Hessian in Online Learning

- Typically, the Hessian of the error surface pertaining to a multilayer perceptron trained with the back-propagation algorithm has the following composition of eigenvalues:
  - a small number of small eigenvalues,
  - a large number of medium-sized eigenvalues, and
  - a small number of large eigenvalues.

Role of Hessian in Online Learning

- It has been shown that optimal learning rate is inverse of largest eigenvalue of H.

$$\alpha_{opt} = \frac{1}{\lambda_{\max}}$$

- Learning time of backpropagation is sensitive to the ratio $\lambda_{\max}/\lambda_{\min}$, where $\lambda_{\max}$ and $\lambda_{\min}$ are largest and non-negative smallest eigenvalues of H.
- For the inputs with non-zero mean, ratio $\lambda_{\max}/\lambda_{\min}$ is large. This is the reason behind normalizing inputs such that mean becomes zero.

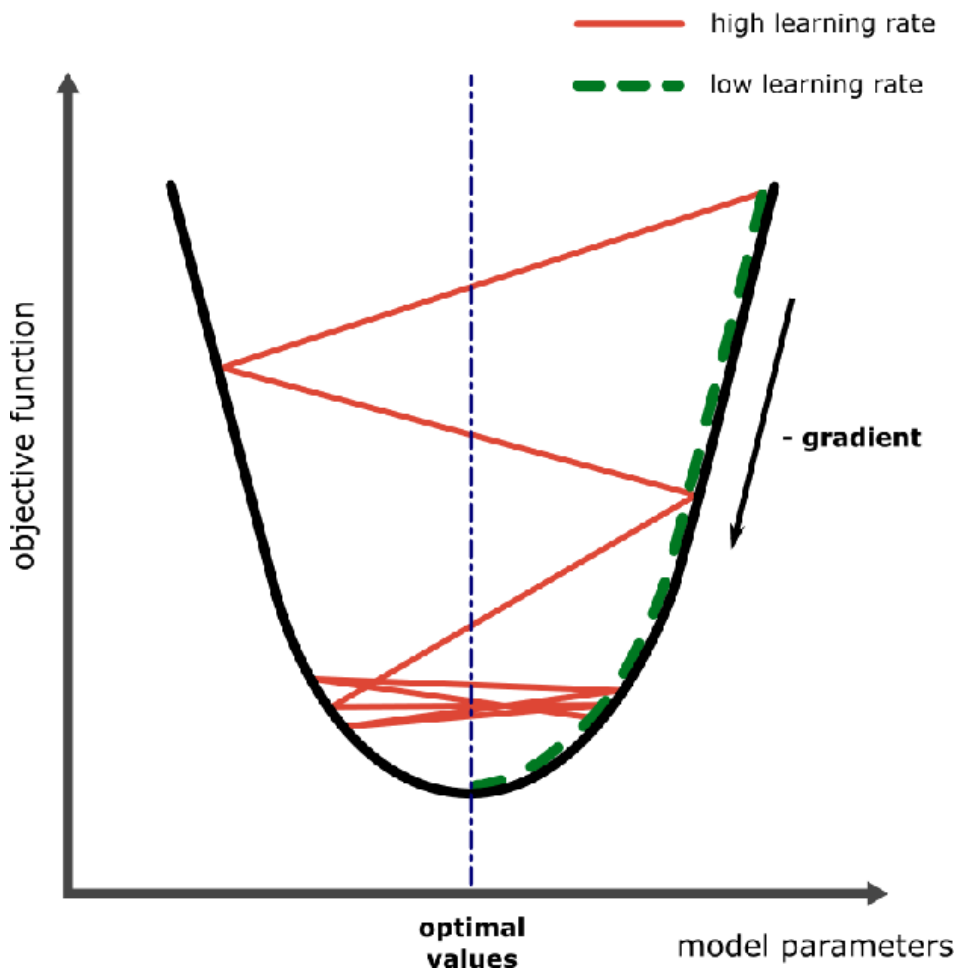
Role of Hessian in Online Learning

- Normalized inputs are only for first hidden layer of MLP. Rest of the hidden layers as well as output layer of MLP takes output of previous layer as input.
- If logistic activation function is used it squashes output in the range [0,1]. Hence, it is not likely to have zero-mean. On the other hand, if hyperbolic tangent activation function is used it squashes output in the range [-1,1]. Hence, it is likely to have zero-mean.
- This is the reason behind faster convergence of learning algorithm when Tanh activation function is used.

Annealing and Adaptive Control of Learning Rate

- One of the key hyper parameters to set in order to train a neural network is the *learning rate*.
- If learning rate is set too low, training will progress very slowly as we are making very tiny updates to the weights in our network.
- However, if learning rate is set too high, it can cause undesirable divergent behavior in our loss function.

Annealing and Adaptive Control of Learning Rate



Annealing and Adaptive Control of Learning Rate

Learning Rate Annealing

- One approach of using learning rate value properly is to start with large value of learning rate and reduce that the value of learning rate according to some predefined schedule. This approach is called learning rate annealing.
- Three commonly used learning rate annealing techniques or learning rate schedules are:
  - Time Based Decay
  - Step Decay
  - Exponential Decay

Annealing and Adaptive Control of Learning Rate

Time Based Decay

- Mathematical formulation of time-based learning rate decay is given below:

$$\alpha = \frac{\alpha_0}{1 + decay\_rate \times \#epoch}$$

Example

Let α_0=0.2 Decay-Rate=1

For Epoch=1 α=0.2/2=0.1

For Epoch=2 α=0.2/3=0.67

For Epoch=4 α=0.05=0.2/4=0.5

So on

Annealing and Adaptive Control of Learning Rate

Step Decay

- Mathematical formulation of step decay of learning rate is given below. A typical way is to to drop the learning rate by half every 10 epochs.

$$\alpha = \alpha_0 \times drop\_rate^{(\lfloor epoch\#/epochs\_drop \rfloor)}$$

Example

Let α_0=0.2 drop_rate=0.5 epochs_drop=5

For Epoch=5 $\alpha=0.2 \times 0.5^{5/5}=0.1$

For Epoch=10 $\alpha= 0.2 \times 0.5^{10/5}=0.05$

For Epoch=15 $\alpha= 0.2 \times 0.5^{15/5}=0.025$

So on

Annealing and Adaptive Control of Learning Rate

Exponential Decay

- Mathematical formulation of exponential decay of learning rate is given below.

$$\alpha = \alpha_0 e^{(-k \times \#epoch)}$$

Example

Let α_0=0.2 k=0.1

For Epoch=1 α=0.181

For Epoch=2 α=0.162

For Epoch=3 α=0.148

So on

Annealing and Adaptive Control of Learning Rate

Adaptive Control of Learning Rate

- Learning rate annealing techniques reduces learning rate equally for all parameters and applies the same learning rate with all parameters.
- However, in many practical situations, we need to change value of different parameters by different extent.
- Thus, the better approach is to use the techniques that gives learning rate a chance to adapt.

Annealing and Adaptive Control of Learning Rate

Adaptive Control of Learning Rate

- The main idea behind adaptive learning rate is to use larger learning rate for updating parameters that are modified with small scale in past and use smaller learning rate for updating parameters that are modified with large scale in the past.
- Four commonly used learning rate adaptation techniques are:
  - Adagrad
  - Adadelta
  - RMSProp
  - Adam

Annealing and Adaptive Control of Learning Rate

Adagrad

- In Adagrad, the variable v, called cache, keeps track of sum of the squared gradients, which in turn is used to normalize the parameter update.

$$v_t = v_{t-1} + dw_t^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \varepsilon}} dw_t$$

Annealing and Adaptive Control of Learning Rate

Adagrad

- The effect of above equations is that Weights receiving high gradients will have their effective learning rate reduced. On the other hand, weights receiving small gradients will have their effective learning rate increased.

Annealing and Adaptive Control of Learning Rate

Adadelta

- Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size W.

Annealing and Adaptive Control of Learning Rate

RMSProp

- Adagrad decays the learning rate very aggressively (as the denominator grows). As a result, after a while, parameters receiving larger gradients will start receiving very small updates because of the decayed learning rate.
- To avoid this why not decay the denominator and prevent its rapid growth.
- RMSProp stands for root mean squared propagation and avoids monotonically increasing denominator of Adagrad.

Annealing and Adaptive Control of Learning Rate

RMSProp

- The central idea of RMSProp is keep the moving average of the squared gradients for each weight. And then divide the learning rate by root mean square of the squared gradients.

$$v_t = \beta v_{t-1} + (1-\beta) dw_t^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \in}} dw_t$$

Annealing and Adaptive Control of Learning Rate

Adam

- Adam update can be considered as RMSprop with momentum. In place of raw and noisy gradient vector *dw*, weighted average of gradients are used.

  Weighted average of gradients

  $$v_t = \beta_1 v_{t-1} + (1-\beta_1) dw_t$$

  Weighted average of squared gradients

  $$s_t = \beta_2 s_{t-1} + (1-\beta_2) dw_t^2$$

Annealing and Adaptive Control of Learning Rate

Adam

Bias Correction

$$v_t = \frac{v_t}{(1-\beta_1^t)} \qquad S_t = \frac{S_t}{(1-\beta_2^t)}$$

Finally, update weights as below:

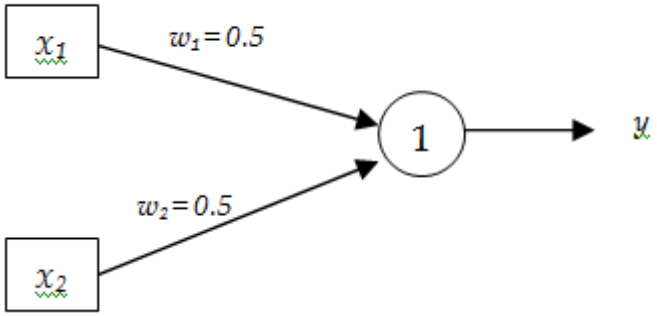
$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{s_t + \in}} v_t$$

Annealing and Adaptive Control of Learning Rate

Example

- Consider following simple ANN with logistic activation function. Assume that $\alpha = 0.2 \quad \beta = 0.8 \quad \beta_1 = 0.8 \quad \beta_2 = 0.8$. Calculate weight updates for the given training Sample using (1) Adagrad (2) RMSProp and (3) Adam

x_1	x_2	t
1	0.6	0.8
0.5	0.5	0.6
0.2	0.8	0.3



Annealing and Adaptive Control of Learning Rate

Solution

For Adagrad

For Training example: (1,0.6,0.8)

$$y_{in} = 0.5*1 + 0.5*0.6 = 0.8$$

$$y = \frac{1}{1+\exp(-y_{in})} = 0.6899$$

$$dw_1 = (y-t)\times y\times(1-y)\times x_1 = -0.02354$$

$$dw_2 = (y-t)\times y\times(1-y)\times x_2 = -0.01412$$

$$v_1 = -0.02354^2 = 0.000554$$

$$v_2 = -0.01412^2 = 0.000199$$

$$w_1 = w_1 - \frac{0.2}{\sqrt{0.000554}}\times(-0.02354) = 0.7$$

$$w_2 = w_2 - \frac{0.2}{\sqrt{0.000199}}\times(-0.01412) = 0.7$$

Annealing and Adaptive Control of Learning Rate

Solution

For Adagrad

For Training example: (0.5,0.5,0.6)

$$y_{in} = 0.7 * 0.5 + 0.7 * 0.5 = 0.7$$

$$y = \frac{1}{1 + \exp(-y_{in})} = 0.6682$$

$$dw_1 = (y - t) \times y \times (1 - y) \times x_1 = 0.007559$$

$$dw_2 = (y - t) \times y \times (1 - y) \times x_2 = 0.007559$$

$$v_1 = 0.000554 + 0.007559^2 = 0.000611$$

$$v_2 = 0.000199 + 0.007559^2 = 0.000257$$

$$w_1 = w_1 - \frac{0.2}{\sqrt{0.000611}} \times 0.007559 = 0.639$$

$$w_2 = w_2 - \frac{0.2}{\sqrt{0.000257}} \times 0.007559 = 0.606$$

Annealing and Adaptive Control of Learning Rate

Solution

For Adagrad

For Training example:

(0.2,0.8,0.3)

$$y_{in} =$$

$$y = \frac{1}{1+\exp(-y_{in})} =$$

$$dw_1 = (y-t)\times y\times(1-y)\times x_1 =$$

$$dw_2 = (y-t)\times y\times(1-y)\times x_2 =$$

$$v_1 =$$

$$v_2 =$$

$$w_1 = ??? = 0.531$$

$$w_2 = ??? = 0.412$$

Annealing and Adaptive Control of Learning Rate

Solution

For RMSProp

For Training example:(1,0.6,0.8)

$$y_{in} = 0.5*1 + 0.5*0.6 = 0.8$$

$$y = \frac{1}{1+\exp(-y_{in})} = 0.6899$$

$$dw_1 = (y-t)\times y\times(1-y)\times x_1 = -0.02354$$

$$dw_2 = (y-t)\times y\times(1-y)\times x_2 = -0.01412$$

$$v_1 = 0.8*0 + (1-0.8)*(-0.02354)^2 = 0.000111$$

$$v_2 = 0.8*0 + (1-0.8)*(-0.01412)^2 = 0.0000399$$

$$w_1 = w_1 - \frac{0.2}{\sqrt{0.000111}}\times(-0.02354) = 0.947$$

$$w_2 = w_2 - \frac{0.2}{\sqrt{.0000399}}\times(-0.01412) = 4.971$$

Annealing and Adaptive Control of Learning Rate

Solution

For RMSProp

For Training example:(0.5,0.5,0.6)

$$y_{in} = 0.947 * 0.5 + 4.971 * 0.5 =$$

$$y = \frac{1}{1 + \exp(-y_{in})} = ??$$

$$dw_1 = (y - t) \times y \times (1 - y) \times x_1 = ???$$

$$dw_2 = (y - t) \times y \times (1 - y) \times x_2 = ???$$

$$v_1 = 0.8 * 0.000111 + (1 - 0.8) * (dw_1)^2 = ????$$

$$v_2 = 0.8 * 0.0000399 + (1 - 0.8) * (dw_2)^2 = ???$$

$$w_1 = w_1 - \frac{0.2}{\sqrt{v_1}} \times (dw_1) = ???$$

$$w_2 = w_2 - \frac{0.2}{\sqrt{v_2}} \times (dw_2) = ???$$

Annealing and Adaptive Control of Learning Rate

Solution

For RMSProp

For Training example:(0.2,0.8,0.3)

$$y_{in} = ???$$

$$y = \frac{1}{1+\exp(-y_{in})} = ??$$

$$dw_1 = (y-t)\times y\times(1-y)\times x_1 = ???$$

$$dw_2 = (y-t)\times y\times(1-y)\times x_2 = ???$$

$$v_1 = 0.8 * ??? + (1-0.8) * (dw_1)^2 = ????$$

$$v_2 = 0.8 * ??? + (1-0.8) * (dw_2)^2 = ???$$

$$w_1 = w_1 - \frac{0.2}{\sqrt{v_1}} \times (dw_1) = ???$$

$$w_2 = w_2 - \frac{0.2}{\sqrt{v_2}} \times (dw_2) = ???$$

Annealing and Adaptive Control of Learning Rate

Solution

For Adam

??????---try yourself

Generalization

- We train neural networks on some training data and expect that it will perform well on the test data. This feature of neural network is called generalization.
- This essentially means how good our model is at learning from the given data and applying the learnt information elsewhere.
- Overfitting is the one of the main reason that hinders generalization capability of neural network.

Generalization

- When a neural network learns too much from the training dataset, it will become specialized on the training dataset but will be unable to make accurate prediction for test data. Such neural network are said to be overfitted.
- Generalization is influenced by three factors:
  1. The size of the training sample and how representative the training sample is
  2. The architecture of the neural network, and
  3. The physical complexity of the problem

Generalization

- Clearly, we have no control over the 3rd factor. In the context of the first two factors, we may view the issue of generalization from two different perspectives:
  - If the architecture of the network is fixed and the issue to be resolved is that of determining the size of the training sample needed for a good generalization to occur.
  - If the size of the training sample is fixed, and the issue of interest is that of determining the best architecture of network for achieving good generalization.

Function approximation

- Function approximation is a technique for estimating an unknown underlying function using historical or available observations from the domain.
- Artificial neural networks can be trained to approximate a function.
- In supervised learning, a dataset is comprised of inputs and outputs, and the supervised learning algorithm learns how to best map examples of inputs to examples of outputs.

Function approximation

- This mapping is governed by a mathematical function, called the **mapping function**, and a supervised learning algorithm seeks to best approximate this function.
- Neural networks are an example of a supervised learning algorithm and hence can be used to approximate the function represented by training data.
- Neural networks achieves this by calculating the error between the predicted outputs and the actual outputs and minimizing this error during the training.

Function approximation

- ANNs can be viewed as Universal approximator. In theory, they can be used to approximate any function.
- For example consider the following training data that can be approximated by a simple univariate function.

x	y
1	2
2	5
3	10
4	17

Function approximation

- A neural network can be trained to approximate the function that can map above inputs to outputs and that ANN can then be used to predict output for the value that is not present in above training data.
- Universal approximation theorem states that "ANN can *compute any function, no matter how complicated it is*".
- No matter what the function, there is guaranteed to be a neural network so that for every possible input x, the value f(x) (or some close approximation) is output from the network. This statement holds even if the function has many inputs, $y=f(x_1,\ldots,x_m)$, and many outputs.

Cross-Validation

- In machine learning, datasets are divided into three sets: Training Set, Validation Set, and Test Set. Split ratio of dataset into three sets is not fix. It depends upon size of the dataset. If we have very large dataset split ratio can be 98:1:1. Otherwise, we can opt for split ratio 80:10:10 or 70:15:15 so on.
- Training set is used to train neural network and update weights on the basis of error. Validation set is used to compute prediction error using weights resulted from training. If the error is higher than a user-defined threshold then the whole training-validation epoch is repeated.

Cross-Validation

- The testing set is then used to measure the performance of the network. This is the data that was never used throughout the training and validation phase.
- *Cross-validation is a re-sampling procedure used to evaluate machine learning models on a limited data sample*.
- k-Fold Cross-Validation (Muti-Fold Cross-Validation) is widely used for cross validation. If we choose k=10, it becomes 10-fold cross validation.

Cross-Validation

- In this approach, the original sample is randomly partitioned into k equal sized subsamples. Out of the k subsamples, a single subsample is used as the testing set, and the remaining $k - 1$ subsamples are used as training+validation data.
- The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the testing data. The k results can then be averaged to produce a single estimation.

Cross-Validation

- The advantage of this method is that all observations are used for both training and testing, and each observation is used for testing exactly once. 10-fold cross-validation is commonly used, but in general k remains an unfixed parameter.
- For example, setting $k = 2$ results in 2-fold cross-validation. In 2-fold cross-validation, we randomly shuffle the dataset into two sets d_0 and d_1, so that both sets are equal size. We then train on d_0 and test on d_1, followed by training on d_1 and testing on d_0.

Cross-Validation

- When $k = n$ (the number of observations), k-fold cross-validation is equivalent to leave-one-out cross-validation. In this case, *n - 1 examples are used to train the model, and the model is validated by testing* it on the example that is left out. This approach is suitable when we have small number of training example.
- To make the cross-validation procedure concrete, let's look at a worked example. Imagine we have a data sample with 6 observations:

  {(1,3),(2,5),(3,7),(4,9),(5,11),(6,13)}
- Let k=3. That means we will shuffle the data and then split the data into 3 groups.

Cross-Validation

- Let k=3. That means we will shuffle the data and then split the data into 3 groups.

  Fold 1= {(1,3),(5,11)}

  Fold 2= {(6,13),(3,7)}

  Fold 3= {(2,5),(4,9)}
- Three models are trained and evaluated with each fold given a chance to be the held out test set. For example:
  - **Model 1**: Trained on Fold 1 + Fold 2, Tested on Fold 3
  - **Model 2**: Trained on Fold 2 + Fold 3, Tested on Fold 1
  - **Model3**: Trained on Fold 1 + Fold 3, Tested on Fold 2

Cross-Validation

- When $k = n$ (the number of observations), k-fold cross-validation is equivalent to leave-one-out cross-validation. In this case, *n - 1 examples are used to train the model, and the model is validated by testing* it on the example that is left out. This approach is suitable when we have small number of training example.
- Holdout method is alternative to K-Fold cross validation. In this approach, we randomly divide data points to two sets d_0 and d_1, usually called the training set and the test set, respectively. Usually, the test set is smaller than the training set. We then train on d_0 and test on d_1.

Cross-Validation

- In typical cross-validation, results of multiple runs of model-testing are averaged together. In contrast, the holdout method, involves a single run.
- It should be used with caution because without such averaging of multiple runs, one may achieve highly misleading results. Predictive accuracy will tend to be unstable since it will not be smoothed out by multiple iterations.

Complexity Regularization and Network Pruning

Complexity Regularization

- Regularization refers to the set of techniques employed while training neural network so that it can generalize over data that hasn't seen before.
- These techniques includes adding restrictions on the parameter values, adding extra terms in the objective function etc.
- While training a neural network model, it must learn the weights of the network (i.e. the model parameters).
- The longer we train the network, the more specialized the weights will become to the training data and may lead to overfitted model.

Complexity Regularization and Network Pruning

Complexity Regularization

- The weights will grow in size in order to handle the specifics of the examples seen in the training data.
- Large weights make the network unstable. Minor variation or statistical noise on the inputs will result in large differences in the output.
- Neural network with large values of weights represents very complex model. Using too complex model is one of the main source of Overfitting.
- Thus, learning algorithm should be updated to encourage the network toward using small weights.

Complexity Regularization and Network Pruning

Complexity Regularization

- One way to do this is to change the calculation of loss used in the optimization of the network to also consider the size of the weights as below.

  $L(x) = L(x) + \lambda \sum w^2$, *where* λ *is regularization parameter*

- This strategy enforces the learning algorithm to keep the weights as small as possible. Therefore, this procedure of regularization is called **weight decay procedure or weight regularization.**
- Larger weights result in a larger penalty, in the form of a larger loss score. The optimization algorithm will then push the model to have smaller weights.

Complexity Regularization and Network Pruning

Complexity Regularization

- The Lambda Hyperparameter has a value between 0 (no penalty) and 1 (full penalty).
- If the penalty is too strong, the model will underestimate the weights and underfit the training data. If the penalty is too weak, the model will be allowed to overfit the training data.
- So, In practical applications of complexity regularization, the regularization parameter is assigned a value somewhere between these two limiting cases.

Complexity Regularization and Network Pruning

Network Pruning

- Pruning is a technique in neural network that aids in the development of smaller and more efficient neural networks. It's a model optimization technique that involves eliminating unnecessary values in the weight matrix. This results in compressed neural networks that run faster.
- There are two hypotheses on neural networks that motivate pruning:

Complexity Regularization and Network Pruning

Network Pruning

- Most networks are actually over-parameterized and we can safely assume that deleting redundant weights won't harm performance too much.
- The second one is that not all weights contribute equally to the output prediction. We can assume that lower magnitude weights will have a lower importance to the network.
- Optimal brain surgeon (OBS) is the one of the widely used technique for neural network pruning. This method is. is based on second order derivative of output w.r.t. weight (i.,e. Hessian).

Complexity Regularization and Network Pruning

OBS Working

1) Train the multilayer perceptron to a minimum in mean square error
2) Compute the inverse Hessian (H^{-1})
3) Eliminate weights with small saliencies

$$S_i = \frac{w_i^2}{2[H^{-1}]_{i,i}}, \Delta w = -\frac{w_i^2}{[H^{-1}]_{i,i}} H^{-1} 1_i, \text{ where } 1_i \text{ is unit vvector whose all elements are zero and ith element is } 1.$$

4) Update the Weights of neural network using above equation.
5) Repeat until no more weights can be deleted without a large increase in the mean-square error.

Virtues and Limitations of Backpropagation

- **Connectionism:** The back-propagation algorithm is an example of a connectionist paradigm that relies on local computations. In this paradigm computation is performed by the neuron is influenced solely by those neurons that are in physical contact with it.
- **Feature Detector:** Hidden neurons of a multilayer perceptron trained with the backpropagation algorithm play a critical role as feature detectors.
- **Function Approximation:** A multilayer perceptron trained with the back-propagation algorithm can be viewed as a nested sigmoid function and it can be used to approximate any function.

Virtue and Limitations of Backpropagation

- **Computational Efficiency**: Computational complexity of backpropagation algorithm is O(W), where W is number of synaptic weights. This means computational complexity of the algorithm is linear. Thus, we can say that backpropagation algorithm is computationally efficient.
- **Sensitivity Analysis:** The sensitivity is defined as the mathematical expectation of the output errors of the MLP due to weight perturbations. Sensitivity can be analyzed by computing partial derivatives. Since, backpropagation is efficient in computing partial derivatives, we can say that it can efficiently compute sensitivity of input-output mapping in neural networks.

Virtue and Limitations of Backpropagation

- **Robustness:** Backpropagation algorithm minimizes the maximum energy gain from the disturbances to the estimation errors. Therefore it is robust algorithm.
- **Convergence:** Backpropagation algorithm is stochastic in nature. It has a tendency to zigzag its way about the true direction to a minimum on the error surface. Therefore, it tends to converge slowly. For fast convergence we have to use learning rate annealing techniques or adaptable learning rate techniques.

Virtue and Limitations of Backpropagation

- **Local Minima:** If error surface has local minima then backpropagation algorithm may trapped in local minima and may be unable to find global minima.
- **Scaling:** With the increased number of hidden layers and neurons backpropagation algorithm suffer from vanishing/exploding gradient problem that slows down learning rate.

Supervised Learning Viewed as Optimization Problem

- The learning problem is formulated in terms of the minimization of a loss function, *f*. It is a function that measures the performance of a neural network on a data set.
- The loss function is, in general, composed of an error and a regularization terms. The error term evaluates how a neural network fits the data set. The regularization term is used to prevent Overfitting by controlling the complexity of the neural network.

Supervised Learning Viewed as Optimization Problem

- The loss function depends on the adaptive parameters (biases and synaptic weights) in the neural network.
- The learning problem for neural networks is formulated as searching of a parameter vector $w*$ at which the loss function f takes a minimum value.
- The necessary condition states that if the neural network is at a minimum of the loss function, then the gradient is the zero vector.

Supervised Learning Viewed as Optimization Problem

- The loss function is, in general, a non-linear function of the parameters. As a consequence, it is not possible to find closed training algorithms for the minima.
- Instead, we consider a search through the parameter space consisting of a succession of steps. At each step, the loss will decrease by adjusting the neural network parameters.
- We can use backpropagation or Newton's methods for finding optimal values of weight that leads to minimum loss.

Supervised Learning Viewed as Optimization Problem

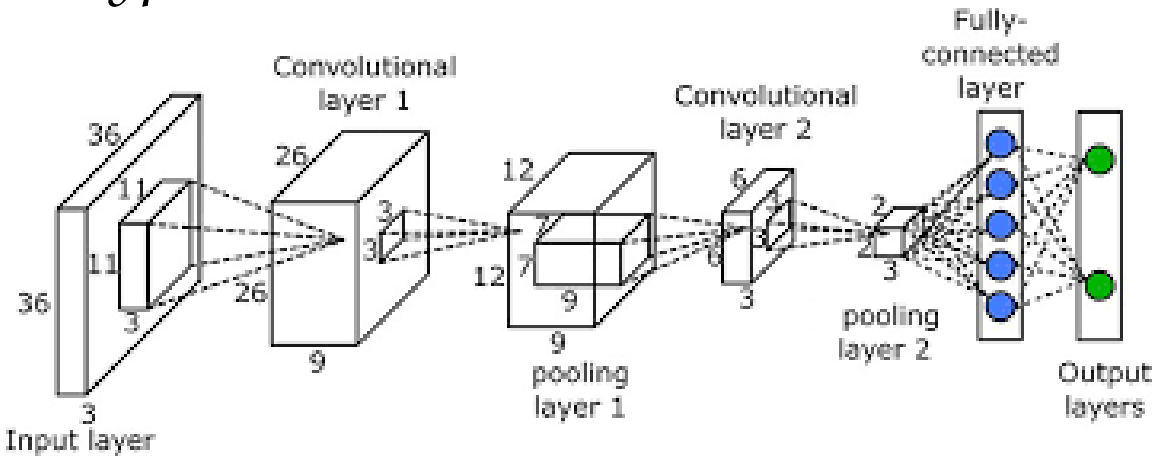
- Newton's method is second order optimization method that uses following equation for updating weights in each iteration.

$$w_{i+1} = w_i - \alpha(H_i^{-1} \times g_i), \textit{where } H_i \textit{ is hessian and } g_i \textit{ is gradient}$$

- Newton's method is not preferred algorithm for MLP training due to following reasons.
  - It requires computation of inverse of hessian and hence is computationally expensive.
  - Inverse of hessian may not exist always.
  - Convergence is not guaranteed.

Convolutional Neural Network(CNN)

- CNN is a type of multilayer neural network specially designed for detecting, recognizing, and classifying images.
- Every CNN consist following three types of layers: *Convolution Layer, Pooling layer, and Fully-connected Layer. Typical a*rchitecture of CNN is shown below:



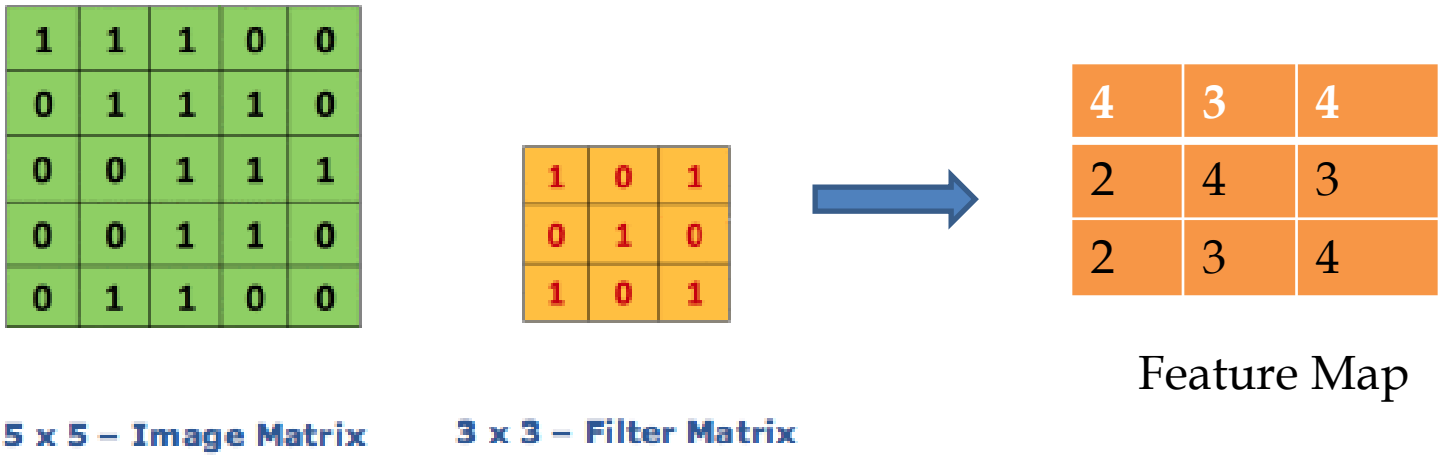
Convolutional Neural Network(CNN)

Convolution Layer

- Convolution layer extracts features from an input image. It preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel and produces an image with convolved features.
- Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below.
- The matrix formed by sliding the filter over the image and computing the dot product is called Feature Map or Convolved Feature.

Convolutional Neural Network(CNN)

Convolution Layer



- Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

Convolutional Neural Network(CNN)

Pooling Layer

- Spatial Pooling or simply Pooling reduces the dimensionality of each feature map but retains the most important information.
- It is also called sub-sampling or down-sampling. Spatial Pooling can be of different types: *Max, Average, Sum etc.*
- In case of Max Pooling, we define a spatial neighborhood (for example, a 3×3 window) and take the largest element from the rectified feature map within that window.

Convolutional Neural Network(CNN)

Pooling Layer

- Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

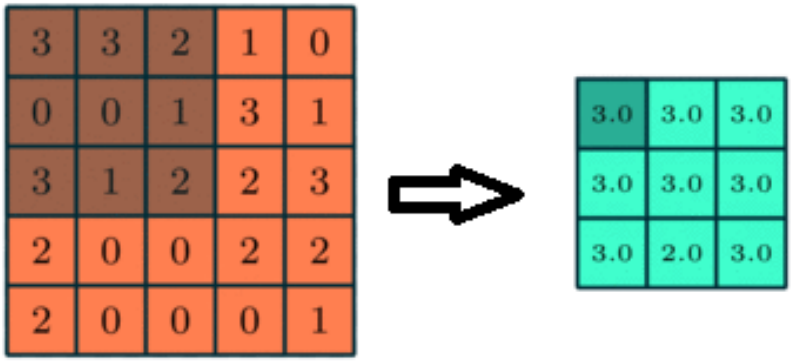


Figure: 3x3 Max Pooling over 5x5 convolved feature

Convolutional Neural Network(CNN)

Fully Connected Layer

- The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer. The term *"Fully Connected"* implies that every neuron in the previous layer is connected to every neuron on the next layer.
- The output from the Convolutional and Pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

Convolutional Neural Network(CNN)

Fully Connected Layer

- The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one.
- The Fully-Connected layer learns non-linear function in output produced by Convolutional and Pooling Layer.
- We should flatten the image into a column vector. The flattened output is fed to the Fully Connected Layer and backpropagation can be applied to train the network.

Convolutional Neural Network(CNN)

Example

Consider following 6x6 image and 3x3 filter as below. Compute feature using the feature and pooled feature map using 2x2 window. Use Max pooling.

255	200	120	89	0	180
210	230	170	165	87	76
49	120	115	125	165	140
20	35	32	42	37	78
55	65	75	45	35	69
190	180	160	150	155	165

1	1	0
0	1	0
0	1	1

Non-Linear Filtering

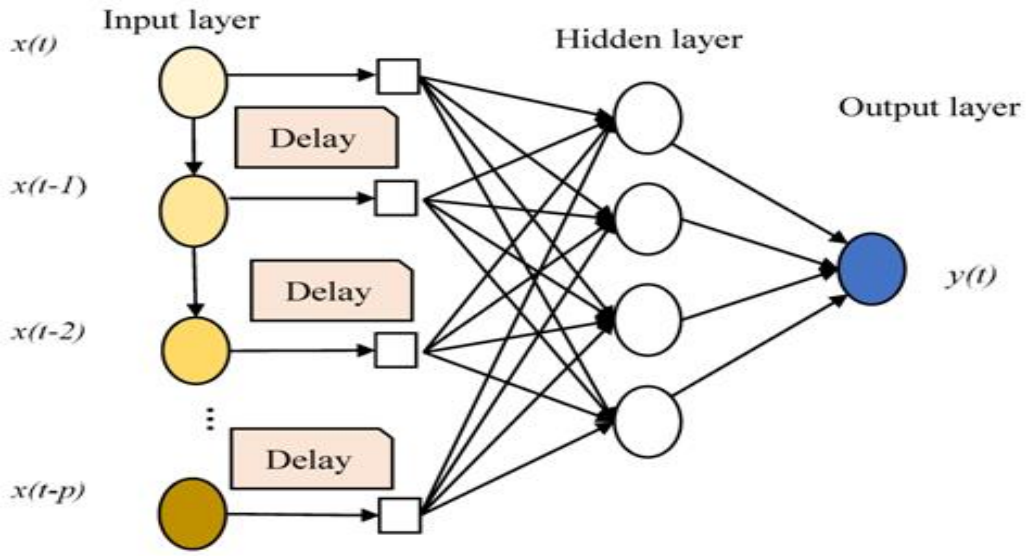
- Linear filtering is the filtering method in which the value of output is linear combinations of the neighboring data. it can be done with convolution or Fourier multiplication.
- MLPs are static neural network that can only be used for recognizing structural or spatial patterns in the data. Thus, MLP acts as linear filters.
- A nonlinear filter is a filter that cannot be done with convolution or Fourier multiplication.

Non-Linear Filtering

- Non-linear filtering is required for recognizing temporal patterns, Where requirement is to process patterns that evolve over time.
- In such systems response at a particular instant of time depends not only on the present value of the input signal, but also on past values.
- Thus, dynamic neural networks or neural networks with short-term memory is required for recognizing temporal patterns.
- Recurrent neural networks (RNN) and time-delay neural networks (TDNN) are dynamic neural networks.

Non-Linear Filtering

- Time delay neural network (TDNN) is a *Multilayer Feed-Forward Artificial Neural Network Architecture* whose purpose is to model context at each layer of the network.
- Typical architecture of TDNN is given below:



Large Scale Vs Small Scale Learning Problems

- A supervised-learning problem is said to be of a small-scale kind when the *size of the training sample is the constraint imposed* on the learning process.
- A supervised-learning problem is said to be of a large-scale kind when the *computing time is the active budget constraint imposed on the learning process.*
- There are three variables available to the designer of a learning machine:
  - the number of training examples- *N*
  - the permissible size *network functions-K*
  - the computational error- e

Large Scale Vs Small Scale Learning Problems

- With the active budget constraint being the number of examples, the design options in Small-Scale learning problems are as follows:
  - Reduce the estimation error by making *N as large as the budget permits.*
  - Adjust the size K to the extent deemed to be reasonable.
- **However, w**ith the active budget constraint being the training time, the designing Large-Scale learning problem is difficult. This is because if we want to reduce e, we must increase N or K or both. But, increasing N or K or both increases training time.