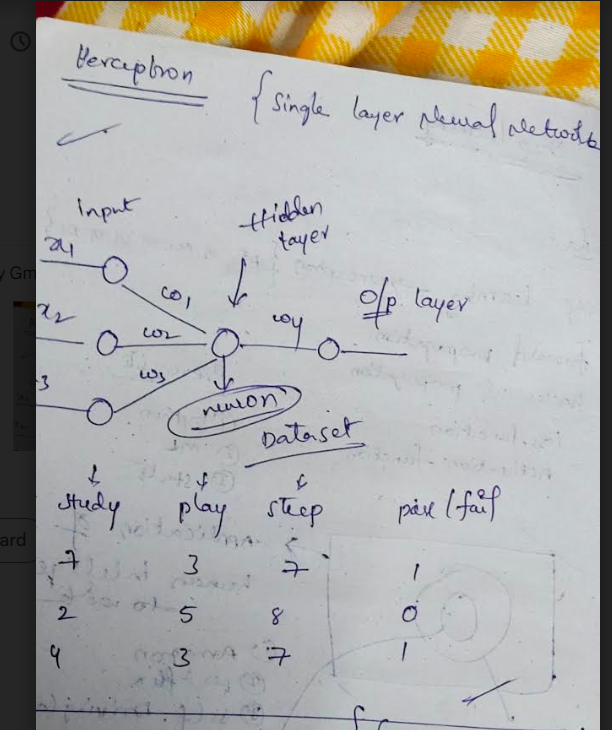
1. Study and document the evolution of ANN from Perceptron with derivation.

Deep learning is a branch of [machine learning](https://www.geeksforgeeks.org/introduction-machine-learning/) which is completely based on [artificial neural networks](https://www.geeksforgeeks.org/tag/neural-network/), as neural network is going to mimic the human brain so deep learning



2. Document problems generally faced in training an ANN and their solution

provided in the lecture with derivation.

**1. Feedforward Neural Network**

The feedforward neural network is one of the most basic artificial neural networks. In this ANN, the data or the input provided travels in a single direction. It enters into the ANN through the input layer and exits through the output layer while hidden layers may or may not exist. So the feedforward neural network has a front propagated wave only and usually does not have backpropagation.

**2. Recurrent Neural Network**

The Recurrent Neural Network saves the output of a layer and feeds this output back to the input to better predict the outcome of the layer. The first layer in the RNN is quite similar to the feed-forward neural network and the recurrent neural network starts once the output of the first layer is computed. After this layer, each unit will remember some information from the previous step so that it can act as a memory cell in performing computations.

**3. Convolutional Neural Network**

A Convolutional neural network has some similarities to the feed-forward neural network, where the connections between units have weights that determine the influence of one unit on another unit. But a CNN has one or more than one convolutional layers that use a convolution operation on the input and then pass the result obtained in the form of output to the next layer. CNN has applications in speech and image processing which is particularly useful in computer vision.

**4. Modular Neural Network**

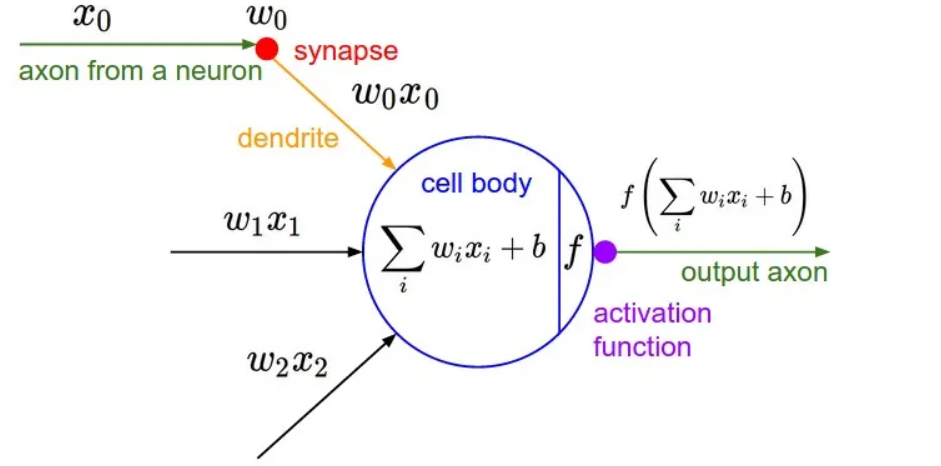
A Modular Neural Network contains a collection of different neural networks that work independently towards obtaining the output with no interaction between them. Each of the different neural networks performs a different sub-task by obtaining unique inputs compared to other networks. The advantage of this modular neural network is that it breaks down a large and complex computational process into smaller components, thus decreasing its complexity while still obtaining the required output.

**5. Radial basis function Neural Network**

Radial basis functions are those functions that consider the distance of a point concerning the center. RBF functions have two layers. In the first layer, the input is mapped into all the Radial basis functions in the hidden layer and then the output layer computes the output in the next step. Radial basis function nets are normally used to model the data that represents any underlying trend or function.

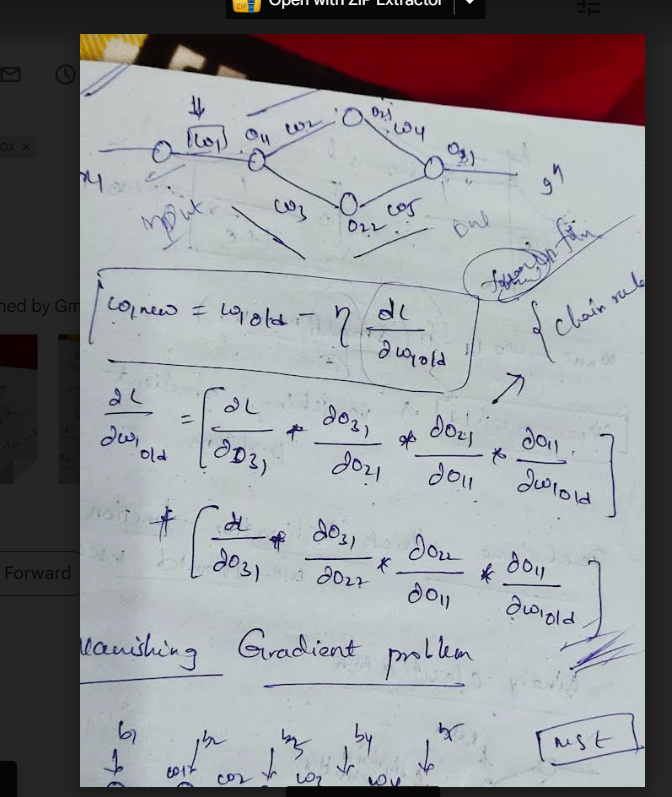
3. Compare and document different available activation functions.

It is necessary to start by introducing the non-linear activation functions, which is an alternative to the best known **sigmoid**  function. It is important to remember that many different conditions are important when evaluating the final performance of activation functions. It is necessary to draw attention to the importance of mathematics and the derivative process at this point



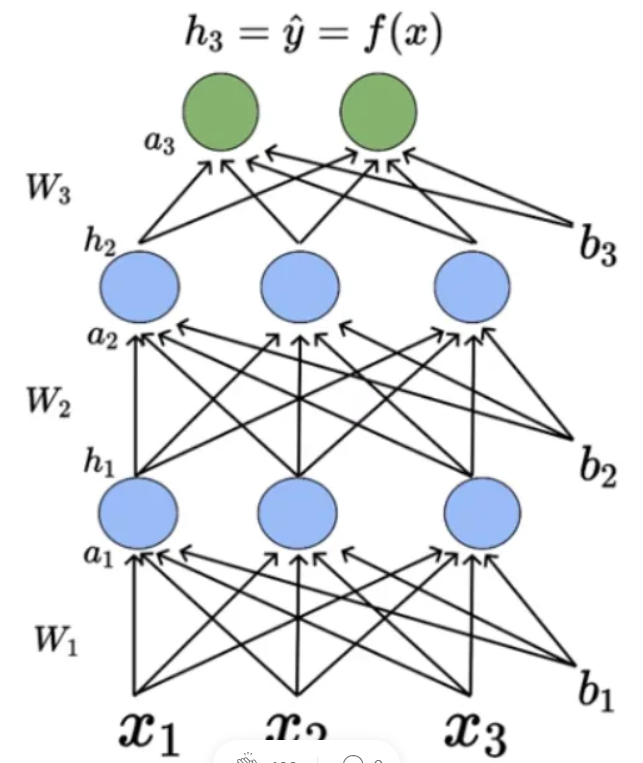
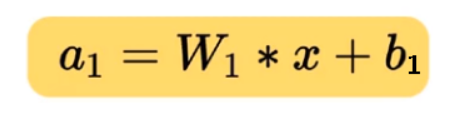
4 Compare and document different available weight initialization techniques.

Weights ate calculated in single neural perceptron& multi nueral perceptron



5. Observe and document results before and after applying Batch Normalization.

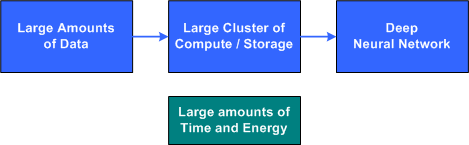
By normalizing the inputs we are able to bring all the inputs features to the same scale. In the neural network, we need to compute the pre-activation for the first neuron of the first layer a₁₁. We know that pre-activation is nothing but the weighted sum of inputs plus bias. In other words, it is the dot product between the first row of the weight matrix W₁ and the input matrix **X** plus bias b₁₁



he mathematical equation for pre-activation at each layer ‘i’ is given by,

6) Observe and document results before and after applying Transfer Learning.

Since the introduction of deep learning, there's been a renewed interest in neural networks for a range of applications. Deep learning has solved problems viewed as impossible not more than a decade ago. But deep learning neural networks require large clusters of compute servers, large amounts of training data, and a large amount of time to train the deep neural network.

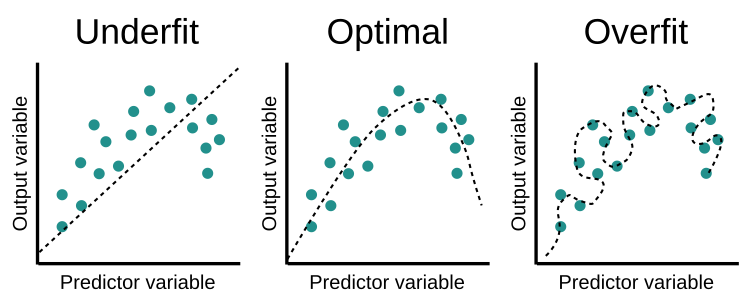


he deep neural network is represented by tens of millions of weights that connect the many layers of neurons of the networks together. These weights (typically real values) are adjusted during the training process and applied to inputs (including inputs from intermediary layers) to feed forward to an output classification. The basic idea of transfer learning is then to start with a deep learning network that is pre-initialized from training of a similar problem. Using this network, a smaller duration of training is required for the new, but related, problem

7)Observe and document use of Early Stopping and Check-pointing.

Deep learning is a compelling field of computer science used to solve many real-life problems such as object detection and product recommendation with great accuracy. It involves various complex algorithms which try to imitate the functioning of the human brain. To get high accuracy, we must train our model on a large amount of training data. It may seem that the more we train our model on the training data, the more will be the accuracy. But if you train your model for long, the prediction accuracy decreases. This is due to a problem known as overfitting. Let us look at this problem and how we can tackle this problem by using early stopping.

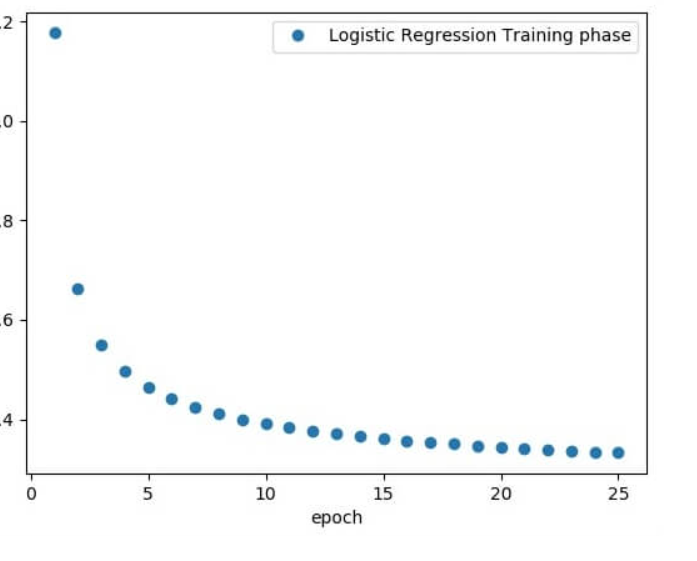
Overfitting is a common problem in the realm of deep learning. It happens when our model fits very closely to our training data. It will lead to excellent training accuracy on the trained dataset. But outside the dataset, the predictions will be poor. Our model will just memorize the training data and will not perform well for any unseen data.



8) Compare and document different available Optimizers and their derivation.

1. # Import the MINST dataset
2. from tensorflow.examples.tutorials.mnist **import** input\_data
3. mnist = input\_data.read\_data\_ ("/tmp/data/", one\_hot=True)
5. **import** tensorflow as tf
6. **import** matplotlib.pyplot as plt
7. # Parameters
8. learning\_rate = 0.01
9. training\_epochs = 25
10. batch\_size = 100
11. display\_step = 1
13. # tf Graph Input
14. x = tf.placeholder("float", [none, 784]) # MNIST data image of shape 28\*28 = 784
15. y = tf.placeholder("float", [none, 10]) # 0-9 digits recognition => 10 classes
16. # Create model
17. # Set model weights
18. W = tf.Variable(tf.zeros([784, 10]))
19. b = tf.Variable(tf.zeros([10]))
20. # Constructing the model
21. activation=tf.nn.softmaxx(tf.matmul (x, W)+b) # Softmax
22. of function
23. # Minimizing error using cross entropy
24. cross\_entropy = y\*tf.log(activation)
25. cost = tf.reduce\_mean\ (-tf.reduce\_sum\ (cross\_entropy, reduction\_indice = 1))
26. optimizer = tf.train.GradientDescentOptimizer(learning\_rate).minimize(cost)
27. #Plot settings
28. avg\_set = []
29. epoch\_set = []
30. # Initializing the variables where init = tf.initialize\_all\_variables()
31. # Launching the graph
32. with tf.Session() as sess:
33. sess.run(init)
35. # Training of the cycle in  the dataset
36. **for** epoch in range(training\_epochs):
37. avg\_cost = 0.
38. total\_batch = **int**(mnist.train.num\_example/batch\_size)
40. # Creating loops at all the batches in the code
41. **for** i in range(total\_batch):
42. batch\_xs, batch\_ys = mnist.train.next\_batch(batch\_size)
43. # Fitting the training by the batch data sess.run(optimizr,  feed\_dict = {
44. x: batch\_xs, y: batch\_ys})
45. # Compute all the average of loss avg\_cost += sess.run(cost, \ feed\_dict = {
46. x: batch\_xs, \ y: batch\_ys}) //total batch
47. # Display the logs at each epoch steps
48. **if** epoch % display\_step==0:
49. print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format (avg\_cost))
50. avg\_set.append(avg\_cost) epoch\_set.append(epoch+1)
51. print ("Training phase finished")
53. plt.plot(epoch\_set,avg\_set, 'o', label = 'Logistics Regression Training')
54. plt.ylabel('cost')
55. plt.xlabel('epoch')
56. plt.legend()
57. plt.show()
59. # Test the model
60. correct\_prediction = tf.equal (tf.argmax (activation, 1),tf.argmax(y,1))
62. # Calculating the accuracy of dataset
63. accuracy = tf.reduce\_mean(tf.cast (correct\_prediction, "float")) print
64. ("Model accuracy:", accuracy.eval({x:mnist.test.images, y: mnist.test.labels}))

Output



9. Observe and document the use of various loss functions.

In the context of an optimization algorithm, the function used to evaluate a candidate solution (a set of weights) is referred to as the objective function.

We may seek to maximize or minimize the objective function, meaning that we are searching for a candidate solution that has the highest or lowest score respectively.

Typically, with neural networks, we seek to minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply “loss.”

*The function we want to minimize or maximize is called the objective function or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function.*

The cost or loss function has an important job in that it must faithfully distill all aspects of the model down into a single number in such a way that improvements in that number are a sign of a better model.

*The cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared.*

*10) Observe and document results before and after applying various*

*regularization techniques like l1, l2, and dropout techniques. (Will be covered this*

*weekend)*

L1 regularization is the preferred choice when having a high number of features as it provides sparse solutions. Even, we obtain the computational advantage because features with zero coefficients can be avoided.

Mathametical formulas

For instance, we define the simple linear regression model Y with an independent variable to understand how L2 regularization works.

For this model, W and b represents **“weight”**and **“bias”** respectively, such as

**W= w1, w2, w3, ......... wn**

And,

**b=b1, b2, b3, ......... bn**

And Ŷ is the predicted result such that

**Ŷ= w1 x1 +w2 x2 +......+wn xn, + b**

The below function calculates an error without the regularization function

**Loss= Error (Y, Ŷ**