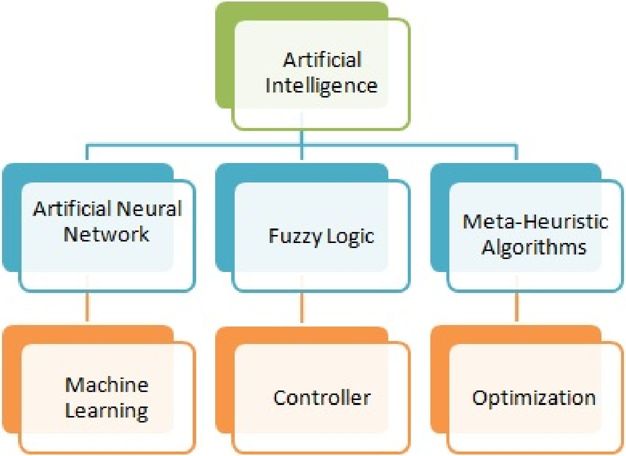
Fundamentals of Artificial Neural Networks

Dr.Venkataramana Veeramsetty
Center for AI and Deep Learning
Assistant Professor in Dept. of EEE
SR University Warangal

December 10, 2020

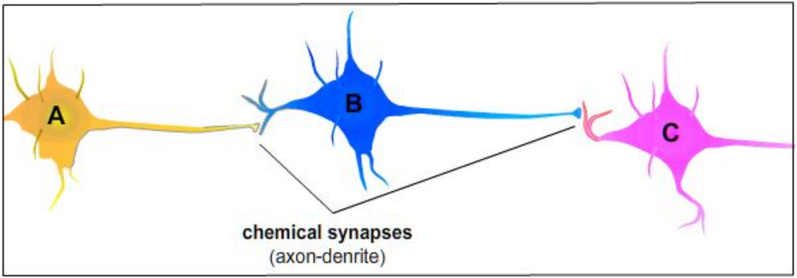
AI - Artificial Intelligence



Email : dr.vvr.research@gmail.com

What is an artificial neural network?

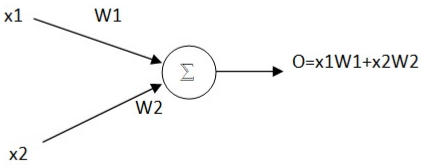
- Artificial neural networks are one of the main tools used in machine learning. As the "neural" part of their name suggests, they are brain-inspired systems which are intended to replicate the way that we humans learn.
- Like our human brain has millions of neurons in a hierarchy and Network of neurons which are interconnected with each other via Axons and passes Electrical signals from one layer to another called synapses.



Email : dr.vvr.research@gmail.com
(I
nsetty)

Neuron Modeling

- It is linear neuron, but most of the problems are non linear



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

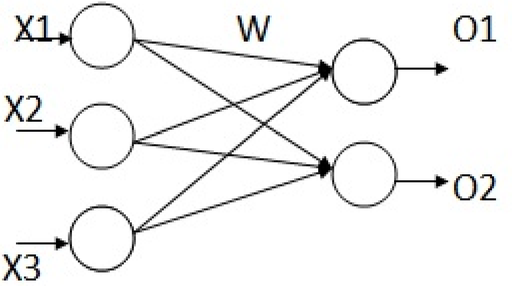
ANN Classification

√ Feed Forward Neural Network
- Single Layer Feed Forward Neural Network
- Multi-layer Feed Forward Neural Network

√ Feed Back Neural Network (Or) Recurrent Neural Network
- Single Layer Feed Back Neural Network
- Multi-layer Feed Back Neural Network

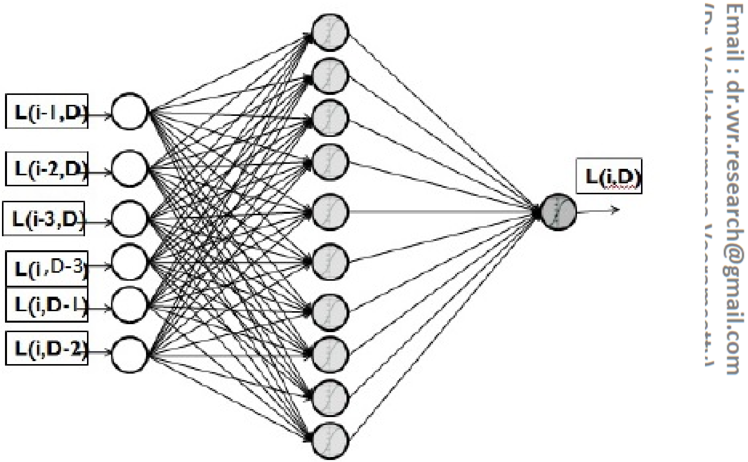
Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Single Layer Feed Forward Neural Network

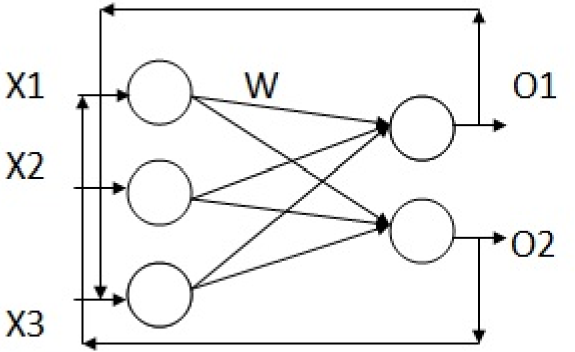


Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Multi-Layer Feed Forward Neural Network

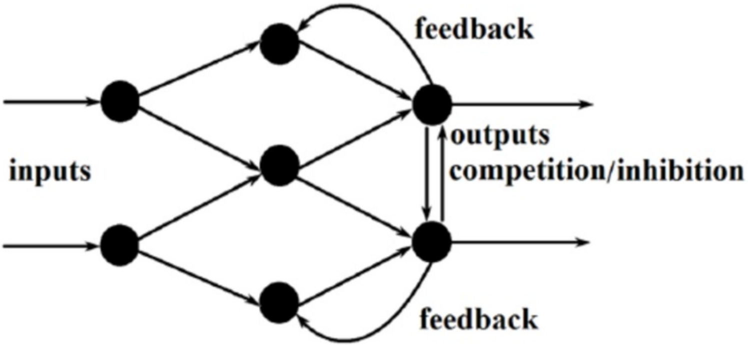


Single Layer Feed Back Neural Network



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

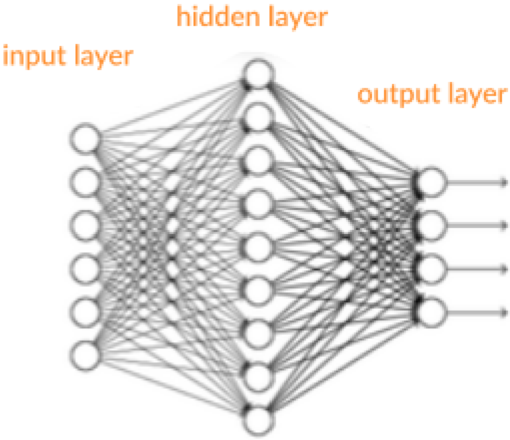
Multi-Layer Feed Back Neural Network



Email : dr.vvr.research@gmail.com

Shallow Neural Network

- Input layer:1, Hidden layer: 1, Output layer: 1

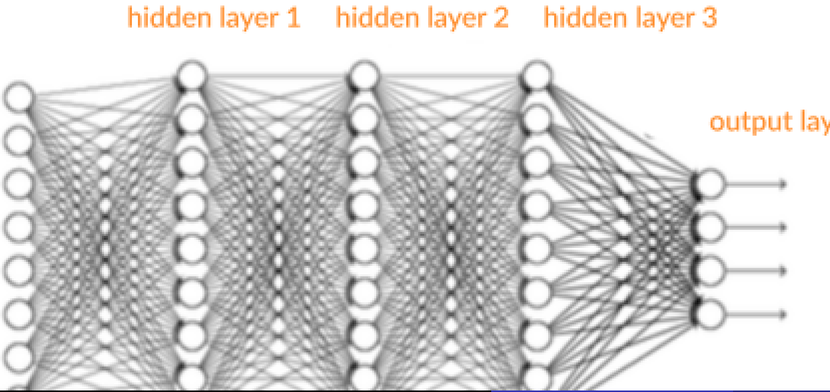


Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Deep Neural Network

- Input layer:1, Hidden layer: 3-10, Output layer: 1
- Beyond 10 predictive power may starts to decline

Email : dr.v
(Dr. Venkata



Neural Network Activation Function

Neural network activation functions are crucial components in deep learning. Activation functions determine

- Output of deep learning model
- Accuracy
- Computational efficiency

Activation functions also have impact on neural network's ability to converge and the convergence speed

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

- Activation functions are mathematical equations that determine the output of a neural network.
- The function is attached to each neuron in the network, and determines whether it should be activated (“fired”) or not, based on whether each neuron’s input
- Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.

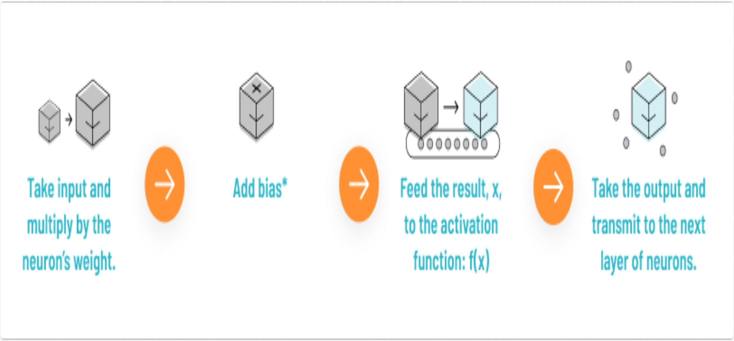
What is need of Activation Function in ANN

- Activation functions are required for ANN in order learn non linear functional mapping between input and output
- A Neural Network without Activation function would simply be a Linear regression Model, which has limited power and does not performs good most of the times
- Without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos , audio , speech etc.

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

- Important feature of a Activation function is that it should be differentiable to perform back propagation optimization strategy while propagation backwards in the network to compute gradients of Error(loss) with respect to Weights and then accordingly optimize weights using Gradient descend or any other Optimization technique to reduce Error

What activation function do in NN



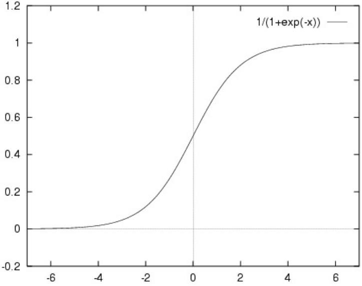
Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Activation Functions

- Sigmoid
- Tan Hyperbolic
- ReLU
- Leaky ReLU
- Softmax

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Sigmoid Activation function



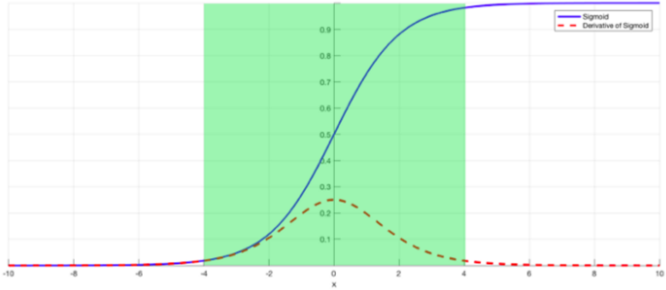
$$f(x)=\frac{1}{1+e^{-x}}$$

Generally used in output layer for regression problems

Advantages	Disadvantages
Smooth gradient	Vanishing gradient problem
Clear prediction	Computationally burden
Bounded output values	Sigmoids have slow convergence

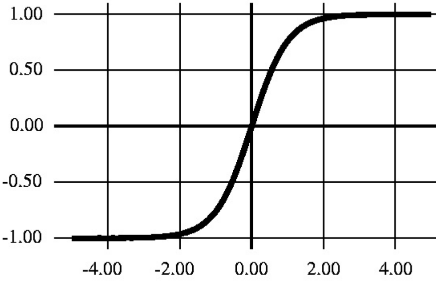
Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Vanishing Gradient Problem



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Hyperbolic Tangent function



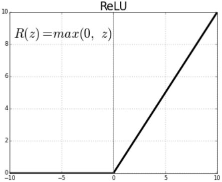
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1}$$

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Advantages	Disadvantages
Zero centered	Vanishing gradient problem
	Computationally burden
	Sigmoids have slow convergence

Generally used in hidden layer

ReLu- Rectified Linear units



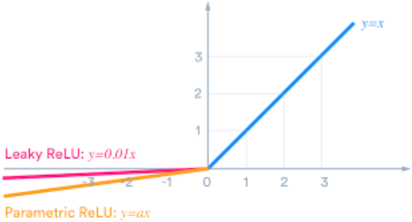
$$R(z) = max(0, z), if \quad z > 0 \tag{2}$$

$$R(z) = 0, if \quad z <= 0 \tag{3}$$

Advantages	Disadvantages
Computationally efficient	Dead neuron
Generally used in hidden layer	

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Leaky ReLU



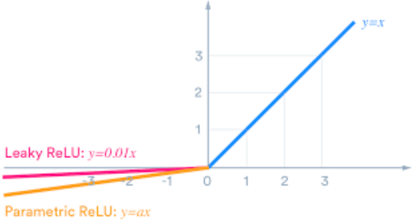
for $\alpha = 100$ Leaky ReLu similar to ReLu. best $\alpha = 5.5$

Email : dr.vvr.research@gmail.com (Dr. Venkataramana Veeramsetty)

$$f(z) = \begin{cases} z & z >= 0 \\ \frac{z}{\alpha} & z < 0 \end{cases} \quad \textit{where} \quad \alpha \quad \textit{is} \quad \textit{a} \quad \textit{small} \quad \textit{value} \qquad (4)$$

Advantages	Disadvantages
Prevents Dead neuron	Results not consistent for negative inputs
Generally used in hidden layer	

Randomized Leaky ReLU



$$f(z) = \begin{cases} z & z >= 0 \\ \alpha * z & z < 0 \end{cases} \tag{5}$$

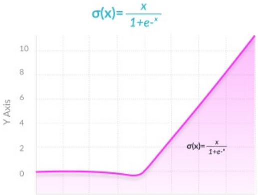
$$\alpha = \frac{l + u}{2} :: l, u \epsilon 0, 1 \tag{6}$$

- Prevents Dead neuron
- Combats over fitting problem

Swish

- Swish is a new, self-gated activation function discovered by researchers at Google
- Performing well over ReLU for computer vision problems

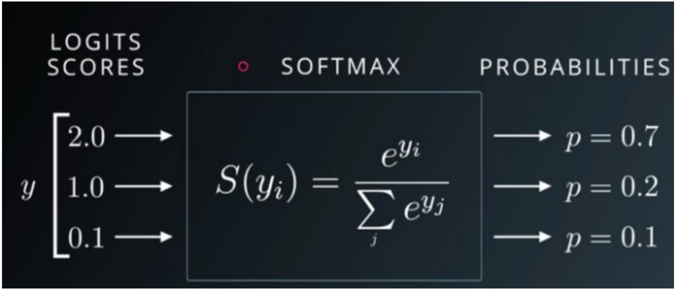
Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)



$$R(z) = \frac{z}{1 + e^{-z}} \quad (7)$$

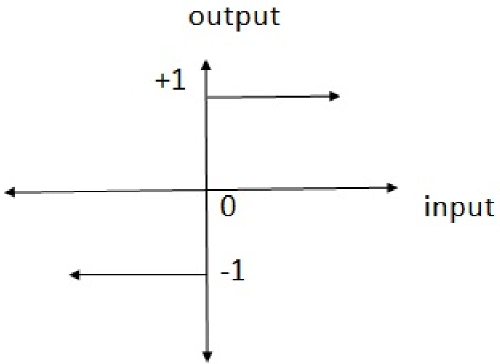
Softmax

- Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes



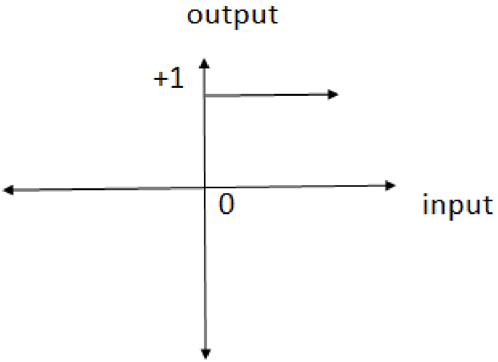
Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Bipolar activation function



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Binary activation function



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

- The problem with a this function is that it does not allow multi-value outputs

Research Problem

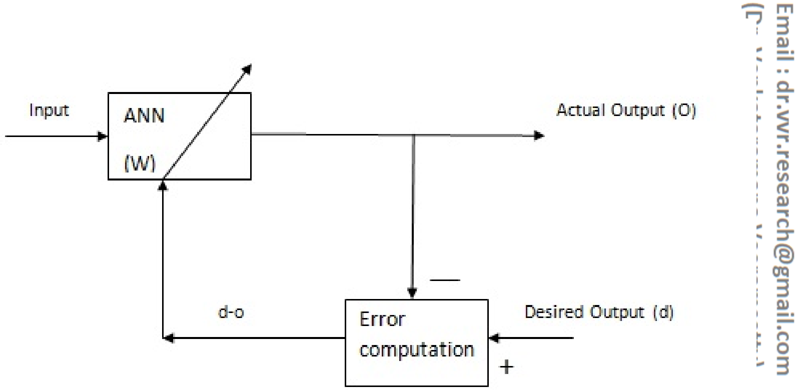
- Optimal activation function for a certain neural network and to even automatically combine activation functions to achieve the highest accuracy. This is a very promising field of research because it attempts to discover an optimal activation function configuration automatically, whereas today, this parameter is manually tuned.

Neural Network Learning Rules

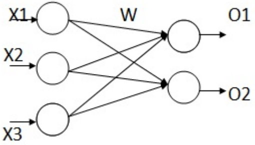
- √ Supervised Learning
  - Delta learning rule
  - Perceptron learning rule
  - Widro-hoff learning rule
- √ Unsupervised Learning
  - Competitive learning rule
- Reinforcement learning

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Supervised Learning



Delta learning rule



- Use Sigmoid Activation function

$$net = X * W^T \tag{8}$$

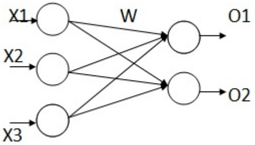
$$O = \frac{1}{1 + exp(-net)} \tag{9}$$

$$\Delta W = \eta * (d - O) * O(1 - O) * X \tag{10}$$

$$W = W + \Delta W \tag{11}$$

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Perceptron learning rule



$$net = X * W^T \tag{12}$$

- Use Linear Activation function like bi-polar or binary activation functions

For Bipolar activation function

$$O = -1 \ \ if \ \ net < 0 \ \ and \ \ O = 1 \ \ if \ \ net > 0 \quad (13)$$

$$\Delta W = \eta * (d - O)) * X \quad (14)$$

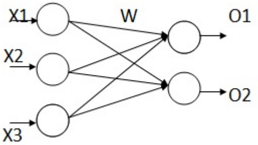
For Binary activation function

$$O = 0 \ \ if \ \ net < 0 \ \ and \ \ O = 1 \ \ if \ \ net > 0 \quad (15)$$

$$\Delta W = \eta * (d - O) * X \quad (16)$$

$$W = W + \Delta W \quad (17)$$

Widro-hoff learning rule



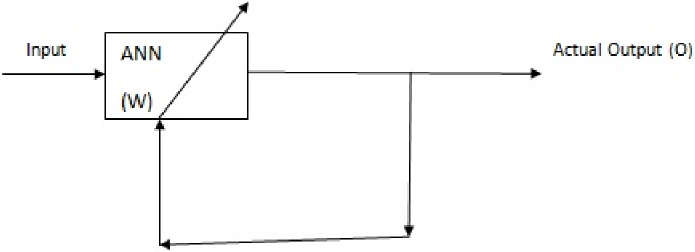
- Independent of activation function

$$O = X * W^T \tag{18}$$

$$\Delta W = \eta * (d - O) * X \tag{19}$$

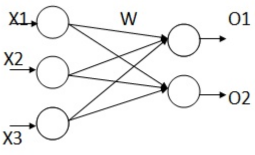
$$W = W + \Delta W \tag{20}$$

Unsupervised Learning



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Competitive Learning



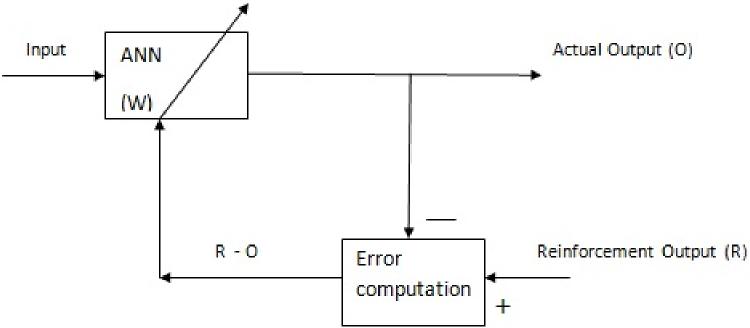
- Weights connected to winning output neuron will be updated

$$\Delta W = \eta * (X - W^{WinningNeuron}) \tag{21}$$

$$W = W + \Delta W \tag{22}$$

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Reinforcement Learning



Steps to follow to solve problem with AI

Step-1: Prepare data based on how you want to solve the problem

Step-2: Split the data as training and testing with 95% and 5% respectively. or 70%, 15% and 15% if validation is considered (Numbers not mandatory). **This can be done randomly**.

Step-3: Design architecture based on how you want to solve the problem

Steps to follow to solve problem with AI

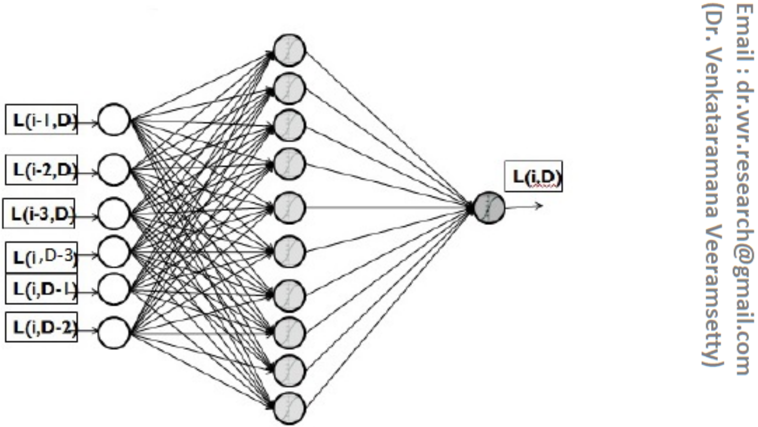
Step-4: Train the network with training data and proper algorithm. Training the network means updating the weights based on error

Step-5: If validation data is available then use it during training to identify the network performance. Validating the network means testing the network while training

Step-6: Test the performance of the network with test data.

Step-6: If training, validation and testing accuracy are good then we can use the network for prediction in real time

Back Propagation Algorithm



Back Propagation Algorithm Cont.

- Let assume W_{ij} is a weight matrix between input layer (i) and hidden layer (j) and W_{jk} is a weight matrix between hidden layer (j) and output layer (k)
- X represents input to the network, d represents target output and O represents actual output given by ANN
- Sigmoid activation function is used to map non linear relation between input and output

Back Propagation Algorithm Cont.

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step-1: Calculate net_j for hidden layer by using equation (23)

$$net_j = X * W_{ij}^T \tag{23}$$

Step-2: Calculate output of hidden layer using equation (52)

$$O_j = \frac{1}{1 + e^{-net_j}} \tag{24}$$

Back Propagation Algorithm Cont.

Step-3: Calculate net_0 for output layer by using equation (80)

$$net_0 = O_j * W_{jk}^T \tag{25}$$

Step-4: Calculate output of output layer using equation (34)

$$O_k = \frac{1}{1 + e^{-net_o}} \tag{26}$$

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Back Propagation Algorithm Cont.

Step-5: Update the weights between hidden layer and output layer using equation (35)

$$W_{jk} = W_{jk} + \Delta W_{jk} \tag{27}$$

where

$$\Delta W_{jk} = \eta * (d_k - O_k) * O_k * (1 - O_k) * O_j \tag{28}$$

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Back Propagation Algorithm Cont.

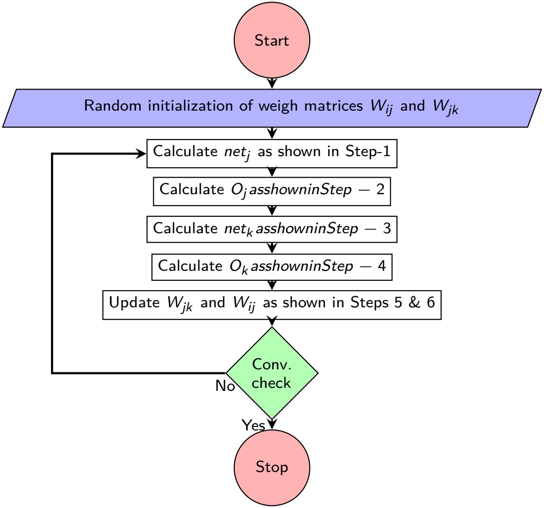
Step-6: Update the weights between hidden layer and input layer using equation (39)

$$W_{ij} = W_{ij} + \Delta W_{ij} \tag{29}$$

where

$$\Delta W_{ij} = \eta * \sum_{k=1}^{n_o} (d_k - O_k) * O_k * (1 - O_k) * W_{jk} * O_j * (1 - O_j) * X \tag{30}$$

Back Propagation Algorithm Cont.

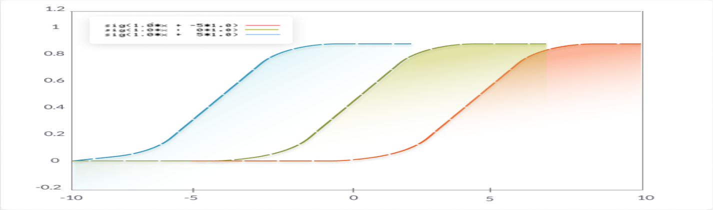


Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

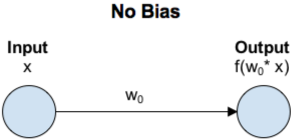
Conv. check is $\max(\Delta W_{ij}, \Delta W_{jk}) < 0.0001$

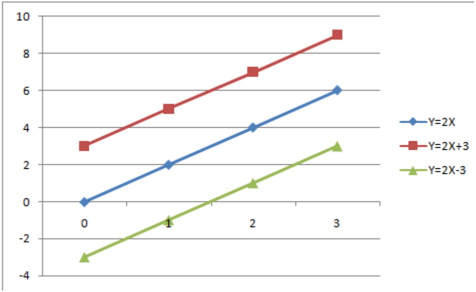
What is bias neuron?

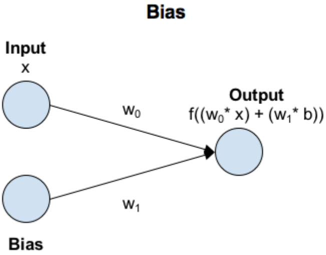
- Bias is one of the neuron which always feed a constant value(is generally 1, some times a random value) to each hidden and output layers
- It helps the activation function to shift in right or left or up or down direction



Email : drvvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)


No Bias
Input
x
Output
$f(w_0 * x)$
w_0


10
8
6
4
2
0
-2
-4
0
1
2
3
Y=2X
Y=2X+3
Y=2X-3


Bias
Input
x
Output
$f((w_0 * x) + (w_1 * b))$
w_0
w_1
Bias

Back Propagation Algorithm with bias neuron

Email : dr.vvr.research@gmail.com (Dr. Venkataramana Veeramsetty)

Step-1: Calculate net_j for hidden layer by using equation (31)

$$net_j = X * W_{ij}^T + b_j \tag{31}$$

Step-2: Calculate output of hidden layer using equation (52)

$$O_j = \frac{1}{1 + e^{-net_j}} \tag{32}$$

Back Propagation Algorithm with bias Cont.

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step-3: Calculate net_0 for output layer by using equation (33)

$$net_0 = O_j * W_{jk}^T + b_o \tag{33}$$

Step-4: Calculate output of output layer using equation (34)

$$O_k = \frac{1}{1 + e^{-net_o}} \tag{34}$$

Back Propagation Algorithm with bias Cont.

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step-5: Update the weights between hidden layer and output layer using equation (35)

$$W_{jk} = W_{jk} + \Delta W_{jk} \tag{35}$$

where

$$\Delta W_{jk} = \eta * (d_k - O_k) * O_k * (1 - O_k) * O_j \tag{36}$$

Step-6: Update the bias parameter between hidden layer and output layer using equation (37)

$$b_o = b_o + \Delta b_o \tag{37}$$

where

$$\Delta b_o = \eta * (d_k - O_k) * O_k * (1 - O_k) \tag{38}$$

Email : dr.vvresearch@gmail.com
(Dr. Venkataramana Veeramsetty)

Back Propagation Algorithm with bias Cont.

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step-7: Update the weights between hidden layer and input layer using equation (39)

$$W_{ij} = W_{ij} + \Delta W_{ij} \tag{39}$$

where

$$\Delta W_{ij} = \eta * \sum_{k=1}^{n_o} (d_k - O_k) * O_k * (1 - O_k) * W_{jk} * O_j * (1 - O_j) * X \tag{40}$$

Step-8: Update the bias parameter between hidden layer and input layer using equation (41)

$$b_j = b_j + \Delta b_j \tag{41}$$

where

$$\Delta b_j = \eta * \sum_{k=1}^{n_o} (d_k - O_k) * O_k * (1 - O_k) * W_{jk} * O_j * (1 - O_j) \tag{42}$$

ANN for binary classification problem

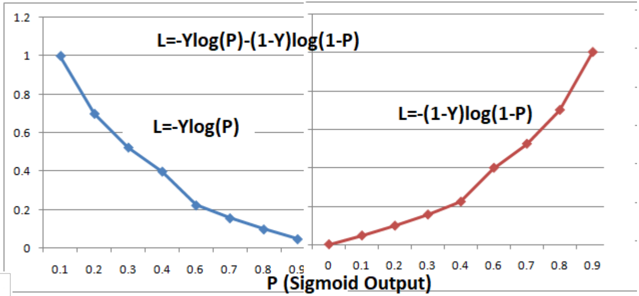
- Cannot use MSE for binary classification problem as it will be a non-convex function so we are going with binary cross entropy loss function instead of MSE

√ **Binary Cross Entropy Loss :**

$$L = -y * log(p) - (1 - y) * log(1 - p) \qquad (43)$$

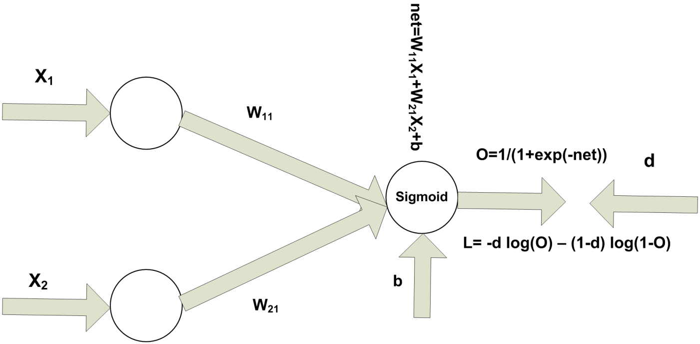
To calculate probability p we can use sigmoid activation in output layer

Predicted output ($\hat{Y}$)	$L\|_{Y=1}$=-log($\hat{Y}$)	$L\|_{Y=0}$=-log(1-$\hat{Y}$)
0	∞	0
0.1	1	0.045757
0.2	0.69897	0.09691
0.3	0.522879	0.154902
0.4	0.39794	0.221849
0.6	0.221849	0.39794
0.7	0.154902	0.522879
0.8	0.09691	0.69897
0.9	0.045757	1
1	0	∞



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Single Layer ANN for binary Classification



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step1: Read dataset X=$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, Y=$(d)$, η. Initialize weights W=$\begin{pmatrix} W_{11} \\ W_{21} \end{pmatrix}$ and bias "b"

Step2: Compute net input to output neuron using equation (44)

$$net = W_{11}X_1 + W_{21}X_2 + b = W^T X + b = \sum_{i=1}^{n_i} W_{i1}X_i + b \tag{44}$$

Step3: Compute output (O) using equation (65)

$$O = \frac{1}{1 + e^{-net}} \tag{45}$$

Step4: Compute binary cross entropy loss using equation (66)

$$L = -d\log(O) - (1-d)\log(1-O) \tag{46}$$

Step5: Compute change in weight (ΔW) using equation (67)

$$\Delta W = \begin{pmatrix} \Delta W_{11} \\ \Delta W_{21} \end{pmatrix} = \eta(d - O) \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \tag{47}$$

Step6: Compute change in bias (Δb) using equation (68)

$$\Delta b = \eta(d - O) \tag{48}$$

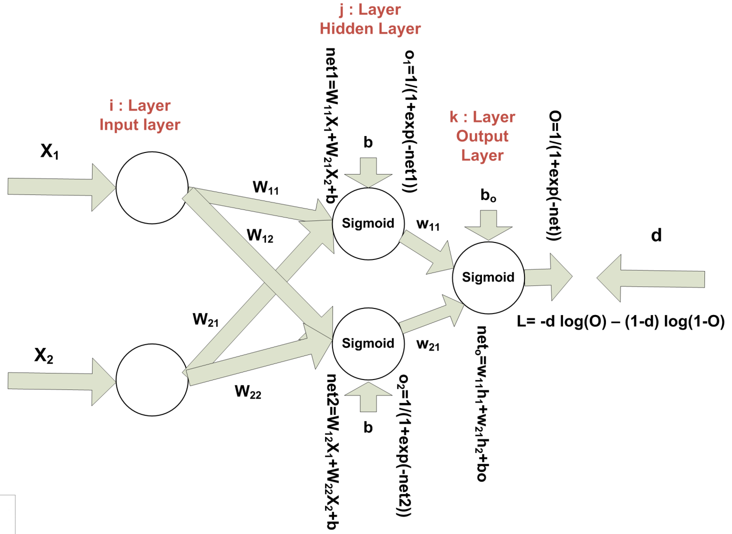
Step7: Update weight matrix using equation (69) and bias using equation (70)

$$W = W + \Delta W \tag{49}$$

$$b = b + \Delta b \tag{50}$$

p8: Repeat steps 2,3,4,5,6,7 till convergence check

Multiple layer ANN for classification



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Back Propagation Algorithm for binary classification

Step-1: Calculate net_j for hidden layer by using equation (71)

$$net_j = W_{ij}^T * X + b_j \tag{51}$$

Step-2: Calculate output of hidden layer using equation (72)

$$O_j = \frac{1}{1 + e^{-net_j}} \tag{52}$$

refer to hidden neuron

Back Propagation Algorithm for binary classification Cont.

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step-3: Calculate net_0 for output layer by using equation (73)

$$net_0 = w_{jk}^T * O_j + b_o \tag{53}$$

Step-4: Calculate output of output layer using equation (54)

$$O = \frac{1}{1 + e^{-net_o}} \tag{54}$$

Back Propagation Algorithm for binary classification Cont.

Step-5: Update the weights between hidden layer and output layer using equation (55)

$$W_{jk} = W_{jk} + \Delta W_{jk} \tag{55}$$

where

$$\Delta W_{jk} = \eta * (d - O) * O_j \tag{56}$$

Step-6: Update the bias parameter between hidden layer and output layer using equation (57)

$$b_o = b_o + \Delta b_o \tag{57}$$

where

$$\Delta b_o = \eta * (d_k - O_k) \tag{58}$$

Back Propagation Algorithm for binary classification Cont.

Step-7: Update the weights between hidden layer and input layer using equation (79)

$$W_{ij} = W_{ij} + \Delta W_{ij} \tag{59}$$

where

$$\Delta W_{ij} = \eta * (d - O) * W_{jk} * O_j * (1 - O_j) * X \tag{60}$$

Step-8: Update the bias parameter between hidden layer and input layer using equation (81)

$$b_j = b_j + \Delta b_j \tag{61}$$

where

$$\Delta b_j = \eta * (d - O) * O_j * (1 - O_j) * W_{jk} \tag{62}$$

Single layer ANN for Multiple Classification

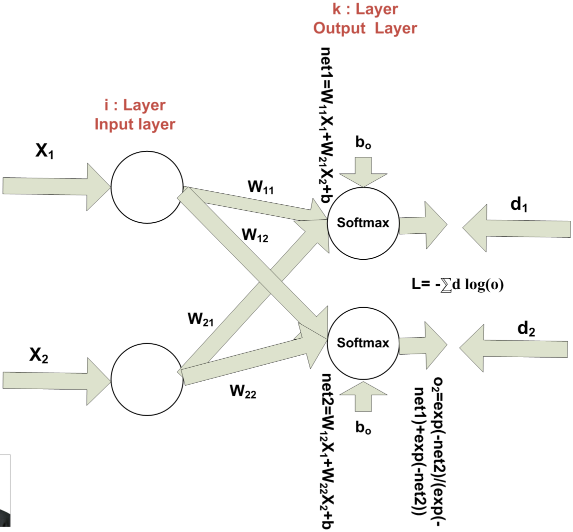
For multiple classifications, categorical cross entropy is used

√ **Categorical Cross Entropy Loss :**

$$L(X_i, Y_i) = -\sum_{j=1}^{n} Y_{ij} log(p_{ij}) \tag{63}$$

where p_{ij} calculated using softmax activation function

Single layer ANN for Multiple Classification



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Derivatives of activation functions

Activation function	f(x)	f'(x)
Sigmoid	$\frac{1}{1+e^{-x}}$	f(x)*(1-f(x))
Softmax	$\frac{e^x}{\sum e^x}$	f(x)*(1-f(x))

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step1: Read dataset X=$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, Y=$(d)$, η. Initialize weights W=$\begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{pmatrix}$ and bias "b"

Step2: Compute net input to each output neuron

$$\begin{pmatrix} net_1 \\ net_2 \end{pmatrix} = W^T X + b \tag{64}$$

Step3: Compute output (O_k) for each output neuron

$$O_k = \frac{e^{net_k}}{\sum_{i=1}^{n_o} e^{net_i}} \tag{65}$$

Step4: Compute categorical cross entropy loss

$$L = -\sum_{i=1}^{n_o} d_i log(O_i) \tag{66}$$

[Weights which are connected to output neuron corresponding to class will be updated]

Step5: Compute change in weight (ΔW) using equation (67)

$$\Delta W = \begin{pmatrix} \Delta W_{11} \\ \Delta W_{21} \end{pmatrix} = \eta d * (1 - O) \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \tag{67}$$

Step6: Compute change in bias (Δb) using equation (68)

$$\Delta b = \eta d * (1 - O) \tag{68}$$

Step7: Update weight matrix using equation (69) and bias using equation (70)

$$W = W + \Delta W \tag{69}$$

$$b = b + \Delta b \tag{70}$$

p8: Repeat steps 2,3,4,5,6,7 till convergence check

Back Propagation Algorithm for Categorical classification

Step-1: Calculate net_j for hidden layer by using equation (71)

$$net_j = W_{ij}^T * X + b_j \tag{71}$$

Step-2: Calculate output of hidden layer using equation (72)

$$O_j = \frac{1}{1 + e^{-net_j}} \tag{72}$$

refer to hidden neuron

Back Propagation Algorithm for Categorical classification Cont.

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step-3: Calculate net_k for output layer by using equation (73)

$$net_k = w_{jk}^T * O_j + b_k \tag{73}$$

Step-4: Calculate output of output layer using equation (74)

$$O_k = \frac{e^{net_k}}{\sum e^{net_k}} \tag{74}$$

Back Propagation Algorithm for Categorical classification Cont.

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Step-5: Update the weights between hidden layer and output layer using equation (75)

$$W_{jk} = W_{jk} + \Delta W_{jk} \tag{75}$$

where

$$\Delta W_{jk} = \eta * \sum_{k=1}^{n_k} d_k * (1 - O_k) * O_j \tag{76}$$

Step-6: Update the bias parameter between hidden layer and output layer using equation (77)

$$b_k = b_k + \Delta b_k \tag{77}$$

where

$$\Delta b_k = \eta * \sum_{k=1}^{n_k} d_k * (1 - O_k) \tag{78}$$

Back Propagation Algorithm for categorical classification Cont.

Step-7: Update the weights between hidden layer and input layer using equation (79)

$$W_{ij} = W_{ij} + \Delta W_{ij} \tag{79}$$

where

$$\Delta W_{ij} = \eta * O_j * (1 - O_j) * \sum_{k=1}^{n_k} d_k (1 - O_k) * W_{jk} * X \tag{80}$$

Step-8: Update the bias parameter between hidden layer and input layer using equation (81)

$$b_j = b_j + \Delta b_j \tag{81}$$

where

$$\Delta b_j = \eta * O_j * (1 - O_j) * \sum_{k=1}^{n_k} d_k (1 - O_k) * W_{jk} \tag{82}$$

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

- **Training Set:** A training set is a group of sample inputs feed into the neural network in order to train the model. The neural network learns from inputs and finds weights for the neurons that can result in an accurate prediction.
- **Training Error:** The error is the difference between the known correct output for the inputs and the actual output of the neural network. During the course of training, the training error is reduced until the model produces an accurate prediction for the training set.
- **Validation Set:** A validation set is another group of sample inputs which were not included in training and preferably are different from the samples in the training set.
- **Validation Error:** The difference between correct prediction for the validation set with the actual model prediction is the validation error.

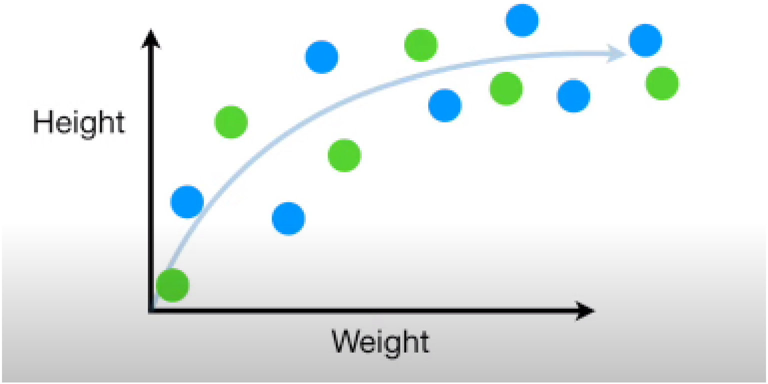
Bias Vs. Variance

- √ High Bias:: High training loss
- √ Low Bias:: Low training loss
- √ High Variance:: High Validation loss
- √ Low Variance:: Low Validation loss

Model with low bias and low variance is good for real time application

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Let consider a model to predict height based on weight.

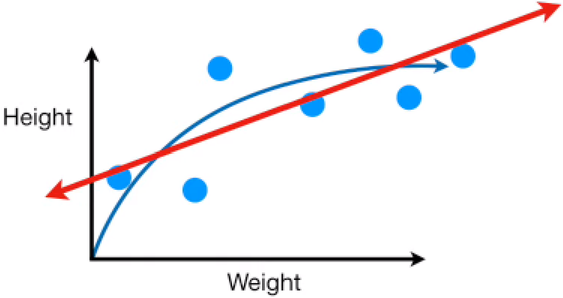


Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

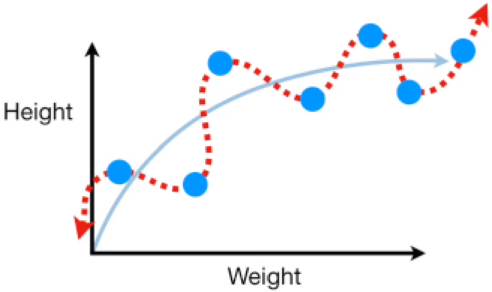
Blue dots: Training Set
Green dots: Testing/Validation Set

If we consider a linear regression model, it may not find the true relation between weight and height leads high training loss. This is called **high bias**.

INABILITY OF MODEL TO FIND THE TRUE RELATION BETWEEN INPUT AND OUTPUT BASED ON TRAINING DATA IS CALLED HIGH BIAS

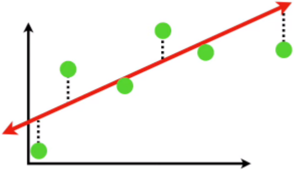


If we consider a polynomial regression model, which fits almost all samples in dataset that leads low training loss. This is called **low bias**. ABILITY OF MODEL TO FIND THE EXACT RELATION BETWEEN INPUT AND OUTPUT BASED ON TRAINING DATA IS CALLED LOW BIAS

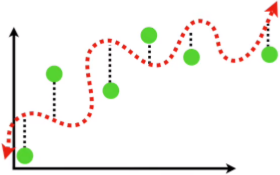


Model performance on validation dataset

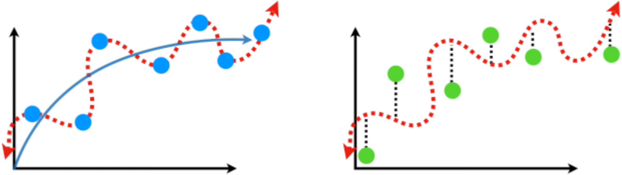
- Low testing loss with linear regression model. It is called low variance



- High testing loss with polynomial regression model. It is called high variance



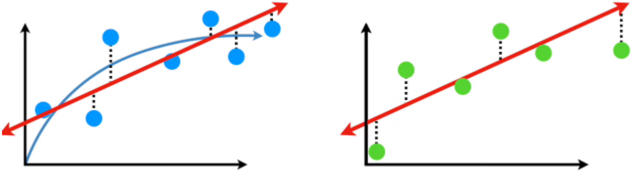
Polynomial regression model fits really well for training dataset but not for the testing dataset. This issue is called **over fitting**[Low Bias and High



Variance].

Email : dr.vvr.research@gmail.com

Linear regression model not fits well for training dataset but may be performing well on the testing dataset. This issue is called **under fitting**[High Bias and High Variance].



Email : dr.vvr.research@gmail.com
(Dr.Venkataramana Veeramsetty)

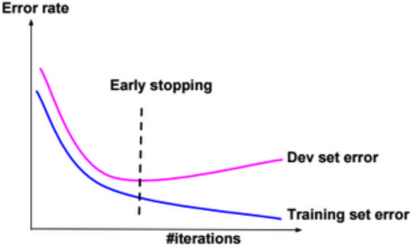
Methods to Avoid Over fitting in Neural Networks

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

√ Reduce complexity of neural network
- Reduce number of hidden layers
- Reduce number of hidden neurons (Dropout)

√ Retraining Neural Networks
- Running the same neural network model on the same training set, but each time with different initial weights.

√ Multiple Neural Networks
- Train several neural networks in parallel, with the same structure but each with different initial weights, and average their outputs.



√ Early Stopping

- The error on the validation set begins to rise
- If the training is unsuccessful, for example in a case where the error rate increases gradually over several iterations.
- If the training's improvement is insignificant, for example, the improvement rate is lower than a set threshold

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

√ Regularization

- Involves slight modification in the error function
- Add a term to the error function, which is intended to decrease the weights and biases, smoothing outputs and making the network less likely to overfit
- Properly tune performance ration (γ) between 0 and 1,so that network will not overfit
- Make self adaptive performance ration (γ), so that network will perform well

√ Dropout

- Kill some percentage of hidden neurons in each training iteration

Methods to Avoid under fitting in Neural Networks

√ Increase complexity of neural network

- Increase number of hidden layers
- Increase number of input neurons [Keep less or zero dropout]

√ Improve training set

- Increase number of samples
- Increase variance in the training set

√ Regularization

- Keep low (zero or appox.zero) regularization performance ratio

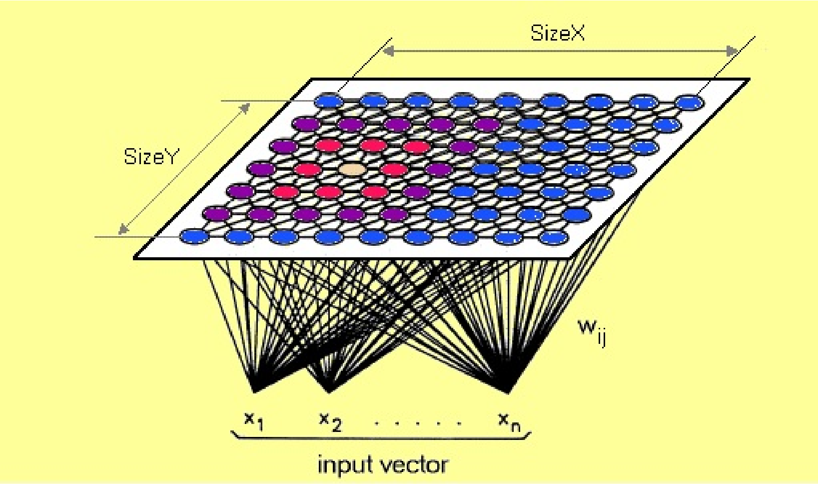
Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Kohonen Self Organizing Maps

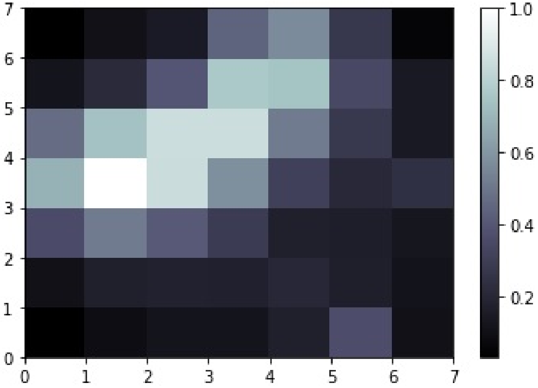
- The Self-Organizing Map was developed by professor Kohonen. The SOM has been proven useful in many applications
- One of the most popular neural network models. It belongs to the category of competitive learning networks.
- Use the SOM for clustering data without knowing the class memberships of the input data. The SOM can be used to detect features inherent to the problem and thus has also been called SOFM, the Self-Organizing Feature Map.

Kohonen Self Organizing Maps Cont.

- The weights are adjusted such that topology closed output nodes sense similar inputs. This is called as self organizing.


SizeX
SizeY
w_{ij}
x_1 x_2 . . . . . x_n
input vector

Clustering Data



Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

SOM Algorithm

Step-1 Initialize the weights between input and output layers W_{ij}

Step-2 Set input radius N_c and learning rate η

Step-3 Present input pattern to the network and calculate euclidean distance using equation (83)

$$ED_j = \sqrt{\sum_{i=1}^{n_f}(X_{pi} - Wji)} \tag{83}$$

Step-4 Update the weights connected to winning neuron using equations (84) and (85)

$$\Delta W_{ij} = \eta(X - W_{ij}^{WinningNeuron}) \tag{84}$$

$$W_{ij}^{WinningNeuron} = W_{ij}^{WinningNeuron} + \Delta W_{ij} \tag{85}$$

Winning neuron is the neuron which has minimum euclidean distance

Step-5 Repeat Steps 3 and 4 for all patterns

Step-6 For every 10 iterations update radius **until** N_c **= 0** and learning rate using equations (86) and (87). and repeat steps 3 and 4

$$N_c = N_c - 1 \tag{86}$$

$$\eta = \eta - 0.02 \tag{87}$$

Example for K-SOM

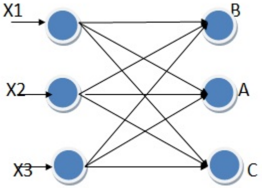


Table 1 : Input Patterns

Pattern	X1	X2	X3
1	1.1	1.7	1.8
2	0	0	0
3	0	0.5	1.5
4	1	0	0
5	0.5	0.5	0.5
6	1	1	1

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Random Weight Matrix

Table 2 : Random Weight Matrix

	1	2	3
W_a	0.2	0.7	0.3
W_b	0.1	0.1	0.9
W_c	1	1	1

Neighborhood Radius

Iteration=1 : D(t)=1

Iteration $>$ 1: D(t)=0

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Learning rate

Iteration 1, η=0.6
Iteration 2, η==0.25
Iteration>2 η=0.1

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Results

Iter	Pattern	ED_A	ED_B	ED_C	η	D(t)
1	1.1,1.7,1.8	2.01	2.09	**1.06**	0.6	1
1	0,0,0	1.91	**0.911**	2.308	0.6	1
1	0,0.5,1.5	**1.06**	1.22	1.403	0.6	1
1	1,0,0	1.48	**1.47**	1.819	0.6	1
1	0.5,0.5,0.5	**0.337**	0.397	1.06	0.6	1
1	1,1,1	0.927	0.957	**0.646**	0.6	1

Results

Iter	Pattern	ED_A	ED_B	ED_C	η	D(t)
2	1.1,1.7,1.8	1.413	1.978	**1.231**	0.25	0
2	0,0,0	1.365	**0.795**	2.264	0.25	0
2	0,0.5,1.5	1.116	1.243	**1.077**	0.25	0
2	1,0,0	1.104	**0.737**	1.989	0.25	0
2	0.5,0.5,0.5	0.501	**0.39**	1.305	0.25	0
2	1,1,1	**0.37**	1.097	0.809	0.25	0

Results

Iter	Pattern	ED_A	ED_B	ED_C	η	D(t)
2	1.1,1.7,1.8	1.326	2.13	**0.899**	0.1	0
2	0,0,0	1.456	**0.689**	2.118	0.1	0
2	0,0.5,1.5	1.133	1.33	**0.887**	0.1	0
2	1,0,0	1.177	**0.637**	1.992	0.1	0
2	0.5,0.5,0.5	0.591	**0.372**	1.296	0.1	0
2	1,1,1	**0.278**	1.139	0.804	0.1	0

Loss Functions

√ **Mean Absolute Error (MAE) :**

$$MAE = \frac{\sum_{i=1}^{n} |y^{actual,i} - y^{Predicted,i}|}{n} \tag{88}$$

√ **Mean Square Error (MSE) :**

$$MSE = \frac{\sum_{i=1}^{n} (y^{actual,i} - y^{Predicted,i})^2}{n} \tag{89}$$

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)

Loss Functions

√ **Mean Absolute Percentage Error (MAPE) :**

$$MAE = \frac{\sum_{i=1}^{n} |y^{actual,i} - y^{Predicted,i}| / y^{actual,i}}{n} * 100 \tag{90}$$

√ **Root Mean Square Error (RMSE) :**

$$MSE = \sqrt{\frac{\sum_{i=1}^{n} (y^{actual,i} - y^{Predicted,i})^2}{n}} \tag{91}$$

References

- https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f
- https://www.analyticsvidhya.com/blog/2019/08/det guide-7-loss-functions-machine-learning-python-code/

Email : dr.vvr.research@gmail.com
(Dr. Venkataramana Veeramsetty)