Jin Zhao
BTech 451

Machine Learning

The Intelligent Functional Approach

1 Introduction

Learning is the act of acquiring new, or modify and reinforcing, existing knowledge, behaviours, skills values, or preferences and may involve synthesising different types of information. Since the inception of the computer era, researchers have been striving to implant such capabilities in computers. Solving this problem has been, and remains, a most challenging and fascinating long-range goal in artificial intelligence (AI). The study and computer modelling of learning process in their multiple manifestations constitutes the subject matter of machine learning (ML).

1.1 The objective of machine learning

Machine leaning is a type of artificial intelligence approach, according to Samuel (1959), ML provides computers with the ability to learn without being instructional programmed. Furthermore, ML focuses on the development of computer programs that can teach themselves to reproduce and adjust when exposed to new data. Nick [1] has pointed out, ML process can be subdivided into two parts supervised machine learning and unsupervised machine learning, which both part focused on different conclusions given by a bunch of data. In this report, I will give a basic implementation of both supervised and unsupervised machine learning process。

2 Unsupervised Learning

Unsupervised learning is a type of machine learning which algorithm will be used to draw inferences from datasets consisting of input data without labelled response. In this report I will focus on the most common unsupervised learning method cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data, furthermore, I will implement the k-means algorithm which was first mentioned by MacQueen[2] and the

idea goes back to Steinhaus [3] to compute the Euclidean distance matrix for measuring the similarity of given data set.

2.1.1 k-means algorithm

k-means clustering is a method of vector quantisation which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. According to Tuceryan and Jain [4] results given by k-means algorithm in a partitioning of the data space into Voronoi cells.

The mathematical proof based upon Selim and Lsmail (1984),

Given a set of observations (p1, p2, …, pn), where each observation is a 2- dimensional real vector, k- means clustering aims to partition the n observation into k (k ≤ n) sets S = {S1, S2, …, Sn} so as to minimise the within-cluster sum of squares(the linear function sum of squares will lead the cluster into Voronoi cells), which in other words, its objective is to find the root for:

$$y=\arg_s\min\sum_{i=1}^{k}\sum_{x\in S_i}\|x-\mu_i\|^2$$

Furthermore, In mathematics, a Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane [5]

The mathematical deduction of the Voronoi diagram,

Let X be a metric space with distance function d. Let K be a set of indices and let (P_k) {k ∈ K} be a tuple (ordered collection) of nonempty subsets (the sites) in the space X. The Voronoi cell, or Voronoi region, R_k, associated with the site P_k is the set of all points in X whose distance to P_k is not greater than their distance to the other sites P_j, where j is any index different from k. In other words, if d(x, A) = inf { d(x, a) |a ∈A \ } denotes the distance between the point x and the subset A then,

$$R\_k = \{x \epsilon X |\ d(x, P\_k) \leq d(x, P\_j) \text{ for all } j \neq k \}$$

In Figure 1 the Voroni diagram(generated by http://alexbeutel.com/webgl/voronoi.html) of 14 points and in Figure 2 the outcome of 10000 points in 14 clusters which calculated by k-means algorithm (base on Computing the least euclidean distance between observations and the cluster centroid),

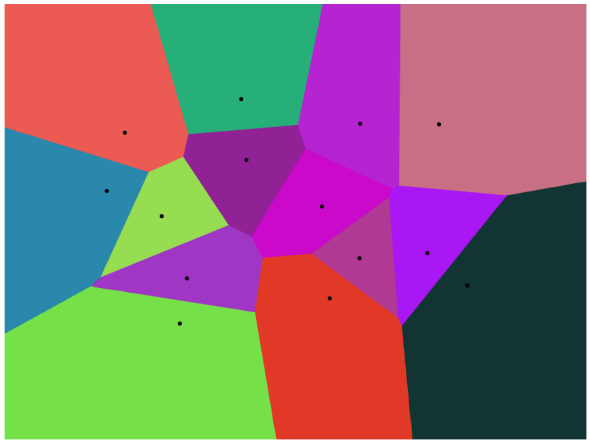


Figure 1

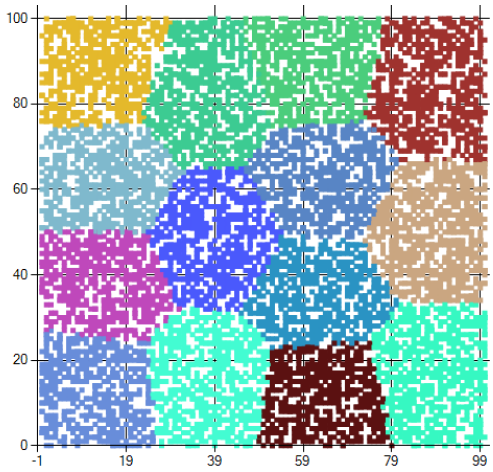


Figure 2

It is clear, that the behaviour of the k-means algorithm measured by the Euclidean distance matrix is similar to the way we generate the Voroni diagram.

2.1.2 Standard Algorithm

The algorithm uses an iterative refinement technique, which the algorithm proceeds two steps:

1. Initialise centroids based upon the k value and assign each observation into the cluster, which firstly I will set each cluster at least one value to make sure that there will not be an empty cluster, after that each data in the raw observation list will be given a random cluster number, the C# implementation is shown below,

```
public static void initializeCentroids (int numberOfClusters,
                                        List<DataPoint> data) {
```

```
    Random random = new Random(numberOfClusters);
    for (int i = 0; i < numberOfClusters; i++) {
        data[i].Cluster = i;
    }
    for (int i = numberOfClusters; i < data.Count; i++) {
        data[i].Cluster = random.Next(0, numberOfClusters);
    }
}
```

The method above takes n (n= number of clusters) elements from the raw observations and assigns them into n different clusters. The remaining observation will be assigned into randomly selected clusters. The method have O(n) for runtime.

Also, in order to calculate the means for all the clusters we have grouped observation data into arrays by their cluster value.

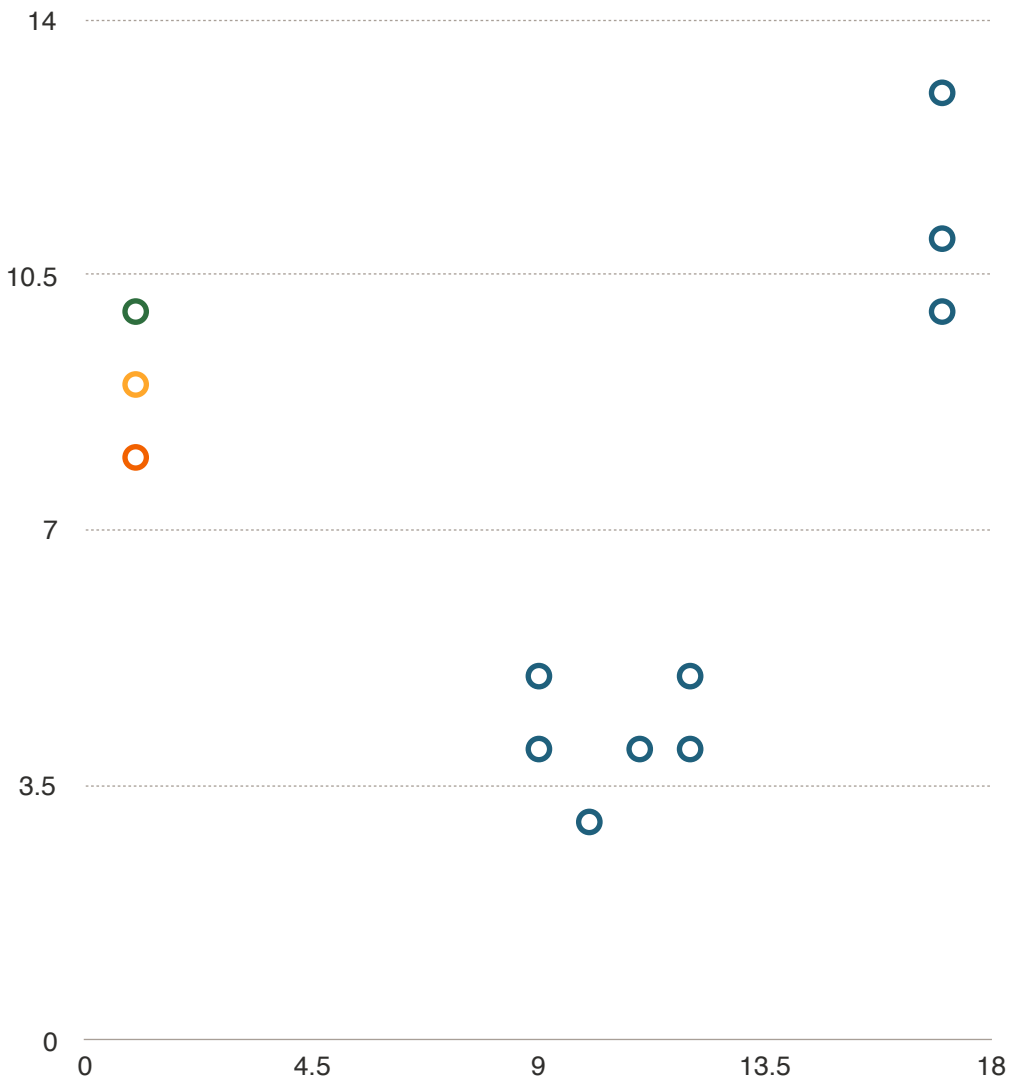
```
public static Boolean calculateMeans ( List<DataPoint> data,
                                       int clusterNumber,
                                       DataPoint[] clusters) {

    if (emptyCluster(data)) return false;
    List<DataPoint>[] clusterList = new
List<DataPoint>[clusterNumber];
    for (int i = 0; i < clusterNumber; i++) {
        clusterList[i] = new List<DataPoint>();
    }
    foreach (DataPoint i in data) {
        clusterList[i.Cluster].Add(i);
    }
    double height = 0.0;
    double weight = 0.0;
    for (int i = 0; i < clusterNumber; i++) {
        foreach (DataPoint dp in clusterList[i]) {
            height += dp.Height;
            weight += dp.Weight;
        }
        DataPoint dpi = new DataPoint();
        dpi.Height = height / clusterList[i].Count();
        dpi.Weight = weight / clusterList[i].Count();
        clusters[i] = dpi;
        height = 0;
        weight = 0;
    }
    return true;
}
```

The calculatedMeans method takes grouped observations as the input and produce an array of the cluster means as the centroid of the cluster, the method has O(n) for runtime.

Overall, step 1 randomly initialises the cluster and produce the mean of each cluster as new data points in terms of the centroid for the cluster. The runtime is O(n).

Figure 3



The demonstration for step 1 is shown in Figure 3,

n initial "means" (in this case n = k= 3) the first 3 observations been assigned to 3 different clusters (represented by 3 different colours, green; yellow and red), the rest observations has been randomly assigned into the clusters (we are not interesting the random selection therefore the colour is shown as blue).

2. In this stage the data point moved from a cluster to a new one. I updated the value stored in its Cluster property based upon compering the difference between the observation and the all the centroids, then I choose the cluster with the minimum difference and move the data point to that cluster. The following code is responsible for this,

```
public static bool updateClusterMembership ( int numberOfClusters,
                                    List<DataPoint> data,
                                    DataPoint[] clusters) {
 bool changed = false;
 double[] distances = new double[numberOfClusters];
 for (int i = 0; i < data.Count; i++) {
    for (int k = 0; k < numberOfClusters; k++) {
       distances[k] = elucidanDistance(data[i], clusters[k]);
}
    int newClusterId = miniIndex(distances);
    if (newClusterId != data[i].Cluster) {
       changed = true;
       data[i].Cluster = newClusterId;
    }
 }
 if (changed == false)
    return false;
 if (data.Count()==0) return false;
 return true;
}
```

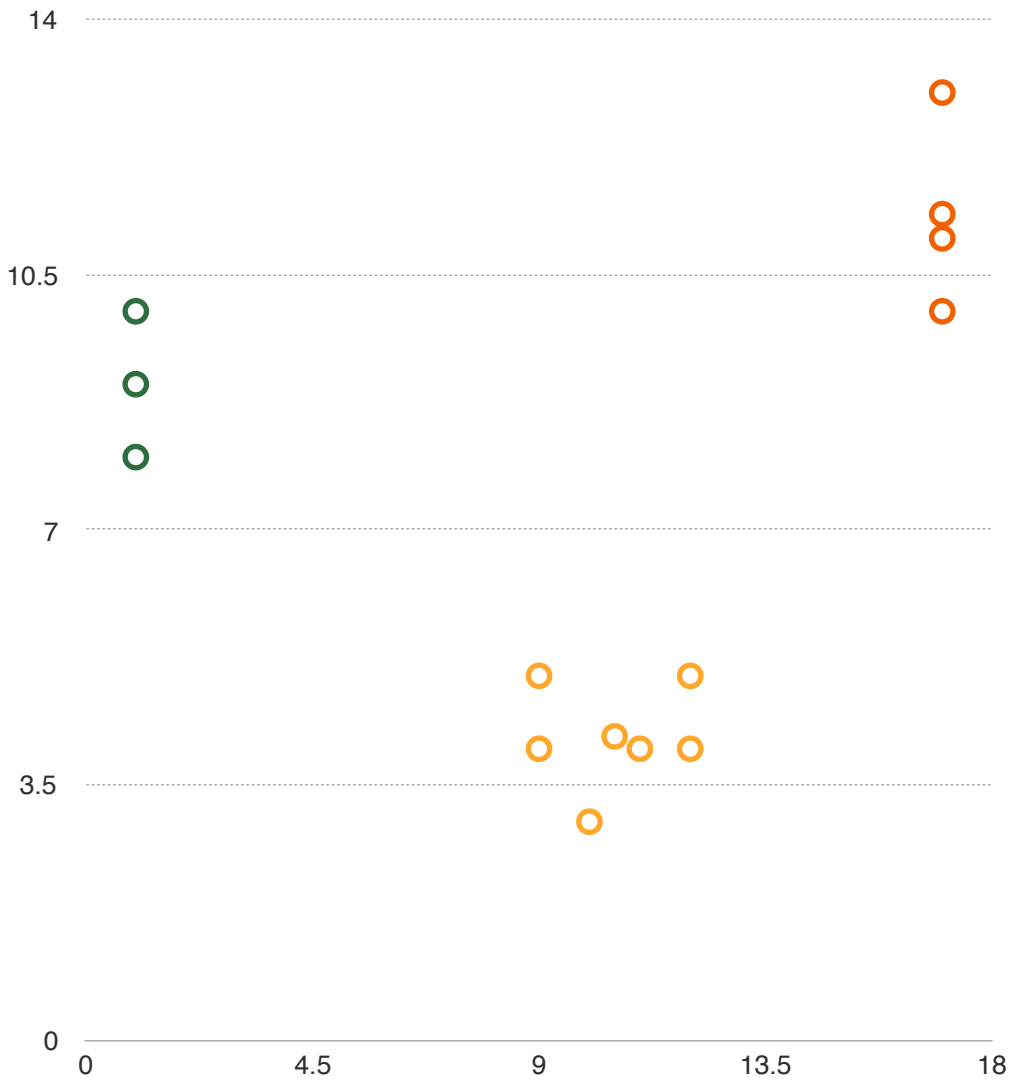
Computing the least euclidean distance and putting the data into the right cluster. The sum of squares will be given as the squared euclidean distance, the C# implementation is shown below,

```
public static double euclideanDistance ( DataPoint data,
                                DataPoint potentialMean) {
   double diff = 0.0;
   diff = Math.Pow(data.Height - potentialMean.Height, 2);
   diff += Math.Pow(data.Weight - potentialMean.Weight, 2);
   return Math.Sqrt(diff);
}
```

Thus, the outer-most loop iterates the items of all the observations, the next loop within this one calculates the Elucidan distance between each item in the data collection and the means of clusters stored in cluster array the result of each of the these comparisons in an array called distances, after that, the code takes the minimal value in the array and checks whether this data point is already in that cluster. If the data point is not in the cluster with the minimum distance I move the data there by changing its cluster property. The step overall runtime is O(n).

The demonstration of updating cluster membership method is given in Figure 4,

Figure 4



The value of centroid of each of the *k* clusters becomes the new mean, in Figure 2 the mean of green is (1, 9), the mean of yellow is (10.5, 4.17) and the mean for read is (17, 10.33).

Hint, the idea behind the K-means Clustering algorithm is that we are trying to move the items into more suitable clusters until there is no change in cluster membership happens

Also, if the observation has pre-defined clusters the algorithm will be able to distinguish the hidden patten, in Figure 5 I have a data set with three "pre-defined" clusters, and I set the k value to 3 for the algorithm the result is given below,

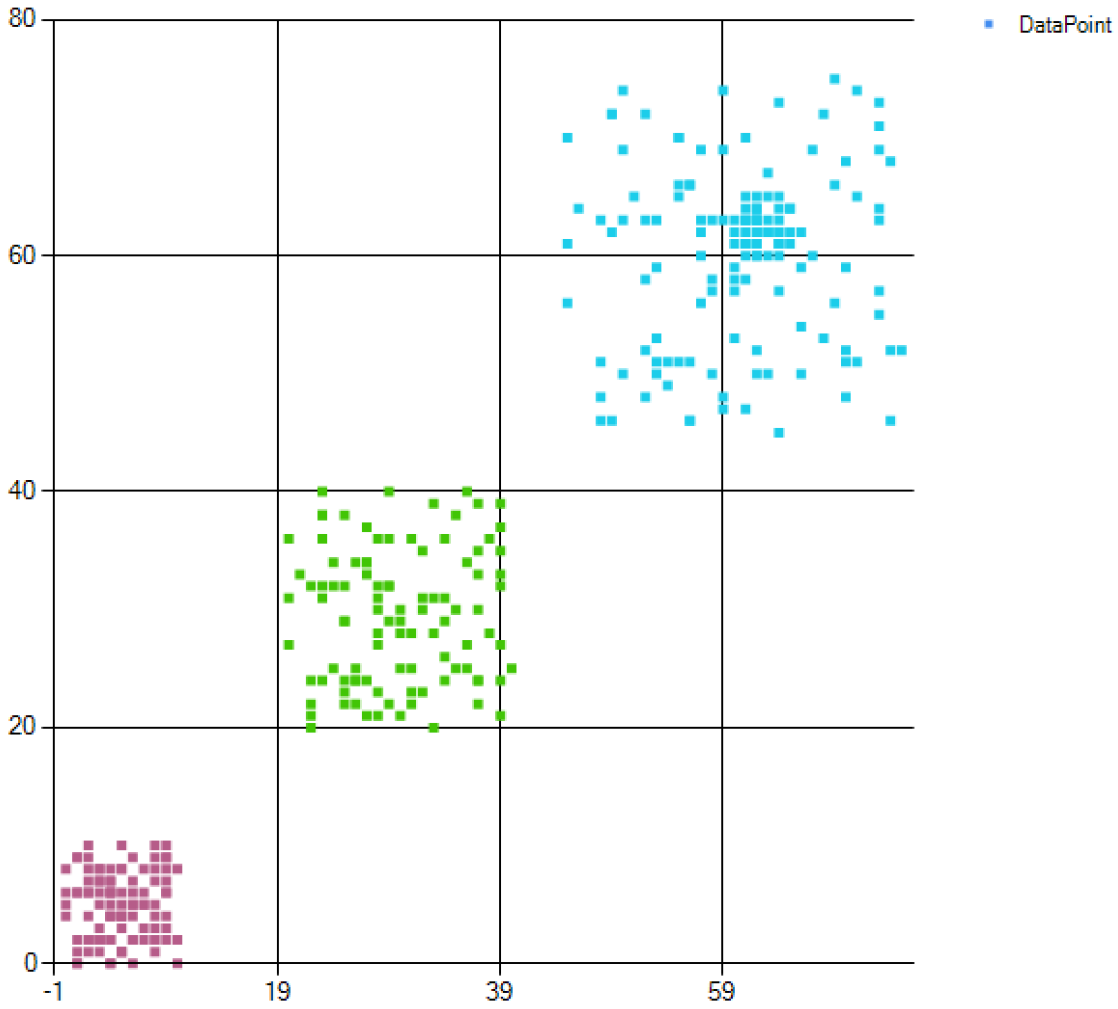


Figure 5

However if the k value is greater than 3, then some big cluster will start to fragment into smaller parts and the demonstration is shown in Figure 6,

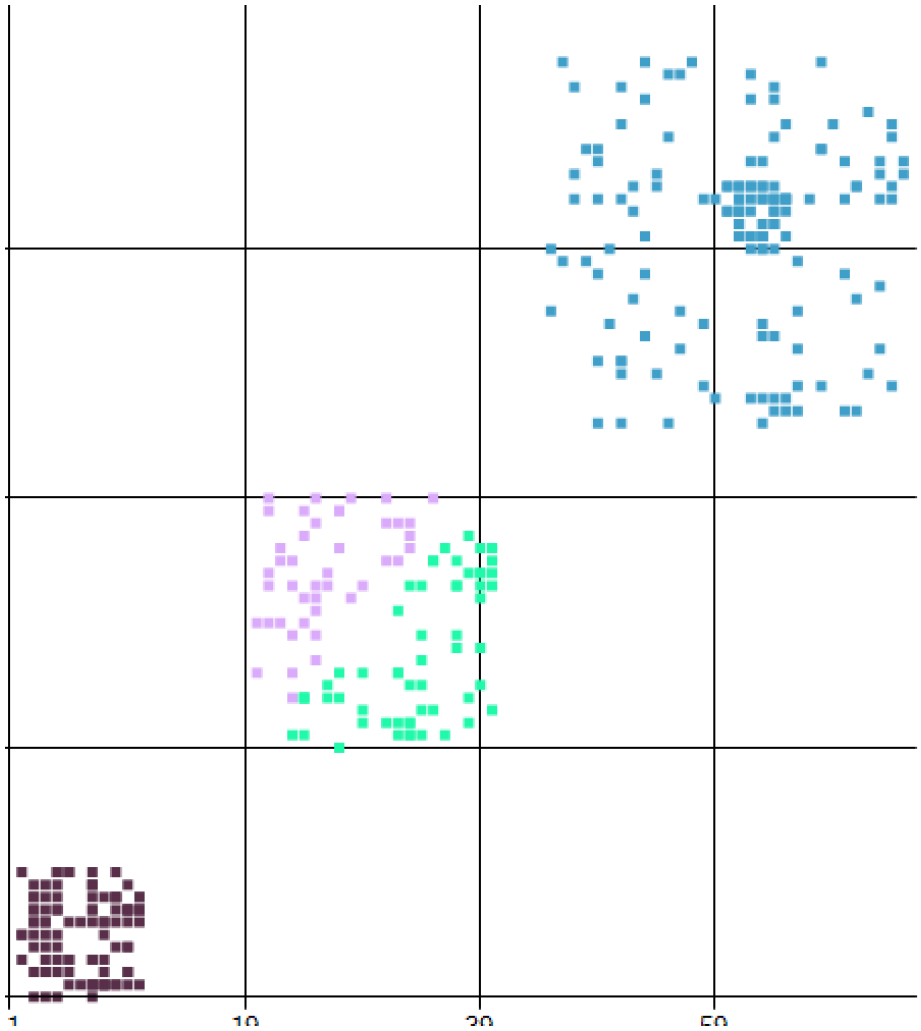


Figure 6

Similar to Figure 6 with greater k value the big data will start to fragment into smaller parts but the small clusters will hold unchanged, and the demonstration is shown in Figure 7,

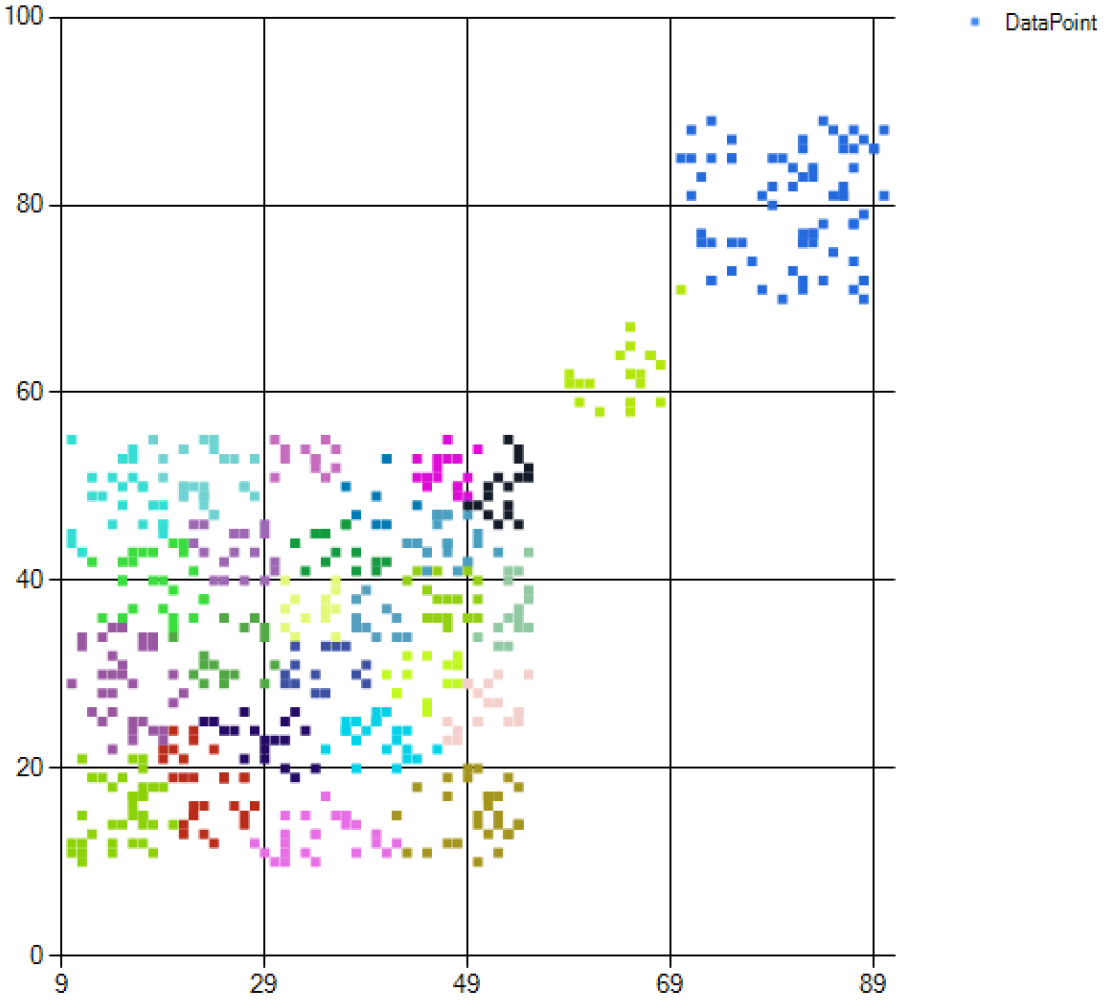


Figure 7

2.1.3 A coin-flipping experiment

In order to illustrate what k-means algorithm can do we consider a simple coin-flipping experiment in which we are given a pair of coins A and B of unknown biases, θ_A and θ_B respectively (that is, on any given flip, coin A will land on heads with probability and tails with probability 1- θ_A and similarly for coin B). Our goal is to estimate the biases by repeating the following procedure 10 times: randomly choose one of the two coins (with equal probability), and perform ten independent coin tosses with the selected coin. Thus, the entire procedure involves a total of 100 coin tosses. The following code is responsible for this,

```
static void Main(string[] args) {
 KMeans kmeans = new KMeans(2);
 double A_Biases = 0.9;            // The probability to head(A)
 double B_Biases = 0.1;            // The probability to head(B)
 int expRand = 10;                 // Rand number
 int toess = 10;                   // Toss for a rand
 double [][] observations = new double [expRand][];
 for (int i = 0; i < expRand; i++) {
    double X = oneTime( A_Biases, B_Biases,toess);
    observations[i] = new double[2];
    observations[i][0] = dataNormalization(X,toess);
    observations[i][1] = 10 - observations[i][0];
    toess = toess + 10;
 }
 int[] labels = kmeans.Compute(observations);
 var temp = kmeans.Clusters.Centroids;
 for (int i=0;i<temp.Length;i++) {
    for (int j=0;j<temp[i].Length;j++){
       Console.Write(temp[i][j]+" ");
    }
 }
}
```

The code above references the Accord.MachineLearning library(from https://www.google.co.nzurlsa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwj-0ebykZHNAhVkxqYKHWKJCdgQFggmMAA&url=http%3A%2F%2Faccord-framework.net%2F&usg=AFQjCNELitjh3lNuYxGYyed5FUNKCvi8ug). Library offers the framework for k-means algorithm which generating the clusters toward the euclidean distance.

2.1.4 Experiment hypothesis

Assuming k-means algorithm is going to generate 2 clusters in terms of 2 coins's toss, the centroids for the cluster will be the bias for the coin (θ_B and θ_A). For example if experiment one takes A coin, and got results for TTTHTHTHTH which will be recorded as point (4, 6) which x= number for heads and y = number for tails. However, assuming k-means algorithm will be performing differently by the difference between two biases ($d = |\theta_B - \theta_A|$) , and it is impossible to have a coin with 100% bias (will only get heads or tails).

2.1.5 Methodology

We set the biases from 0.1 to 0.9 for the possibility to heads and run the experiment for 50 times. We exam the standard deviation and mean regarding to the expecting bias value.

2.1.6 result

Based upon appendix A, we take the total difference between result value and the expected value ($|E\theta_A - R\theta_A| + |E\theta_B - R\theta_B|$). Also the total standard deviation of two values. The result value is shown below,

Difference for 2 biases	Total difference	Total sd
0	0.278	2.269
1	0.79	0.39
2	0.33	0.309
3	0.212	1.09
4	0.131	0.525
5	0.179	1.294
6	0.115	0.279
7	0.101	0.37
8	0.039	0.417

2.1.7 Experiment conclusion

Based upon the results stated above, it is clear that k-means algorithm will preform well when the difference between two biases is considerably large. Generally, the expected bias value is within the range offered by the standard deviation.

In this experiment we group the rand on similar toss together. For e.g. all tosses that have 7 heads are grouped into a single cluster as their bias is vary similar and Other cluster will be similar to this. Therefore, this type of Clustering is called partition or iterative clustering (Dhillon,Guan & Kogan, 2002). The algorithm converges even though sometimes it may take exponential time. Although, in practice it converges very fast. Also the algorithm might get stuck in a local minimum that has an arbitrarily bad cost

2.2 Overall Conclusion

The k-means algorithm accepts two inputs. The data itself, and "k", the number of cluster. The output is k clusters with input data partitioned among them. There are different measures for the notion of similarity or dissimilarity (AKA distance) like the Euclidean distance, Manhattan Distance or Cosine similarity etc. The reason why we need so many different measurements is in the "real world" things are a bit complicated when the items are not points but objects with some attributes (for e.g. the content of an article, location of a person). Thus, in this case we need to design our own distance measure in order to suit the scenarios. For instance, we want to check if news articles talk about same topic, a simple distance measurement can be to find the entities in an article and compare their frequencies. Saying two articles with common words like Donald Trump, Hillary Clinton, test "might" talk about same topic. For a real world application, finding a good distance measure requires lot of work, because the measurement should be efficient but also reasonably accurate.

The algorithm do have some random generating, because the initialising centroids, according to Bradley and Fayyad (1998), initial points is very critical to avoid local minima and a good set of initial points are important as it influences both quality of clusters and also the running time of the algorithm. Paper by Arthur and Vassilvitskii shows that if picking the next candidate centre with probability

proportional to its minimum distance squared from the current set of clusters allows for provable guarantees on the quality of the solution returned by k-means. However, the actual results will be slightly different but the trade-off in performance is worth to consider.

There is no way to guarantee the running time, cause the algorithm might run into a local local optimum, but generally the k-means algorithm can take exponential time to converge.

As my project , I did clustering of simple scenario with three Gaussian distributions, non-linear scenario and some random data set. The task showed something like this: Lets say if you record the informations like height, weight or age we can group the information into clusters and the cluster can be represented into patterns (Bishop, 2006), in other words, we like to think of ourselves as individuals but we are actually a pattern with online DNA which includes we married or single, we educated or not, we are rich or poor. The data mining cluster can be the tool to reveal our true nature.

2.3 Regression Machine Learning Systems

In regression machine learning systems I will be interested in the best fitted linear function y= kx + b, for given cost function $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_{t,i}) - y)^2$,and in this case I have a set of points with 2 dimensional values (x,y). The initial prediction has been given as y= 12 +0.2x (k= 0.2 and b=12), the graphic view is shown in Figure 8.

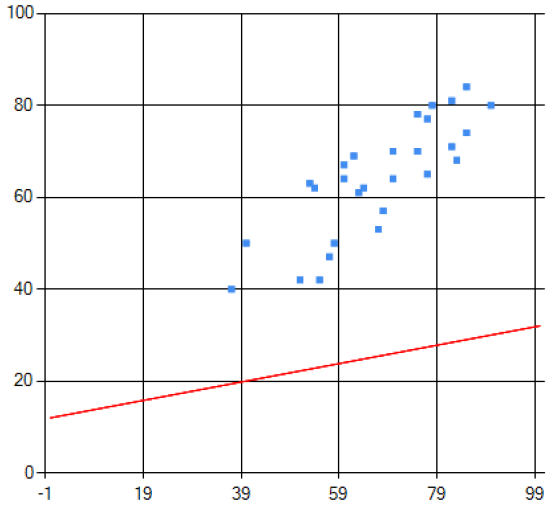


Figure 8.

In order to optimize the value in y= kx +b; we reference the simplex, the truncated Newton method and the BFGS method from the DotNumerics libraries (from http://www.dotnumerics.com/NumericalLibraries/). Three methods will optimise the standard form function for given cost function. The process is shown below.

2.3.1 Simplex Method

```
Simplex simplex = new Simplex();
double[] simplexMinimum = simplex.ComputeMin(bananaFunction,
                                              initialGuess);

public double bananaFunction(double[] x) {
    double sum = 0;
    for (int i = 0; i < setOfPoints.Length; i++) {
        Func<double, double, double, double> h = (b, y, z) => b + y *          z;
        sum = sum + setOfPoints[i].Y - h(x[1], x[0], setOfPoints[i].X);
    }
    double result = Math.Pow(sum, 2) / (2 * setOfPoints.Length);
    return result;
}
```

The C# code above allows simplex.ComputeMin() method to take the initialGuess(k=0.2, b=12) and the cost function(given by method bananaFunction()) as input to generate an array of best fitted k value and b value, and we will take the final values either after 1000 evaluation or the difference between E_i and E_i+1 is small than 1e -5. The results are shown in Figure 9.

Simplex Method

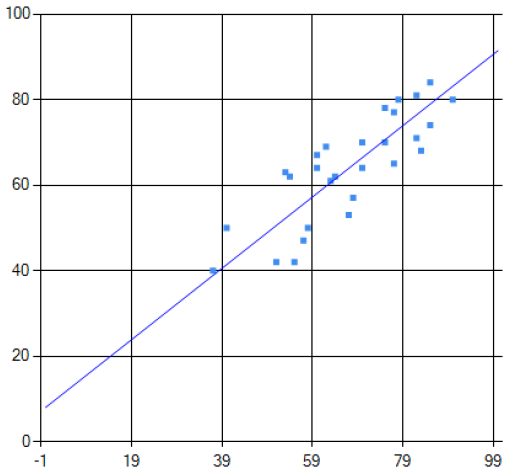


Figure 9.

2.3.2 Truncated Newton Method

```
TruncatedNewton tnewt = new TruncatedNewton();
double[] tnetMinimum = tnewt.ComputeMin(bananaFunction,
                                        bananaGradient,
                                        initialGuess);
public double[] bananaGradient(double[] x) {
    double sum = 0;
    double sum2 = 0;
    double[] relust = new double[2];
    for (int i = 0; i < setOfPoints.Length; i++) {
        Func<double, double, double, double> h = (b, y, z) => b + y * z;
        Func<double, double, double> h0 = (a, b) => 1.0;
        Func<double, double, double> h1 = (a, b) => a;
        sum = sum + (h(x[1], x[0], setOfPoints[i].X) - setOfPoints[i].Y);
        sum2 = sum2 + (h(x[1], x[0], setOfPoints[i].X) - setOfPoints[i].Y) * x[0];
    }
    relust[0] = sum / setOfPoints.Length;
    relust[1] = sum2 / setOfPoints.Length;
    return relust;
}
```

The C# code above allows tnewt.ComputeMin() method to take bananaFunction(given by the code in simplex section), bananaGradient(given by bananaGradient which takes input array with b value, k value and output the differentiations for standard form linear equation y=kx+b) and the initialGuess(k=0.2 and b=12) to generate an array of best fitted k value and b value for given cost function. The results are shown in Figure 10.

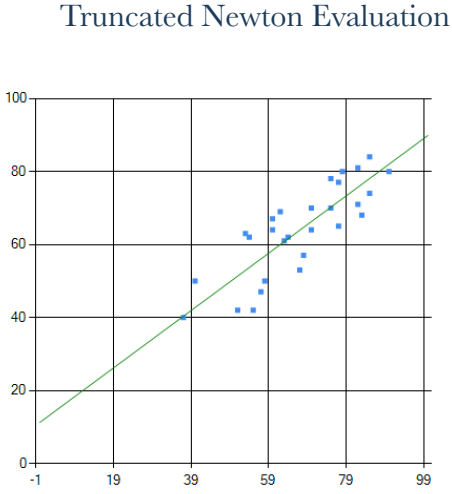


Figure 10

2.3.3 BFGS Method

```
L_BFGS_B lbfgsb = new L_BFGS_B();
double[] lbfgsbMin = lbfgsb.ComputeMin(bananaFunction,
                                       bananaGradient,
                                initialGuess);
```

Similarly, the C# code above allows bfgsb.ComputeMin() method to take bananaFunction (given by bananaFunction() in the simplex method section), bananaGradient (given by bananaGradient() in Truncated newton method section) and the initial Guess(where k=0.2 and b= 12) to compute the desired output, which is shown in Figure 11.

BFGS Evaluation

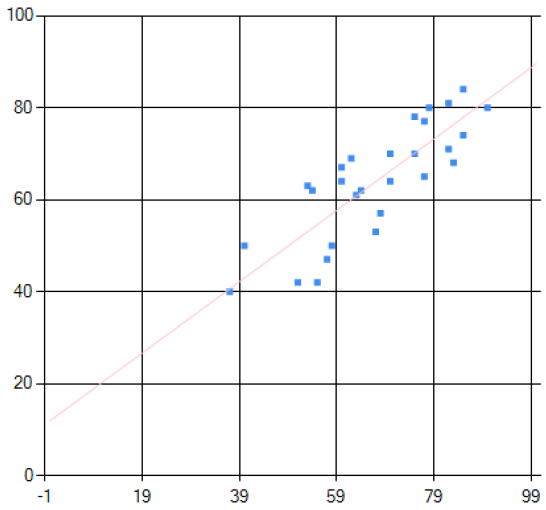


Figure 11.

2.3.4 Conclusion

The evaluation above gives a clear implementation of the basic ML, the system will begin with initial guess where k=0.2 and b=12, for every evaluation in the process both k and b value will be adjusted under hidden algorithm in the method which referenced in the DotNumerics libraries. Three methods return an almost same result, therefore, the implementation is a successful implantation of the regression machine learning systems.

3 Supervised Learning

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output[6]. Supervised learning problems are categorised into "regression" and "classification" problems [7]. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

3.1 Classification Machine Learning Systems

In classification machine learning system, I am interest in yes or no prediction which in this case has been shown in the truth table for a given logic gate, where I take input matrix {1 ,0 ,0 } with an additional value 1 in the first digit, and the output will be differently by {1 ,0 ,1 } different kinds of desired logic gates. For example, the desired output for the {1 ,1 ,0 } NAND gate will be { 1, 1, 1,0}. In order to do this, I will have an algorithm of {1 ,1 ,1 } training devices, which takes input matrix and the desired output to compute the weight value for each input digit, furthermore the weight will be shown as a standard linear equation y= kx + b, in other words, assuming there is a linear y= kx+b can separate the 1 values from 0 values in the output. For instance the mathematical proof for NAND gate is shown in Figure 5 (threshold t=0.5, learning rater= 0.1).

Input values	Wights values	Sum of(input *weight) if S> t then 1, else 0	Desired output	Error	Correction
Round 1					
{ 1, 0, 0}	{ 0, 0 , 0 }	0	1	1	0.1
{ 1, 0, 1}	{ 0.1, 0 , 0 }	0	1	1	0.1
{ 1, 1, 0}	{ 0.2, 0 , 0.1 }	0	1	1	0.1
{ 1, 1, 1}	{ 0.3, 0.1 , 0.1 }	0	0	0	0
Round 2					
{ 1, 0, 0}	{ 0.3, 0.1 , 0.1 }	0	1	1	0.1
{ 1, 0, 1}	{ 0.4, 0.1 , 0.1 }	0	1	1	0.1
{ 1, 1, 0}	{ 0.5, 0.1 , 0.2 }	1	1	0	0
{ 1, 1, 1}	{ 0.5, 0.1 , 0.2 }	1	0	-1	-0.1
....					
Round 7					
{ 1, 0, 0}	{ 0.7, -0.2 ,-0.2 }	1	0	0	0
{ 1, 0, 1}	{ 0.7, -0.2 ,-0.2 }	0	1	1	0.1
{ 1, 1, 0}	{ 0.8, -0.2 ,-0.1 }	1	0	0	0
{ 1, 1, 1}	{ 0.8, -0.2 ,-0.1 }	0	0	0	0
Round 8					
{ 1, 0, 0}	{ 0.8, -0.2 ,-0.1 }	1	0	0	0
{ 1, 0, 1}	{ 0.8, -0.2 ,-0.1 }	1	0	0	0
{ 1, 1, 0}	{ 0.8, -0.2 ,-0.1 }	1	0	0	0
{ 1, 1, 1}	{ 0.8, -0.2 ,-0.1 }	0	0	0	0

Figure 12 illustrates the theoretical process of the tanning device, since the algorithm predicts yes or no (1 or 0) as output, we set the threshold value to (1+0)/2 =0.5, and the correction for the weight values happens when both input digits is 1 and the error is not 0(eg. in round 1, input {1, 0,1} with error = 1 and weight { 0.1, 0, 0}, then after this evaluation the wight values will become to { 0.1+1*0.1, 0 , 0 +1*0.1}.

The C# implementation is shown below.

```
public static Tuple<double[], int, bool, string> tranningDivace(double[][] inputs,
                        double[] outputs,
string name) {
    int count = 0;
    bool fixPoint = true;
    double threshold = 0.5;
    double learning_rate = 0.1;
    double[] weights = new double[inputs[1].Length];
    while (true && count < 20) {
        fixPoint = count + 1 == 20 ? false : true;
        count++;
        double error_count = 0;
        for (int i = 0; i < inputs.Length; i++) {
            double[] arr1 = new double[inputs[1].Length];
            for (int j = 0; j < inputs[i].Length; j++) {
                arr1[j] = inputs[i][j];
            }
            double num = 0;
            if (dot_product(arr1, weights) != threshold) {
                num = dot_product(arr1, weights) > threshold ? 1 : 0;
            } else {
                num = 0.5;
            }
            double error = outputs[i] - num;
            if (error != 0) {
                error_count += 1;
                int index = 0;
                foreach (double q in arr1) {
                    weights[index] += learning_rate * error * q;
                    index++;
                }
            }
        }
        if (error_count == 0) {
            return Tuple.Create(weights, count, fixPoint, name);
        }
    }
    return Tuple.Create(weights, count, fixPoint, name);
}

public static double dot_product(double[] values, double[] weights) {
    double sum;
    sum = values.Zip(weights, (X, Y) => X * Y).Sum();
    return sum;
}
```

(In order to make the implementation more stable I've set up a limit value where when the value returned by dot_product(Sum * weight in Figure 5) = 0.5, system will return 0.5).

The outputs for NAND are shown below.

```
------------------------------------------------------------
        Weights for NAND  Round: 8  FixPoint: True
 0.7                        -0.15                        -0.1

                Truth table for NAND

       0                    0                    1
       0                    1                    1
       1                    0                    1
       1                    1                    0
------------------------------------------------------------
```

Also the output for AND, OR and XOR.

```
------------------------------------------------------------
        Weights for AND  Round: 3  FixPoint: True
 0.2                         0.2                         0.2

                Truth table for AND

       0                    0                    0
       0                    1                    0
       1                    0                    0
       1                    1                    1
------------------------------------------------------------

------------------------------------------------------------
        Weights for OR  Round: 3  FixPoint: True
 0.35                        0.2                         0.25

                Truth table for OR

       0                    0                    0
       0                    1                    1
       1                    0                    1
       1                    1                    1
------------------------------------------------------------

------------------------------------------------------------
        Weights for XOR  Round: 20  FixPoint: False
 0.35                        0.15                         0.15

                True table for XOR

       0                    0                    0
       0                    1                    1
       1                    0                    1
       1                    1                    1
------------------------------------------------------------
```

According to the outputs above system successfully worked for the NAND, OR and AND, but not for XOR gate, base upon the implementation design above we can assume that the output can be shown as 4 points in the 2 dimensional coordinate system and there is a line can separate all the 1 values from the 0 values, the graphic view is shown in Figure 13.

According to Figure 6 there is highly unlikely to separate the value (0,0) (1,1) and (1,0) (0,1) with a single linear function, also the mathematical proof is,

Assuming the is a line y = kx+b that can separate the value (0,0) (1,1) and (1,0) (0,1).

Figure 13

Therefore,

k+ b > 0 where there k!=0.

k < 0

b < 0

Thus according to the functions above there is highly unlikely to have a k and b with k +b>0 and k<0, b<0.

In order to implement a XOR I will compose an NAND and OR to a AND. The truth table is shown below,

Input Values	**NAND (A)**	**OR(B)**	**A and B**
(**0 , 0**)	1	0	0
(**0 , 1**)	1	1	1
(**1 , 0**)	1	1	1
(**1 , 1**)	0	1	0

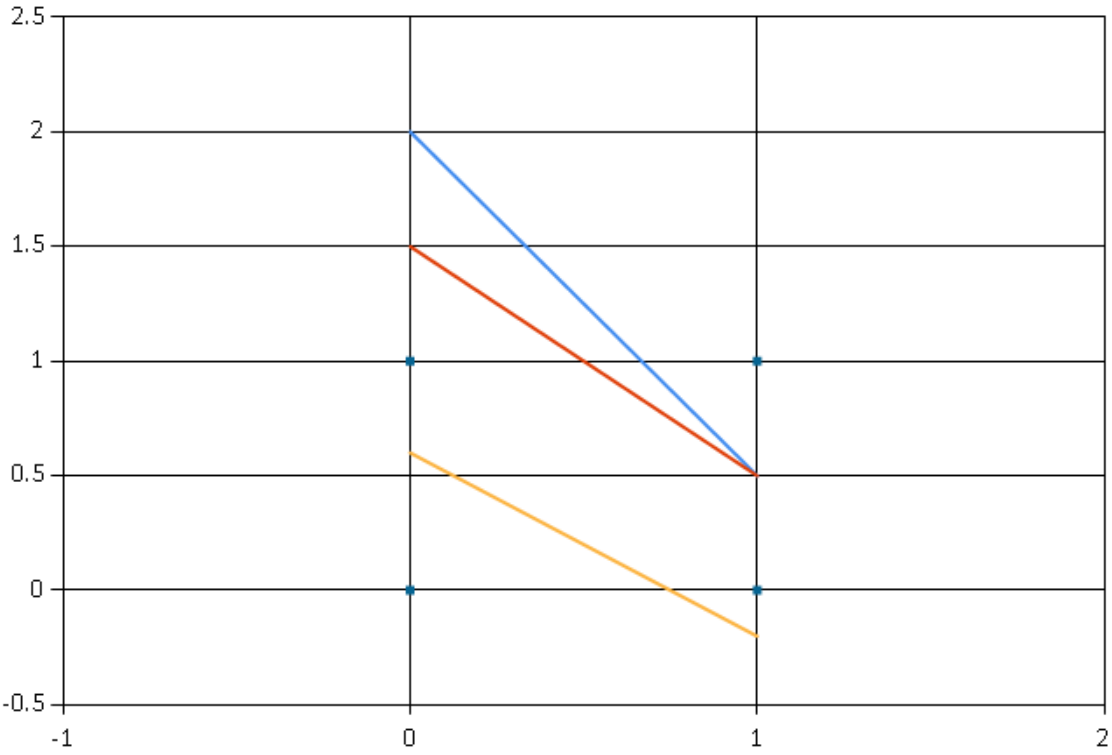
And the C# implementation is:

```
public static double[][] xorComp(double[][] input,
                    Tuple<double[], int, bool, string> nand,
                    Tuple<double[], int, bool,string>or,
                    Tuple<double[], int, bool, string> and) {
   double[][] tbXor = new double[4][];
   for (int i = 0; i < 4; i++) {
      tbXor[i] = new double[3];
      foreach (double[] arr in input) {
         tbXor[i][0] = 1.0;
         tbXor[i][1] = dot_product(input[i], nand.Item1) > 0.5 ? 1.0 : 0.0;
         tbXor[i][2] = dot_product(input[i], or.Item1) > 0.5 ? 1.0 : 0.0;
      }
   }
   return tbXor;
```

The output is:

```
------------------------------------------------------------
                Truth table for XOR

        0               0               0
        0               1               1
        1               0               1
        1               1               0
```




```
------------------------------------------------------------
```

According to the output above we can compute the XOR by combining NAND, OR and AND.

3.2 Conclusion

The process above shows a clear implementation of a basic Classification machine learning systems, the algorithm will auto correct the weight values under certain learning rate value and eventually the dot product of input value and weights will give us the desired output value, however for the XOR I've used a more complex model where the model will take the result of 2 evaluation and put into a new process and the results show a clear implementation.

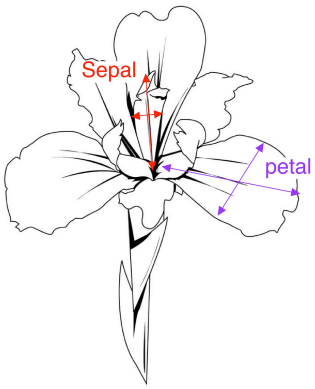
4 Neural Networks

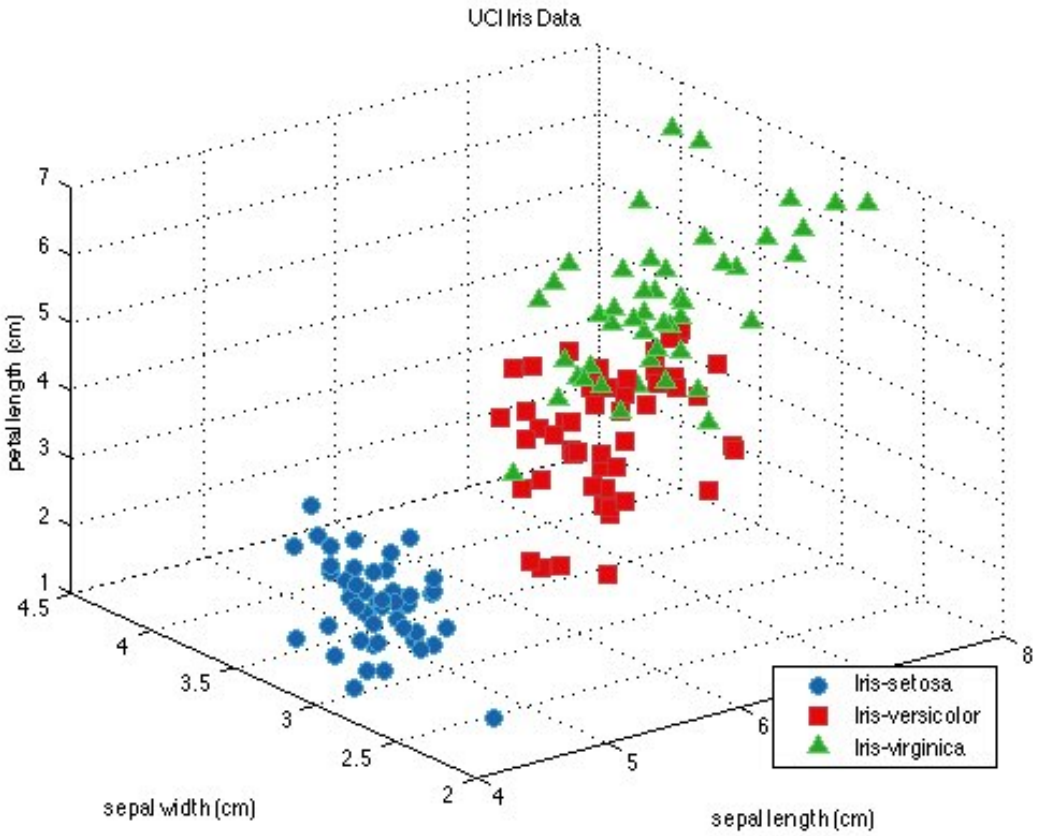
4.1 Introduction to the Neural Networks

This section provides a basic experiment of neural networks and neural network programming using the Encog Artificial Intelligence Framework and the Computational Network Toolkit (CNTK) [13]. Encog [14] is an AI framework that is available for both Java and Microsoft. NET and the Computational Network Toolkit is a unified deep-learning toolkit that describes neural networks as a series of computational steps via a directed graph.

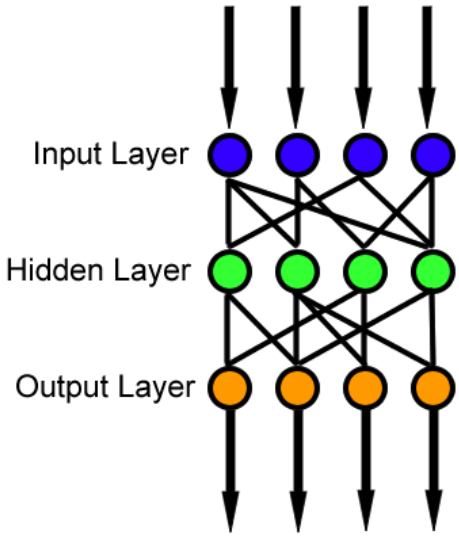
4.2 Experiment With Iris Flower Dataset

Both Encog and CNTK will be used to analyse the Iris flower data set (aka Fisher's Iris dataset) which is a multivariate data set introduced by Fisher [15]. The dataset consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versiclor). Four features were measured from each sample: the tenth and the width of the sepals and petals, in centimetres. Based on the linear discriminant model created by Fisher[15] this data set became a typical test case for many statistical classification techniques in machine learning.





The plotting using Matlab is shown above, the dataset been analysed by the feed-forward algorithm, which the networks consisted of multiple layers of computational units, usually interconnected in a feed-forward way. Each neurone in one layer has directed connection to the neurones of the subsequent layer, which a similar neurone network was described by McCulloch and Pitt [16].



4.2.1C# Experiment with Encog

The purpose is to create a program that generates a mode to predict the type of Iris, based on the four measurements. Also, this program will allow us to easily change the model type to any different model including, Feed-forward Neural Network, NEAT Neural Network and Support Vector Machine. The reason why we need different model types is sometimes the data is normalised differently by different type mode used. The program will split the training data into a training and validation set. The validations will be held until the end to see how er we can predict data that the model was not trained on.

Firstly, we define a VersatileMLDataSet object that will load from CSV file. We define the five columns of the Iris data set. The following code is responding for such,

```
VersatileDataSource source = new CSVDataSource( irisFile , false ,CSVFormat .
                                                        DecimalPoint ) ;
var data = new VersatileMLDataSet ( source ) ;
data . DefineSourceColumn (”sepal–length”, 0,ColumnType.Continuous); data .
DefineSourceColumn (”sepal–width”, 1, ColumnType.Continuous);
data . DefineSourceColumn (”petal–length”, 2,ColumnType.Continuous);
data . DefineSourceColumn (”petal–width”, 3, ColumnType.Continuous);
ColumnDefinition outputColumn = data . DefineSourceColumn (
”species”, 4,ColumnType . Nominal ) ;
data . Analyze () ;
```

The Analyse method reads the entire file and determines the minimum, maximum, mean and standard deviations for each column. Sense we are going to prodding the non-numeric column (species) we use the feedforward neural network to normalise the data, cause there is no negative values in numeric columns. The following C# code accomplishes this,

```
data . DefineSingleOutputOthersInput (columnMPG) ;
var model = new EncogModel ( data ) ;
model . SelectMethod ( data , MLMethodFactory . TypeFeedforward ) ;
model . Report = new ConsoleStatusReportable () ;
data . Normalize () ;
```

Then before we fit the model we hold back part of the data for the validation set. We choose to hold back 30%, and we chose to randomise the data set with a fix seed value, which ensures that we get the same training and validation set each time. This is a matter of preference. Finally, we fit the model with a k-fold cross-validation of size 5. The code below accomplishes the task,

```
data . LeadWindowSize = 1;
data.LagWindowSize = WindowSize;
model . HoldBackValidation (0.3 , false , 1001) ;

var bestMethod = ( IMLRegression ) model . Crossvalidate ( 5, false);
model . HoldBackValidation (0.3 , true , 1001) ;
model . SelectTrainingType ( data ) ;
var bestMethod = (IMLRegression) model.Crossvalidate(5, true);
```

The Cross-validation breaks the training dataset into 5 different combinations of training and validation data, and the Cross-validation process do not use the validation data that we previously set said, cause those data are for a final validation. At the end of the cross-validation training the cross-validated error should be displayed. The cross-validation error is an estimate how the model might perform on data that is not trained on. The implementation is showing below,

```
Console.WriteLine(@"Training error: "+ model.CalculateError(bestMethod,
model.TrainingDataset));
Console.WriteLine(@"Validation error:"+
model.CalculateError(bestMethod, model.ValidationDataset));
NormalizationHelper helper = data . NormHelper ;
Console . WriteLine ( helper . ToString ()) ;
```

From this point, the model has been trained and the best model can be saved using normal serialisation. Also, the data should be normalise in order to be used in the model, and the data should be denormalised for further use.

The output from this program will look similar to the following. First the program downloads the data set and begins training. Training occurs over 5 foods. Each fold uses a

separate portion of the training data as validation. The remaining portion of the training data is used to train the mode for that fold. Each fold gives a different model; we choose the model with the best validation score. We train until the validation score cease to improve, which helps to prevent over-fitting.

```
1/5 : Fold #1
1/5 : Fold #1/5: Iteration #1 Training Error : 1.34751708, Validation Error : 1.42040606
1/5 : Fold #1/5: Iteration #2 Training Error : 0.99412971, Validation Error : 1.42040606
…
1/5 : Fold #1/5: Iteration #47 Training Error : 0.03025748, Validation Error : 0.00397662
1/5 : Fold #1/5: Iteration #48 Training Error : 0.03007620, Validation Error : 0.00558196
```

For instance, the first fold trains for 48 iterations before it stops, and the validation error is a decent result. After fold 5 inc complete, the cross-validated score will be displayed which is the average of all 5 validation sores. This should give us a reasonable estimate of how well the model might perform on data that is was not trained with. Using the best mode from the 5 folds we now evaluate it with the training data and true validation data that we set aside earlier

```
Training error : 0.023942862952610295
Validation error : 0.061413317688009464
```

The training error and the validation error is shown above, the shows the training error is better than the validation error, this is because the model always tends to perform better on data that trained with.

Finally, we loop over the entire dataset and display predictions. 149/150 is correctly predicted and only 1 is mis-predicted. The following example shows the results,

```
[5.1, 3.5, 1.4, 0.2] –> predicted: Iris–setosa(correct: Iris–setosa)
```

```
[4.9, 3.0, 1.4, 0.2] –> predicted: Iris–setosa(correct: Iris–setosa)
...
[7.0, 3.2, 4.7, 1.4] –> predicted: Iris–versicolor(correct: Iris– versicolor)
[6.4, 3.2, 4.5, 1.5] –> predicted: Iris–versicolor(correct: Iris– versicolor)
[5.9, 3.2, 4.8, 1.8] –> predicted: Iris–virginica(correct: Iris– versicolor)  ?
...
[6.3, 3.3, 6.0, 2.5] –> predicted: Iris–virginica(correct: Iris–virginica)
```

4.2.2 Experiment with CNTK

Similarly, the purpose is to create a program that generates a model to predict the type of Iris, based on the four measurements. In order to perform this in CNTK, we take 10% of the dataset as the test dataset, which is using the model to predict the outcome, and 90% of the dataset as the training dataset, which will be using the training model.

Firstly, we need to define a training block, which contains 3 sub-blocks, the network builders, learners and the data readers. The network builders create a network using CNTK's network description language [13], in this case, we will be using the SimpleNetworkBuilder, which is the simplest Brainscript in CNTK [13] which creates one of the predefined networks with limited customisation. The coding is inspired by CNTK example simple-2D[13].

```
SimpleNetworkBuilder = [
    # 4 input, 2 50-element hidden, 3 output
    layerSizes = 4:50*2:3
    trainingCriterion = "CrossEntropyWithSoftmax"
    evalCriterion = "ErrorPrediction"
    layerTypes = "Sigmoid"
    initValueScale = 1.0
    applyMeanVarNorm = true
    uniformInit = true
    needPrior = true
]
```

The learner uses the *stochastic gradient descent(SGD)* algorithm to train the model, which is the desired trainer for most applications. In this case, the SGD has 5 parameters, the epochSize = 0 means epochSize is the size of the training set, the minibatchSize parameter

denotes the number of samples between model updates, which a sample here is defined as one vector or tensor flying through the system. The learning-rate and momentum parameter define the behaviour of the learning algorithm, and the maxEpochs is the maximum number of epochs to run.

```
SGD = [
    epochSize = 0
    minibatchSize = 25
    learningRatesPerMB = 0.5:0.2*20:0.1
    momentumPerMB = 0.9
    maxEpochs = 10
]
```

The reader is how the CNTK reads the data in this case is the CNTKTextFormatReader which will read the text- based CNTK format and it supports multiple inputs combined in the same file.

```
reader = [
    readerType = "CNTKTextFormatReader"
    file = "$DataDir$/Iris_Data_Train.txt"
    input = [
        features = [
            dim = 4       # 4-dimensional input data
            format = "dense"
        ]
        labels = [
            dim = 3       # 3-dimensional labels
            format = "dense"
        ]
    ]
```

Sense the train block defines the training network and the algorithm behaviour, the test block will be easily defined, cause the test block is a part of training dataset.

```
Simple_Demo_Test = [
    action = "test"
    reader = [
        readerType = "CNTKTextFormatReader"
        file = "$DataDir$/Iris_Data_Test.txt"
        input = [
            features = [
                dim = 4        # two-dimensional input data
                format = "dense"
            ]
            labels = [
                dim = 3        # two-dimensional labels
                format = "dense"
            ]
        ]
```

Finally, the output method will read the test data and map the outputs from the algorithm to human readable labels.

```
Simple_Demo_Output=[
    action = "write"

    reader = [
        readerType = "CNTKTextFormatReader"
        file = "$DataDir$/Iris_Data_Test.txt"
        input = [
            features = [
                dim = 4
                format = "dense"
            ]
            labels = [
                dim = 3
                format = "dense"
            ]
        ]
    ]
```

```
        outputNodeNames = PosteriorProb : labels

    outputPath = "$OutputDir$/SimpleOutput"

    format = [
        type = "category"
        labelMappingFile = "$DataDir$/IrisMapping.txt"
        sequenceEpilogue = "\t// %s\n"

    ]
]
```

The output of this program is 15/15, which means CNTK successfully predicted the type of iris, also it is important to point out, I intentionally include the mis-predicted data in Encog experiment in the test dataset and the CNTK gives the correct outcome of that data.

4.2.3 Experiment Conclusion

Since we get the decent result from both CNTK and Encog, then is pretty safe to say both CNTK and Encog will be able to solve the Iris problem, however, the dataset size is relatively small the difference in usage of CPU and memory is not obvious. Also, it is unwise to run the dataset under multi-GPU or parallel computing, cause it will lead for saving more time to analyse the data.

However, the Encog library does not require to customise our neural network, which means the network is already pre-coded into the library, on the other hand, we build the simple network in CNTK with 2 hidden layers and each layer contains 50 element nodes. Thus, in this case when dataset gets complicated and the size of the dataset get increased, in this case, is safe to say CNTK will be more flexible. Furthermore, we get 100% accuracy in CNTK tests but 149/150 accuracy by using Encog, which I think the reason is we generated a considerably complex network by sacrifice some of the performance.

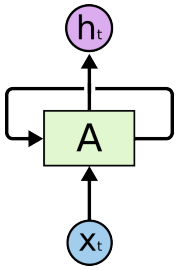
Overall as an ML student, I think both CNTK and Encog have their specialties, but the Encog is well-documented and CNTK just comes out last year. In further study, if I am facing

simple task I will prefer to use Encog instead of CNTK, but if I have the complex dataset with a lot of data in it I will use the CNTK.

4.3.1Experiment Using Long Short-term Memory Network

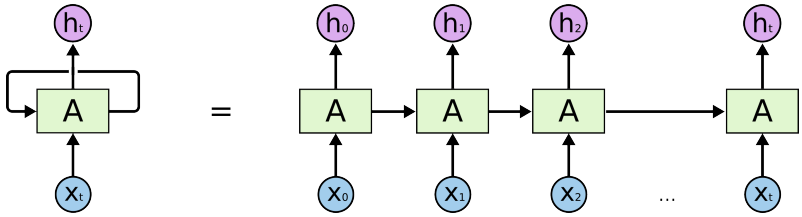
Neural network based approaches have recently produced record-setting performances in natural language understanding tasks such as word labelling. In the word labelling task, a tagger is used to assign a label to each word in an input sequence. Specifically, simple recurrent neural networks (RNN) and convolutional neural networks (CNNs) have shown to significantly outperform the previous state-of-the- art conditional random fields (CRFs). This paper investigates using long short-term memory (LSTM) neural networks, They were introduced by Hochreiter & Schmidhuber [17], which contain input, output and forgetting gates and are more advanced than simple RNN, for the word labelling task. To explicitly model output label dependence, we propose a regression model on top of the LSTM unnormalised scores. We also propose to apply LSTM to the Iris dataset [15](mentioned in pervious section) and the ATIS dataset validated the effectiveness of the proposed models.

The recurrent neural networks (RNN) address this issue. They are networks with loops in them, allowing information to persist.



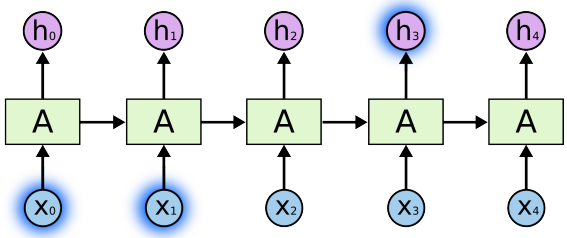
In the above diagram, a chunk of neural network A, looks at some input x and outputs a value h. A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, it is not different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Thus, is the loop is unrolled,



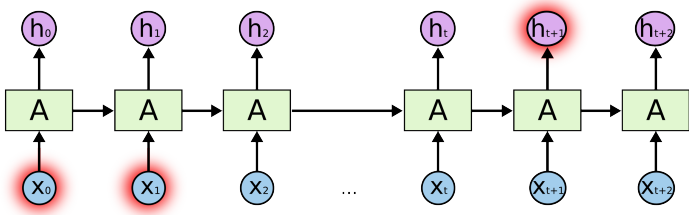
This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. Sometimes, only recent information need to be looked to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the *sky*," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France… I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

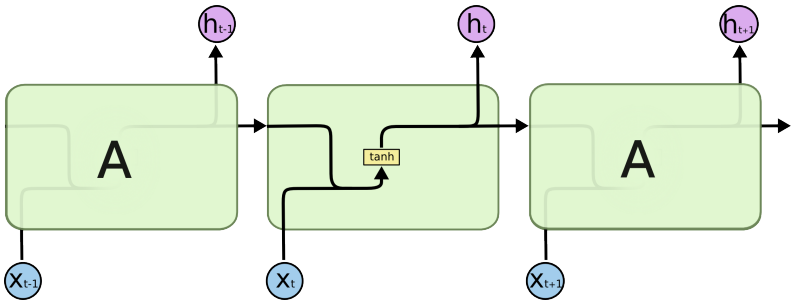
Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



In theory, RNNs are absolutely capable of handling such "long-term dependencies." A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter [18] and Bengio, et al. [19], who found some pretty fundamental reasons why it might be difficult.

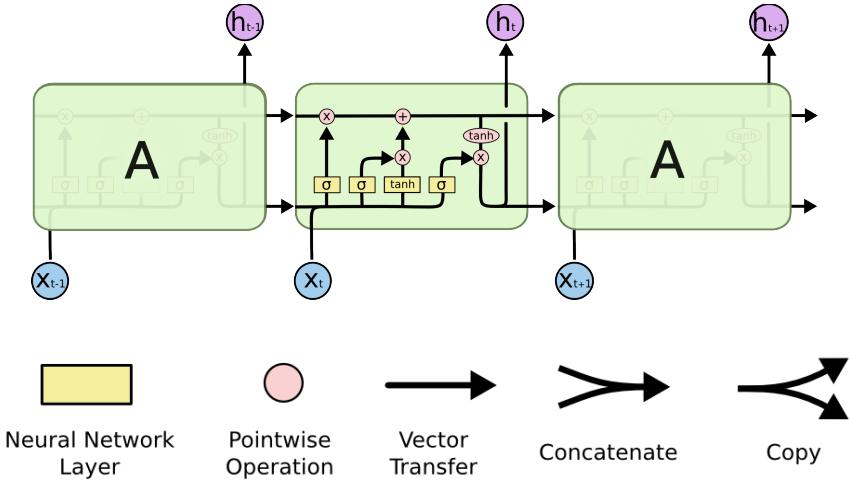
Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber [17] and were refined and popularised by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent point-wise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

In particular, RNNs have attracted much attention because of their superior performance in language modelling and understanding tasks. In common with feed-forward neural networks(mentioned in pervious section), an RNN maintains a representation for each word as a high-dimensional real-valued vector. Critically, similar words tend to be close with each other in this continuous vector space. Thus, adjusting the model parameters to increase the objective function for a particular training example tends to improve performance for similar words in the similar contexts.

In this paper we focus on spoken language understanding (SLU), in particular, word labelling with semantic information. For example, for the sentence "I want to fly from Seattle to Paris," the goal is to label the word "Seattle" and 'Paris" as the departure and arrival cities of a trip, respectively.

In this paper we apply LSTM neural networks to the SLU tasks. LSTM has some advanced properties compared to the simple RNN. It consists of a layer of inputs connected to a set of hidden memory cells, a connected set of recurrent connections amongst the hidden memory cells, and a set of output nodes. Importantly, input to and output of the memory cells are modulated in a context- sensitive way. To avoid the gradient diminishing and exploding problem, the memory cells are linearly activated and propagated between different time steps. We further imply the basic LSTM architecture to the Iris dataset to exam the performance in the arbitral dataset.

4.3.1 Recurrent Neural Network with LSTM

RNNs incorporate discrete-time dynamics. The long short- term memory (LSTM) RNN has been shown to perform better at finding and exploiting long range dependencies in the data than the simple RNN. One difference from simple RNN is that the LSTM uses a memory cell with linear activation function to store information. Note that the gradient-based error propagation scales errors by the derivative of the unit's activation function times the weight that the forward signal weight through. Using linear activation functions allows the

LSTM to preserve the value of errors because its derivative with regard to the error is one. This to some extent avoids the error exploding and diminishing problems as the linear memory cells maintains unscaled activation and error derivatives across arbitrary time lags.

The data is ATIS [22], which consists of 944 unique words and 127 different output tags which can be found in CNTK project [20, 21], in the training section. Output has 127 dimension, each corresponding to a semantic tag in ATIS. Unseen words in test will be mapped to the similar tags. A file provides such mapping from one word to the other, which is useful to map low-frequency input or unseen input to a common input. In this case, the common input is provided in the test dataset. To understand the dataset format a sample is provided below,

BOS i would like to find a flight from charlotte to Las Vegas that makes a stop in St. Louis EOS

it is converted into the following text,

Sequence	Past word	Current word	Next word	Label
1	1:1	1:1	12:1	126:1
1	1:1	12:1	39:1	126:1
1	12:1	39:1	28:1	126:1
1	39:1	28:1	3:1	126:1
1	28:1	3:1	86:1	126:1
1	3:1	86:1	15:1	126:1
1	86:1	15:1	10:1	126:1
1	15:1	10:1	4:1	126:1
1	10:1	4:1	101:1	126:1
1	4:1	101:1	3:1	48:1
1	101:1	3:1	92:1	126:1
1	3:1	92:1	90:1	78:1
1	92:1	90:1	33:1	123:1
1	90:1	33:1	338:1	126:1
1	33:1	338:1	15:1	126:1
1	338:1	15:1	132:1	126:1

1	15:1	132:1	17:1	126:1
1	132:1	17:1	72:1	126:1
1	17:1	72:1	144:1	71:1
1	72:1	144:1	2:1	119:1
1	144:1	2:1	2:1	126:1

where the first column identifies the sequence (sentence) ID, which is the same for all words of the same sentence. There are four input streams: PW, CW, NW, L. The input "PW" represents the previous word ID, "CW" for current word, and "NW" for next word. Input name "L" is for labels. Words "BOS" and "EOS" denote beginning of sentence and end of sentences respectively.

Each line above represents one sample (word). E.g. the meaning of this line:

- the sequence ID is 1
- the current word is "charlotte" whose word ID is 101
- the previous word is "from" whose ID is 4
- the next word is "to" whose ID is 3
- the semantic label is "B-fromloc.city_name" whose label Id is 48.

And in this example, we use BrainScript to create one-layer LSTM with embedding for slot tagging. The consolidated config file is ATIS.cntk. One can check the file (with some comments) for details, especially how the reader is configured in ATIS.cntk.

```
    reader=[
readerType = "CNTKTextFormatReader" file = "$DataDir$/ATIS.train.cntk.sparse"
miniBatchMode = "partial" randomize = true
input = [
featuresPW = [
alias = "PW" # previous word dim = $wordCount$
format = "sparse"
]
featuresCW = [
alias = "CW" # current word dim = $wordCount$
format = "sparse"
]
featuresNW = [
alias = "NW" # next word dim = $wordCount$
format = "sparse"
]
```

```
            labels = [
            alias = "L" # label dim = $labelCount$

    format = "sparse" ]

        ]

    ]
```

The above section tells CNTK to use CNTKTextFormatReader to read data from the file "$DataDir/ATIS.train.cntk.sparse". The same input names (PW, CW, NW, L) are used to refer inputs (features and labels) provided in data files. The input is read into different feature vectors: featuresPW, featuresCW, featuresNW and labels. These vectors are later used to build LSTM node with BrainScript[12] as follows.

```
featuresPW = Input(inputDim)
featuresCW = Input(inputDim)
featuresNW = Input(inputDim)
features = RowStack(featuresPW : featuresCW : featuresNW)

labels=Input(labelDim, tag="label")

# embedding layer
emb = LearnableParameter(embDim, featDim)

 featEmbedded = Times(emb, features)
# build the LSTM stack
lstmDims[i:0..maxLayer] = hiddenDim
NoAuxInputHook (input, lstmState) = BS.Constants.None

 lstmStack = BS.RNNs.RecurrentLSTMPStack (lstmDims,

cellDims=lstmDims,
featEmbedded,
inputDim=embDim, previousHook=BS.RNNs.PreviousHC,
augmentInputHook=BS.RNNs.NoAuxInputHook,

augmentInputDim=0, enableSelfStabilization=false)

lstmOutputLayer = Length (lstmStack)-1 LSTMoutput = lstmStack[lstmOutputLayer].h
```

The output with minibatch is 0.01884559* 10984, which means we successfully analysed 10800 words(including beginning and ending tags).

4.3.2 Iris dataset with LSTM model

We further introduce the LSTM model to the Iris dataset, but to make the dataset understandable for the CNTK reader, the format of dataset has been changed into decimal number. Which is shown below,

1 |SL 4.8 | SW 3.0 | PL 1.4 | PW 0.1 |S 1

where the first column identifies the sequence (sentence) ID, which is the same for all words of the iris flowers. There are five input streams: SL, SW, PL, PW, S. The input "SL" represents Sepal length, "SW" for Sepal width, "PL" for Petal length and “PW" for Petal width. Input name "S" is for Species. Since, there is only 150 data we reduce the size of the network by changing the input dimension.

Each line above represents one sample (flower). E.g. the meaning of this line:

- the sequence ID is 1 and 1 only since there is not a combination of Iris
- the Sepal length is 4.8 cm.
- the Sepal width is 3.0 cm.
- the Petal length is 1.4 cm.
- the Petal width is 0.1 cm.
- the semantic Species is 1 which indicates Iris setosa.

We build the neural network inspired by the ATIS example[22] provided by CNTK project. The output with minibatch is 00000* 15, which means we successfully analysed 15 flowers which randomly selected in the dataset as the test dataset. The distribution of the the dataset flows 5/5/5 (5 form the iris setosa specie, 5 form the iris virginica and 5 form the iris versiclor). Hint, in this experiment the dataset does not have to be closely related.

4.3.3 Experiment Conclusion

The ATIS experiment shows we are able to build the LSTM neural network in order to solve the spoken language understanding (SLU) problem, it successfully analysed 10984 tags with only 2% error rate. This programme can be further used in .Net web recognising and complex time series dataset classification problem. Furthermore, we successfully analysed the Iris dataset with LSTM model, which as fast as section 4.2 the simpleNetWork with feed-forward algorithm.

4.4 Experiment With Image Reader Experiment

This is the part of the CNTK where we will using CNTK more to its full potential. In previous part modes have been created to solve simple binary and multi-class classification problems. Though those models achieved good accuracy, they will not perform as well on harder real-world problems. One principal reason is that the decision boundaries between the classes are not typically linear. In this report I will learn to build more complex models and creating an image classification system using the MNIST dataset[23] as our benchmark.

4.4.1 The MNIST Dataset

MNIST is a popular image dataset of handwritten digits. It is divided into a training set of 60,000 examples, and a test set of 10,000 examples. This dataset is a subset of the original data from NIST, pre-processed and published by LeCun et al[23]. The MNIST dataset[23] has become a standard benchmark for machine learning methods because it is real-world data, yet it is simple and requires minimal efforts in pre-processing and formatting. Each instance of the data consists of a 28x28 pixel image representing one digit, with a label between 0 and 9. Here are some examples:

Furthermore, CNTK comes with a python script that fetches and prepares the MNIST data for CNTK consumption. By looking through the Python script[24], we found the Python script takes the input image and re-shaped it to a 28*28 matrix with pixel's grey-scale.

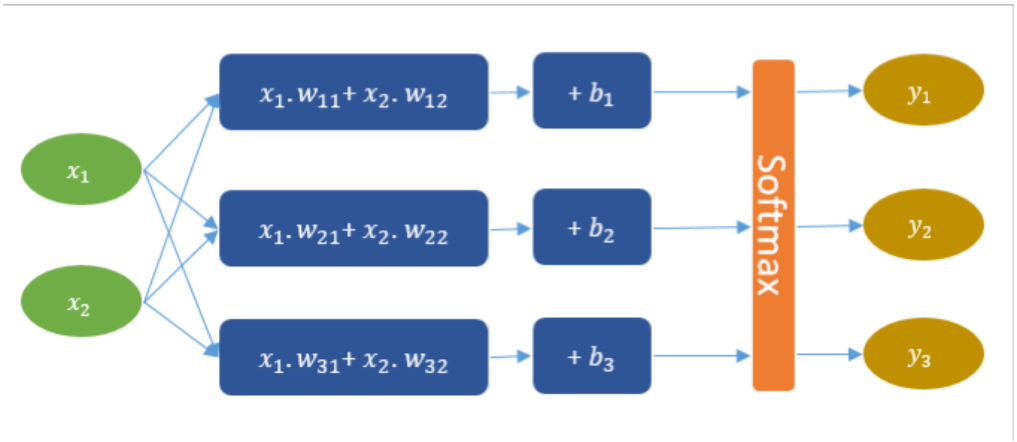
4. 4.2 Building the neural networks

In the previous work the softmax regression[12] can learn to separate data with more than two classes. However, the separation boundaries are linear. However in most realistic work the boundaries are trickier than linear, in that case, the feature space can be distorted in order to bring the data closer to being linearly separable, and this is what a hidden layer can do for us. So basically, the softmax regression solution can be inserted in a hidden layer connected to the network's input. Such a layer will learn to apply a feature mapping that projects the data into a space where it is (hopefully) linearly separable. Then, the next layer will receive an easier problem to deal with using its linear decision boundaries.

The softmax function is a generalisation of the logistic function. It maps a vector of real values to a probability distribution. It is widely used for multi-class classification problems[24]. In this context, if we are trying to predict the class of an instance out of k possible classes, the model would compute k-dimensional vector whose components represent the confidence scores for each class. Then, the softmax function would map those scores to a posterior probability distribution over the possible classes using the following formula[25]:

$$P(y = j|X) = \frac{e^{X^T W_j}}{\sum_{k=1}^{K} e^{X^T W_k}}$$

where x is our input vector, and w is the weight matrix, that includes both the model parameters and the bias. We want to define a computational network that looks like the one in the figure below. This network can be understood as a combination of three linear models, each of which will be trained to separate one of the three classes from the other two. Then, on top, we have the softmax layer that squashes the linear model scores into a probability distribution. Thus, for each instance, the network outputs three probability values that sum up to one. For example, if a given instance is of class (1), the model might output the following probabilities (95%, 3%, 2%), which basically means that the probability of the instance belonging to class (1) is 95%.



Let's get back to the task at hand: classifying images of hand-written digits. To do so, we will build our first neural network with CNTK. Starting simple, our network will only have one hidden layer., in CNTK's MINIST example[23], we apply a scaling of (1.0 / 256.0) on the features in order to have their values within the range 0 to 1. This normalisation helps SGD to converge faster with better predictive performance. Then, we specify the topology of our network. The follow code is responsible for such,

```
BrainScriptNetworkBuilder= [ include "Shared.bs"

featDim = 28 * 28 # number of pixels labelDim = 10 # number of distinct labels

features = Input (featDim)
featScaled = Constant (1.0 / 256.0) .* features labels = Input (labelDim)

hiddenDim = 200

h1 = DNNSigmoidLayer (featDim, hiddenDim, featScaled, 1) z = DNNLayer (hiddenDim,
labelDim, h1, 1)

ce = CrossEntropyWithSoftmax (labels, z) errs = ErrorPrediction (labels, z) top5Errs =
ErrorPrediction (labels, z, topN=5)

featureNodes = (features) labelNodes = (labels) criterionNodes = (ce) evaluationNodes =
(errs) outputNodes = (z)

]
```

With a shared BrainScript,

```
DNNLayer (inDim, outDim, x, parmScale) = [
W = Parameter (outDim, inDim, init="uniform", initValueScale=parmScale,

initOnCPUOnly=true)
b = Parameter (outDim, 1, init="fixedValue", value=0) z=W*x+b

].z
```

Also the SGD (Stochastic Gradient Descent) block[2] tells CNTK how to optimise the network to find the best parameters. This includes information about mini-batch size (so the computation is more efficient), the learning rate, and how many epochs to train. Here is the SGD block for our example:

```
SGD = [
epochSize = 60000
minibatchSize = 32 learningRatesPerSample = 0.003125

momentumAsTimeConstant = 0

maxEpochs = 30

]
```

where the epochSize indicates the number of samples to use in each epoch. An intermediate model and other check point information are saved for each epoch. The minibatchSize means using minibatch size of 128 for the first two epochs and then 1024 for the rest, which in other words, it is the number of samples processed between model updates. The learningRatePerSample means the learning rates per epoch with which each sample's gradient updates the model, similarly the momentumAsTime constant is CNTK specifies momentum in a minibatch-size agnostic way as the time constant. The maxEpochs indicts the maximum number of epochs to run.

To run the example we got:

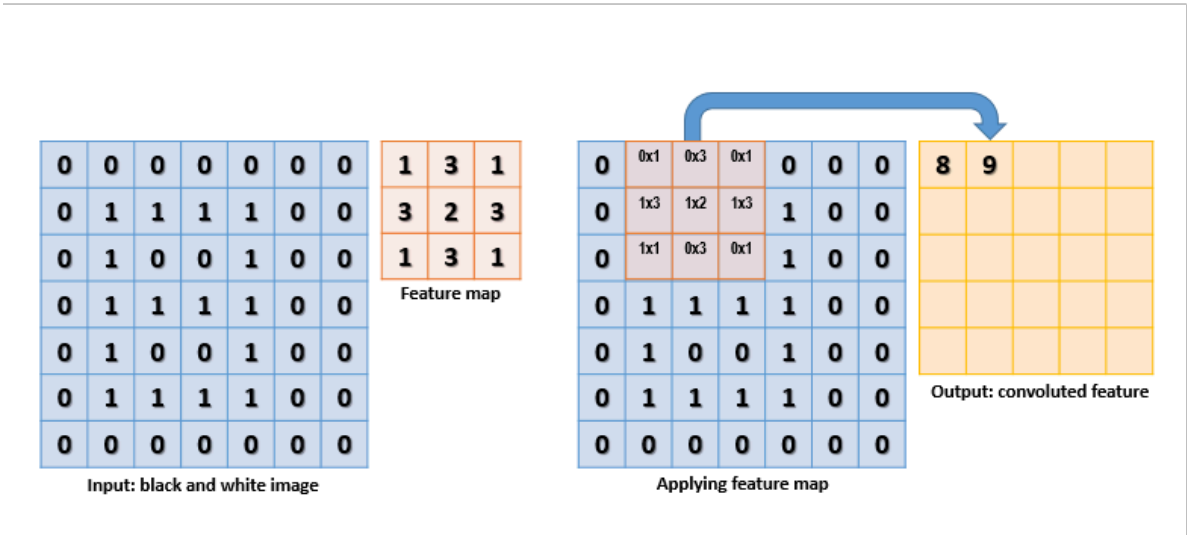
```
Final Results: Minibatch[1-625]: errs = 2.250% * 10000; top5errs = 0.040% * 10000
```

This model has an error of *2.25%,* which is not too bad for image recognition. However, by using Convolutional Neural Networks, we can improve the result even better.

4.4.3A deeper network with Convolutional Neural Networks

As we have seen, a simple neural network can achieve impressive results on the MNIST dataset[22]. However, these results are not all that good when compared to what is out there in the literature . But we can do much better if we introduce some concepts from CNN theory[26].

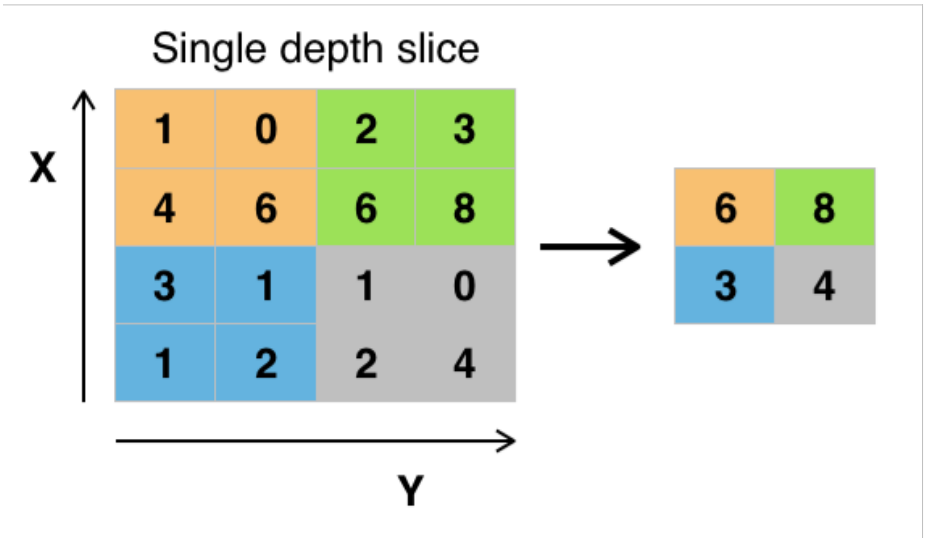
A convolutional neural network (CNN) consists of model layers that apply local filters and are stacked in a certain order. A convolutional layer applies a learned local linear filter, e.g. of [3*3] pixels, to every pixel position in an image. Such filter capture local patterns thanks to the local connectivity of its units. Whereas the input image has a single dimension (gray-scale level), hidden layers maintain their spatial layout but store an entire activation vector for each pixel position, where each dimension of the activation vector has its own filter kernel, or feature map[27]. A feature map is basically a sliding window over sub-regions of the layer's inputs, where each application of the map results in one dimension of the output activation vector. A feature map is computed by performing a dot product between its filter parameters and the corresponding input rectangle, which is slid across the entire 2D plane. This linear filter operation is called convolution. Each feature map is called a depth slice. Here is a simple example of how the feature map is applied.



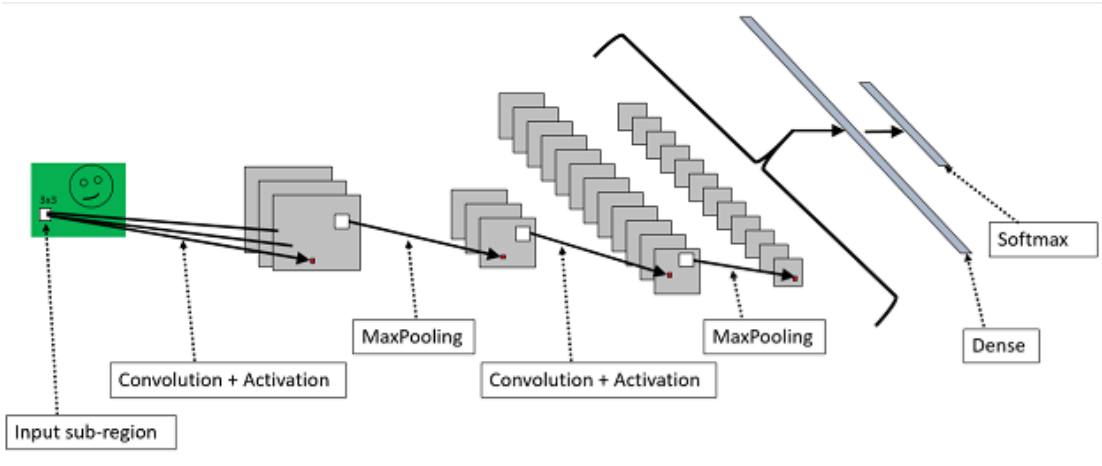
To do so, an activation function layer has been used, which A non-linear activation function is then applied to each unit output element of the convolutional layer. One of the most commonly used functions is the Rectified Linear Unit[28], or ReLU, which is simply

max(0,*x*). Its practical advantage over a sigmoid function is that it does not suffer from the vanishing gradient problem, and therefore learning can be more efficient.

The "Max Pooling" layer[28] has been also used after the activation function that aims at reducing dimensionality. It divides the input into a set of non-overlapping regions, where for each region it outputs the maximum activation value (independently for each depth slice). By reducing each region into a single point, the image dimension is reduced. What this achieves is two-fold: (1) it reduces the number of parameters and thus helps controlling overfitting; and (2) it selects the salient activation values regardless of their location in the region, which helps training models that are more resilient to things like rotation / translation. Here is an example (from Wikipedia[25]) of max pooling with a window size of [2*2] (which would reduce a hidden layer of [28 * 28 *16]to [14 * 14 *16].



Finally, after cascading several convolutional, activation function, and MaxPooling layers, a CNN will have one or more fully connected, or dense, layers. Every unit in a dense layer has connections to all activations of the previous layer, similar to regular neural networks. Furthermore, he softmax layer for classification. We know the softmax from the first part of the tutorial. The output is a probability distribution over the possible classes. Below is a chart of a CNN with two alternating convolution / activation and MaxPooling layers, one dense layer, and one softmax layer given by CNTK example[22].



4.4.4 The Network Definition

We define each sample as a 28* 28matrix rather than a vector. This is because a CNN exploits local correlations in the image. Thus, we need to preserve this information. Second, in addition to the layer we saw in the previous network, we define a cascade of convolutional and max-pooling layers. We have two of each type. The core layer is ConvReLULayer which is defined as a function in Shared.bs. Here is what this macro looks like:

```
ConvReLULayer (inp, outMap, inWCount, kW, kH, hStride, vStride) = [

convW = Parameter (outMap, inWCount, init="uniform", initValueScale=wScale)
convB = ImageParameter (1, 1, outMap, init="fixedValue", value=bValue,
                                                imageLayout="$imageLayout$")

conv = Convolution (convW, inp, kW, kH, outMap, hStride, vStride, zeroPadding=false,
                                                imageLayout="$imageLayout$")

act = RectifiedLinear (conv + convB)

].act
```

The Convolution built-in function that convolves (filters) the image with a kernel.We will also need the MaxPooling operation. This allows us to put together the final network definition of our CNN that will learn to classify images of hand-written digits. Note the reason why we have 2 MaxPooling layer is the method reduce the matrix by half util we get [7*7], further reduce will cause a redundant in dataset:

```
wScale = 10 ; bValue = 1 # in Shared.bs imageW = 28
imageH = 28
labelDim = 10

features = ImageInput (imageW, imageH, 1, imageLayout="$imageLayou$")

featScaled = Constant(1/256) .* features)
labels = InputValue (labelDim)
kW1 = 5

kH1 = 5
cMap1 = 16
hStride1 = 1
vStride1 = 1
conv1_act = ConvReLULayer (featScaled, cMap1, 25, kW1, kH1, hStride1, vStride1)

pool1W = 2
pool1H = 2
pool1hStride = 2
pool1vStride = 2
pool1 = MaxPooling (conv1_act, pool1W, pool1H, pool1hStride, pool1vStride,
                                                imageLayout="$imageLayout$")

kW2 = 5
kH2 = 5
cMap2 = 32
hStride2 = 1
vStride2 = 1
conv2_act = ConvReLULayer (pool1, cMap2, 400, kW2, kH2, hStride2, vStride2, 10, 1)

pool2W = 2
pool2H = 2
pool2hStride = 2
pool2vStride = 2
pool2 = MaxPooling (conv2_act, pool2W, pool2H, pool2hStride, pool2vStride,
                                                imageLayout="$imageLayout$")

h1Dim = 128
h1 = DNNSigmoidLayer (512, h1Dim, pool2, 1)
```

```
ol = DNNLayer (h1Dim, labelDim, h1, 1)
ce = CrossEntropyWithSoftmax (labels, ol)

errs = ErrorPrediction (labels, ol)
featureNodes = (features)
labelNodes = (labels)
criterionNodes = (ce)
evaluationNodes = (errs)
outputNodes = (ol)
```

The output for the test results on the console:

```
Final Results: Minibatch[1-625]: errs = 0.83% * 10000;

ce = 0.02825477 * 10000
```

With an error of *0.83%*, this model (unsurprisingly) greatly outperforms the previous one. It took only 2-3 minutes and 15 epochs to train on a single GPU, and it took 15-20 minutes and 15 epochs to train on CPU-only.

4.4.5 Conclusion

After analysing all the samples in CNTK project[22], in an addition to wade variety of built-in computation nodes, CNTK provides a plug-in architecture allowing users to define their own computation nodes. There are 5 algorithms in CNTK: Feed-Forward, CNN, RNN, LSTM and Sequence-to-Sequence. Compare to other toolkits like Encog, CNTK does offer a highly concentrated approach which focus on solving analysing both similarity and accuracy of the dataset.

Unfortunately, as a Machine Learning toolkit, CNTK does not have a produce method(for now), that can further use the analysing output to actually produce something. As a analysing tool CNTK has a wade combination of analysing tricks, which in my perspective it is easy to use and easy to learn. However, the reader in CNTK is confused, in LSTM mode experiment the ATIS[22] dataset has been manually reshaped into CNTK supported reader format.

Hence, in my perspective, CNTK do need a better data reader and CNTK should be used as a part of a project not all of the project, in other words, we should pack the CNTK result into other program or other code.

5 Overall Conclusion

For this year, my focus for Btech 451 is mainly on the algorithms for Machine Learning (ML). Like I said in the beginning of this project, ML is coming into its own, with a growing recognition that ML can play a key role in a wide range of critical applications, such as data mining, natural language processing, image recognition, and expert systems. ML provides potential solutions in all these domains and more, and is set to be a pillar of our future civilisation , also in 1997, Tom Mitchell gave a "well-posed" definition that has proven more useful to engineering types: "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. [31]". For the past year, we've covered much of the basic theory underlying the field of Machine Learning here, including the traditional libraries Accord and Encog, also the new technology CNTK, but of course, we have only barely scratched the surface. However, to implement the theories to the real life like the Iris dataset experiment and two- coins flipping problem, gives us a deeper understanding of the topics. One thing for sure, there are many libraries and approaches in ML and no one is perfect, to choice the must suitable method for the scenarios appears to be the most necessary step in the way of solving ML problems.

The traditional method like the Accord library and the Encog library can be considered as the first step to the ML world, cause it is not only well- documented and it is based on textbook programming languages like C# and Java. However, these method has its limitations, for instance, when I use this method it feels a bit method-oriented, the user must understand how this method works and when it can be used.

Cleary, CNTK appears to be the next thing for both Microsoft research and AL community. However, in my perspective, CNTK is still in it's beta phase, and it needs to be improved. For instance, CNTK is computational toolkit which only provides us a way of analysing which means it will not actually producing something, which is perfectly okey for traditional method, cause it has so many other interface and libraries as the backup. Sometimes CNTK feels a bit isolated by it's own kind, unlike the Caffe [32] developed by the Berkeley Vision and Learning Centre it has a forward method to actually predict something really meaningful. However, CNTK does have the capability to involve GPU and multi-GPUs to the computing part, and it is easy to implement. This capability generally reduces the run-time by 90% on a considerably big dataset., which means the CNTK has a better potential than the traditional algorithms.

Hence, in my point of view, ML is the calculator for the mathematicians, the microscope for the doctors, or even the canvas for the artist and fingerboard for the musicals. It still needs a better mind to drive, to navigate.

Note. All the experiments was run on the Microsoft Surface Pro 4 with Intel Core i5 and 8 GB Ram.

Reference

[1] Carbonell, J. G., Michalski, R. S., & Mitchell, T. M. (1983). An overview of machine learning. In *Machine learning* (pp. 3-23). Springer Berlin Heidelberg.

[2] Samuel, L. (1959). Some studies in machine learning using the game of checkers. IBM Journal of research and development, 3(3), 210-229.

[3] Nick, M (2016). An Introduction to Machine Learning Theory and Its Applications: A Visual Tutorial with Examples. [Weblog]. Retrieved 4 April 2016, from https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer

[4] Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of machine learning*. MIT press.

Zhu, X. (2005). Semi-supervised learning literature survey.

[5] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar (2012) *Foundations of Machine Learning*, The MIT Press ISBN 9780262018258.

[6] Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, *37*(3), 277-296.

[7] Rosenblatt, Frank (1957), The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[8] Norvig, P. R., & Intelligence, S. A. (2002). A modern approach.

[9] Dhillon, I. S., Guan, Y., & Kogan, J. (2002). Iterative clustering of high dimensional text data augmented by local search. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on* (pp. 131-138). IEEE.

[10] Bradley, P. S., & Fayyad, U. M. (1998, July). Refining Initial Points for K-Means Clustering. In *ICML* (Vol. 98, pp. 91-99).

[11] Arthur, D., & Vassilvitskii, S. (2007, January). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 1027-1035). Society for Industrial and Applied Mathematics.

[12] Bishop, C. M. (2006). Pattern Recognition. *Machine Learning*.

Nick, M (2016, no-date). An Introduction to Machine Learning Theory and Its Applications: A Visual Tutorial with Examples. [Weblog]. Retrieved 4 April 2016, from https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer

[13] *Microsoft/CNTK*. (2016). *GitHub*. Retrieved 22 October 2016, from https://github.com/Microsoft/CNTK

[14] *Encog Machine Learning Framework*. (2016). *Heatonresearch.com*. Retrieved 22 October 2016, from http://www.heatonresearch.com/encog/

[15] The iris dataset can be download within follow link, https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

[16] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115-133.

[17] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

[18] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München*, 91.

[19] Bengio, Y., Paolo Frasconi, and P. Simard. "The problem of learning long-term dependencies in recurrent networks." *Neural Networks, 1993., IEEE International Conference on*. IEEE, 1993.

[20] *Microsoft/CNTK*. (2016). *GitHub*. Retrieved 22 October 2016, from https://github.com/Microsoft/CNTK/blob/master/Examples/Text/ATIS/Data/ATIS.vocab

[21] *Microsoft/CNTK*. (2016). *GitHub*. Retrieved 22 October 2016, from https://github.com/Microsoft/CNTK/blob/master/Examples/Text/ATIS/Data/ATIS.label

[22] *Microsoft/CNTK*. (2016). *GitHub*. Retrieved 22 October 2016, from https://github.com/Microsoft/CNTK/tree/master/Examples/Text/ATIS

[23] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. (2016). Yann.lecun.com. Retrieved 22 October 2016, from http://yann.lecun.com/exdb/mnist/

[24] Wang, X., Fouhey, D., & Gupta, A. (2015). Designing deep networks for surface normal estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 539-547).

[25] Softmax Regression - Ufldl. (2016). Ufldl.stanford.edu. Retrieved 22 October 2016, from http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression

[26] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

[27] Dahl, G. E., Sainath, T. N., & Hinton, G. E. (2013, May). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 8609-8613). IEEE.

[28] "Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation". *DeepLearning 0.1*. LISA Lab. Retrieved 31 August 2013.

[29] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

[30] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02), 107-116.

[31] Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, *45*, 37.

[32] *Caffe | Deep Learning Framework*. (2016). *Caffe.berkeleyvision.org*. Retrieved 23 October 2016, from http://caffe.berkeleyvision.org

R	5	5	5	6	5	7	5	8	5	9	4	9	3	9	2	9	1	9
1	4.68	6.83	4.79	6.03	4.82	6.93	5.19	8.18	5.00	8.44	3.77	8.04	3.16	9.18	1.88	8.99	1.01	9.21
2	4.74	7.25	3.8	6.03	3.75	7.36	5.05	8.27	5.25	9.00	4.75	8.88	3.32	8.90	1.82	8.66	0.92	9.07
3	6.66	4.99	4.09	5.84	4.64	6.58	4.28	8.40	4.78	9.08	4.4	9.29	2.7	9.17	2	8.90	1.22	8.79
4	5.73	4.26	4.92	6.45	4.76	7.46	4.74	8.24	5.42	8.86	3.59	8.94	2.97	9.38	1.99	9.02	1.17	9.05
5	3.87	5.62	5.19	6.4	5.56	7.11	4.75	7.98	4.6	9.24	3.95	9.01	3.06	9.34	2.05	8.80	0.72	8.90
6	4.43	6.33	5.1	6.33	5.53	7.52	4.09	7.52	4.94	9.27	3.98	9.65	2.6	8.84	1.86	9.02	1.22	9.04
7	3.9	5.47	4.45	6.16	4.44	6.9	4.7	8.00	5.20	8.79	4.1	9.10	2.95	9.22	1.71	9.05	0.71	9.03
8	4.47	4.21	5.45	6.87	3.04	6.49	4.93	7.73	4.94	9.58	3.89	9.38	3.03	8.99	1.56	8.78	1.22	9.15
9	4.93	6.68	4.66	6.38	4.88	7.09	5.09	8.28	4.74	9.23	4.35	8.65	2.78	8.69	1.91	9.16	0.99	8.75
10	5.49	4.14	4.49	6.25	4.75	7.50	4.43	8.27	5.24	8.91	4.01	7.76	3.05	9.02	2.35	9.11	0.88	8.84
11	5.4	3.87	4.31	5.98	4.89	7.19	5.28	8.02	4.79	9.15	3.78	9.04	2.97	8.71	1.97	9.4	0.95	8.79
12	4.37	5.75	4.12	5.45	5.00	7.45	4.92	8.34	5.33	9.01	3.98	8.78	2.99	8.9	1.69	9.03	0.97	9.36
13	4.81	5.96	5.00	7.33	5.90	8.25	5.16	7.34	4.71	8.54	4.01	8.76	3.20	8.68	2.05	9.05	1.06	9.55
14	5.05	4.33	3.25	6.00	5.05	7.14	5.19	8.02	4.64	9.12	3.88	8.89	3.26	8.97	2.20	9.00	1.00	8.68
15	4.5	5.63	5.1	5.98	4.35	6.75	5.54	8.03	4.75	9.00	3.83	9.21	3.25	9.29	2.47	9.05	0.81	8.72
16	5.5	4.06	5.17	6.48	5.4	7.59	4.66	8.14	5.14	8.93	3.71	9.32	3.18	9.06	2.52	8.86	1.18	9.28
17	6.1	4.83	3	5.49	5.27	6.43	5.58	7.79	4.36	9.16	3.71	8.82	3.05	9.17	2.02	9.33	0.56	9.31
18	5.14	3	4.91	6.37	4.55	6.30	5.17	7.87	5.28	9.04	3.54	8.99	3.01	9.05	1.90	8.61	1.18	8.58
19	5.11	4.29	4	5.98	3.27	6.19	5.19	7.89	4.84	9.20	3.91	8.88	3.06	9.5	2.34	8.98	0.93	8.92
20	4.49	5.67	4.26	5.51	4.68	7.34	4.47	8	5.38	8.85	4.3	8.99	2.28	9.14	2.68	9.12	0.91	8.63
21	5.04	3.5	5.16	7	5.11	7.55	5.38	7.69	4.70	8.93	3.64	9.11	2.96	8.66	1.97	9.07	1.11	8.88
22	4.92	3	4.12	5.73	5.07	6.94	5.31	8.33	4.97	9.02	4.10	8.92	2.62	8.79	2.18	8.92	1.06	9.09
23	4	5.44	4.87	6.02	5.18	7.21	4.59	7.49	4.49	9.08	5	8.69	3.49	10	2.56	9.31	0.95	9.58
24	4.47	5.67	4.71	6.16	5.13	7.81	4.69	8.28	4.59	9.11	4.06	9.41	2.06	8.81	2.11	8.75	1.14	8.88
25	4.73	8.5	5.15	5.93	4.81	7.06	5.21	8.38	4.37	9.00	3.98	9.2	3.20	8.87	2.38	9.25	0.92	9.2

R	5	5	5	6	5	7	5	8	5	9	4	9	3	9	2	9	1	9
26	4.73	8.5	5.15	5.93	4.81	7.06	5.21	8.38	4.37	9.00	3.98	9.2	3.20	8.87	2.38	9.25	0.92	9.2
27	5.12	2.75	0	5.89	5.34	6.47	5.13	8.16	5.18	8.94	4.02	8.98	3.02	8.89	2.08	9.41	1.00	9.25
28	3.16	4.97	4.06	5.88	4.49	6.81	5.07	8.04	4.93	8.75	4.48	8.98	2.52	9.01	2.51	8.97	1.04	8.86
29	5.64	3.33	2	5.7	5.09	7.21	4.76	7.85	5.36	8.66	4.41	8.92	2.76	9.29	1.74	9.02	1.38	8.91
30	5.98	4.2	4.56	6.17	4.13	7.08	4.43	7.64	4.64	8.73	4.32	8.9	2.99	8.97	1.96	8.97	0.98	8.83
31	5.5	4.93	3.96	5.76	4.99	8.12	4.5	7.18	4.78	9.17	3.55	9.06	2.88	8.95	2.03	9.35	1.20	8.83
32	6.83	4.95	4.29	5.88	4.89	7.11	4.55	7.76	5.08	9.08	3.86	9.07	2.83	9.16	2.04	9.17	0.77	9.01
33	4.05	6.1	5.35	7.33	5.06	6.30	4.90	8.01	5.67	9.18	3.31	8.89	2.83	9.03	2.11	9.40	1.21	8.90
34	6.58	4.62	4.29	5.6	4.76	7.25	5.20	8	4.79	9.19	4.06	9.16	2.30	9.18	2.27	8.77	0.72	9.19
35	5.04	4.08	5.14	6.23	5.02	6.91	5.25	8.13	5.12	8.64	3.36	8.68	3.39	8.87	1.97	9.13	0.96	8.86
36	4.13	5.34	4.84	6.06	4.46	7.4	4.64	7.84	5.08	8.70	3.36	8.89	3.26	8.91	1.76	8.54	1.09	8.95
37	4.56	5.78	4.37	6.04	5.00	7.09	4.62	8.1	4.38	9.28	4.13	8.67	3.22	9.13	1.42	9.11	1.01	9.10
38	5.65	4.70	5.5	8	4.86	7.35	4.43	7.70	4.89	8.71	4.07	8.57	2.50	8.79	2.31	8.98	1.15	9.74
39	4.03	5.69	5.46	6.46	4.99	7.5	3.7	7.58	5.12	9.12	3.65	8.88	3.25	8.97	2.18	9.17	1.13	9.03
40	5.33	8	4.94	6.67	5.64	7.32	4.29	8.33	5.46	8.68	3.49	8.81	2.57	9.44	2.66	9.01	0.83	9.01
41	5.17	3.27	4.66	6.05	5.33	6.96	5.37	8.00	4.58	8.71	4.8	9.28	2.99	8.68	1.79	9.02	1.19	8.79
42	4.66	6.66	4.33	5.64	5.46	6.81	5	8.18	4.68	8.90	3.73	4.75	3.45	9.18	1.83	8.86	1.15	9.06
43	4.2	5.61	5.14	5.93	5.16	7.02	4.73	8.07	5.35	8.48	3.98	8.69	2.45	9.25	2.13	8.72	1.20	8.82
44	4	5.29	4.42	6.18	5.17	7.00	5.87	8.59	5.05	8.88	3.3	8.87	2.89	9.08	1.93	9.22	0.76	9.04
45	4.99	6.6	4.64	6.03	2	6.25	5.05	8.09	4.57	9.04	3.78	9.01	2.90	9.16	1.97	9.14	0.98	9.06
46	4.56	8	5.47	6.83	4.63	6.67	4.07	7.67	4.49	8.70	4.12	8.72	2.69	8.86	2.07	8.87	0.88	9.22
47	4.24	5.56	3.64	5.83	5.17	8.02	6.09	8	5.01	9.25	3.42	8.68	3.05	8.79	1.88	8.93	0.46	9.33
48	5.75	4.72	4.41	6.22	4.55	7.56	4.05	7.00	4.28	9.25	3.72	8.95	2.37	9.05	1.66	9.12	1.29	9.32
49	4.5	5.63	4.96	6.21	1	5.9	4.98	7.91	3.4	8.96	7.7	9.08	2.65	9.01	2.75	8.98	0.97	9.1
50	3.98	5.14	4.58	6.07	4.72	6.29	3	7.62	5.23	8.79	3.56	8.6	2.59	8.41	1.84	9.25	0.98	9.12