

DELHI TECHNOLOGICAL UNIVERSITY

Representation Learning and Nature Encoded Fusion Technique for Heterogeneous Sensor Networks

EC-353: Computer Vision

Anish Sachdeva

MC/2K16/013





Representation Learning and Nature Encoded Fusion Technique for Heterogeneous Sensor Networks

11.12.2019

Anish Sachdeva

DTU/2K16/MC/013

Under Prof. Dr. Rajiv Kapoor

With assistance from PhD Students Rahul Kuma and Nikhil Singh



Acknowledgements

I would like to express my special thanks of gratitude to my teacher Prof Dr. Rajiv Kapoor of the Electronics and Communication Engineering department as well as the research faculty under Prof. Dr. Rajiv Kapoor at Delhi Technological University who gave me a golden opportunity to do this wonderful project on the Representation Learning and Nature Encoded Fusion for Heterogeneous Sensor Networks, which exposed me to the fascinating world of Neural networks and deep learning.

Secondly i would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

Index

1. Implementation.....	3-19
a. Driver Program.....	3-4
b. Implementation of Auxiliary Functions.....	4-9
c. Implementation of Belief propagation Algorithm.....	9-19
2. Output.....	20
3. Introduction and Terminology.....	21-27
a. Heterogeneous Sensor Network.....	21
b. Multimodal Data Fusion.....	22
c. Sensor Fusion.....	23
d. Representation Learning.....	23
e. Neural Networks.....	24-25
f. Neyman-Pearson Lemma.....	25
g. Belief Propagation.....	26
h. Gaussian Beleif Propogation.....	27
4. About the Authors.....	28
a. Longwei Wang	
b. Qilin Liang	
5. Bibliography.....	29

Implementation

Driver Program

```
% Extracting features from the audio signal
[audio_info, audio_features] = getInfoAndPlotGraph(331,
'recording_cv.m4a');
disp('Audio details'); disp(audio_info);

[frames, video] = getFramesFromVideo('video_introduction.mp4');
qualities = getImageQualities(video.NumFrames, frames);

mle_mfcc = mle(audio_info.mfcc);
mle_ssim = mle(qualities);

pd = makedist('Normal','mu',0,'sigma',1);
pdf_mfcc = pdf(pd, audio_info.mfcc);
pdf_mfcc(14 : 49) = 0;
pdf_ssim = pdf(pd, qualities);

llr = log(pdf_mfcc / pdf_ssim);
disp('LLR of the heterogenous system: '); disp(llr);

[p0, p1] = getCorrespondingProbabilities(llr);

kl = KLDiv(pdf_mfcc, pdf_ssim);
disp('Kullback-Leibler divergence: '); disp(kl);
```

```

cross_corr = xcorr(pdf_mfcc, pdf_ssim);
subplot(332); plot(cross_corr);

mfcc_shifted(2:50) = pdf_mfcc;
cross_self = xcorr(pdf_mfcc, mfcc_shifted);
cross_corr(98:99) = 0;
sum_corr = cross_self + cross_corr;
subplot(333); plot(sum_corr);

pdf_mfcc_shifted(2:50) = pdf_mfcc;

audio_bp = belief_propagation(length(audio_info.mfcc),
audio_info.mfcc);
video_bp = belief_propagation(length(qualities), qualities);
llr_bp = belief_propagation(5, [1, -2, -1, -2, -1]);
llr_bp2 = belief_propagation(5, [1, -1, -2, -5, -6]);

subplot(334); plot(audio_bp);
subplot(335); plot(video_bp);
subplot(336); plot(llr_bp);
subplot(337); plot(llr_bp2);

```

Implementation of Auxiliary Functions

```

function [p0, p1] = getCorrespondingProbabilities(log_likelyhood)
    p0 = exp(log_likelyhood) / (1 + exp(log_likelyhood)) ;
    p1 = 1 - p0;
end

```

```

function indexes = getImageQualities(num_frames, frames)
    reference_image = getReferenceImage('video_introduction.mp4',
50);
    indexes = [];
    for i = 1:num_frames
        image = frames(1:720, 1280*(i-1) + 1:1280*i, 1:3);
        ssimval = ssim(image, reference_image );
        indexes = [indexes ssimval];
    end
end

```

```

function image = getReferenceImage(fileName, nTest)
    image = uint8(getBack(fileName, nTest, 'median'));
end

```

```

function backGrnd = getBack(fileName, nTest, method)
    if nargin < 2, nTest = 20; end
    if nargin < 3, method = 'median'; end
    v = VideoReader(fileName);
    nChannel = size(readFrame(v), 3);
    tTest = linspace(0, v.Duration-1/v.FrameRate , nTest);
    %allocate room for buffer
    buff = NaN([v.Height, v.Width, nChannel, nTest]);
    for fi = 1:nTest
        v.CurrentTime =tTest(fi);
        % read current frame and update model
        buff(:, :, :, mod(fi, nTest) + 1) = readFrame(v);
    end
    switch lower(method)

```

```

        case 'median'
            backGrnd = nanmedian(buff, 4);
        case 'mean'
            backGrnd = nanmean(buff, 4);
        end
    end
end

function [frames, video] = getFramesFromVideo(fileName)
    video = VideoReader(fileName);
    frames = [];
    for img = 1:video.NumFrames
        b = read(video, img);
        frames = [frames b];
    end
end

function [audio_info, audio_features] =
    getInfoAndPlotGraph(subplotPosition, fileName)
    plotPitchGraph(subplotPosition, fileName);
    [audio_info, audio_features] = getAudioInfo(fileName);
end

function plotPitchGraph(subplotPosition, fileName)
    [audioIn, fs] = audioread(fileName);
    [audioInfo, audioFeatures] = getAudioInfo(fileName);

    t = linspace(0, size(audioIn, 1)/fs, size(audioFeatures, 1));
    subplot(subplotPosition);
    plot(t, audioFeatures(:, audioInfo.pitch));

```



```

        title('Pitch');
        xlabel('Time (s)');
        ylabel('Frequency (Hz)');
    end

function [audio_info, audio_features] = getAudioInfo(fileName)
    [audio, fs] = audioread(fileName);

    aFE = audioFeatureExtractor( ...
        "SampleRate",fs, ...
        "Window",hamming(round(0.03*fs),"periodic"), ...
        "OverlapLength",round(0.02*fs), ...
        "mfcc",true, ...
        "mfccDelta",true, ...
        "mfccDeltaDelta",true, ...
        "pitch",true, ...
        "spectralCentroid",true);

    audio_features = extract(aFE, audio);
    audio_info = info(aFE);
end

function dist = KLDiv(P,Q)
    % dist = KLDiv(P,Q) Kullback-Leibler divergence of two discrete
    probability
    % distributions
    % P and Q are automatically normalised to have the sum of one
    on rows
    % have the length of one at each

```

```

% P =  n x nbins
% Q =  1 x nbins or n x nbins(one to one)
% dist = n x 1

if size(P,2)~=size(Q,2)
    error('the number of columns in P and Q should be the same');
end

if sum(~isfinite(P(:))) + sum(~isfinite(Q(:)))
    error('the inputs contain non-finite values!')
end

% normalizing the P and Q
if size(Q,1)==1
    Q = Q ./sum(Q);
    P = P ./ repmat(sum(P,2),[1 size(P,2)]);
    temp = P.*log(P./repmat(Q,[size(P,1) 1]));
    temp(isnan(temp))=0;% resolving the case when P(i)==0
    dist = sum(temp,2);

elseif size(Q,1)==size(P,1)

    Q = Q ./ repmat(sum(Q,2),[1 size(Q,2)]);
    P = P ./ repmat(sum(P,2),[1 size(P,2)]);
    temp = P.*log(P./Q);
    temp(isnan(temp))=0; % resolving the case when P(i)==0
    dist = sum(temp,2);
end

```

```
end
```

```
function f = gauss_distribution(x, mu, s)
    p1 = -.5 * ((x - mu)/s) ^ 2;
    p2 = (s * sqrt(2*pi));
    f = exp(p1) / p2;
end
```

Implementation of the Belief Propagation Algorithm

```
function result = belief_propagation(size, array)
    n = size;
    m = n-1;
    H=spalloc(n-1,n,2*(n-1));
    for i=1:n-1
        H(i, i)=1;
        H(i, i+1)=1;
    end
    G = generatormatrix(H, m , n);
    spy(H);
    spy(G);
    H2DS(H, m, n);

    k = n - m;
    message = double(rand(k,1) >0.5);
    x = G*message;
    x_tilde = x;
    iscodeword(hard_decision(x_tilde, H), H);

    sigma = 0.5;
```

```

y = x_tilde+randn(n,1) * sigma^2;
TRIES = 100; t = 0;
while iscodeword(hard_decision(y, H), H) && t<TRIES
    y = x_tilde + randn(n,1) * sigma;
    t = t+1;
end

T = 50;
u = 4*y/(2*sigma^2);
if ~any(mod(H*double(y<0),2))
    disp('y already represents a codeword');
    clf ;
else
    disp('performing BP');
    Y = iterate_BP(T,u);
    for k=1:T
        if ~any(mod(H*double(Y(:,k) <0),2))
            strcat('found a codeword at iteration ', num2str(k))
            break;
        end
    end
    plot (0:T,Y,'o-');
end

result = iterate_BP(100, array');

function bool = hard_decision(u, H)
    bool = ~any(mod(H*(double(u<0)),2));
end

```

```

function bool = iscodeword(y, H)
    bool = true;
end

function H = generate_H(m,n,d)
    H = sparse(m,n);
    H = mod(H,2);
    while not( all (sum(H,1)>=2) && all (sum(H,2)>=2))
        H = H+abs(sprand(m,n,d))>0;
        H = mod(H,2);
    end
end

function [G] = generatormatrix(H, m , n)
    Hp = H;
    colperm = 1:n;
    for j=1:m
        i=min(find(Hp( j :m, j )));
        if isempty(i)
            k=min(max(find(Hp(j ,:)) , j ));
            if isempty(k)
                disp(['problem in row', num2str(j ,0)]);
                continue;
            end

            temp = Hp(: , j );
            Hp(: , j )=Hp(: , k );
            Hp(: , k )=temp;
        end
    end
end

```

```

        temp=colperm(k );
        colperm(k)=colperm( j );
        colperm( j)=temp;
    end

    i=i+j-1;
    if (i~=j)
        temp = Hp(j ,:);
        Hp(j ,:)=Hp(i ,:);
        Hp(i ,:)=temp;
    end

    K= find(Hp(: , j ));
    K= K(find(K~=j));

    if isempty(K)
        t1=full (Hp(j ,:));
        for k=K
            t2=full (Hp(k ,:));
            temp=xor(t1 , t2 );
            Hp(k ,:)=sparse(temp);
        end
    end

end

A = Hp(: ,m+1:n);
[b ,invperm] = sort(colperm);
G = [A; speye(n-m)];
G = G(invperm, :);

```

end

% The belief propogation algorithm

function [] = H2DS(H, m, n)

global B P S q;

q = nnz(H);

P = spalloc(q ,q , (sum(H,2)-1)' * sum(H,2));

S = spalloc(q ,q , (sum(H,1)-1) * sum(H,1)');

k=0;

for j=1:n

I=find(H(:, j));

for x=1:length(I)

for y=x+1:length(I)

P(k+x ,k+y)=1;

P(k+y,k+x)=1;

end

end

k=k+length(I);

end

k=0;

for i=1:m

J=find(H(i ,:));

for x=1:length(J)

for y=x+1:length(J)

S (k+x ,k+y)=1;

```

        S (k+y,k+x)=1;
    end
end
k=k+length(J);
end

B=spalloc(q ,n,q );
b=[];
for k=1:m
    b=[b find(H(k , : ) ) ] ;
end
B=sparse ([1: q]',' , b', ones(q ,1) ,q ,n);
end

function y = S(x)
    global S_ q;

    y=ones(q ,1);
    for i=1:q
        for j=find(S_(i ,:))
            y(i) = y(i) * tanh(x(j)/2);
        end
    end
    y=2*atanh(y);
end

function y = iterate_BP(T,u)
    global B P S q;

```



```

x1_k = zeros(q ,1);
x2_k = zeros(q ,1);
x1_k_1 = zeros(q ,1);
x2_k_1 = zeros(q ,1);

y=zeros(n,T+1);
for t=1:T
    x1_k_1 = P*x2_k + B*u;
    x2_k_1 = S(threshold_1(x1_k));
    y(:, t) = B' * x2_k + u;
    x1_k = x1_k_1;
    x2_k = x2_k_1;
end
end

function val = threshold_2(number)
    if number < 0
        number = 1;
    end
    val = int8(number) + 1;
    disp(val);
end

function val = threshold_1(number)
    val = int8(abs(number)) + 1;
end

function plot_BP_output(y, mono, filename)
    T=size(y);

```

```

T = T(2)-1;

clf ;
if nargin>1
    plot (0:T,y,'ko-') % monochrome
else
    plot (0:T,y,'o-') % c o l o r
end
grid on;
axis ([0 T min(min(y))-0.5 max(max(y))+0.5])
LEGEND=[];
for k=1:n
    LEGEND = [LEGEND ; strcat('output', num2str(k))];
end
legend(LEGEND)
xlabel('time');
ylabel('LLR');

if nargin==3
    if isstr (filename )
        saveas(gcf , filename , 'eps');
    end
end
end

function [H, final_column_weights , final_row_weights] =
MacKayNealCreateCode(n,r ,v ,h)

m = floor(n*(1-r ));
H = zeros ([m,n]);
alpha = [];

```

```

for i = 1:length(v)
    for j = 1:( floor(v(i)*n))
        alpha = [ alpha i ];
    end
end

while (length(alpha) ~= n)
    alpha = [ alpha i ];
end
beta = [];

for i=1:length(h)
    for j=1:(floor(h( i )*m))
        beta = [beta, i ];
    end
end

while(length(beta) ~= m)
    beta = [beta i];
end

for i = 1:n
    c = [];
    beta_temp = beta;
    for j = 1: alpha( i )
        temp_row = randi(1 ,1 ,[1 ,m]);
        while ((( beta_temp(temp_row) == 0) &&
(max(beta_temp) > 0)) || (( beta_temp(temp_row) <= -1)))
            temp_row = mod(temp_row+1,m)+1;
        end
    end
end

```

```

        end
        c = [c temp_row];
        beta_temp(temp_row) = -10;
    end


    for k = 1:length(c)
        beta(c(k)) = beta(c(k))-1;
    end

    for j = 1:alpha(i)
        H(c(j), i) = 1;
    end
end

column_weights = H'*ones(m,1);
for i = 1:max(column_weights)
    count = 0;
    for j = 1:length(column_weights)
        if (column_weights(j) == i)
            count = count + 1;
        end
    end
    final_column_weights(i) = count/length(column_weights);
end

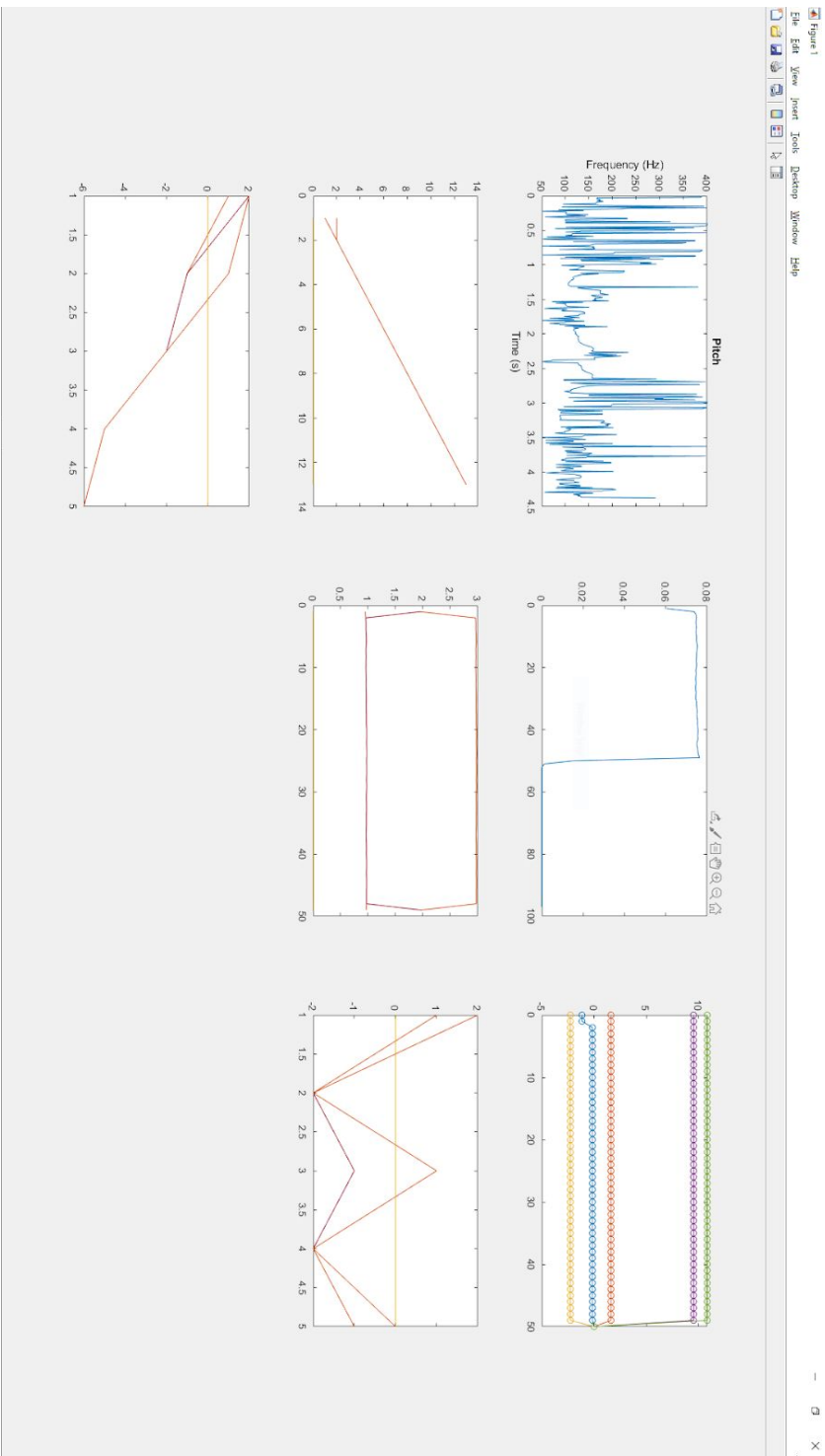
row_weights = H*ones(n,1);
for i = 1:max(row_weights)
    count = 0;
    for j = 1:length(row_weights)

```



```
        if (row_weights(j) == i)
            count = count + 1;
        end
    end
    final_row_weights(i) = count/length(row_weights);
end
end
end
```

Output



Introduction and Terminology in the Research Paper


Target detection based on heterogeneous sensor networks was considered in this paper. Fusion problem is investigated to fully take advantage of the information of multi-modal data. The sensing data may not be compatible with each other due to heterogeneous sensing modalities, and the joint PDF of the sensors is not easily available. A two-stage fusion method was proposed to solve the heterogeneous data fusion problem. First, the multi-modality data is transformed into the same representation form by a certain linear or nonlinear transformation. Since there is a model mismatch among the different modalities, each modality is trained by an individual statistical model. In this way, the information of different modalities is preserved. Then, the representation is used as the input of the probabilistic fusion. The probabilistic framework allows data from different modalities to be processed in a unified information fusion space. The inherent inter-sensor relationship is exploited to encode the original sensor data on a graph. Iterative belief propagation is used to fuse the local sensing belief. The more general correlation case is also considered, in which the relation between two sensors is characterized by the correlation factor.

In this paper a new technique was introduced to combine and enhance the nature encoded model of digital image processing for heterogeneous sensors such that the probability distribution for each sensor and modality is different.

And this new method is used to better understand and represent the same data using information coming from an array of heterogeneous networks with different modalities.

Heterogeneous Sensor Networks

Heterogeneous sensor networks with multiple sensing modalities are gaining increasing popularity because they can provide several advantages for performance improvement in different realistic scenarios. Fusion of data from heterogeneous modalities, observing a




certain phenomenon, has been shown to improve the performance of many surveillance and monitoring tasks. The key motivation is that sensors of different modalities will provide richer information than a single sensor, or even several sensors of the same modality. Take the audio-visual fusion in human speech communication for example. Speech with the help of visual cues from the lip movements can enhance the intelligibility of speech. The aid of visual cues increase redundancy and makes the speech more robust to noise and interruptions.

Two sensors are said to be heterogeneous if their respective observation models cannot be described by the same probability density function. The question of how to integrate the data from such diversity of modalities. If the joint probability density function (PDF) under each hypothesis is known a priori, the optimal performance by the Neyman-Pearson rule for detection (binary hypothesis testing) can be easily obtained. However, in practice, this information may not be available. This usually happens when the dimensionality of the sample data is high and when there are not enough training samples to accurately estimate the joint PDF. If the data from heterogeneous sensors are independent under certain hypothesis and the local individual sensing data is correctly received in the fusion center, the optimal fusion decision can be obtained by the conventional product rule. However, the problem becomes complicated when the condition that independence among the sensors does not hold. In this paper, the scenario that the joint distribution between the sensors being unknown is considered. This is commonly seen in heterogeneous sensor networks, i.e., sensors with disparate sensing modalities. For example, it is not immediately clear how one could model the joint distributions between data of an audio and a video sensor monitoring a common target of interest.

Multimodal Data Fusion

Data fusion is the process of integrating multiple data sources to produce more consistent, accurate, and useful information than that provided by any individual data source.

Data fusion processes are often categorized as low, intermediate, or high, depending on the processing stage at which fusion takes place. Low-level data fusion combines several sources of raw data to produce new raw data. The expectation is that fused data is more **informative** and **synthetic** than the original inputs.



Data fusion processes are often categorized as low, intermediate, or high, depending on the processing stage at which fusion takes place. Low-level data fusion combines several sources of raw data to produce new raw data. The expectation is that fused data is more **informative** and **synthetic** than the original inputs.

Sensor Fusion

Sensor fusion is the combining of sensory data or data derived from disparate sources such that the resulting information has less uncertainty than would be possible when these sources were used individually. The term *uncertainty reduction* in this case can mean more accurate, more complete, or more dependable, or refer to the result of an emerging view, such as stereoscopic vision (calculation of depth information by combining two-dimensional images from two cameras at slightly different viewpoints).

The data sources for a fusion process are not specified to originate from identical sensors. One can distinguish *direct fusion*, *indirect fusion* and fusion of the outputs of the former two. Direct fusion is the fusion of sensor data from a set of heterogeneous or homogeneous sensors, soft sensors, and history values of sensor data, while indirect fusion uses information sources like *a priori* knowledge about the environment and human input.

Sensor fusion is also known as *(multi-sensor) data fusion* and is a subset of *information fusion*.

Representation Learning

In machine learning, feature learning or representation learning is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task.

Feature learning is motivated by the fact that machine learning tasks such as classification often require input that is mathematically and computationally convenient to process. However, real-world data such as images, video, and sensor data has not yielded to attempts to algorithmically define specific features. An alternative is to discover such features or representations through examination, without relying on explicit algorithms.

Feature learning can be either supervised or unsupervised.

- In supervised feature learning, features are learned using labeled input data. Examples include supervised neural networks, multilayer perceptron and (supervised) dictionary learning.
- In unsupervised feature learning, features are learned with unlabeled input data. Examples include dictionary learning, independent component analysis, [autoencoders](#), matrix factorization and various forms of clustering.

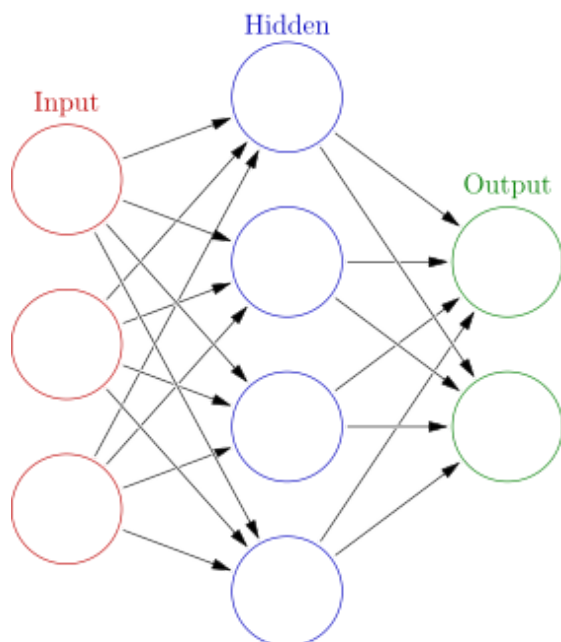
Neural Networks

Artificial neural networks (ANN) or connectionist systems are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge of cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the examples that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the [neurons](#) in a biological brain. Each connection, like the [synapses](#) in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it.

In ANN implementations, the "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called *edges*. Neurons and edges typically have a *weight* that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a **human brain** would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. ANNs have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games, medical diagnosis and even in activities that have traditionally been considered as reserved to humans, like painting.



Neyman-Pearson Lemma

In statistics, the **Neyman-Pearson lemma** was introduced by Jerzy Neyman and Egon Pearson in a paper in 1933.

Suppose one is performing a hypothesis test between two simple hypotheses $H_0: \theta = \theta_0$ and $H_1: \theta = \theta_1$ using the likelihood-ratio test with η threshold which rejects H_0 in favour of H_1 at a significance level of $\alpha = P(\Lambda(x) \leq \eta \mid H_0)$,

Where,

$$\Lambda(x) := \frac{\mathcal{L}(\theta_0 \mid x)}{\mathcal{L}(\theta_1 \mid x)}$$

and $\mathcal{L}(\theta | x)$ is the likelihood function. Then, the Neyman–Pearson lemma states that the likelihood ratio, $\Lambda(x)$ is the **most powerful test** at significance level α .

Belief Propagation

Belief propagation, also known as **sum-product message passing**, is a message-passing algorithm for performing inference on graphical models, such as Bayesian networks and Markov random fields. It calculates the marginal distribution for each unobserved node (or variable), conditional on any observed nodes (or variables). Belief propagation is commonly used in artificial intelligence and information theory and has demonstrated empirical success in numerous applications including low-density parity-check codes, turbo codes, free energy approximation, and satisfiability.

The algorithm was first proposed by Judea Pearl in 1982, who formulated it as an exact inference algorithm on trees, which was later extended to polytrees. While it is not exact on general graphs, it has been shown to be a useful approximate algorithm.

If $X=\{X_i\}$ is a set of discrete random variables with a joint mass function p , the marginal distribution of a single X_i is simply the summation of p over all other variables:


$$p_{X_i}(x_i) = \sum_{\mathbf{x}': x'_i = x_i} p(\mathbf{x}').$$

However, this quickly becomes computationally prohibitive: if there are 100 binary variables, then one needs to sum over $2^{99} \approx 6.338 \times 10^{29}$ possible values. By exploiting the polytree structure, belief propagation allows the marginals to be computed much more efficiently.

Gaussian Belief Propagation

Gaussian belief propagation is a variant of the belief propagation algorithm when the underlying **distributions are Gaussian**. The first work analyzing this special model was the seminal work of Weiss and Freeman.

The GaBP algorithm solves the following marginalization problem:


$$P(x_i) = \frac{1}{Z} \int_{j \neq i} \exp(-1/2 x^T A x + b^T x) dx_j$$

About the Authors

LONGWEI WANG

He is currently pursuing a Ph.D. degree in electrical engineering with The University of Texas at Arlington. His current research interest includes signal processing and wireless communications.

QILIAN LIANG

Received the B.S. degree from Wuhan University, in 1993, the M.S. degree from the Beijing University of Posts and Telecommunications, in 1996, and the Ph.D. degree from the University of Southern California (USC), in 2000, all in electrical engineering. Prior to joining the faculty of The University of Texas at Arlington, in 2002, he was a Member of Technical Staff at Hughes Network Systems Inc., San Diego, CA, USA. He is currently a Distinguished University Professor with the Department of Electrical Engineering, The University of Texas at Arlington. He has published over 300 journal and conference papers and seven book chapters, and holds six U.S. patents pending. His research interests include radar sensor networks, wireless sensor networks, wireless communications, compressive sensing, smart grid, signal processing for communications, and fuzzy logic systems and applications. He received the 2002 IEEE TRANSACTIONS ON FUZZY SYSTEMS Outstanding Paper Award, the 2003 U.S. Office of Naval Research (ONR) Young Investigator Award, the 2005 UTA College of Engineering Outstanding Young Faculty Award, 2007, 2009, and 2010 U.S. Air Force Summer Faculty Fellowship Program Award, the 2012 UTA College of Engineering Excellence in Research Award, and the 2013 UTA Outstanding Research Achievement Award, and he was inducted into the UTA Academy of Distinguished Scholars, in 2015.

Bibliography

1. <https://www.google.com>
2. <https://www.wikipedia.com>
3. <https://www.bing.com>
4. <https://www.springer.com/in>
5. <https://in.mathworks.com/help/matlab/>
6. <https://www.python.org/>
7. <https://www.tensorflow.org/federated>
8. <https://github.com/tensorflow>
9. <https://www.ieee.org/>
10. <https://ieeexplore.ieee.org/document/8673763>
11. <https://www.jetbrains.com/pycharm/>
12. <https://keras.io/>