Human Activity Recognition Using Sensor Data Through Deep Learning Techniques

Submitted By
Nirmal Raj Eganathan
Deekshith Basvoju
Batch 56 – INSOFE, Bengaluru.

Guide: Dr . Soumen Manna, Senior Data Scientist.

International School of Engineering, Bengaluru.
25th September 2019

TABLE OF CONTENTS

ABSTRACT

Human-centred computing is an emerging research field that aims to understand human behaviour and integrate users and their social context with computer systems. One of the most recent, challenging and appealing applications in this framework consists in sensing human body motion using smartphones to gather context information about people actions. In this context, we describe in this work an Activity Recognition database, built from the recordings of 30 subjects doing Activities of Daily Living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors, which is released to public domain on a well-known on-line repository.

Human Activity Recognition database built from the recordings of 30 subjects performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors.

1.INTRODUCTION

1.1. DATA SET INFORMATION

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cut off frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

Follow this link to see a video of the 6 activities recorded in the experiment with one of the participants

1.2. METHODOLOGY FOR DATA COLLECTION

A set of experiments were carried out to obtain the HAR dataset. A group of 30 volunteers with ages ranging from 19 to 48 years were selected for this task. Each person was instructed to follow a protocol of activities while wearing a waist-mounted Samsung Galaxy S II smartphone. The six selected ADL were standing, sitting, laying down, walking, walking downstairs and upstairs. Each subject performed the protocol twice: on the first trial the smartphone was fixed on the left side of the belt and on the second it was placed by the user himself as preferred. There is also a separation of 5 seconds between each task where individuals are told to rest, this facilitated repeatability (every activity is at least tried twice) and ground trough generation through the visual interface. The tasks were performed in laboratory conditions but volunteers were asked to perform freely the sequence of activities for a more naturalistic dataset.

1.3. PROBLEM STATEMENT

Human activity recognition (HAR) is gaining importance due to wearables and sensors data associated with it. Different from traditional Pattern Recognition methods, deep learning can largely relieve the effort on designing features and can learn much more high-level and meaningful features by training an end-to-end neural network. You need to find the dataset from open resources which can provide you human activity data. Train a deep net model on this data and analyse your findings.

2. RELATED WORK

Many HAR systems adopted various classification algorithms like the decision tree, Markov models, domain transform, fuzzy logic, support vector machine (SVM), regression methods, artificial neural networks (ANN), K-nearest neighbour (KNN), etc. [1]. Wei et al. [2] proposed the two-layer Hidden Markov Model (HMM) for continues and long-term daily activity monitoring which uses wearable body sensors. Andreu and Angelo [3] suggested the fuzzy based algorithm for real-time human activity recognition. In this work, rule-based and self-learning fuzzy classifier extracts class information from wearable wireless accelerometers. After the extensive analysis, Janidarmian et al. [4] found that the KNN with ensembles methods shows the best robust results among the other machine-learning models. Rodriguez et al. [5] reported Bio Harness and smartphone-based activity recognition using decision tree for the classification. Zebin et al. [6] conducted the comparison between different HAR algorithms for the inertial sensor data. Tang and Sazonov [9] compared the ANN and SVM classification algorithms for the smart shoe. From the accuracy results, Tang et al. concluded that the ANN classification algorithms perform well compared with the SVM algorithm. Therefore, we adopted ANN as an activity classification algorithm for the workforce monitoring.

Many applications adopted ANN for classification, control and calibration tasks in biomedical instrumentation [7], [8], control systems [10], [11], non-conventional energy production etc. However, the efficient implementation and real-time execution of ANN models on embedded platforms are still a challenging problem because it involves plenty of nonlinear activation functions, numerous amounts of additions and multiplications. The implementation of these algorithms demands high performance parallel computing devices like FPGA's. Because of an inherently parallel architecture of ANN algorithms, FPGA

devices are becoming a favourable choice compared with sequential devices. The modern FPGA has specialized blocks like DSP and BRAM to handle complex mathematical operations. The soft-core and hard-core processor are also available to enhance sequential capability. This makes the FPGA technology most suitable for the implementation of soft-computing algorithms (like ANN) for real-time applications.

3.Ensemble Techniques: -

3.1Random Forest Classifier: -

Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object. Suppose training set is given as: [X1, X2, X3, X4] with corresponding labels as [L1, L2, L3, L4], random forest may create three decision trees taking input of subset for example,

[X1, X2, X3]
[X1, X2, X4]
[X2, X3, X4]

So finally, it predicts based on the majority of votes from each of the decision trees made.

Why use Random Forest Machine Learning Algorithm?

1. It maintains accuracy when there is missing data and is also resistant to outliers.ie we could have run this model on our original data without removing customers with NA values.
2. Capable of handling numerical, binary and categorical features, without scaling, transformation or modification. Our data has variables which have largely different values
3. Implicit feature selection as it gives estimates on what variables are important in the classification. We have few features and the models select what it presumes well.

Pros and cons: -

1. Both training and prediction are very fast, because of the simplicity of the underlying decision trees. In addition, both tasks can be straightforwardly parallelized, because the individual trees are entirely independent entities.
2. The multiple trees allow for a probabilistic classification: a majority vote among estimators gives an estimate of the probability.
3. The nonparametric model is extremely flexible, and can thus perform well on tasks that are under-fit by other estimators.

A primary disadvantage of random forests is that the results are not easily interpretable: that is, if you would like to draw conclusions about the *meaning* of the classification model, random forests may not be the best choice.

3.2 Xg-Boost:

Xg-Boost (extreme Gradient Boosting) is an advanced implementation of the gradient boosting algorithm. Xg-Boost has proved to be a highly effective ML algorithm, extensively used in machine learning competitions and hackathons. Xg-Boost has high predictive power and is almost 10 times faster than the other gradient boosting techniques. It also includes a variety of regularization which reduces overfitting and improves overall performance. Hence it is also known as **regularized boosting** technique.

Let us see how Xg-Boost is comparatively better than other techniques:

1. Regularization:
   Standard GBM implementation has no regularization like Xg-Boost. Thus Xg-Boost also helps to reduce overfitting.
2. Parallel Processing:
   Xg-Boost implements parallel processing and is faster than GBM. Xg-Boost also supports implementation on Hadoop.
3. High Flexibility:
   Xg-Boost allows users to define custom optimization objectives and evaluation criteria adding a whole new dimension to the model.
4. Handling Missing Values:
   Xg-Boost has an in-built routine to handle missing values.
5. Tree Pruning:
   Xg-Boost makes splits up to the max_depth specified and then starts pruning the tree backward and removes splits beyond which there is no positive gain.
6. Built-in Cross-Validation:
   Xg-Boost allows a user to run cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

3.3 Bagging Classifier:

Bagging (Bootstrap Aggregating) is a widely used an ensemble learning algorithm in machine learning. The algorithm builds multiple models from randomly taken subsets of train dataset and aggregates learners to build overall stronger learner. In this post, we'll learn how to classify data with Bagging Classifier class of a sklearn library in Python.

4.Support Vector Machines:

SVM can be used to solve both classification and regression problems.
The idea of SVM is to find nonlinear boundaries by constructing a linear boundary in a large, transformed version of the feature space.
SVM's are classified into two categories:

1. Linear SVM's – In linear SVM's the training data i.e. classifiers are separated by a hyper-plane.

2.Non-Linear SVM's- In non-linear SVM's it is not possible to separate the training data using a hyper-plane. Under such conditions, the training data is too complex that it is impossible to find a representation for every feature vector.

Let's look at the basic terminology used with SVM:

Kernel: The function used to map a lower dimensional data into higher dimensional data.

Hyper Plane: In SVM this is basically a separation line between the data classes. In SVR, it is the line that will help us to predict the continuous value or target value.

Boundary Line: In SVM there are two lines other than hyper plane which creates a margin. The support vectors can be on the boundary line or outside.

Support Vectors: This are the data points which are closest to the boundary.

5.Dimensionality Reduction Techniques Used

5.1 PCA – Principle Component Analysis

The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables while retaining as much as possible of the variation present in the data set. This is achieved by transforming to a new set of variables, the Principle Components *(PCs)*, which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables.

Mathematics Behind PCA

PCA can be thought of as an unsupervised learning problem. The whole process of obtaining principle components from a raw dataset can be simplified in six parts.

1. Take the whole dataset consisting of *d+1 dimensions* and ignore the labels such that our new dataset becomes *d dimensional*.
2. Compute the *mean* for every dimension of the whole dataset.
3. Compute the *covariance matrix* of the whole dataset.
4. Compute *eigenvectors* and the corresponding *eigenvalues*.
5. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ *dimensional* matrix **W.**
6. Use this $d \times k$ *eigenvector matrix* to transform the samples onto the new subspace.

Follow this for detail information: - https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643

5.2 LDA – Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting ("curse of dimensionality") and also reduce computational costs.

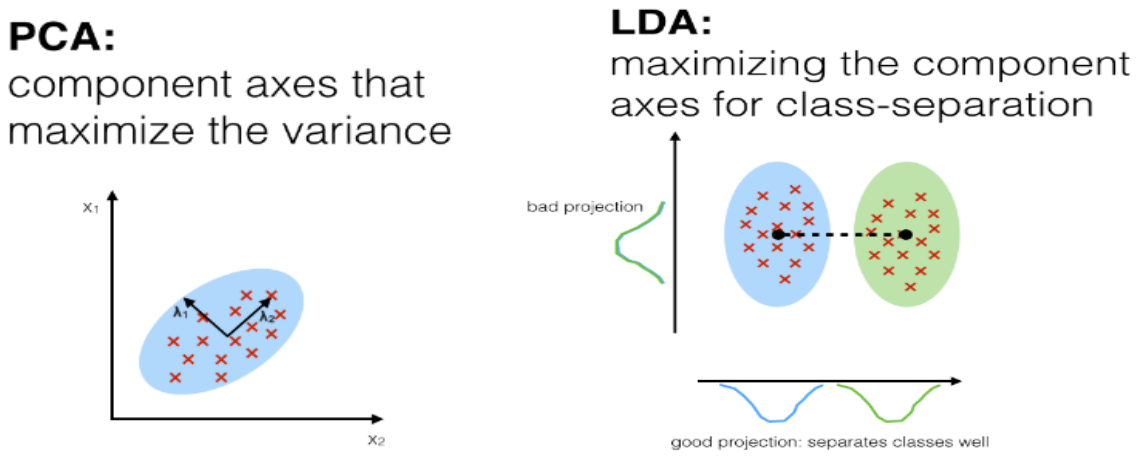
The general LDA approach is very similar to a Principal Component Analysis, but in addition to finding the component axes that maximize the variance of our data (PCA), we are additionally interested in the axes that maximize the separation between multiple classes (LDA).

So, in a nutshell, often the goal of an LDA is to project a feature space (a dataset n-dimensional samples) onto a smaller subspace **k** (where $\mathbf{k} \leq \mathbf{n-1}$) while maintaining the class-discriminatory information.

In general, dimensionality reduction does not only help reducing computational costs for a given classification task, but it can also be helpful to avoid overfitting by minimizing the error in parameter estimation ("curse of dimensionality").

Principle Component Analysis vs. Linear Discriminant Analysis

Both Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) are linear transformation techniques that are commonly used for dimensionality reduction. PCA can be described as an "unsupervised" algorithm, since it "ignores" class labels and its goal is to find the directions (the so-called principal components) that maximize the variance in a dataset. In contrast to PCA, LDA is "supervised" and computes the directions ("linear discriminants") that will represent the axes that that maximize the separation between multiple classes.



What is a "good" feature subspace?

Let's assume that our goal is to reduce the dimensions of a d-dimensional dataset by projecting it onto a (k)-dimensional subspace (where k<d). So, how do we know what size we should choose for k (k = the number of dimensions of the new feature subspace), and how do we know if we have a feature space that represents our data "well"?

Later, we will compute eigenvectors (the components) from our data set and collect them in a so-called scatter-matrices (i.e., the in-between-class scatter matrix and within-class scatter matrix).

Each of these eigenvectors is associated with an eigenvalue, which tells us about the "length" or "magnitude" of the eigenvectors.

If we would observe that all eigenvalues have a similar magnitude, then this may be a good indicator that our data is already projected on a "good" feature space.

And in the other scenario, if some of the eigenvalues are much larger than others, we might be interested in keeping only those eigenvectors with the highest eigenvalues, since they contain more information about our data distribution. Vice versa, eigenvalues that are close to 0 are less informative and we might consider dropping those for constructing the new feature subspace.

Mathematics Behind LDA

LDA can be thought of as supervised learning problem. The whole process of obtaining linear Discriminants from a raw dataset can be simplified in five parts.

1. Compute the d-dimensional mean vectors for the different classes from the dataset.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
3. Compute the eigenvectors (e1, e2…. ed) and corresponding eigenvalues (λ1, λ2...,λd) for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d × k dimensional matrix W (where every column represents an eigenvector).
5. Use this d × k eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: Y=X×W (where X is a n × d dimensional matrix representing the n samples, and y are the transformed n × k - dimensional samples in the new subspace).

6. HARDWARE ARCHITECTURE MLP

The hardware design of the MLP classifier is broadly divided into two parts. The elementary unit of the architecture is a perceptron model and a united network of these perceptron's is MLP model. A group of multiple and independent perceptron's are referred as a perceptron layer. The feature information is processed and forwarded from one layer to another completely connected layer. This multiple layer of perceptron's is used to solve complex classification problems like activity recognition. This type of network is also known as fully connected feed-forward neural network. In this work, the final MLP activity classifier is constructed from elementary perceptron's using Xilinx system generator toolbox. The detailed mathematical formulation and hardware design of a perceptron and MLP classifier have been discussed in the following two subsections.

6.1. Perceptron Model

A perceptron basically takes some input values, called "features" and represented by the values x_1, x_2,...$x_n x_1$,x_2,...x_n in the following formula, multiplies them by some factors called "weights", represented by w_1,w_2,...wnw_1,w_2,...w_n, takes the sum of these multiplications and depending on the value of this sum outputs another value o:

$$O = f(w_1 x_1 + w_2 x_2 + w_3 x_3 + w_{14} x_4 \dots \dots \dots . + w_n x_n)$$

There are a few types of perceptron's, differing in the way the sum results in the output, thus the function f () in the above formula.

We will investigate the math evolved and discuss its limitations, thereby setting the ground for the future articles. The basic maths formula for perceptron

$$f(x) = \begin{cases} 1, if \ w_1 x_1 + w_2 x_2 + w_3 x_3 + w_{14} x_4 .. + w_i x_i \dots + w_n x_n > b \\ 0, \quad Otherwise \end{cases}$$

So, what the perceptron basically does is take some linear combination of input values or features, compare it to a threshold value b, and return 1 if the threshold is exceeded and zero if not.

The feature vector is a group of characteristics describing the objects we want to classify.

In other words, we classify our objects into two classes: a set of objects with characteristics (and thus a feature vector) resulting in in output of 1, and a set of objects with characteristics resulting in an output of 0.

If you search the internet on information about the perceptron you will find alternative definitions which define the formula as follows:

$$f(x) = \begin{cases} +1, if \ w_1 x_1 + w_2 x_2 + w_3 x_3 + w_{14} x_4 \dots + w_i x_i .. + w_n x_n > b \\ -1, \quad Otherwise \end{cases}$$

We will see further this does not affect the workings of the perceptron

Taking a linear combination of input variables:

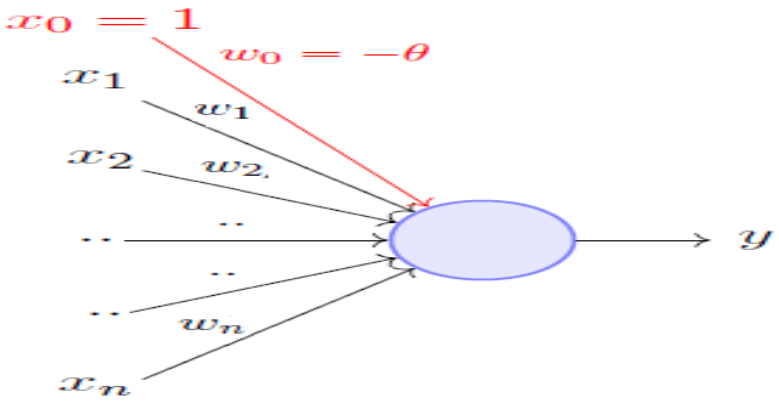
This is the definition of a *Linear Combination*: it is the sum of some terms multiplied by constant values. In our case the terms are the features and the constants are the weights.

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + w_{14} x_4 \dots + w_i x_i .. + w_n x_n$$

This is the definition of a Linear Combination: it is the sum of some terms multiplied by constant values. In our case the terms are the features and the constants are the weights.

If we substitute the subscript by a variable ii, then we can write the sum as

$$\sum_{i=1}^{n} w_i x_i$$



6.2. Multi-Layer Perceptron

The MLP network consists of input, output and hidden layers. Each hidden layer consists of numerous perceptron's which are called hidden units

A single-hidden layer MLP contains an array of perceptron’s.

The output of hidden layer of MLP can be expressed as a function

$$(f(x) = G (W^T x + b))$$

(f :R^D \right arrow R^L),
where D is the size of input vector (x)
(L) is the size of the output vector
(G) is activation function.

In case the activation function G is a sigmoid function then a single-layer MLP consisting of just the output layer is equivalent to a logistic classifier

$$fi(x) = \frac{e^{w_i x + b_i}}{\sum_j e^{w_j x + b_j}}$$

Each unit of input layer corresponds to element of input vector. Each output unit of logistic classifier generate a prediction probability that input vector belong to a specified class.

6.3 Feed Forward Neural Network:

Let us first consider the most classical case of a single hidden layer neural network

The number of inputs to hidden layer is (d) and number of outputs of hidden layer are (m) The hidden layer performs mapping of vector of dimensionality (d) to vector of dimensionality (m).

Each unit of hidden layer of a MLP can be parameterized by weight matrix and bias vector (W, b) and an activation function (\math Cal{G}). The output of a hidden layer is activation function applied to linear combination of input and weight vector.

Dimensionality of weight matrix and bias vector are determined by desired number of output.

If the number of inputs to hidden layer/dimensionality of input is (\math Cal{M}) and number of outputs is (\math Cal{N}) then dimensionality of weight vector in (\math Cal{N x M}) and that of bias vector is (\math Cal{N}x_1).

We can consider that hidden layer consists of (\math Cal{N}) hidden units, each of which accepts a (\math Cal{M}) dimensional vector and produces a single output.

The output is the affine transformation of the input layer followed by the application of function $f(x)$, which is typically a nonlinear function like sigmoid of inverse tan hyperbolic function.

The vector valued function (h(x)) is the output of the hidden layer.

$$h(x) = f(W^T x + c)$$

The output layer of MLP is typically Logistic regression classifier, if probabilistic outputs are desired for classification purposes in which case the activation function is the SoftMax regression function.

Single Hidden Layer Multi-Layer Perceptron's:

Let,

- (h_{i-1}) denote the input vector to the ith layer
- (h_{i}) denote the output vector of the ith layer.
- (h_{0})=x is vector that represents input layer
- (h_{n}=y) is output layer which produces the desired prediction output.
- (f(x)) denote the activation function

Thus, we denote the output of each hidden layer as

$$h_k(x) = f\left(b_k + w_k^T h_{i-1}(x)\right) = f(a_k)$$

Considering sigmoid activation function, gradient of function wrt arguments can be written as

$$\frac{\partial h_k(x)}{\partial a_k} = f(a_k)(1 - f(a_k))$$

The computation associated with each hidden unit (i) of the layer can be denoted as

$$h_{k,i}(x) = f\left(b_{k,i} + W_{k,i}^T h_{i-1}(x)\right) = f(a_k(x))$$

The output layer is a Logistic regression classifier. The output is a probabilistic output denoting the confident that input belongs to the predicted class. The cost function defined for the same is defined as negative log likelihood over the training data

$$L = -log(p_y)$$

The idea is to maximize (p_{y}= P(Y =y_{i} \| x)) as estimator of conditional probability of the class (y) given that input is (x).This is the cost function for training algorithm.

6.4 Back-Propagation Algorithm:

The Back-Propagation Algorithm is recursive gradient algorithm used to optimize the parameters MLP wrt to defined loss function. Thus, our aim is that each layer of MLP the hidden units are computed so that cost function is maximized.

Like in logistic regression we compute the gradients of weights wrt to the cost function . The gradient of the cost function wrt all the weights in various hidden layers are computed. Standard gradient based optimization is performed to obtain the parameters that will minimize the likelihood function.

The output layer determines the cost function. Since we are using Logistic regression as output layer. The cost function is the Softmax function. Let L denote the cost function.

There is nothing different we do in back-propagation algorithm that any other optimization technique. The aim is to determine how the weights and biases change in the network

$$\frac{\partial L}{\partial W_{k,i,j}} \text{ and } \frac{\partial L}{\partial W_{k,i,j}}$$

Output Layer:

$$L = -\log(f(a_{k,i}))$$

$$\frac{\partial \mathrm{L}}{\partial \mathrm{a_{k,i}}} = \frac{\partial L}{\partial \mathrm{h_{k,i}}} \frac{\partial \mathrm{h_{k,i}}}{\partial \mathrm{a_{k,i}}} = -\frac{1}{h_{k,i}} * h_{k,i} * (1 - h_{k,i}) = (h_{k,i} - 1)$$

$$\frac{\partial L}{\partial a_{k,i}} = h_{k,j} - 1_{y=y_i}$$

The above expression can be considered as the error in output. When (y=y_{i}) the error is (1-p_{i}) and then (y - y_{i}) the error in prediction is (p_{i}).

Hidden Layer:

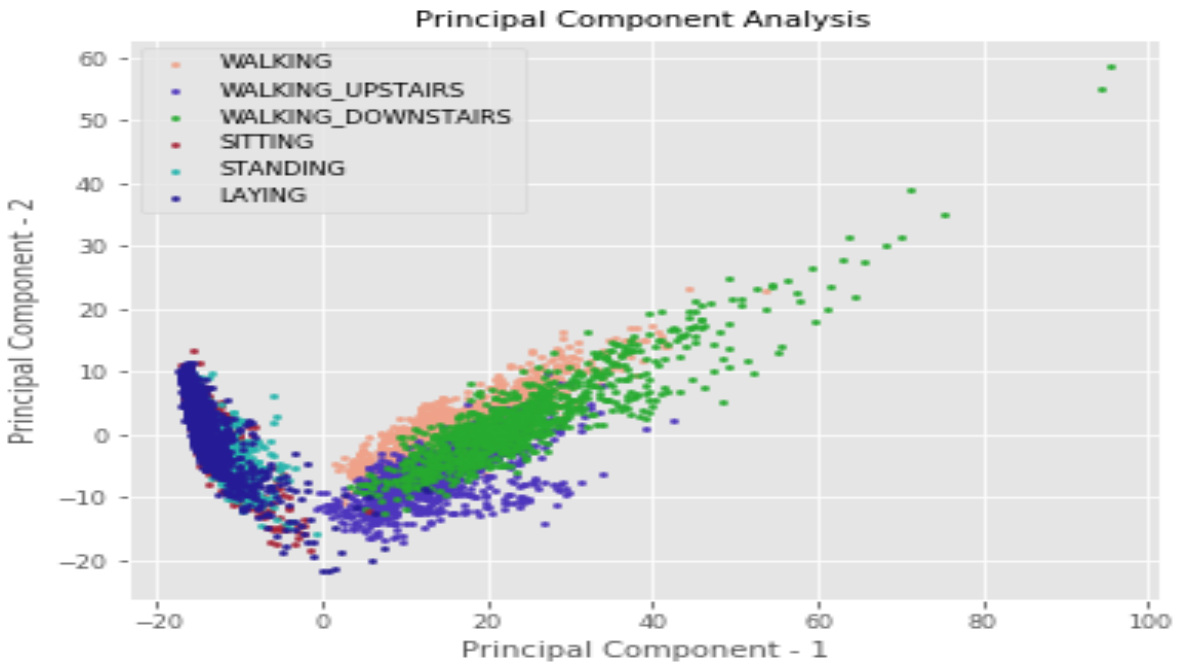
$$\frac{\partial L}{\partial a_{k-1,j}} = \frac{\partial y}{\partial h_{k-1,j}} \frac{\partial h_{k-1,j}}{\partial a_{k-1,j}}$$

Thus, the idea is to start computing gradients from the bottom most layer. To compute the gradients of the cost function wrt parameters at the i^{th} layer we need to know the gradients of cost function wrt parameters at $(i+1)^{th}$ layer.

We start with gradient computation at the logistic classifier level. The propagate backwards, updating the parameters at each layer.

7.Reports

PCA



From the above figure it is clearly seen that the two principle components explain only 56% of the variance that can explain the dependent variable. Though only two components explaining 56% we can add more principle components that it explains more than 90% variance that explains the classes well.

As you can see from the picture classes WALKING and WALKING_DOWNSTAIRS and WALKING_UPSTAIRS are more related than the other classes.

Ensemble Model Results when applied PCA

```
Random_Forest_Tuned : _Train_Accuracy : 0.9182535364526659
Random_Forest_Tuned : _Test_Accuracy : 0.833050559891415
Duration: 0:03:37.507408
-------------------------------------------------
Xg_boost : _Train_Accuracy : 0.971436343852013
Xg_boost : _Test_Accuracy : 0.8836104513064132
Duration: 0:01:01.973543
-------------------------------------------------
Bagging Classifier : _Train_Accuracy : 0.9957834602829162
Bagging Classifier : _Test_Accuracy : 0.8218527315914489
Duration: 0:00:08.362479
```

Network Models Approached

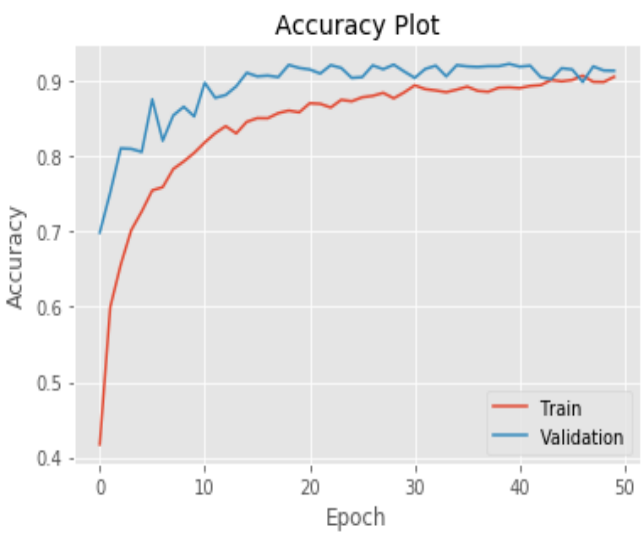
Architecture

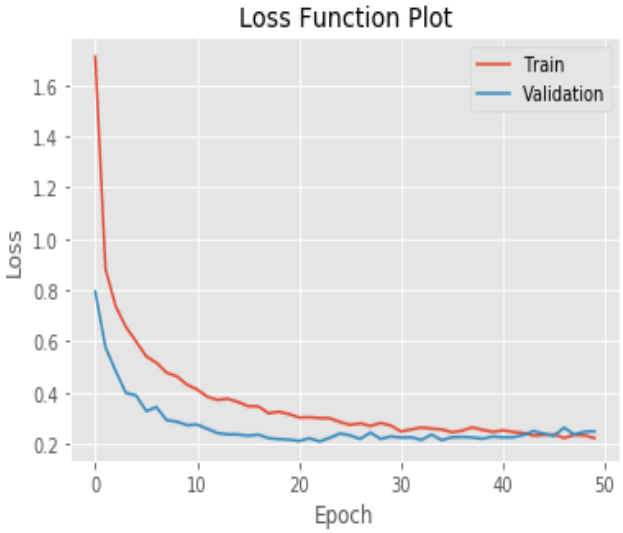
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 45)                3195
_________________________________________________________________
dropout_1 (Dropout)          (None, 45)                0
_________________________________________________________________
dense_2 (Dense)              (None, 25)                1150
_________________________________________________________________
dropout_2 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 156
=================================================================
Total params: 4,501
Trainable params: 4,501
Non-trainable params: 0
_________________________________________________________________
```

Results when running the architecture on epoch of 50

```
Epoch 50/50
7352/7352 [==============================] - 0s 50us/step - loss: 0.2213 - acc: 0.9057 - val_loss: 0.2481 - val_acc: 0.9138
```

Learning Curve for Above Result





After Finding Best Learning Rate

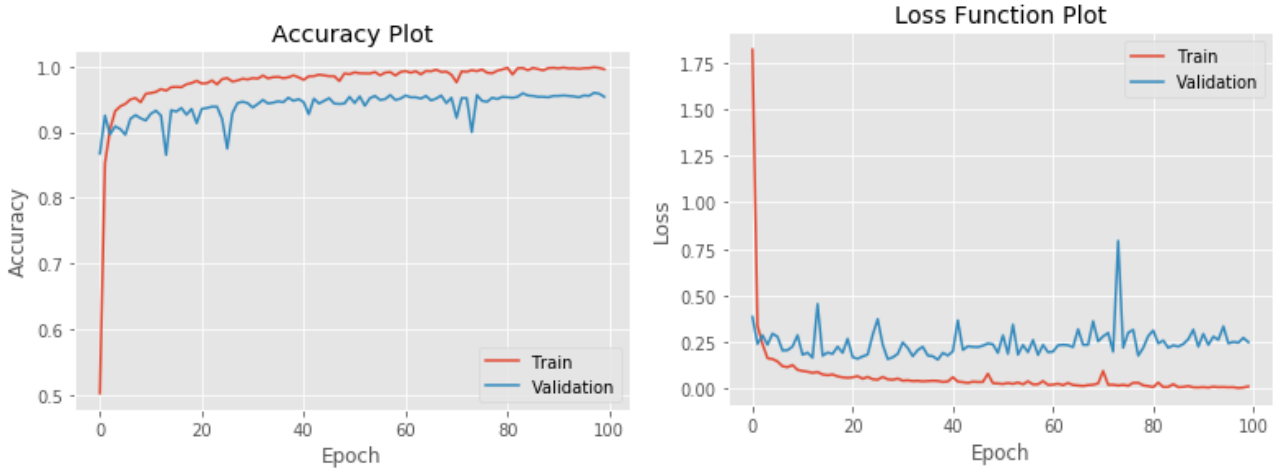
Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 25)                1775
_________________________________________________________________
dense_35 (Dense)             (None, 15)                390
_________________________________________________________________
dense_36 (Dense)             (None, 6)                 96
=================================================================
Total params: 2,261
Trainable params: 2,261
Non-trainable params: 0
_________________________________________________________________
```

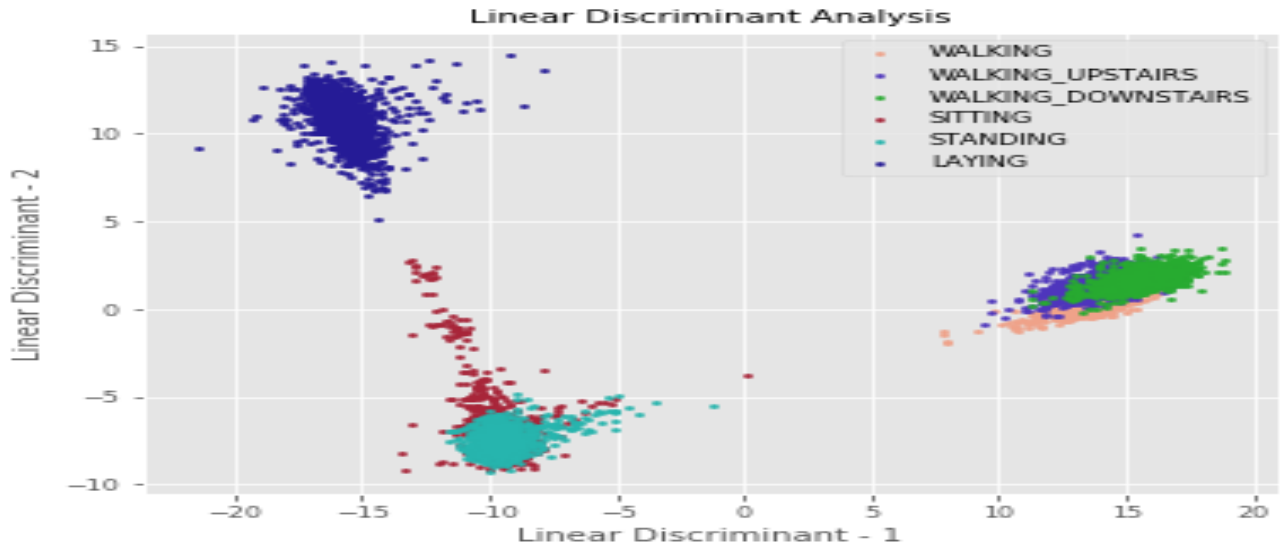
Results when running the architecture on epoch of 100

```
Epoch 100/100
5881/5881 [==============================] - 0s 67us/step - loss: 0.0122 - acc: 0.9957 - val_loss: 0.2498 - val_acc: 0.9538
```

Learning Curve for Above Result



LDA



From the above picture

1.Movement based Activities are clustered much

2.Sitting and Standing are clustered

3.Laying is Completely away from other activities

This is explained by two Linear Discriminantor

Ensemble Model Results when applied LDA

```
Random_Forest_Tuned : _Train_Accuracy : 0.9868063112078346
Random_Forest_Tuned : _Test_Accuracy : 0.9599592806243638
Duration: 0:00:59.677413
-------------------------------------------------
Xg_boost : _Train_Accuracy : 0.9912948857453754
Xg_boost : _Test_Accuracy : 0.9538513742789277
Duration: 0:00:05.205298
-------------------------------------------------
Bagging Classifier : _Train_Accuracy : 0.9982317736670294
Bagging Classifier : _Test_Accuracy : 0.9626739056667798
Duration: 0:00:00.311018
```

SVM Model Results when applied LDA

```
Support Vector Machines_Tuned : _Train_Accuracy : 0.9870783460282916
Support Vector Machines_Tuned : _Test_Accuracy : 0.9643705463182898
Duration: 0:00:47.653725
```

Network Models Approached

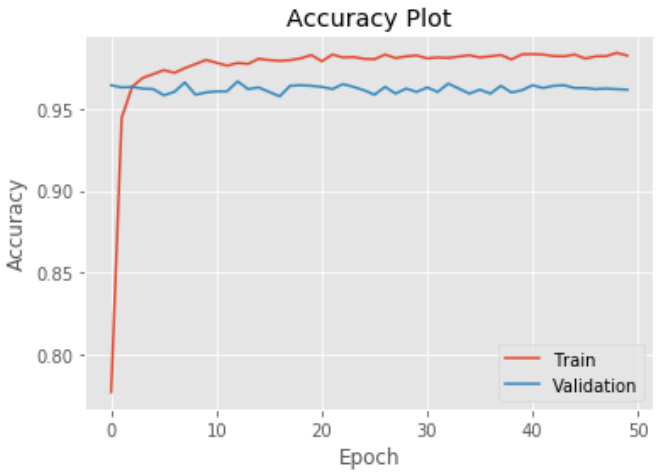
Architecture

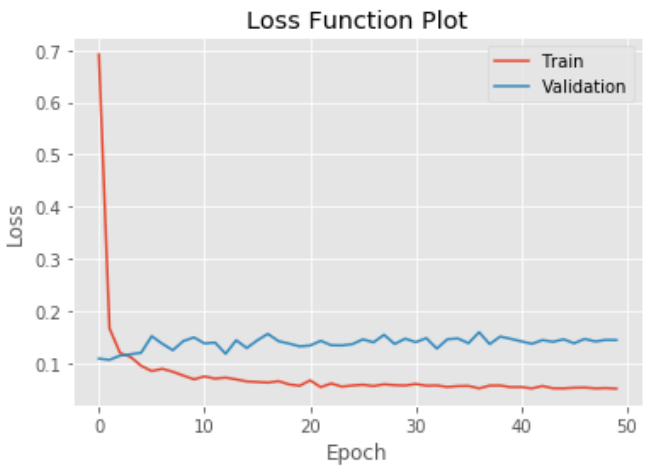
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                384
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 390
=================================================================
Total params: 4,934
Trainable params: 4,934
Non-trainable params: 0
_________________________________________________________________
```

Results when running the architecture on epoch of 50

```
Epoch 50/50
7352/7352 [==============================] - 0s 38us/step - loss: 0.0506 - acc: 0.9825 - val_loss: 0.1439 - val_acc: 0.9617
```

Learning Curve for Above Result





After Finding Best Learning Rate

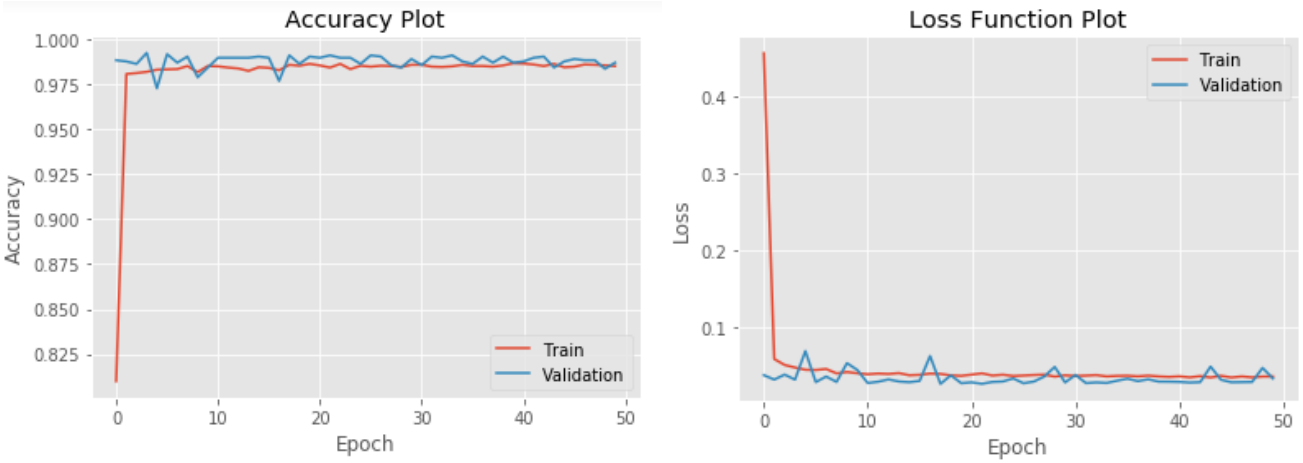
Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_64 (Dense)             (None, 25)                150
_________________________________________________________________
dense_65 (Dense)             (None, 15)                390
_________________________________________________________________
dense_66 (Dense)             (None, 6)                 96
=================================================================
Total params: 636
Trainable params: 636
Non-trainable params: 0
_________________________________________________________________
```

Results when running the architecture on epoch of 50

```
Epoch 50/50
5881/5881 [==============================] - 0s 31us/step - loss: 0.0351 - acc: 0.9852 - val_loss: 0.0327 - val_acc: 0.9871
```

Learning Curve for Above Result



Conclusion

Whenever using a machine learning algorithm, evaluation metrics for the model have to be chosen cautiously: we must use the metrics that give us the best overview of how well our model is doing with regards to our goals. Similarly, for network models Parameters decides the way to learn the patterns clearly and predicts better than machine learning algorithms.

Deep neural network model's works on large data set with a best parameters values and optimizers can give a best result.

Understanding that the finding best learning rate, weight decay, batch-size, batch-normalization are the parameters and finding the best values will give a better result when compared to the Machine Learning Algorithms.

Reference

- O. D. Lara, M. A. Labrador, "A survey on human activity recognition using wearable sensors", *IEEE Commun. Surveys Tuts.*, vol. 15, pp. 1192-1209, 3rd Quart. 2013.
- H. Wei, J. He, J. Tan, "Layered hidden Markov models for real-time daily activity monitoring using body sensor networks", *Knowl. Inf. Syst.*, vol. 29, no. 2, pp. 479-494, 2011.
- J. Andreu, P. Angelov, "Real-time human activity recognition from wireless sensors using evolving fuzzy systems", *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ)*, pp. 1-8, Jul. 2010.
- M. Janidarmian, A. R. Fekr, K. Radecka, Z. Zilic, "A comprehensive analysis on wearable acceleration sensors in human activity recognition", *Sensors*, vol. 17, no. 3, pp. 529, 2017.
- C. Rodriguez et al., "IoT system for human activity recognition using BioHarness 3 and smartphone", *Proc. Int. Conf. Future Netw. Distrib. Syst.*, pp. 37, 2017.
- T. Zebin, P. J. Scully, K. B. Ozanyan, "Evaluation of supervised classification algorithms for human activity recognition with inertial sensors", *Proc. IEEE Sensors*, pp. 1-3, Oct. 2017.
- A. T. Özdemir, K. Danişman, "Fully parallel ANN-based arrhythmia classifier on a single-chip FPGA: FPAAC", *Turkish J. Elect. Eng. Comput. Sci.*, vol. 19, no. 4, pp. 667-687, 2011.
- Mellal, M. Laghrouche, H. T. Bui, "Field programmable gate array (FPGA) respiratory monitoring system using a flow microsensor and an accelerometer", *Meas. Sci. Rev.*, vol. 17, no. 2, pp. 61-67, 2017.
- W. Tang, E. S. Sazonov, "Highly accurate recognition of human postures and activities through classification with rejection", *IEEE J. Biomed. Health Inform.*, vol. 18, no. 1, pp. 309-315, Jan. 2014.
- J. J. M. Nolasco, J. A. P. Medina, "LabVIEW-based classic fuzzy and neural controller's algorithm design applied to a level control prototype", *IEEE Latin Amer. Trans.*, vol. 15, no. 6, pp. 1154-1162, Jun. 2017.
- H.-C. Huang, C.-H. Chiang, "An evolutionary radial basis function neural network with robust genetic-based immunecomputing for online tracking control of autonomous robots", *Neural Process. Lett.*, vol. 44, no. 1, pp. 19-35, 2016.