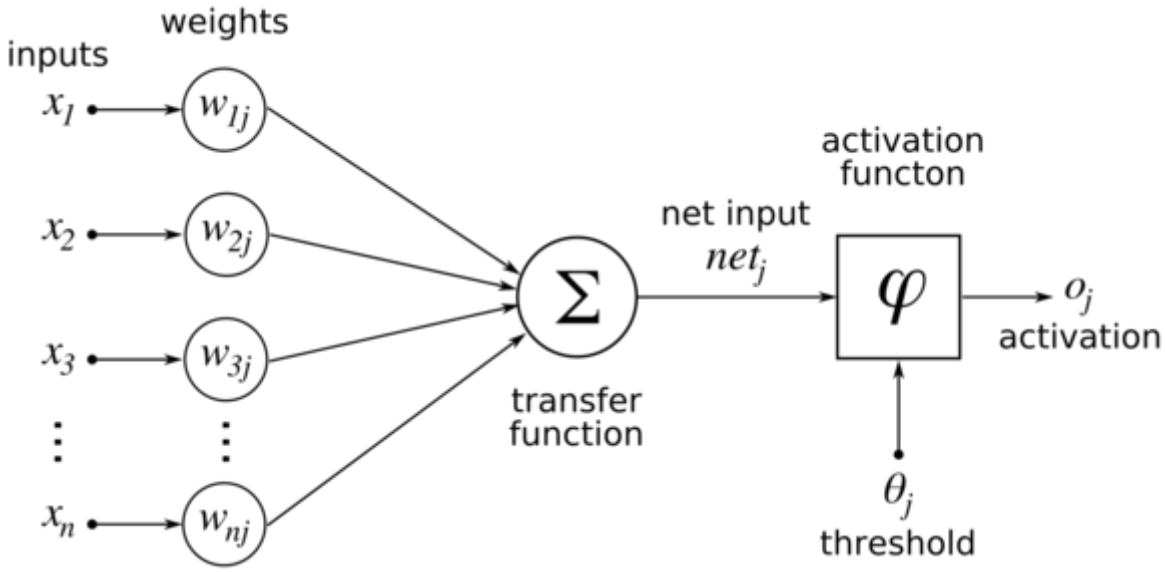
How to improve performance of Neural Networks



Neural networks have been the most promising field of research for quite some time. Recently they have picked up more pace. In earlier days of neural networks, it could only implement single hidden layers and still we have seen better results.
Deep learning methods are becoming exponentially more important due to their demonstrated success at tackling complex learning problems. At the same time, increasing access to high-performance computing resources and state-of-the-art open-source libraries are making it more and more feasible for enterprises, small firms, and individuals to use these methods.

Neural network models have become the center of attraction in solving machine learning problems.

Now, What's the use of knowing something when we can't apply our knowledge intelligently. There are various problems with neural networks when we

implement them and if we don't know how to deal with them, then so-called "Neural Network" becomes useless.

Some Issues with Neural Network:

1. Sometimes neural networks fail to converge due to low dimensionality.
2. Even a small change in weights can lead to significant change in output. sometimes results may be worse.
3. The gradient may become zero . In this case , weight optimization fails.
4. Data overfitting.
5. Time complexity is too high. Sometimes algorithm runs for days even on small data set.
6. We get the same output for every input when we predict.

So what next!!

One day I sat down(I am not kidding!) with neural networks to check What can I do for better performance of neural networks. I have tried and tested various use cases to discover solutions.
Let's dig deeper now. Now we'll check out the proven way to improve the performance(Speed and Accuracy both) of neural network models:

1. Increase hidden Layers

we have always been wondering what happens if we can implement more hidden layers!! In theory, it has been established that many of the functions will converge in a higher level of abstraction. So it seems more layers better results

Multiple hidden layers for networks are created using the mlp function in the RSNNS package and neuralnet in the neuralnet package. As far as I know, these are the only neural network functions in R that can create multiple hidden layers(I am not talking about Deep Learning here). All others use a single hidden layer. Let's start exploring the neural net package first.

I won't go into the details of the algorithms. You can google it yourself about their training process. I have used a data set and want to predict Response/Target variable. Below is a sample code for 4 layers.

R code

A. Neuralnet Package

```
library(neuralnet)

set.seed(1000000)

multi_net = neuralnet(action_click~ FAL_DAYS_last_visit_index+NoofSMS_30days_index+offer_index+Days_last_SMS_index+camp_catL3_index+Index_weekday , algorithm= 'rprop+', data=train, hidden = c(6,9,10,11) ,stepmax=1e9 , err.fct = "ce" ,linear.output =F)
```

I have tried several iteration. Below are the confusion matrix of some of the results

```
Case 1: One layer
      0  1
0 14 12
1 10 14
Case 2: Two layers
      0  1
0 14 12
1 10 14
Case 3: Four layers
    FALSE TRUE
0      11   15
1       8   16
Case 4: Six layers
    FALSE TRUE
0      12   14
1       7   17
Case 5: Seven layers
    FALSE TRUE
0      11   15
1       8   16
Case 6: Nine layers
      0  1
0 14 12
1  8 16
```

B. RSNNS Package

library(RSNNS)
set.seed(10000)

a = mlp(train[,2:7], train$action_click, size = c(5,6), maxit = 5000,

initFunc = “Randomize_Weights”, initFuncParams = c(-0.3, 0.3),

learnFunc = “Std_Backpropagation”, learnFuncParams = c(0.2,0),

hiddenActFunc = “Act_Logistic”, shufflePatterns = TRUE, linOut = FALSE)

I have tried several iteration. Below are the confusion matrix of some of the results.

```
Case 1: Two layer
      0   1
0 16 10
1 14 10
Case 2: Three layers
      0   1
0 17 10
1 14 10
Case 3: Five layers
    FALSE TRUE
0      11    15
1       8    16
```

From my experiment, I have concluded that when you increase layers, it may result in better accuracy but it's not a thumb rule. You have to just test it with a different number of layers. I have tried several data set with several iterations and it seems neuralnet package performs better than RSNNS. Always start with single layer then gradually increase if you don't have performance improvement .

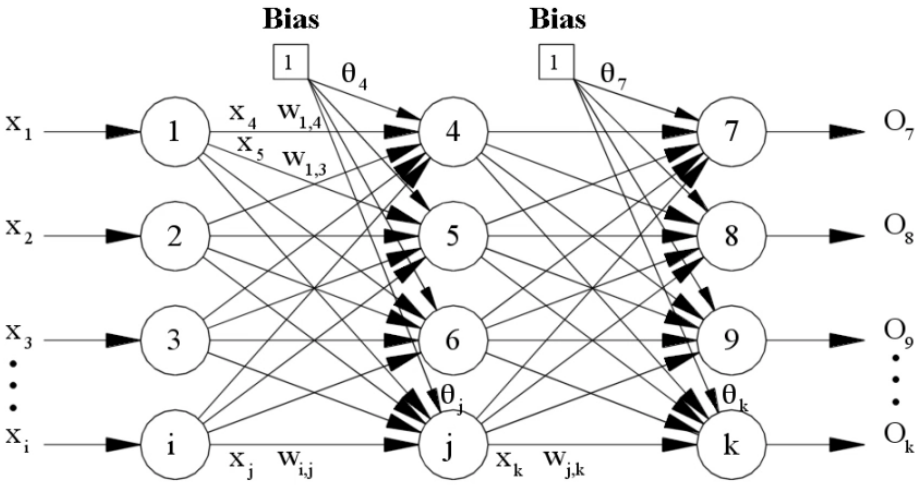


Figure 2 . A multi layered Neural Network

2. Change Activation function

Changing activation function can be a deal breaker for you. I have tested results with sigmoid, tanh and Rectified linear units. Simplest and most successful activation function is rectified linear unit. Mostly we use sigmoid function network. Compared to sigmoid, the gradients of ReLU does not approach zero

when x is very big. ReLU also converges faster than other activation function. You should know how to use these activation function i.e. when you use “tanh” activation function you should categorize your binary classes into “-1” and “1”. The classes encoded in 0 and 1 , won’t work in tanh activation function.

3. Change Activation function in Output layer

I have experimented with trying a different activation function in output layer than that of in hidden layers. In some cases, results were better so its better to try with different activation function in output neuron.

As with the single-layered ANN, the choice of activation function for the output layer will depend on the task that we would like the network to perform (i.e. categorization or regression). However, in multi-layered NN, it is generally desirable for the hidden units to have nonlinear activation functions (e.g. logistic sigmoid or tanh). This is because multiple layers of linear computations can be equally formulated as a single layer of linear computations. Thus using linear activations for the hidden layers doesn’t buy us much. However, using linear activations for the output unit activation function (in conjunction with nonlinear activations for the hidden units) allows the network to perform nonlinear regression.

4. Increase number of neurons

If an inadequate number of neurons are used, the network will be unable to model complex data, and the resulting fit will be poor. If too many neurons are used, the training time may become excessively long, and, worse, the network may overfit the data. When overfitting $ occurs, the network will begin to model random noise in the data. The result is that the model fits the training data extremely well, but it generalizes poorly to new, unseen data. Validation must be used to test for this.

There is no rule of thumb in choosing number of neurons but you can consider this one –

N is number of hidden neurons–

- N = 2/3 the size of the input layer, plus the size of the output layer.
- N < twice the size of the input layer

5. Weight initialization

While training neural networks, first-time weights are assigned randomly. Although weight updation does take place, but sometimes neural network can converge in local minima. When we use multilayered architecture, random weights does not perform well. We can supply optimal initial weights. You should try with different random seed to generate different random weights then choose the seed number which works well for your problem.
You can use methods like Adaptive weight initialization, Xavier weight initialization etc to initialize weights.

The random values of initial synaptic weights generally lead to a big error. So learning is finding a proper value for the synaptic weights, in order to find the minimum value for output error. below figure shows being trapped in local minima in order to find optimal weights–

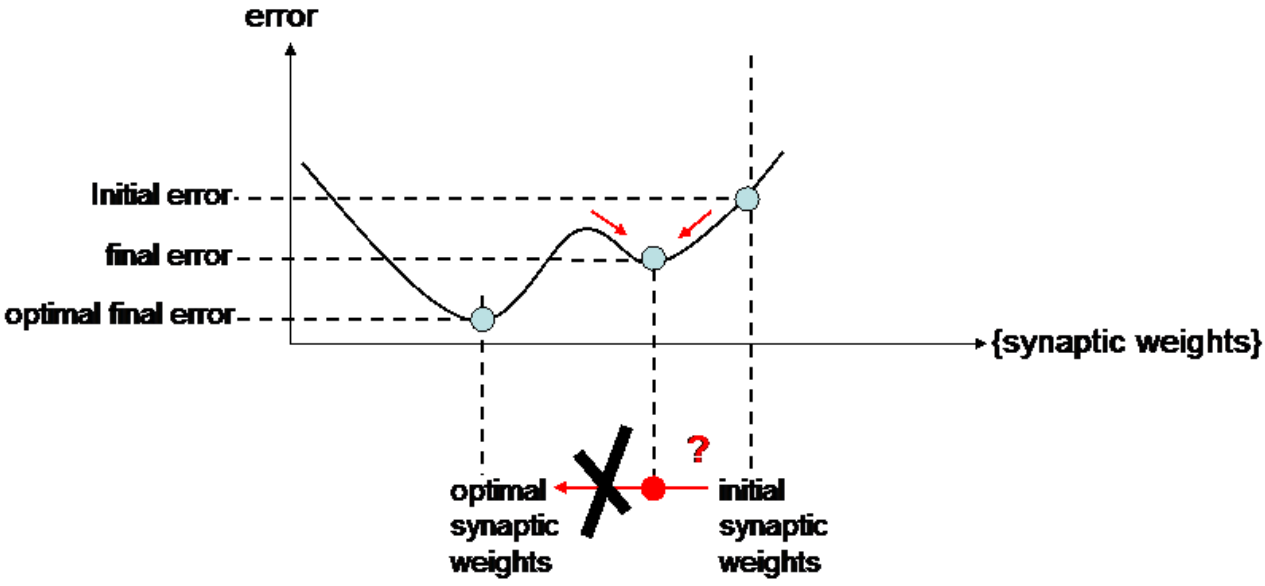


Figure 3: Local minima problem due to random initialization of weights

6. More data

When We have lots of data , then neural network generalizes well. otherwise, it may overfits data. So it's better to have more data. Overfitting is a general problem when using neural networks. The amount of data needed to train a neural network is very much problem-dependent. The quality of training data (i.e., how well the available training data represents the problem space) is as important as the quantity (i.e., the number of records, or examples of input-output pairs). The key is to use training data that generally span the problem data space. For relatively small datasets (fewer than 20 input variables, 100 to several thousand records) a minimum of 10 to 40 records (examples) per input variable is recommended for training. For relatively large datasets (more than 20 000 records), the dataset should be sub-sampled to obtain a smaller dataset that contains 30 – 50 records per input variable. In either case, any "extra" records should be used for validating the neural networks produced.

7. Normalizing/Scaling data

Most of the times scaling/normalizing your input data can lead to improvement. There are a variety of practical reasons why standardizing the inputs can make training faster and reduce the chances of getting stuck in local optima. Also, weight decay and Bayesian estimation can be done more conveniently with standardized inputs. When NN use gradient descent to optimize parameters , standardizing covariates may speed up convergence (because when you have unscaled covariates, the corresponding parameters may inappropriately dominate the gradient).

8. Change learning algorithm parameters

Try different learning rates (0.01 to 0.9). Also try different momentum parameters, if your algorithm supports it (0.1 to 0.9). Changing learning rate parameter can help us to identify if we are getting stuck in local minima.

The two plots below nicely emphasize the importance of choosing learning rate by illustrating two most common problems with gradient descent:

(i) If the learning rate is too large, gradient descent will overshoot the minima and diverge.

(ii) If the learning rate is too small, the algorithm will require too many epochs to converge and can become trapped in local minima more easily.

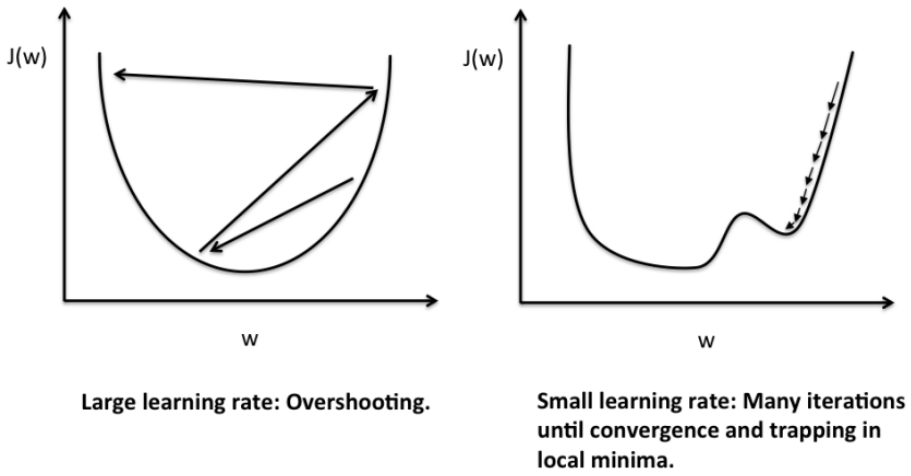
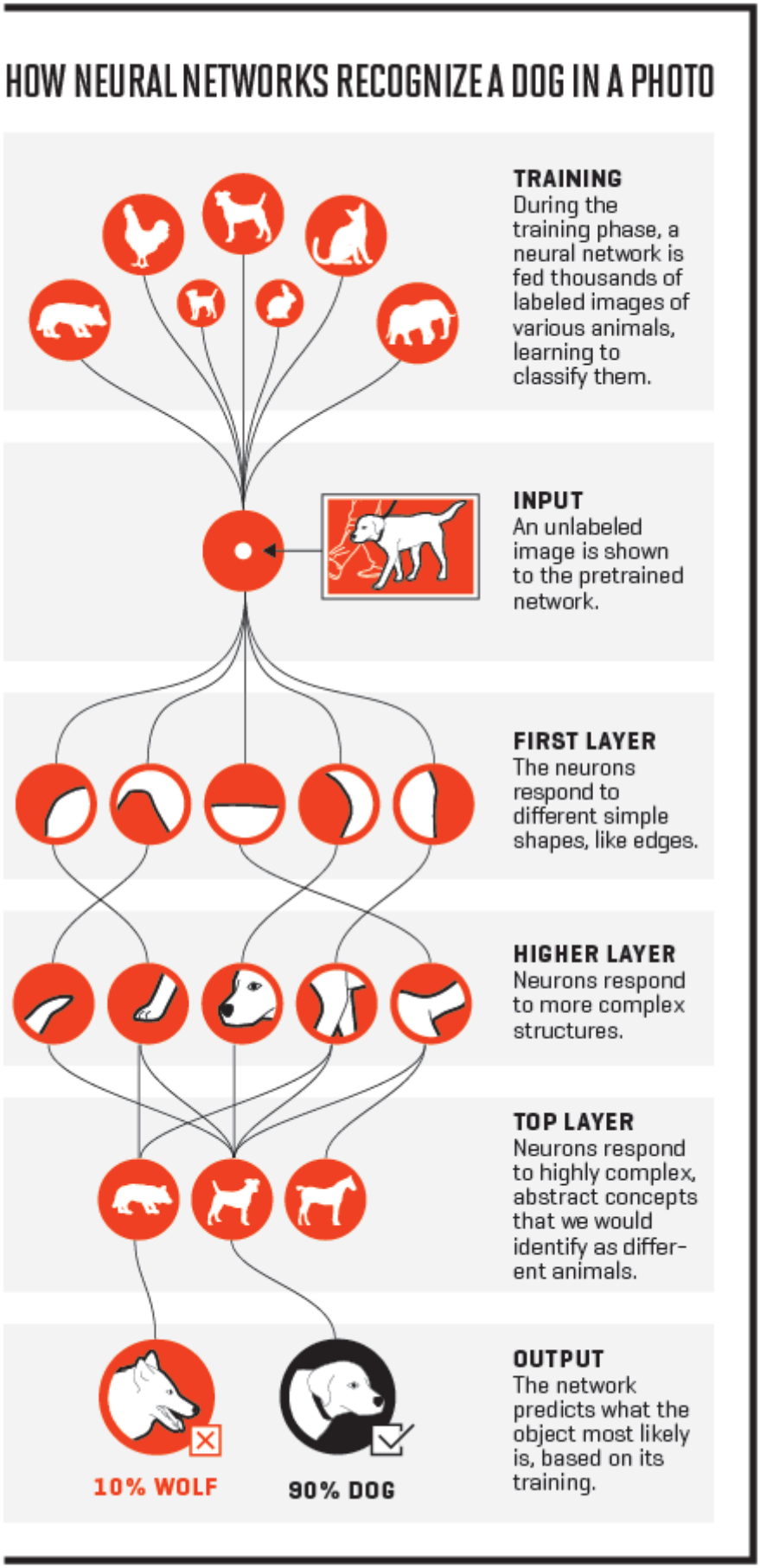
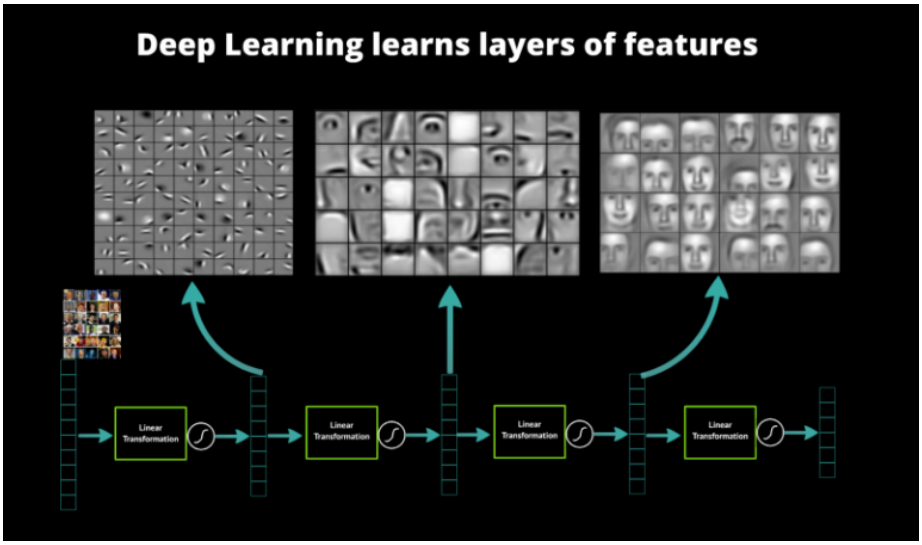


Figure 4 : Effect of learning rate parameter values

9. Deep learning for auto feature generation

Machine learning is one of the fastest-growing and most exciting fields out there, and deep learning represents its true bleeding edge. Usual neural networks are not efficient in creating features. Like other machine learning models, Neural networks algorithm's performance also depends on the quality of features. If we have better features then we would have better accuracy. When we use deep architecture then features are created automatically and every layer refines the features. i.e.


HOW NEURAL NETWORKS RECOGNIZE A DOG IN A PHOTO
TRAINING
During the training phase, a neural network is fed thousands of labeled images of various animals, learning to classify them.
INPUT
An unlabeled image is shown to the pretrained network.
FIRST LAYER
The neurons respond to different simple shapes, like edges.
HIGHER LAYER
Neurons respond to more complex structures.
TOP LAYER
Neurons respond to highly complex, abstract concepts that we would identify as different animals.
OUTPUT
The network predicts what the object most likely is, based on its training.
10% WOLF
90% DOG



10. Misc- You can try with a different number of epoch and different random seed. Various parameters like dropout ratio, regularization weight penalties, early stopping etc can be changed while training neural network models.

To improve generalization on small noisy data, you can train multiple neural networks and average their output or you can also take a weighted average. There are various types of neural network model and you should choose according to your problem. i.e. while doing stock prediction you should first try Recurrent Neural network models.

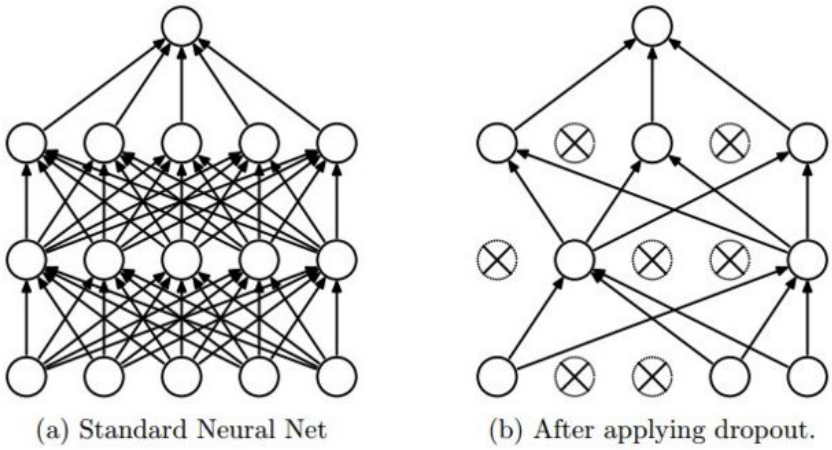


Figure 5 : After dropout, insignificant neurons do not participate in training

References:

1. http://stats.stackexchange.com/
2. http://stackoverflow.com/
3. https://www.quora.com/
4. http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html
5. http://www.nexyad.net/html/upgrades%20site%20nexyad/e-book-Tutorial-Neural-Networks.html

Related:

Deep Learning Explained in laymen terms

In "Deep Learning"

Difference between Usual Machine Learning and Deep Learning Explained!

In "Machine Learning"

Feature Learning , Deep Learning and Machine learning

In "Deep Learning"

Author: Ved

I am working as Data Scientist to assist organizations to take better decisions by providing actionable insights by applying data science,maths and business knowledge to the Data. View all posts by Ved

---

Ved / September 29, 2016 / Deep Learning, Machine Learning, Neural Network, Uncategorized