Understanding Multilayer Perceptron

Srishti

July 17, 2018

Multilayer Perceptron

Neural Network

A massive parallel distributed processor made up of simple processing units that has a neural propensity for storing experimental knowledge and making it available for use.

Mathematically, $f : \mathbb{R}^m \rightarrow \mathbb{R}^p$

Rosenblatt's Perceptron: Single layer neural network.

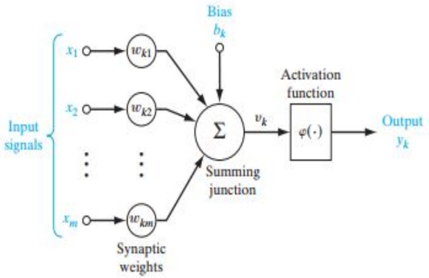


Figure: Signal flow graph of the perceptron

Image courtesy: *Neural Networks and Learning Machines, Simon O. Haykin*

Decision boundary: $\sum_{i=1}^{m} w_{ki}x_i + b_k = 0$

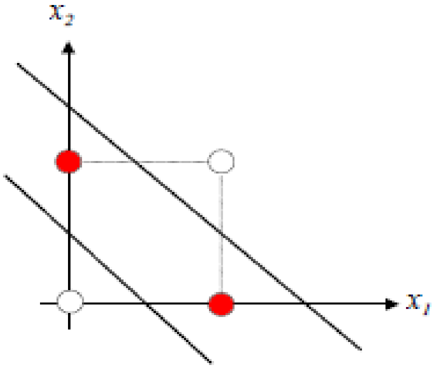


Figure: Non linearly seperable data

Image courtesy: *Google images*

Limitation: Unable to classify non linearly separable patterns.

Multilayer Perceptrons (MLP)

Neural network with one more hidden layers.

Structure Of MLP

Multilayer Perceptrons (MLP)

Neural network with one more hidden layers.

Structure Of MLP

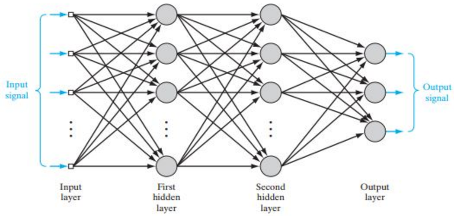


Figure: Architectural graph of a multilayer perceptron with two hidden layers

Image courtesy: *Neural Networks and Learning Machines, Simon O. Haykin*

Method for the training of MLP - Backpropogation-algorithm

Two phases of training:

- forward phase
- backward phase

Methods of Supervised Learning

Let, training sample be $\mathcal{T} = (\boldsymbol{x}(n), \boldsymbol{d}(n))_{n=1}^{N}$
where: $\boldsymbol{x} \in \mathbb{R}^m$
$\boldsymbol{d} \in \mathbb{R}^p$

$y_j(n)$: function signal produced at the output of neuron j in the output layer

Error signal produced at the output of neuron j:

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

where $d_j(n)$ is the ith element of desired response vector $\boldsymbol{d}(n)$

Instantaneous error energy of neuron j:

$$\mathcal{E}_j(n) = \frac{1}{2} e_j^2(n)$$

Total instantaneous energy of the whole network:

$$\begin{aligned}\mathcal{E}(n) &= \sum_{j \in C} \mathcal{E}_j(n) \\ &= \frac{1}{2} \sum_{j \in C} e_j^2(n)\end{aligned}$$

where C : includes all neurons in the output layer

Empirical risk :

$$\begin{aligned}\mathcal{E}_{av}(N) &= \frac{1}{N} \sum_{n=1}^{N} \mathcal{E}(n) \\ &= \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n)\end{aligned}$$

Two methods of supervised learning:

- Batch learning

Two methods of supervised learning:

- Batch learning
  Cost function:

$$\mathcal{E}_{av}(N) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n)$$
$$= \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in c} e_j^2(n)$$

Two methods of supervised learning:

- Batch learning
  Cost function:

$$\begin{aligned}\mathcal{E}_{av}(N) &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n)\\ &= \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in c} e_j^2(n)\end{aligned}$$

  Synaptic weights of MLP are adjusted after one epoch.
- On-line learning

Two methods of supervised learning:

- Batch learning
  Cost function:

$$\mathcal{E}_{av}(N) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n)$$
$$= \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in c} e_j^2(n)$$

  Synaptic weights of MLP are adjusted after one epoch.
- On-line learning
  Cost function:

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

Two methods of supervised learning:

- Batch learning
  Cost function:

$$\begin{aligned}\mathcal{E}_{av}(N) &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n) \\ &= \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in c} e_j^2(n)\end{aligned}$$

  Synaptic weights of MLP are adjusted after one epoch.
- On-line learning
  Cost function:

$$\begin{aligned}\mathcal{E}(n) &= \sum_{j\in C}\mathcal{E}_j(n) \\ &= \frac{1}{2}\sum_{j\in C} e_j^2(n)\end{aligned}$$

  Synaptic weights are adjusted on an example by example basis.

Two methods of supervised learning:

- Batch learning
  Cost function:

$$\begin{aligned}\mathcal{E}_{av}(N) &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n)\\ &= \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in c} e_j^2(n)\end{aligned}$$

  Synaptic weights of MLP are adjusted after one epoch.
- On-line learning
  Cost function:

$$\begin{aligned}\mathcal{E}(n) &= \sum_{j\in C}\mathcal{E}_j(n)\\ &= \frac{1}{2}\sum_{j\in C} e_j^2(n)\end{aligned}$$

  Synaptic weights are adjusted on an example by example basis.
  Weight adjustment are performed using gradient descent.

Back-Propagation Algorithm

Initialization: Choose synaptic weights and bias from uniform distribution whose mean is zero.

Iterate over each training set and do the following:

- Forward Computation
- Backward Computation
- Weight Updation Synaptic weights for n+1 iteration

$$w_{ij}^{(l)}(n+1) = w_{ij}^{(l)}(n) + \eta \Delta w_{ij}^{(l)}(n+1)$$

Forward Computation

Let, training example in the epoch be $\mathcal{T} = (\boldsymbol{x}(n), \boldsymbol{d}(n))$
where: $\boldsymbol{x}(n)$: input vector
$\boldsymbol{d}(n)$: desired response vector

At iteration n:
Induced local field for neuron j in layer l:

$$v_j^{(l)}(n) = \sum_i w_{ij}^{(l)}(n) y_i^{(l-1)}(n)$$

Output signal of neuron j in layer l:

$$y_j^{(l)} = \varphi((v_j)(n))$$

For neuron j in output layer: $y_j(n)^{(L)} = o_j(n)$
Error signal:

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

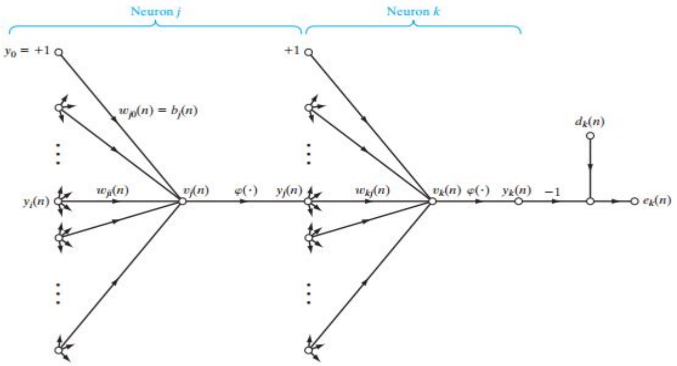


Figure: Signal flow graph highlighting the details of output neuron k

Image courtesy: *Neural Networks and Learning Machines, Simon O. Haykin*

Backward Computation

Applies correction $\Delta w_{ij}(n)$ to the synaptic weight $w_{ij}(n)$

$$\Delta w_{ij}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ij}(n)}$$

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ij}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ij}(n)}$$

For neuron k in output layer:

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ik}^{(L)}(n)} = -e_k^{(L)}(n)\varphi'_k(v_k^{(L)}(n))y_i^{(l-1)}(n)$$

For neuron j in hidden layer:

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ij}^{(l)}(n)} = \varphi_j'(v_j^{(l)}(n))y_i^{(l-1)}(n)\sum_k e_k^{(L)}(n)\varphi_j'(v_k^{(L)}(n))$$

In terms of local gradient:

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\phi_j'(v_j^{(L)}(n)), & \text{for neuron j in output layer L} \\ \phi_j'(v_j^{(l)}(n))\sum_k \delta_k^{(l+1)}(n)w_{jk}^{(l+1)}(n), & \text{for neuron j in hidden layer l} \end{cases}$$

Synaptic weights for $n+1$ iteration

$$w_{ij}^{(l)}(n+1) = w_{ij}^{(l)}(n) + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n)$$

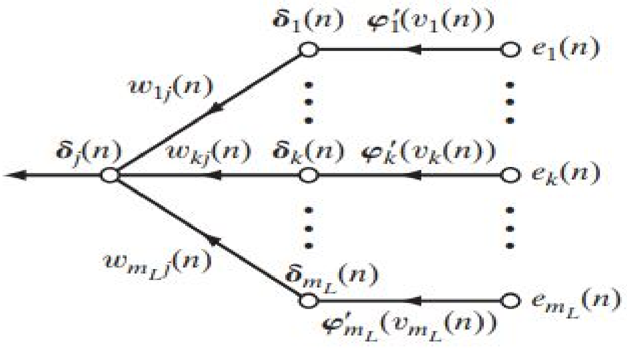


Figure: Signal-flow graph of a part of the adjoint system pertaining to backpropagation of error signals

Image courtesy: *Neural Networks and Learning Machines, Simon O. Haykin*

Case: Sigmoid activation function: $\frac{1}{1+e^{-x}}$

For neuron k in output layer:

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ik}^{(L)}(n)} = -e_k^{(L)}(n) o_k(n)(1 - o_k(n)) y_i^{(l-1)}(n)$$

For neuron j in hidden layer:

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ij}^{(l)}(n)} = y_j^{(l)}(1 - y_j^{(l)}) y_i^{(l-1)}(n) \sum_k e_k^{(L)}(n) o_k(n)(1 - o_k(n))$$

Experiment

Problem: Two class classification

Data Set: Training data - 2000 data samples

Test Data - 650 data samples

Cost Function: $\frac{1}{2}\sum_i (y_i - o_i)^2$

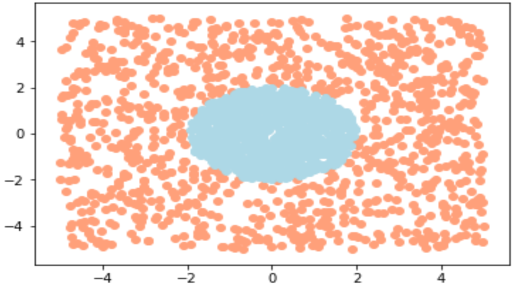


Figure: Training Dataset

Set up:

No of hidden neurons $= 88$

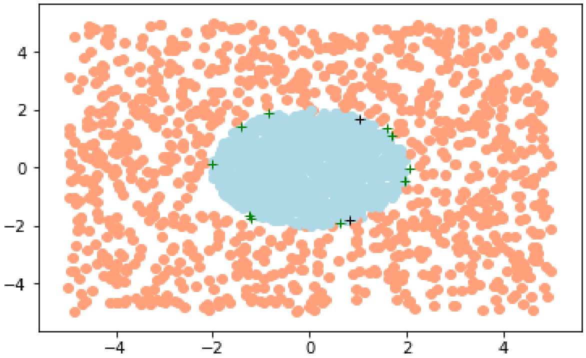
No of output layer neurons $= 1$

No of epochs $= 1000$

Learning rate $= 0.1$

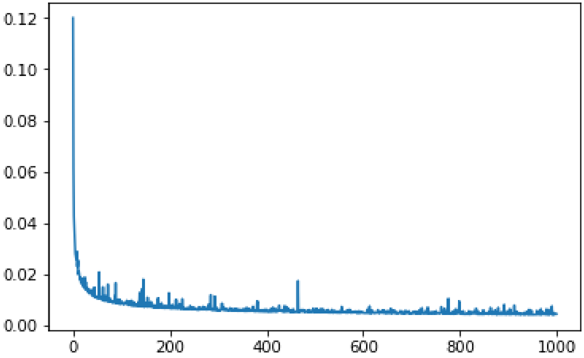
Performance

At epoch 1000:



Note: "+" denotes the misclassified points and "o" denotes the classified points.

Epoch error



Min epoch error :0.002

Test Data:

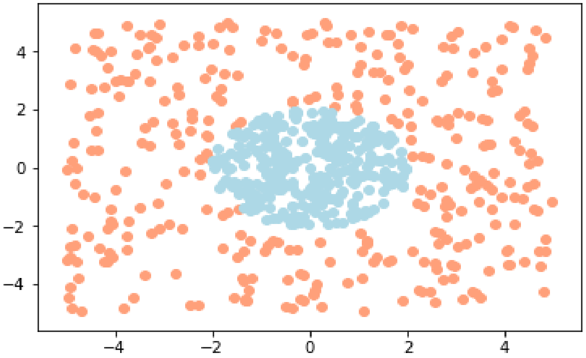


Figure: Actual test data

Note: "+" denotes the misclassified points and region represented by 'lightsalmon' and 'lightblue' is corresponding to the region of the two classes.

Performance of network for test data

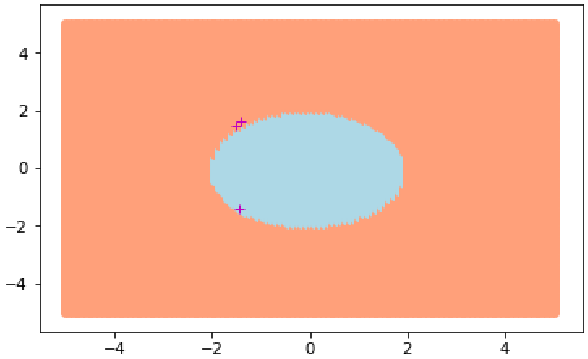


Figure: test data

Accuracy : 98.77

Misclassified points:

x	y
-1.51	1.44
-1.46	-1.39
-1.42	1.59

Convergence Rate of Steepest Descent for Quadratic Function

$$f(x) = \frac{1}{2}x' Hx \qquad \nabla f(x) = Hx \qquad \nabla^2 f(x) = H$$

H: Positive definite and symmetric

Using the steepest descent method:
$x^{k+1} = x^k - \alpha^k \nabla f(x^k) = (I - \alpha^k H)x^k$

Then $\forall k$,
$$f(x^{k+1}) \leq \left(\frac{M-m}{M+m}\right)^2 f(x^k)$$

where,
M: largest eigenvalue of H
m: smallest eigenvalue of H

Disadvantage of MLP

- Huge number of parameters would quickly lead to overfitting.
- Time required for training is very high.
- Don't scale well to full images.

Convolution Neural Network

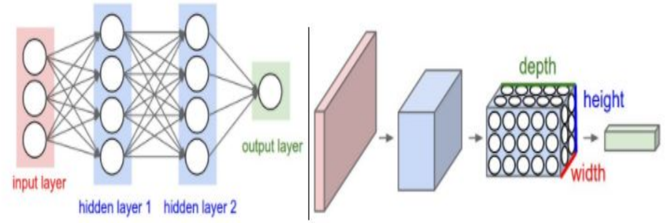
- A convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks
- Applied to analyze visual imagery

Three main types of layers used to build ConvNets architectures:

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer

Similarity to MLP:

- Made up of neurons that have learnable weights and biases
- Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.
- Have a loss function on the last (fully-connected) layer.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Image courtesy: *http://cs231n.github.io/convolutional-networks/*

Architecture of ConvNet

- **INPUT** [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- **CONV layer** will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.

- **RELU layer** will apply an element wise activation function, such as the max(0,x) with threshold zero. This leaves the size of the volume unchanged ([32x32x12]).

- **POOL layer** will perform a down sampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

- **FC (i.e. fully-connected)** layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score. Each neuron in this layer will be connected to all the numbers in the previous volume.

AlexNet

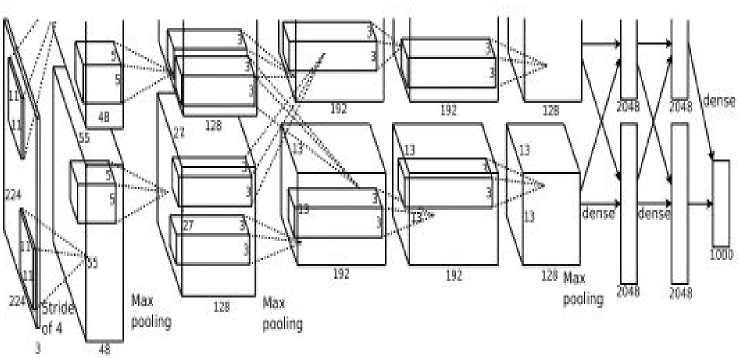


Figure: Architecture [1]

Architecture:

8 - learned layers: The first five are convolutional and the remaining three are fullyconnected.

[227x227x3] **INPUT**
[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0 / **ReLU**
[27x27x96] **MAX POOL1**: 3x3 filters at stride 2
[27x27x96] **NORM1**: Normalization layer
[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2 / **ReLU**
[13x13x256] **MAX POOL2**: 3x3 filters at stride 2
[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1 / **ReLU**
[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1 / **ReLU**
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1 / **ReLU**
[6x6x256] **MAX POOL3**: 3x3 filters at stride 2
[4096] **FC6**: 4096 neurons / **ReLU**
[4096] **FC7**: 4096 neurons / **ReLU**
[1000] **FC8**: 1000 neurons (class scores)

Spaces involved are

1st layer:	$\mathbb{R}^{154587} \to \mathbb{R}^{290400} \to \mathbb{R}^{69984} \to \mathbb{R}^{69984}$
2nd layer:	$\mathbb{R}^{69984} \to \mathbb{R}^{186624} \to \mathbb{R}^{43264} \to \mathbb{R}^{43264}$
3rd layer:	$\mathbb{R}^{43264} \to \mathbb{R}^{64896}$
4th layer:	$\mathbb{R}^{64896} \to \mathbb{R}^{64896}$

5th layer:	$\mathbb{R}^{64896} \to \mathbb{R}^{43264} \to \mathbb{R}^{9216}$
6th layer:	$\mathbb{R}^{9216} \to \mathbb{R}^{4096}$
7th layer:	$\mathbb{R}^{4096} \to \mathbb{R}^{4096}$
8th layer:	$\mathbb{R}^{4096} \to \mathbb{R}^{1000}$

References

A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA: 2009, vol. 3.

S. Kumar, *Neural networks: A classroom approach*. Tata McGraw-Hill Education, 2004.

Thank You