UNIT II INTRODUCTION TO DEEP LEARNING

History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

2 History of Deep Learning [DL]:

- The chain rule that underlies the back-propagation algorithm was invented in the seventeenth century (Leibniz, 1676; L'Hôpital, 1696)
- Beginning in the 1940s, the function approximation techniques were used to motivate machine learning models such as the perceptron
- The earliest models were based on linear models. Critics including Marvin Minsky pointed out several of the flaws of the linear model family, such as its inability to learn the XOR function, which led to a backlash against the entire neural network approach
- Efficient applications of the chain rule based on dynamic programming began to appear in the 1960s and 1970s
- Werbos (1981) proposed applying chain rule techniques for training artificial neural networks. The idea was finally developed in practice after being independently rediscovered in different ways (LeCun, 1985; Parker, 1985; Rumelhart et al., 1986a)
- Following the success of back-propagation, neural network research gained popularity and reached a peak in the early 1990s. Afterwards, other machine learning techniques became more popular until the modern deep learning renaissance that began in 2006
- The core ideas behind modern feedforward networks have not changed substantially since the 1980s. The same back-propagation algorithm and the same approaches to gradient descent are still in use.

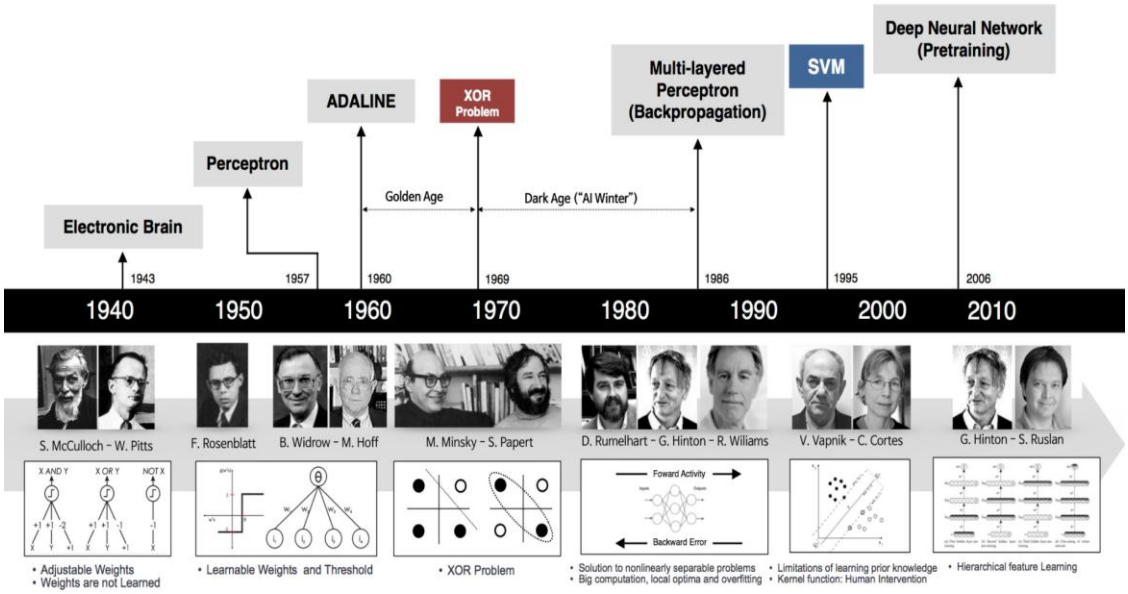
  Most of the improvement in neural network performance from 1986 to 2015 can be attributed to two factors. First, larger datasets have reduced the degree to which statistical generalization is a challenge for neural networks. Second, neural networks have become much larger, because of more powerful computers and better software infrastructure.A small number of algorithmic changes have also improved the performance of neural networks noticeably. One of these algorithmic changes was the replacement of mean squared error with the cross-entropy family of loss functions. Mean squared error was popular in the 1980s and 1990s but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learning community.

  The other major algorithmic change that has greatly improved the performance of feedforward networks was the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units. Rectification using the $\max\{0, z\}$ function was introduced in early neural network models and dates back at least as far as the Cognitron and Neo-Cognitron (Fukushima, 1975, 1980).

  For small datasets, Jarrett et al. (2009) observed that using rectifying nonlinearities is even more important than learning the weights of the hidden layers. Random weights are

sufficient to propagate useful information through a rectified linear network, enabling the classifier layer at the top to learn how to map different feature vectors to class identities. When more data is available, learning begins to extract enough useful knowledge to exceed the performance of randomly chosen parameters. Glorot et al. (2011a) showed that learning is far easier in deep rectified linear networks than in deep networks that have curvature or two-sided saturation in their activation functions.

When the modern resurgence of deep learning began in 2006, feedforward networks continued to have a bad reputation. From about 2006 to 2012, it was widely believed that feedforward networks would not perform well unless they were assisted by other models, such as probabilistic models. Today, it is now known that with the right resources and engineering practices, feedforward networks perform very well. Today, gradient-based learning in feedforward networks is used as a tool to develop probabilistic models. Feedforward networks continue to have unfulfilled potential. In the future, we expect they will be applied to many more tasks, and that advances in optimization algorithms and model design will improve their performance even further.



2.1 A Probabilistic Theory of Deep Learning

Probability is the science of quantifying uncertain things. Most of machine learning and deep learning systems utilize a lot of data to learn about patterns in the data. Whenever data is utilized in a system rather than sole logic, uncertainty grows up and whenever uncertainty grows up, probability becomes relevant.

By introducing probability to a deep learning system, we introduce common sense to the system. In deep learning, several models like Bayesian models, probabilistic graphical models, Hidden Markov models are used. They depend entirely on probability concepts.

Real world data is chaotic. Since deep learning systems utilize real world data, they require a tool to handle the chaoticness.

2.2 Back Propagation Networks (BPN)

2.2.1. Need for Multilayer Networks

- Single Layer networks cannot used to solve Linear Inseparable problems & can only be used to solve linear separable problems
- Single layer networks cannot solve complex problems
- Single layer networks cannot be used when large input-output data set is available
- Single layer networks cannot capture the complex information's available in the training pairs

Hence to overcome the above said Limitations we use Multi-Layer Networks.

2.2.2. Multi-Layer Networks

- Any neural network which has at least one layer in between input and output layers is called Multi-Layer Networks
- Layers present in between the input and out layers are called Hidden Layers
- Input layer neural unit just collects the inputs and forwards them to the next higher layer
- Hidden layer and output layer neural units process the information's feed to them and produce an appropriate output
- Multi -layer networks provide optimal solution for arbitrary classification problems
- Multi -layer networks use linear discriminants, where the inputs are non linear

2.2.3. Back Propagation Networks (BPN)

Introduced by Rumelhart, Hinton, & Williams in 1986. BPN is a Multi-layer Feedforward Network but error is back propagated, Hence the name Back Propagation Network (BPN). It uses Supervised Training process; it has a systematic procedure for training the network and is used in Error Detection and Correction. Generalized Delta Law /Continuous Perceptron Law/ Gradient Descent Law is used in this network. Generalized Delta rule minimizes the mean squared error of the output calculated from the output. Delta law has faster convergence rate when compared with Perceptron Law. It is the extended version of Perceptron Training Law. Limitations of this law is the Local minima problem. Due to this the convergence speed reduces, but it is better than perceptron's. Figure 1 represents a BPN network architecture. Even though Multi level perceptron's can be used they are flexible and efficient that BPN. In figure 1 the weights between input and the hidden portion is considered as Wij and the weight between first hidden to the next layer is considered as V_{jk}. This network is valid only for Differential Output functions. The Training process used in backpropagation involves three stages, which are listed as below

1. Feedforward of input training pair

2. Calculation and backpropagation of associated error

3. Adjustments of weights

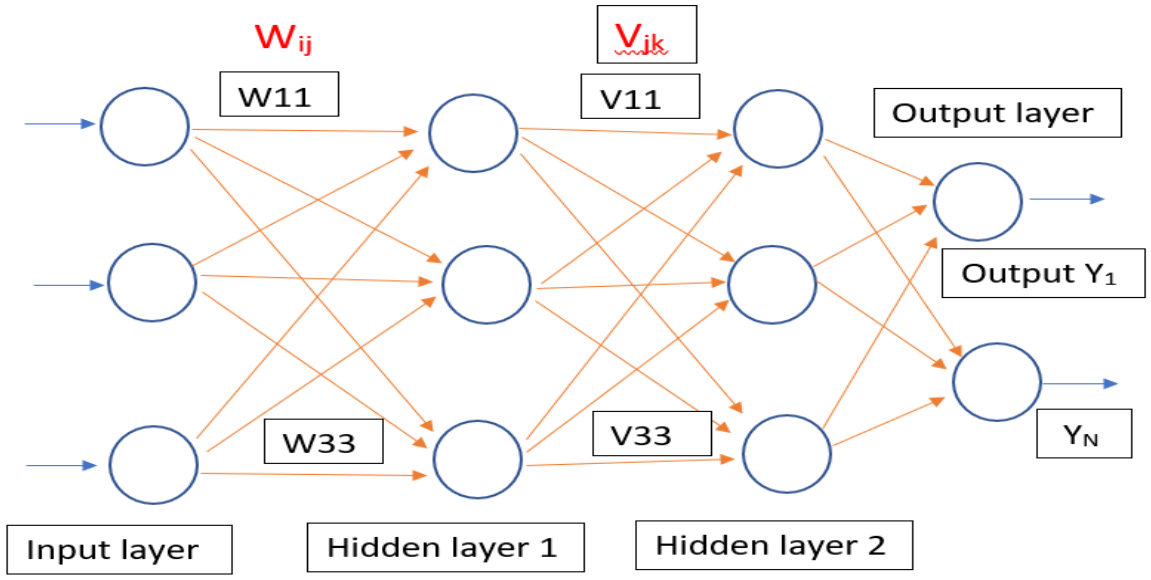


Figure 1: Back Propagation Network

2.2.4. BPN Algorithm

The algorithm for BPN is as classified int four major steps as follows:

1. Initialization of Bias, Weights
2. Feedforward process
3. Back Propagation of Errors
4. Updating of weights & biases

Algorithm:

I. Initialization of weights:

Step 1: Initialize the weights to small random values near zero

Step 2: While stop condition is false , Do steps 3 to 10

Step 3: For each training pair do steps 4 to 9

II. Feed forward of inputs

Step 4: Each input xi is received and forwarded to higher layers (next hidden)

Step 5: Hidden unit sums its weighted inputs as follows

$$Z_{inj} = W_{oj} + \Sigma x_i w_{ij}$$

Applying Activation function

$$Z_j = f(Z_{inj})$$

This value is passed to the output layer

Step 6: Output unit sums it's weighted inputs

$$y_{ink} = V_{oj} + \Sigma Z_j V_{jk}$$

Applying Activation function

$$Y_k = f(y_{ink})$$

III. Backpropagation of Errors

Step 7: $\delta_k = (t_k - Y_k)f(y_{ink})$

Step 8: $\delta_{inj} = \Sigma\, \delta_j V_{jk}$

IV. Updating of Weights & Biases

Step 8: Weight correction is $\Delta w_{ij} = \alpha \delta_k Z_j$

bias Correction is $\Delta w_{oj} = \alpha \delta_k$

V. Updating of Weights & Biases

Step 9: continued:

New Weight is

$$W_{ij(new)} = W_{ij(old)} + \Delta w_{ij}$$

$$V_{jk(new)} = V_{jk(old)} + \Delta V_{jk}$$

New bias is

$$W_{oj(new)} = W_{oj(old)} + \Delta w_{oj}$$

$$V_{ok(new)} = V_{ok(old)} + \Delta V_{ok}$$

Step 10: Test for Stop Condition

2.2.5 Merits

- Has smooth effect on weight correction
- Computing time is less if weight's are small
- 100 times faster than perceptron model
- Has a systematic weight updating procedure

2.2.6. Demerits

- Learning phase requires intensive calculations
- Selection of number of Hidden layer neurons is an issue
- Selection of number of Hidden layers is also an issue
- Network gets trapped in Local Minima
- Temporal Instability
- Network Paralysis
- Training time is more for Complex problems

2.3 Regularization

A fundamental problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs. Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization.

Definition: - "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

❖ In the context of deep learning, most regularization strategies are based on regularizing estimators.

❖ Regularization of an estimator works by trading increased bias for reduced variance.

An effective regularizer is one that makes a profitable trade, reducing variance significantly while not overly increasing the bias.

- Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty Ω(θ) to the objective function J. We denote the regularized objective function by J˜

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

where $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, relative to the standard objective function J. Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.

- The parameter norm penalty Ω that penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized.

2.3.1 L2 Regularization

One of the simplest and most common kind of parameter norm penalty is L2 parameter & it's also called commonly as weight decay. This regularization strategy drives the weights closer to the origin by adding a regularization term $\Omega(\theta) = \frac{1}{2}||w||_2^2$. L2 regularization is also known as ridge regression or Tikhonov regularization. To simplify, we assume no bias parameter, so θ is just w. Such a model has the following total objective function.

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

with the corresponding parameter gradient

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

To take a single gradient step to update the weights, we perform this update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\left(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})\right).$$

Written another way, the update is

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

- We can see that the addition of the weight decay term has modified the learning rule to multiplicatively shrink the weight vector by a constant factor on each step, just before performing the usual gradient update. This describes what happens in a single step.
- The approximation ^J is Given by

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*),$$

Where H is the Hessian matrix of J with respect to w evaluated at w∗.

The minimum of ˆJ occurs where its gradient $\nabla w\hat{J}(w) = H(w - w*)$ is equal to '0'

To study the eff ect of weight decay,

$$\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$
$$(\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$$
$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^*$$

- As α approaches 0, the regularized solution ˜w approaches w*. But what happens as α grows? Because H is real and symmetric, we can decompose it into a diagonal matrix Λ and an orthonormal basis of eigenvectors, Q, such that $H = Q\Lambda Q^T$. Applying Decomposition to the above equation, We Obtain

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*$$
$$= \left[\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^\top\right]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*$$
$$= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*.$$

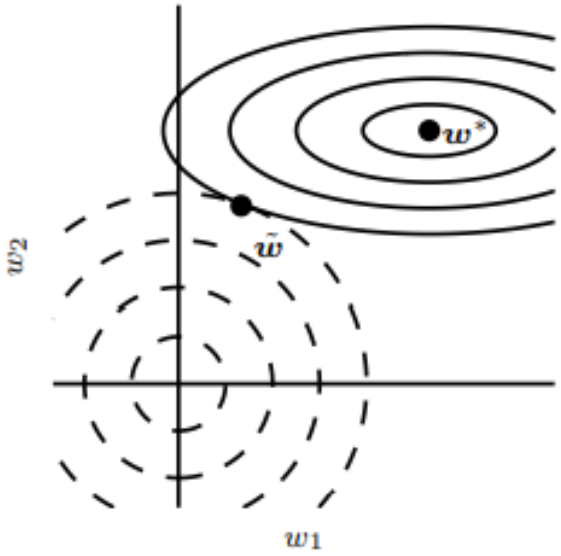


Figure 2: Weight updation effect

The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L 2 regularizer. At the point w˜, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from w∗ . Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w1 close to zero. In the second dimension, the objective function is very sensitive to movements away from w∗ . The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w2 relatively little.

2.3.2 L1 Regularization

While L2 weight decay is the most common form of weight decay, there are other ways to penalize the size of the model parameters. Another option is to use L1 regularization.

- L1 regularization on the model parameter w is defined as the sum of absolute values of the individual parameters.

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|,$$

L1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α. Thus, the regularized objective function J˜(w; X, y) is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

with the corresponding gradient as

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w}), \longrightarrow \text{Eq-1}$$

By inspecting equation 1, we can see immediately that the effect of L 1 regularization is quite different from that of L 2 regularization. Specifically, we can see that the regularization contribution to the gradient no longer scales linearly with each wi ; instead it is a constant factor with a sign equal to sign(wi).

Quadratic approximation of the L 1 regularized objective function decomposes into a sum over the parameters

$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i}(\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha|w_i|\right].$$

The problem of minimizing this approximate cost function has an analytical solution with the following form:

$$w_i = \text{sign}(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}.$$

Consider the situation where w ∗ i > 0 for all i. There are two possible outcomes:

1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed—in direction i—by the L^1 regularization, which pushes the value of w_i to zero.

2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.

2.3.3 Difference between L1 & L2 Parameter Regularization

- L1 regularization attempts to estimate the median of data, L2 regularization makes estimation for the mean of the data in order to evade overfitting.

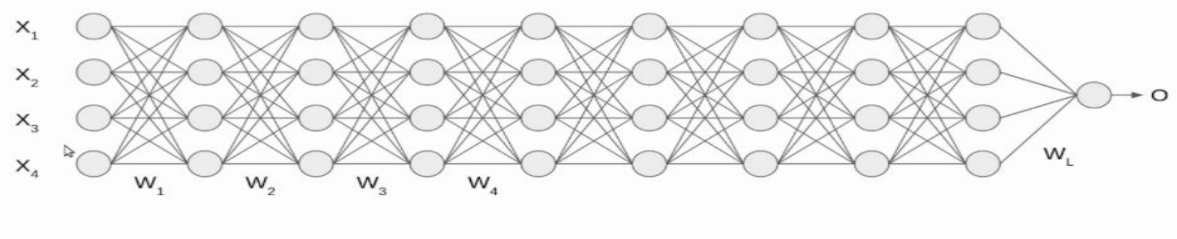
- L1 regularization can add the penalty term in cost function. But L2 regularization appends the squared value of weights in the cost function.
- L1 regularization can be helpful in features selection by eradicating the unimportant features, whereas, L2 regularization is not recommended for feature selection
- L1 doesn't have a closed form solution since it includes an absolute value and it is a non-differentiable function, while L2 has a solution in closed form as it's a square of a weight

S.No	L1 Regularization	L2 Regularization
1	Panelizes the sum of absolute value of weights.	penalizes the sum of square weights.
2	It has a sparse solution.	It has a non-sparse solution.
3	It gives multiple solutions.	It has only one solution.
4	Constructed in feature selection.	No feature selection.
5	Robust to outliers.	Not robust to outliers.
6	It generates simple and interpretable models.	It gives more accurate predictions when the output variable is the function of whole input variables.
7	Unable to learn complex data patterns.	Able to learn complex data patterns.
8	Computationally inefficient over non-sparse conditions.	Computationally efficient because of having analytical solutions.

2.4 Batch Normalization:

It is a method of adaptive reparameterization, motivated by the difficulty of training very deep models.In Deep networks, the weights are updated for each layer. So the output will no longer be on the same scale as the input (even though input is normalized).Normalization - is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale because, we ensure that our model can generalize appropriately.(Normalization is used to bring the input into a balanced scale/ Range).

Let's understand this through an example, we have a deep neural network as shown in the following image.



Initially, our inputs X1, X2, X3, X4 are in normalized form as they are coming from the pre-processing stage. When the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of input X and the weight matrix W.

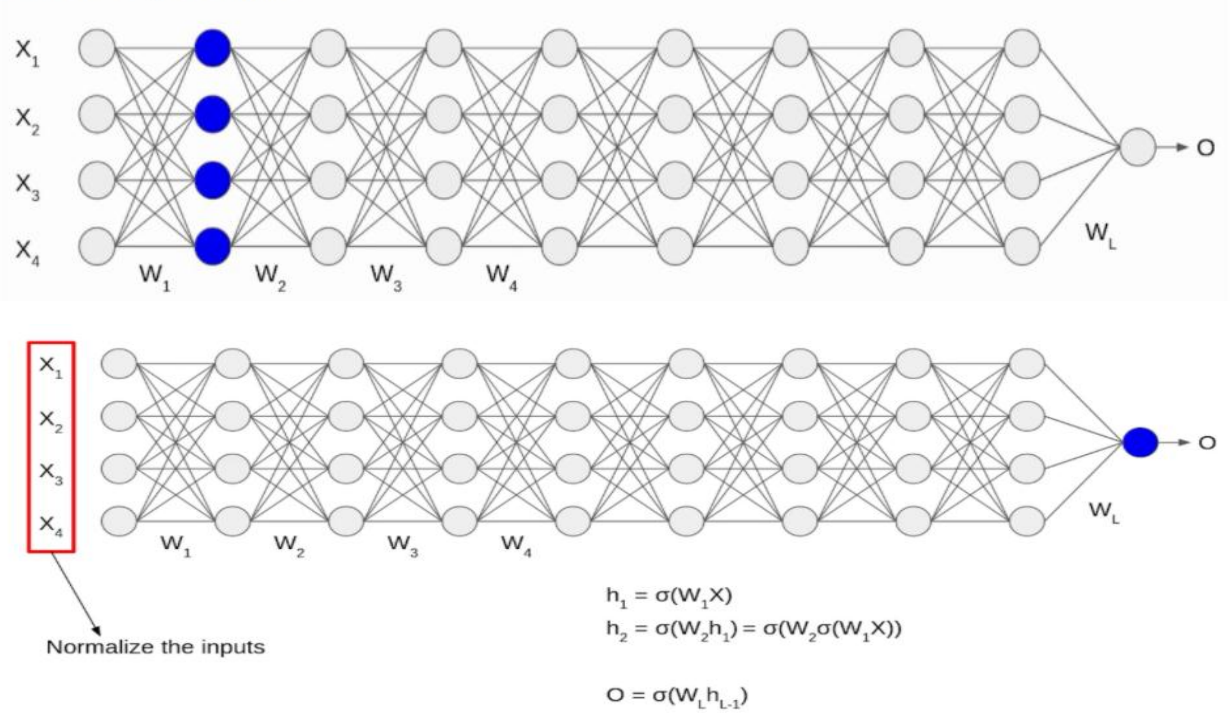


Image Source: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

Even though the input X was normalized but the output is no longer on the same scale. The data passes through multiple layers of network with multiple times(sigmoidal) activation functions are applied, which leads to an internal co-variate shift in the data.

This motivates us to move towards Batch Normalization

Normalization is the process of altering the input data to have mean as zero and standard deviation value as one.

2.4.1 Procedure to do Batch Normalization:

(1) Consider the batch input from layer h, for this layer we need to calculate the mean of this hidden activation.
(2) After calculating the mean the next step is to calculate the standard deviation of the hidden activations.
(3) Now we normalize the hidden activations using these Mean & Standard Deviation values. To do this, we subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).
(4) As the final stage, the re-scaling and offsetting of the input is performed. Here two components of the BN algorithm is used, γ(gamma) and β (beta). These parameters are used for re-scaling (γ) and shifting(β) the vector contains values from the previous operations.

These two parameters are learnable parameters, Hence during the training of neural network, the optimal values of γ and β are obtained and used. Hence we get the accurate normalization of each batch.

2.5. Shallow Networks

Shallow neural networks give us basic idea about deep neural network which consist of only 1 or 2 hidden layers. Understanding a shallow neural network gives us an understanding into what exactly is going on inside a deep neural network A neural network is built using various hidden layers. Now that we know the computations that occur in a particular layer, let us understand how the whole neural network computes the output for a given input *X*. These can also be called the *forward-propagation* equations.

$$Z^{[1]} = W^{[1]T}X + b^{[1]}$$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

$$Z^{[2]} = W^{[2]T}A^{[1]} + b^{[2]}$$

$$\hat{y} = A^{[2]} = \sigma\left(Z^{[2]}\right)$$

1. The first equation calculates the intermediate output **Z[1]** of the first hidden layer.
2. The second equation calculates the final output **A[1]** of the first hidden layer.
3. The third equation calculates the intermediate output **Z[2]** of the output layer.
4. The fourth equation calculates the final output **A[2]** of the output layer which is also the final output of the whole neural network.

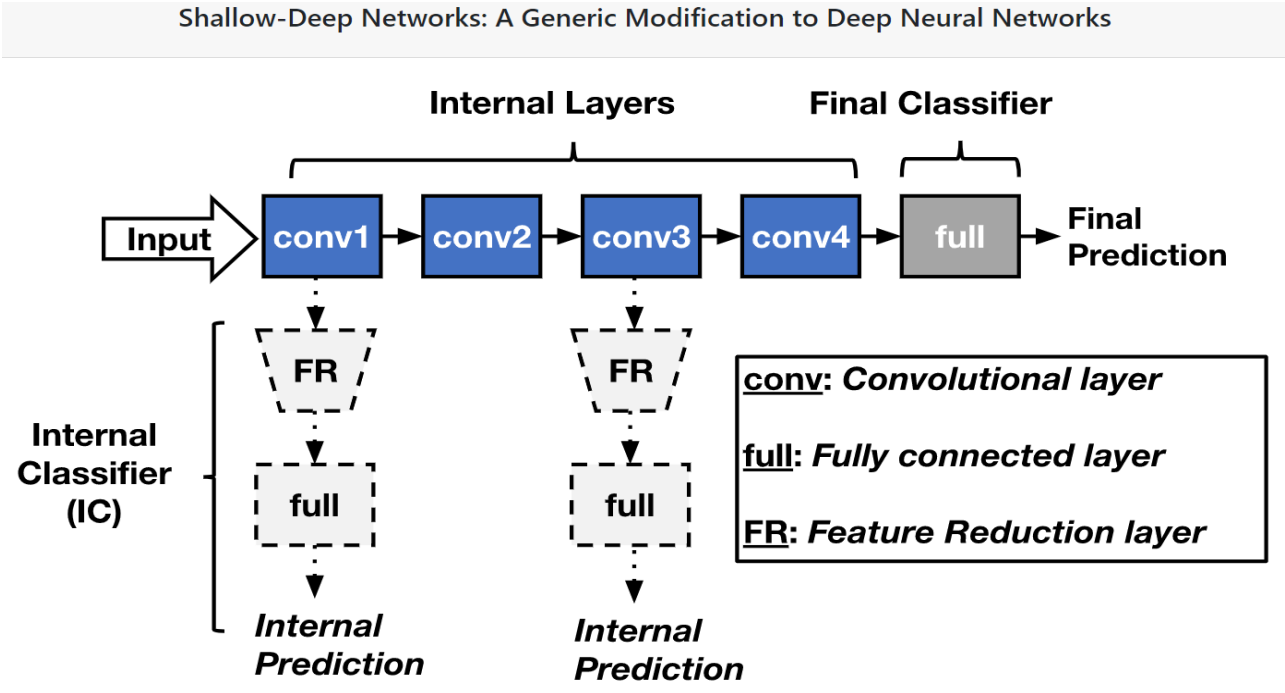


Figure 2:Shallow Networks – Generic Model

2.5.1 Difference Between a Shallow Net & Deep Learning Net:

Sl.No	Shallow Net's	Deep Learning Net's
1	One Hidden layer(or very less no. of Hidden Layers)	Deep Net's has many layers of Hidden layers with more no. of neurons in each layers
2	Takes input only as VECTORS	DL can have raw data like image, text as inputs
3	Shallow net's needs more parameters to have better fit	DL can fit functions better with less parameters than a shallow network
4	Shallow networks with one Hidden layer (same no of neurons as DL) cannot place complex functions over the input space	DL can compactly express highly complex functions over input space
5	The number of units in a shallow network grows exponentially with task complexity.	DL don't need to increase it size(neurons) for complex problems
6	Shallow network is more difficult to train with our current algorithms (e.g. it has issues of local minima etc)	Training in DL is easy and no issue of local minima in DL

Reference Books:

1. B. Yegnanarayana, "Artificial Neural Networks" Prentice Hall Publications.
2. Simon Haykin, "Artificial Neural Networks", Second Edition, Pearson Education.
3. Laurene Fausett, "Fundamentals of Neural Networks, Architectures, Algorithms and Applications", Prentice Hall publications.
4. Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
5. 2. Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
6. 3. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
7. 4. Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

Note: For further reference, kindly refer the class notes, PPTs, Video lectures available in the Learning Management System (Moodle)

********** **ALL THE BEST** **********