Neural Organization and Deep Neural Networks

AI functions of Neural Networks

- Fault tolerance
- Collective decisions
- Combinatorial optimization
- Parallel and distributed processes
- Predictions
- Visions
- Perception
- Natural language processes
- Acoustic processes
- Machine translation
- Motor control
- Classification: pattern recognition
- Neural Regressions and Approximation
- Pattern encoding and decoding
- Memory storage and association

MATLAB R2019a - academic use

HOME | PLOTS | APPS | EDITOR | PUBLISH | VIEW

Search Documentation | Sign In

New | Open | Save | Find Files | Compare | Print — FILE
Go To | Find — NAVIGATE
Insert | Comment | Indent — EDIT
Breakpoints — BREAKPOINTS
Run | Run and Advance | Run Section | Advance | Run and Time — RUN

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ code2019 ▸ code2006 ▸ Apps ▸ NEN ▸ Large_Scaled_NEN

Current Folder

Name

- distance.m
- dline.m
- dome_nba.m
- face_3d_0001.mat
- faceNet_p4_new.fig
- faceNet_p4_new.mat
- faceNet_p4.fig
- faceNet_p4.mat
- faceNet_p4__back.mat
- filtering_3Dface_elstic_nen.m
- gdr_classy.m
- gen_AF.m
- gen_clustered_data.m
- gen_clustered_data2.m
- interpolating_3Dface_elstic...
- large_scaled_nen.m
- large_scaled_nen_pdp_versi...
- learn_en.m
- lip_sub_problem_deep_lerni...
- nbaa.m
- net_40x40.jpg
- p2_face.mat
- p3_face.mat
- p4_face.mat
- p4_face_filtered.mat
- p4_face_large_en_temp2.mat

faceNet_p4_new.fig (Figure)

No details available

Editor - /Users/apple/Desktop/Jiann-Ming Wu/code2019/code2006/Apps/NEN/Large_Scaled...

+11 | call_learning_en.m | remoteparfor.m | plot_subface.m | demo_faceNet_p4_plot.m

```
1  function demo_faceNet_p4_p
2 -
3 -
4 -
5 -
6 -
7 -                                  ,3)
8 -
9 -                                  3Df
10
11                 %plot3(X(:,1),X(:,2),X(:,3
12 -               n = sqrt(size(Net,2));
```

Figure 1

File Edit View Inser Tool Deskto Windo Help

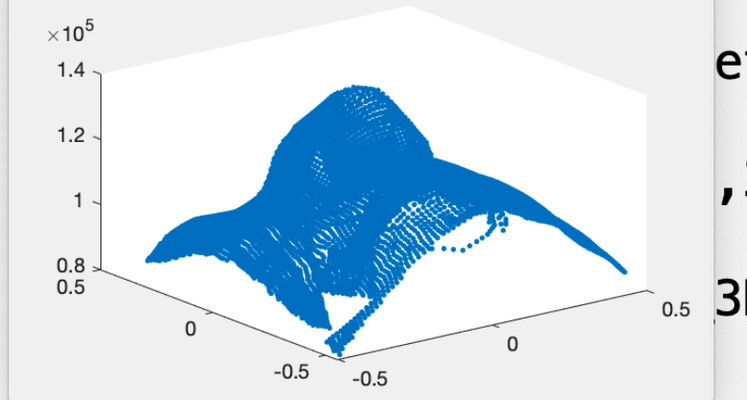
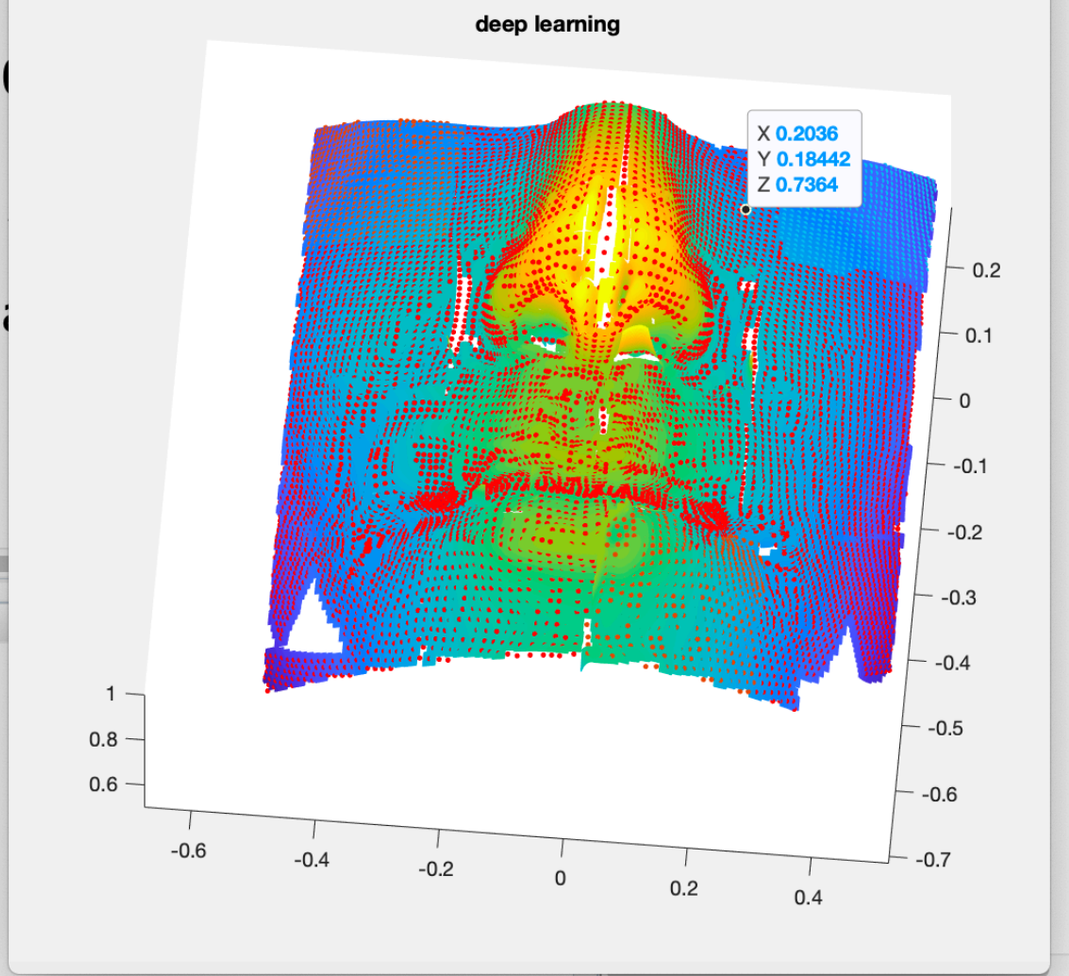


Figure 2

File Edit View Insert Tools Desktop Window Help



Workspace

Name | Value

Command Window

```
>> close all
>> load p4_face.mat
>> plot3(X(:,1),X(:,2),y,'.')
>> open('faceNet_p4_new.fig')
fx >>
```

Ln 1 Col 1

LETTER Communicated by Carsten Peterson

Natural Discriminant Analysis Using Interactive Potts Models

Jiann-Ming Wu
jmwu@server.am.ndhu.edu.tw
Department of Applied Mathematics, National Donghwa University, Shoufeng, Hualien 941,Taiwan, Republic of China

Natural discriminant analysis based on interactive Potts models is developed in this work. A generative model composed of piece-wise multivariate gaussian distributions is used to characterize the input space, exploring the embedded clustering and mixing structures and developing proper internal representations of input parameters. The maximization of a log-likelihood function measuring the fitness of all input parameters to the generative model, and the minimization of a design cost summing up square errors between posterior outputs and desired outputs constitutes a mathematical framework for discriminant analysis. We apply a hybrid of the mean-field annealing and the gradient-descent methods to the optimization of this framework and obtain multiple sets of interactive dynamics, which realize coupled Potts models for discriminant analysis. The new learning process is a whole process of component analysis, clustering analysis, and labeling analysis. Its major improvement compared to the radial basis function and the support vector machine is described by using some artificial examples and a real-world application to breast cancer diagnosis.

MATLAB R2019a - academic use

HOME | PLOTS | APPS | EDITOR | PUBLISH | VIEW

Search Documentation | Sign In

New | Open | Save | Find Files | Compare | Print — FILE

Go To | Find — NAVIGATE

Insert | Comment | Indent — EDIT

Breakpoints — BREAKPOINTS

Run | Run and Advance | Run Section | Advance | Run and Time — RUN

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ code2019 ▸ code2006 ▸ Apps ▸ Classification ▸

Current Folder

Name

demo_breas
demo_digit
demo_iris.a
demo_iris.m
evaluated.m
gdr_class1.
hc.m
hierarchical
k_clusterin
kernel_sort
kld.m
Make_Para.
mccExclude
mfa2_class
mfa_class.m
plotsig.m
potts_class
Potts_net.m
PottsNDA.c
PottsNDA.e
PottsNDA.fi
PottsNDA.m
pottsnda_m
pottsnda_m
setpath.m
testplot3.m

PottsNDA.m (Fu

Application M-fi
PottsNDA.fig

PottsNDA(var
pushbuttonS
pushbuttonE
sliderCategory_Callback(h, e...

34x2 double
34x1 double

Potts Discriminant Analysis

Information of training dataset

FileName : <<None>>

Data Size : <<UnKnown>> Category Size : <<UnKnown>>

IT Lab
Applied Math
NDHU

Load training data

Information of testing dataset

FileName : <<None>>

Data Size : <<UnKnown>> Category Size : <<UnKnown>>

Load testing data

Parameters of PottsDA

Initial Catagory	10
Initial Temperature	8.0
Shrink Temperature	0.98
Constant Weight	7.5

Active parameters

Results

Test : Unknown Non Fit : Unknown Fit : Unknown

Start Training

Save Result

Accurate

Hit point : Unknown Train Rate : Unknown Test Rate : Unknown

Exit

Caffe

Deep learning framework
by BAIR

Created by
Yangqing Jia
Lead Developer
Evan Shelhamer

View On GitHub

Installation

Prior to installing, have a glance through this guide and take note of the details for your platform. We install and run Caffe on Ubuntu 16.04–12.04, OS X 10.11–10.8, and through Docker and AWS. The official Makefile and `Makefile.config` build are complemented by a community CMake build.

Step-by-step Instructions:

- Docker setup *out-of-the-box brewing*
- Ubuntu installation *the standard platform*
- Debian installation *install caffe with a single command*
- OS X installation
- RHEL / CentOS / Fedora installation
- Windows *see the Windows branch led by Guillaume Dumont*
- OpenCL *see the OpenCL branch led by Fabian Tschopp*
- AWS AMI *pre-configured for AWS*

Overview:

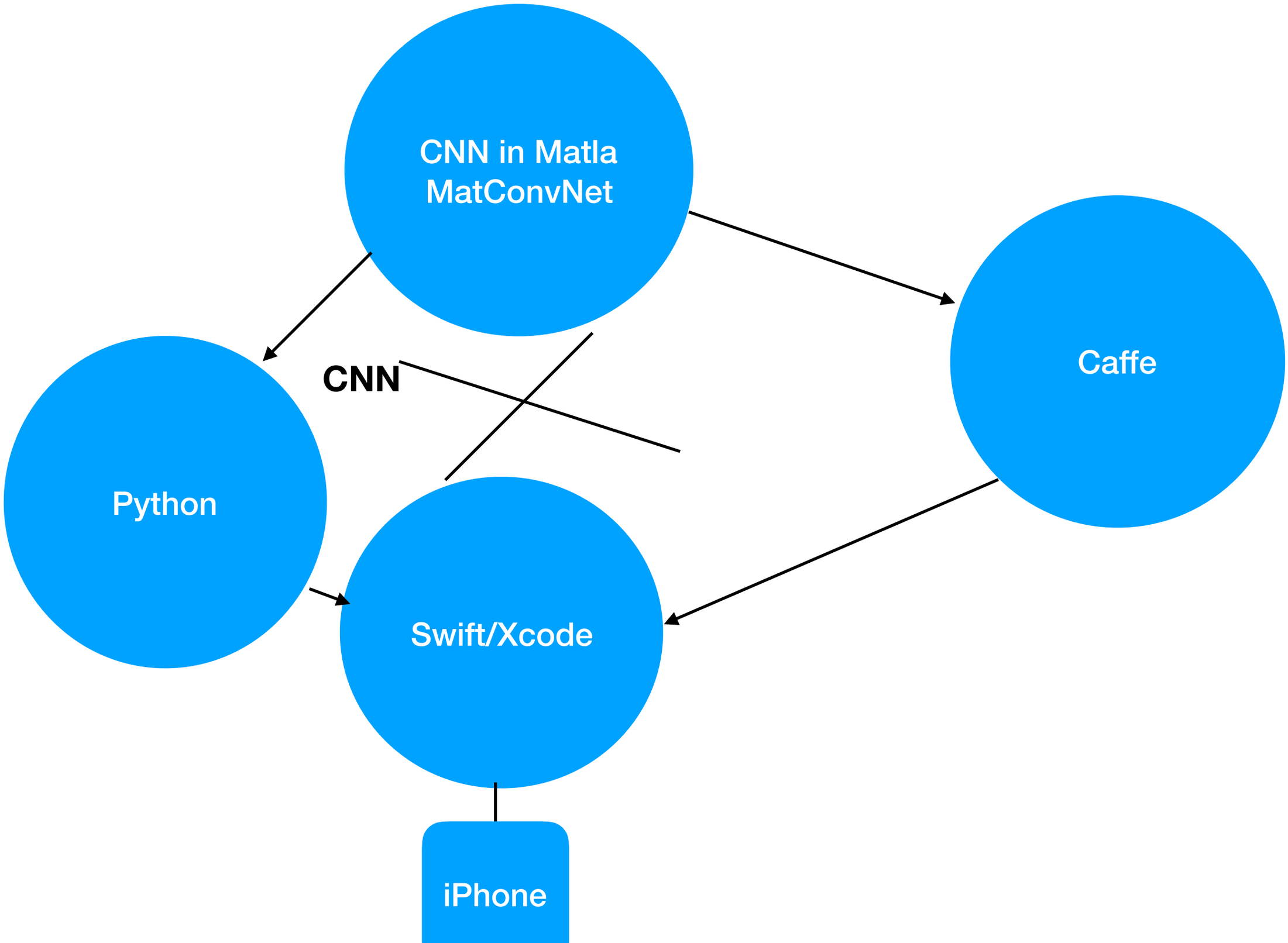
- Prerequisites
- Compilation
- Hardware

When updating Caffe, it's best to `make clean` before re-compiling.

Prerequisites

Caffe has several dependencies:

- CUDA is required for GPU mode.


CNN in Matla
MatConvNet
CNN
Python
Caffe
Swift/Xcode
iPhone

MatConvNet: CNNs for MATLAB

Download Code & issues Pre-trained models Discussion forum

MatConvNet is a MATLAB toolbox implementing *Convolutional Neural Networks* (CNNs) for computer vision applications. It is simple, efficient, and can run and learn state-of-the-art CNNs. Many pre-trained CNNs for image classification, segmentation, face recognition, and text detection are available.

New: 1.0-beta25 released with a new modular system `vl_contrib` for third-party contributions. A partial rewrite of the C++ code and support for recent CuDNN versions is also included.

New: 1.0-beta24 released with bugfixes, new examples, and utility functions.

New: 1.0-beta23 released with `vl_nnroipool` and a Fast-RCNN demo.

New: 1.0-beta22 released with a few bugfixes.

Obtaining MatConvNet

- Tarball for version 1.0-beta25; older versions
- GIT repository
- Citation

Documentation

- Manual (PDF)
- MATLAB functions
- FAQ
- Discussion group

Extensions

MIT CSAIL

6.819/6.869: Advances in Computer Vision

MIT COMPUTER VISION

Fall 2015

Final Project

Final Project is an opportunity for you to apply what you have learned in class to a problem of your interest in computer vision. We strongly recommand a team of 2-4 people (except for the survey, which should be individual-based). There are three project options you can pick from:

Report
Due: December 10, 2015

The report should be 4 - 6 pages (the upper limit of 6 pages is strict!) in CVPR format. It should be structured like a research paper, with sections for Introduction, related work, the approach/algorithm, experimental results, conclusions and references.

You should describe and evaluate what you did in your project, which may not necessarily be what you hoped to do originally. A small result described and evaluated well will earn more credit than an ambitious result where no aspect was done well. Be accurate in describing the problem you tried to solve. Explain in detail your approach, and specify any simplifications or assumptions you have taken. Also demonstrate the limitations of your approach. When doesn't it work? Why? What steps would you have taken have you continued working on it? Make sure to add references to all related work you reviewed or used.

You are allowed to submit any supplementary material that you think it important to evaluate your work, however we do not guarantee that we will review all of that material, and you should not assume that. The report should be self-contained.

Submission: submit your report to stellar as a pdf file named <YOUR_LAST_NAME>.pdf. Submit any supplementary material as a **single zip file** named <YOUR_LAST_NAME>.zip. Add a README file describing the supplemental content.

Option 1: Mini Places Challenge.

Submission:

Trace MatConvNet

按兩下來編輯

Trace MatConvNet

1

MatConvNet

6.6.4 Softmax

Care must be taken in evaluating the exponential in order to avoid underflow or overflow. The simplest way to do so is to divide the numerator and denominator by the exponential of the maximum value:

$$y_{ijk} = \frac{e^{x_{ijk} - \max_d x_{ijd}}}{\sum_{t=1}^{D} e^{x_{ijt} - \max_d x_{ijd}}}.$$

The derivative is given by:

$$\frac{dz}{dx_{ijd}} = \sum_k \frac{dz}{dy_{ijk}} \left(e^{x_{ijd}} L(\mathbf{x})^{-1} \delta_{(k=d)} - e^{x_{ijd}} e^{x_{ijk}} L(\mathbf{x})^{-2} \right), \quad L(\mathbf{x}) = \sum_{t=1}^{D} e^{x_{ijt}}.$$

Simplifying:

$$\frac{dz}{dx_{ijd}} = y_{ijd} \left(\frac{dz}{dy_{ijd}} - \sum_{k=1}^{K} \frac{dz}{dy_{ijk}} y_{ijk} \right).$$

In matrix form:

$$\frac{dz}{dX} = Y \odot \left(\frac{dz}{dY} - \left(\frac{dz}{dY} \odot Y \right) \mathbf{1}\mathbf{1}^\top \right)$$

2

Cnn_Mnist_Init.m

3

```
function net = cnn_mnist_init(varargin)
% CNN_MNIST_LENET Initialize a CNN similar for MNIST
opts.batchNormalization = true ;
opts.networkType = 'simplenn' ;
opts = vl_argparse(opts, varargin) ;
```

Layer configuration

4

```
% optionally switch to batch normalization
if opts.batchNormalization
  net = insertBnorm(net, 1) ;
  net = insertBnorm(net, 4) ;
  net = insertBnorm(net, 7) ;
end

% Meta parameters
net.meta.inputSize = [28 28 1] ;
net.meta.trainOpts.learningRate = 0.001 ;
net.meta.trainOpts.numEpochs = 20 ;
net.meta.trainOpts.batchSize = 100 ;
```

5

```
function tnet = vl_simplenn_tidy(net)
%VL_SIMPLENN_TIDY  Fix an incomplete or outdated SimpleNN network.
%   NET = VL_SIMPLENN_TIDY(NET) takes the NET object and upgrades
%   it to the current version of MatConvNet. This is necessary in
%   order to allow MatConvNet to evolve, while maintaining the NET
%   objects clean. This function ignores custom layers.
%
%   The function is also generally useful to fill in missing default
%   values in NET.
%
%   See also: VL_SIMPLENN().

% Copyright (C) 2014-15 Andrea Vedaldi.
% All rights reserved.
```

6

```
function [net, stats] = cnn_train(net, imdb, getBatch, varargin)
%CNN_TRAIN  An example implementation of SGD for training CNNs
%    CNN_TRAIN() is an example learner implementing stochastic
%    gradient descent with momentum to train a CNN. It can be used
%    with different datasets and tasks by providing a suitable
%    getBatch function.
%
%    The function automatically restarts after each training epoch by
%    checkpointing.
%
%    The function supports training on CPU or on one or more GPUs
%    (specify the list of GPU IDs in the `gpus` option).
```

7

```
% Train for one epoch.
params = opts ;
params.epoch = epoch ;
params.learningRate = opts.learningRate(min(epoch, numel(opts.learningRate)))
params.train = opts.train(randperm(numel(opts.train))) ; % shuffle
params.train = params.train(1:min(opts.epochSize, numel(opts.train)));
params.val = opts.val(randperm(numel(opts.val))) ;
params.imdb = imdb ;
params.getBatch = getBatch ;

if numel(params.gpus) <= 1
  [net, state] = processEpoch(net, state, params, 'train') ;
  [net, state] = processEpoch(net, state, params, 'val') ;
  if ~evaluateMode
    saveState(modelPath(epoch), net, state) ;
  end
  lastStats = state.stats ;
```

8

```
function res = vl_simplenn(net, x, dzdy, res, varargin)
%VL_SIMPLENN  Evaluate a SimpleNN network.
%   RES = VL_SIMPLENN(NET, X) evaluates the convnet NET on data X.
%   RES = VL_SIMPLENN(NET, X, DZDY) evaluates the convnent NET and its
%   derivative on data X and output derivative DZDY (foward+bacwkard pass).
%   RES = VL_SIMPLENN(NET, X, [], RES) evaluates the NET on X reusing the
%   structure RES.
%   RES = VL_SIMPLENN(NET, X, DZDY, RES) evaluates the NET on X and its
%   derivatives reusing the structure RES.
%
%   This function process networks using the SimpleNN wrapper
%   format. Such networks are 'simple' in the sense that they consist
%   of a linear sequence of computational layers. You can use the
%   `dagnn.DagNN` wrapper for more complex topologies, or write your
%   own wrapper around MatConvNet computational blocks for even
%   greater flexibility.
```

9

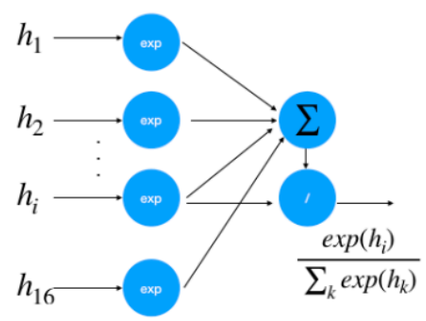
- vl_imreadjpeg.mexmaci64
- vl_imreadjpeg_old.mexmaci64
- vl_nnbilinearsampler.mexmaci64
- vl_nnbnorm.mexmaci64
- vl_nnconv.mexmaci64
- vl_nnconvt.mexmaci64
- vl_nnnormalize.mexmaci64
- vl_nnnormalizelp.mexmaci64
- vl_nnpool.mexmaci64
- vl_nnroipool.mexmaci64
- vl_taccummex.mexmaci64
- vl_tmove.mexmaci64

10

```
%   Y = VL_NNSOFTMAX(X, C) applies the softmax operator followed by
%   the logistic loss the data X. X has dimension H x W x D x N,
%   packing N arrays of W x H D-dimensional vectors.
%
%   C contains the class labels, which should be integers in the range
%   1 to D. C can be an array with either N elements or with dimensions
%   H x W x 1 x N dimensions. In the fist case, a given class label is
%   applied at all spatial locations; in the second case, different
%   class labels can be specified for different locations.

    E = exp(bsxfun(@minus, X, max(X,[],3))) ;
    L = sum(E,3) ;
    Y = bsxfun(@rdivide, E, L) ;

    if nargin <= 1, return ; end

    % backward
    Y = Y .* bsxfun(@minus, dzdY, sum(dzdY .* Y, 3)) ;
```

11

softmax



12

A matrix for Latin Square Encoding

```
function B=LatinSquare(n)
% Latin square encoding

a = 1:1:n^2;
A = reshape(a,n,n);
B = zeros(n^2,n^2);
for i = 1:n^2
   r = mod(i,n);
   if r == 0
      r = n;
   end
   c = ceil(i/n);
   % fprintf('%d %d\n', r,c)
   ls_c = A(:,c)';
   ind_c = find(ls_c ~= i);
   ls = [A(r,:) ls_c(ind_c)];
   B(i,ls) = 1;
end
```

13

Column major

```
a =
    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16

>> reshape(B(11,:),4,4)

ans =
     0     0     1     0
     0     0     1     0
     1     1     1     1
     0     0     1     0

i=11

A =
     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16
```

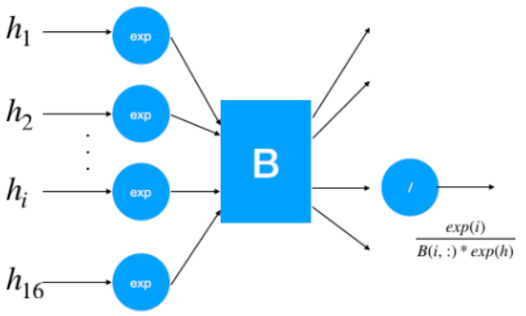
$$\frac{exp(i)}{B(i,:) * exp(h)}$$

```
>> B(11,:)
ans =
     0     0     1     0     0     0     1     0     1     1     1     1     0     0     1     0
```

14

```
B =
     1     1     1     1     1     0     0     0     1     0     0     0     1     0     0     0
     1     1     1     1     0     1     0     0     0     1     0     0     0     1     0     0
     1     1     1     1     0     0     1     0     0     0     1     0     0     0     1     0
     1     0     0     0     1     1     1     1     1     0     1     0     0     0     1     0
     0     1     0     0     1     1     1     1     0     1     0     0     0     1     0     0
     0     0     1     0     1     1     1     1     0     0     1     0     0     0     1     0
     0     0     0     1     1     1     1     0     1     0     0     1     0     0     0     1
     1     0     0     0     1     1     1     0     1     0     0     1     0     0     0     1
     1     0     0     0     0     1     1     1     0     1     1     1     1     0     0     0
     0     1     0     0     0     1     1     1     0     1     1     1     1     0     0     0
     0     0     1     0     0     0     1     1     0     1     1     1     1     0     0     0
     0     0     0     1     0     0     1     1     0     1     1     1     1     0     0     0
     1     0     0     0     1     0     0     1     0     1     1     0     0     1     1     1
     0     1     0     0     0     1     0     1     0     1     0     1     0     1     1     1
     0     0     1     0     0     0     1     0     0     0     1     0     0     1     1     1
     0     0     0     1     0     0     0     1     0     0     0     1     1     1     1     1
```

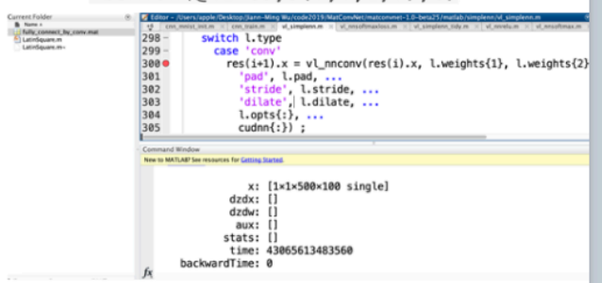
15



16

Multiplication of 4 D array

`bsxfun(@times,S(:,:,1,:),C)`



Use the vl_nnconv function to implement Multiplication of B to exp(h)

17

Maxsoftloss

CX=squeeze(c(1,1,:,:))

```
K>> CX(1:10)

ans =
     9
     5
     1
     0
     3
     9
     4
     5
     4
     6
```

```
K>> c_(1:10)

ans =
    10    16    22    31    44    60    65    76    85    97
```

18

```
K>> squeeze(c(:,:,1,1:10))'

ans =
    10     6     2     1     4    10     5     6     5     7
```

```
K>> squeeze(mass(:,:,1,1:10))'

ans =
  1×10 single row vector
     1     1     1     1     1     1     1     1     1     1
```

19

```
K>> ind = find(mass == 0)

ind =
  0×1 empty double column vector

K>> size(mass)

ans =
     1     1     1   100
```

20

Softmax log-loss or multinomial logistic loss. This loss combines the softmax block and the log-loss block into a single block:

$$\ell(\mathbf{x}, c) = -\log \frac{e^{x_c}}{\sum_{k=1}^{C} e^{x_k}} = -x_c + \log \sum_{k=1}^{C} e^{x_k}. \quad (4.7)$$

Combining the two blocks explicitly is required for numerical stability. Note that, by combining the log-loss with softmax, this loss automatically makes the score compete: $\ell(bx, c) \approx 0$ when $x_c \gg \sum_{k \neq c} x_k$.

This loss is implemented also in the *deprecated* function `vl_softmaxloss`.

```
% compute softmaxloss
xmax = max(x,[],3) ;
ex = exp(bsxfun(@minus, x, xmax)) ;

%n = sz(1)*sz(2) ;
if nargin <= 2
  t = xmax + log(sum(ex,3)) - reshape(x(c_), [sz(1:2) 1 sz(4)]) ;
  y = sum(sum(sum(mass .* t,1),2),4) ;
else
  y = bsxfun(@rdivide, ex, sum(ex,3)) ;
  y(c_) = y(c_) - 1;
  y = bsxfun(@times, y, bsxfun(@times, mass, dzdy)) ;
end
```

$$\ell(\mathbf{x}, c) = -\log \frac{e^{x_c}}{\sum_{k=1}^{C} e^{x_k}} = -x_c + \log \sum_{k=1}^{C} e^{x_k}. \quad (4.7)$$

x_c **denotes receptive fields of units with largest activations**

```
if nargin <= 2
  t = xmax + log(sum(ex,3)) - reshape(x(c_), [sz(1:2) 1 sz(4)]) ;
  y = sum(sum(sum(mass .* t,1),2),4) ;
else
  y: single =
  y     1.5623     - 1;
  y = bsxfun(@times, y, bsxfun(@times, mass, dzdy)) ;
end
```

A scalar

Back propagation

```
case 'loss'
  res(i).dzdx = vl_nnloss(res(i).x, l.class, res(i+1).dzdx) ;

case 'softmaxloss'
  res(i).dzdx = vl_nnsoftmaxloss(res(i).x, l.class, res(i+1).dzdx)
case 'r
  if l.                                  = {'leak', l.leak} ; else leak = {} ; end
  if ~isempty(res(i).x)
    res(i).dzdx = vl_nnrelu(res(i).x, res(i+1).dzdx, leak{:}) ;
```

i: 1x1 double = 11

```
>> size(res(i).x)

ns =
     1     1    10   100

>> res(i+1).dzdx

ns =
     1
```

```
% compute softmaxloss
xmax = max(x,[],3) ;
ex = exp(bsxfun(@minus, x, xmax)) ;

%n = sz(1)*sz(2) ;
if nargin <= 2
  t = xmax + log(sum(ex,3)) - reshape(x(c_), [sz(1:2) 1 sz(4)]) ;
  y = sum(sum(sum(mass .* t,1),2),4) ;
else
  y = bsxfun(@rdivide, ex, sum(ex,3)) ;
  y(c_) = y(c_) - 1;
  y = bsxfun(@times, y, bsxfun(@times, mass, dzdy)) ;
end
```

```
K>> size(y)

ans =

     1     1    10   100
```

$$\frac{dl(\mathbf{x},c)}{dx_c} = -1 + \frac{e^{x_c}}{\sum_k e^{x_k}}$$

$$\frac{dl(\mathbf{x},c)}{dx_h} = \frac{e^{x}_{h}}{\sum_k e^{x_k}}, \quad h \neq c$$

$$\ell(\mathbf{x},c) = -\log\frac{e^{x_c}}{\sum_{k=1}^{C} e^{x_k}} = -x_c + \log\sum_{k=1}^{C} e^{x_k}. \tag{4.7}$$

25

```
  else
    y = bsxfun(@rdivide, ex, sum(ex,3)) ;
    y(c_) = y(c_) - 1;
    y = bsxfun(@times, y, bsxfun(@times, mass, dzdy)) ;
  end
```

y: 1×1×10×100 single array

```
K>> squeeze(b(1:10))'

ans =

  1×10 single row vector

     1     1     1     1     1     1     1     1     1     1
```

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

The result y denotes derivative of loss with respect to external fields of softmax

26

Latin square encoding

27

Softmax changes to SoftLatinSquareMax

A unit in location (i,j) of a 2D latin square has inhibitory units that locate at row i and column j

How to evaluate outputs of softLSmax?
How to define softLSMaxLoss ?
How to calculate derivatives?

28

Column major

```
a =

     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16

>> reshape(B(11,:),4,4)

ans =

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

i=11

```
A =

     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16
```

$$\frac{exp(h_i)}{B(i,:) * exp(h)}$$

```
>> B(11,:)

ans =

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

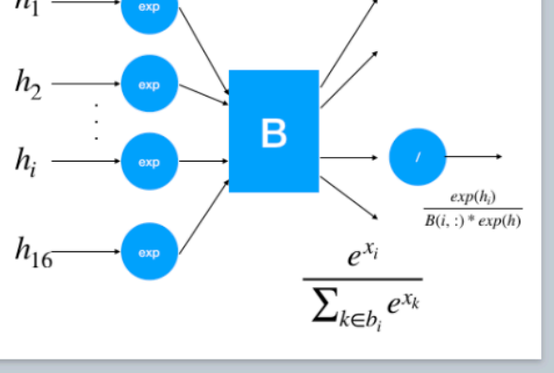
$b_i = \{3,7,9,10,11,12,15\}$

$$\frac{e^{x_i}}{\sum_{k\in b_i} e^{x_k}}$$

29

```
B =

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

30



31

SoftLSMaxLoss

```
>> LS = [ 0 0 1 0; 0 1 0 0; 1 0 0 0; 0 0 0 1]

LS =

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

$c_1 = 3, c_2 = 6, c_3 = 9, c_4 = 16$

Let A collect active bits

$A = \{3,6,9,16\}$

$$l(\mathbf{x},A,B) = -\sum_{i\in A} log\frac{e^{x_i}}{\sum_{k\in b_i} e^{x_k}}$$

b_i collects positions with active bits at the ith row of matrix B

$b_3 = \{1,2,3,4,7,11,15\}$

32

```
% compute a_
A = latinsquare_list(squeeze(c)+1,:);
base = 0:16:1600-16;
A=A'+repmat(base',1,4)';
a_ = A(:);
```

33

$$\ell(\mathbf{x},c) = -\log\frac{e^{x_c}}{\sum_{k=1}^{C} e^{x_k}} = -x_c + \log\sum_{k=1}^{C} e^{x_k}. \tag{4.7}$$

```
y = bsxfun(@rdivide, ex, sum(ex,3)) ;
y(c_) = y(c_) - 1;
y = bsxfun(@times, y, bsxfun(@times, mass, dzdy)) ;
```

$$l(\mathbf{x},A,B) = -\sum_{i\in A} log\frac{e^{x_i}}{\sum_{k\in b_i} e^{x_k}}$$

exB=B'*squeeze(ex);

```
term2 = log(exB(a_))';  % my c
```

$$l(\mathbf{x},A,B) = -\sum_{i\in A} x_i + \sum_{i\in A} log\sum_{k\in b_i} e^{x_k}$$

```
LS =

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

$$\frac{\partial l(\mathbf{x},A,B)}{\partial x_i} = -1 + \frac{e^{x_i}}{\sum_{k\in b_i} e^{x_k}}, \quad i \in A$$

$$= \frac{2e^{x_i}}{\sum_{k\in b_i} e^{x_k}}, \quad i \notin A$$

$A = \{3,6,9,16\}$ $\quad i = 12 \in b_9 \cap b_{16}$

34

Revise codes for LS encoding

Let B=LatinSquare(n) denote an input matrix of SoftLSMaxLoss

35

```
% compute softmaxloss
xmax = max(x,[],3) ;
ex = exp(bsxfun(@minus, x, xmax)) ;

%n = sz(1)*sz(2) ;
if nargin <= 2
  t = xmax + log(sum(ex,3)) - reshape(x(c_), [sz(1:2) 1 sz(4)]) ;
  y = sum(sum(sum(mass .* t,1),2),4) ;
```

1x1x1x100 | 1x1x1x100 | x(c_): one hundred winners

Let a_ collect 400 hundreds of elements, ranging from 1 to 1000

$$l(\mathbf{x},A,B) = -\sum_{i\in A} log\frac{e^{x_i}}{\sum_{k\in b_i} e^{x_k}}$$

exB = B'*squeeze(ex)
term2 = log(exB(a_))
term3 = x(a_)

$$l(\mathbf{x},A,B) = -\sum_{i\in A} x_i + \sum_{i\in A} log\sum_{k\in b_i} e^{x_k}$$

36


Sketch a digit from A to Z
Network input
Network output
M
M
JCSA


Handwriting 99 Multiplication
Sketch a digit from 1 to 9
8
9 x 8 = 72
JCSA

iphone

mlmodel

LetterCore 〉 iPhone

LetterCore | Build LetterCore: **Succeeded** | Today at 7:42 AM

LetterCore 〉 LetterCore 〉 EMNIST_letter.mlmodel

- LetterCore M
  - LetterCore
    - AppDelegate.swift M
    - ViewController.swift A
    - MainView.swift A
    - ShadowView.swift A
    - UIView+Constraints.swift A
    - Float+String.swift A
    - Colors.swift ?
    - EMNIST_letter.mlmodel ?
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist M
  - Products

Filter

Machine Learning Model

Name	EMNIST_letter
Type	Neural Network
Size	9 MB
Author	C.Tien
Description	This CNN model without BatchNorm is for testing of English handwritten-alphabet recognition.
License	unknown

Model Class

EMNIST_letter

Automatically generated Swift model class

Prediction

Name	Type	Description
Inputs		
data	MultiArray (Double 1 x 784 x 1)	Input image multiarray
Outputs		
prob	MultiArray (Double)	output A-Z element

Auto

Filter

All Output

Filter

Identity and Type

Name: EMNIST_letter.mlmodel

Type: Default - CoreML Model

Location: Relative to Group

EMNIST_letter.mlmodel

Full Path: /Users/apple/Desktop/swift_2019/LetterCore36__demo/LetterCore/EMNIST_letter.mlmodel

On Demand Resource Tags

Only resources are taggable

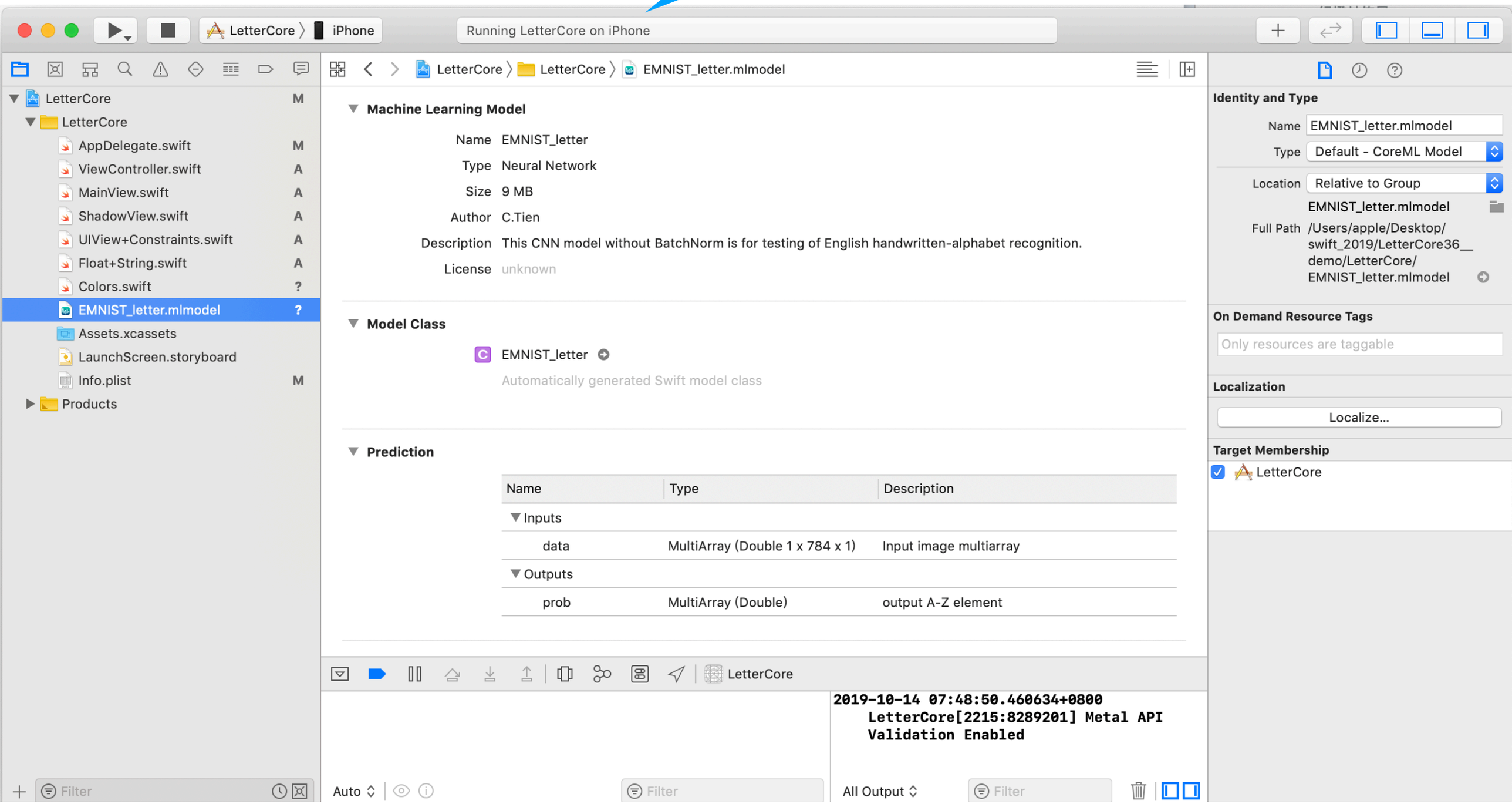
Localization

Localize...

Target Membership

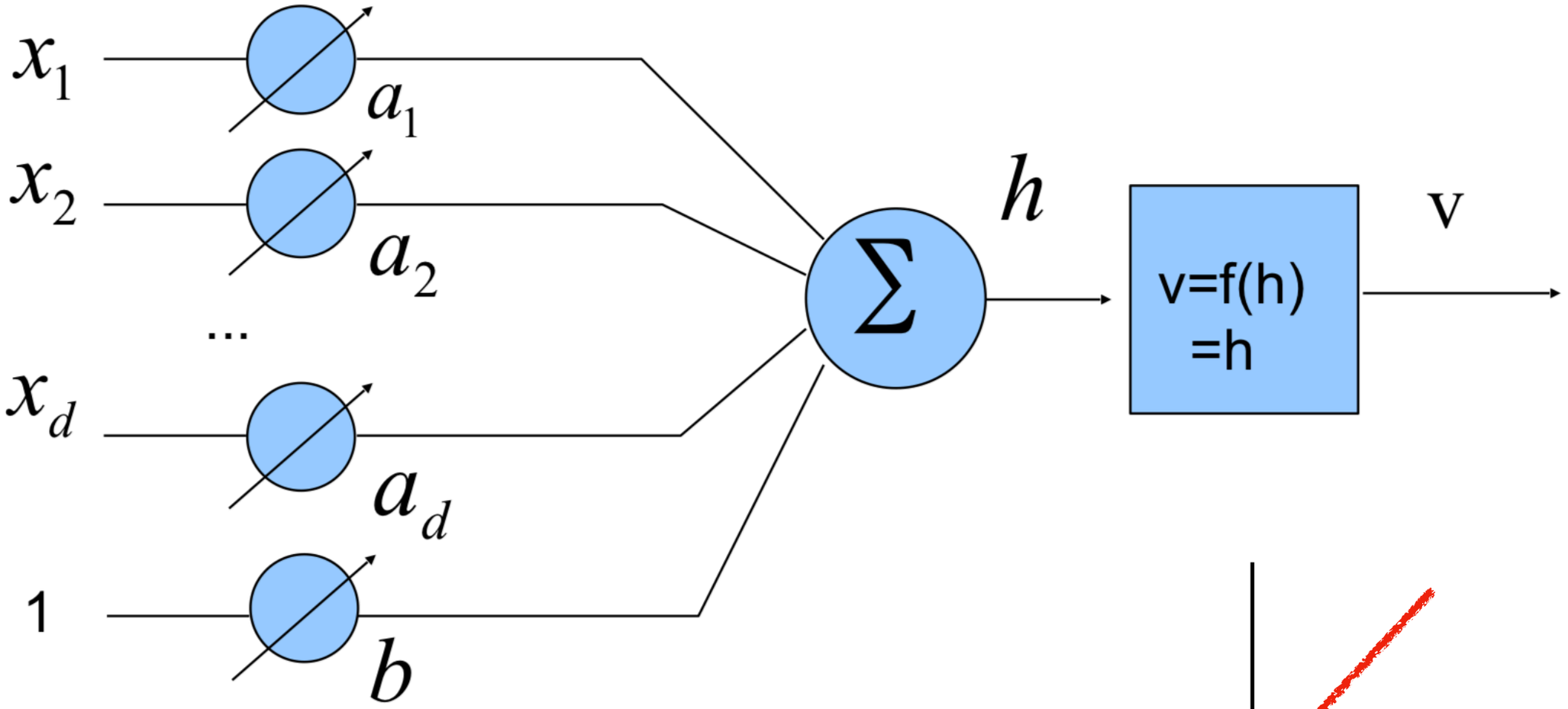
LetterCore

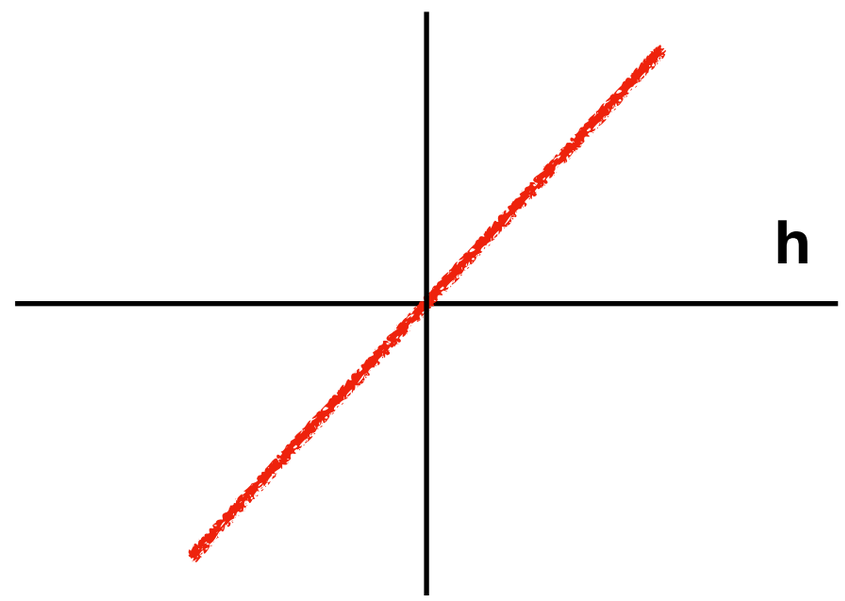
Running on iphone

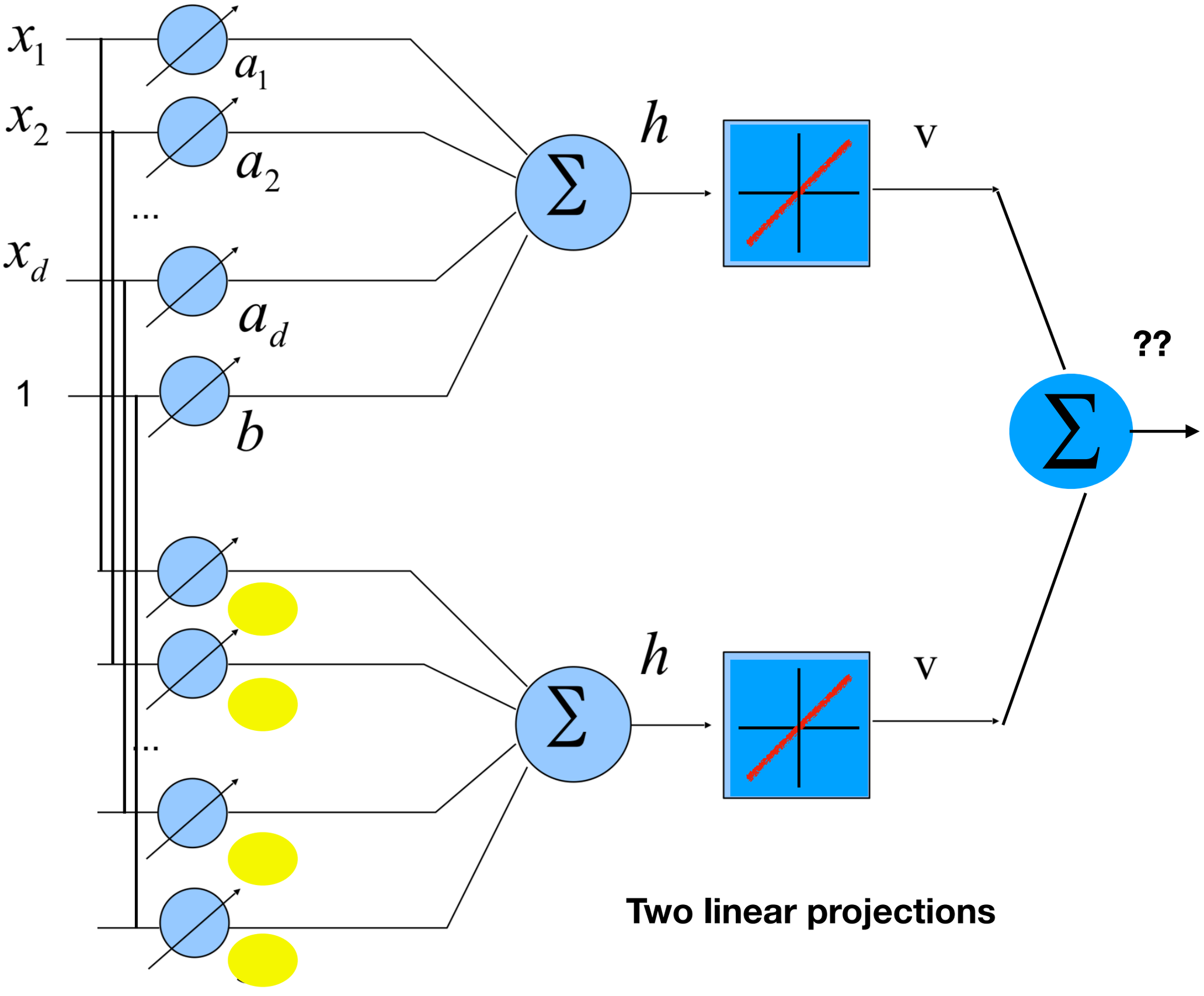


Mathematical Modeling of Neurons

$$v = h = a_1 x_1 + a_2 x_2 + \ldots + a_d x_d + b$$

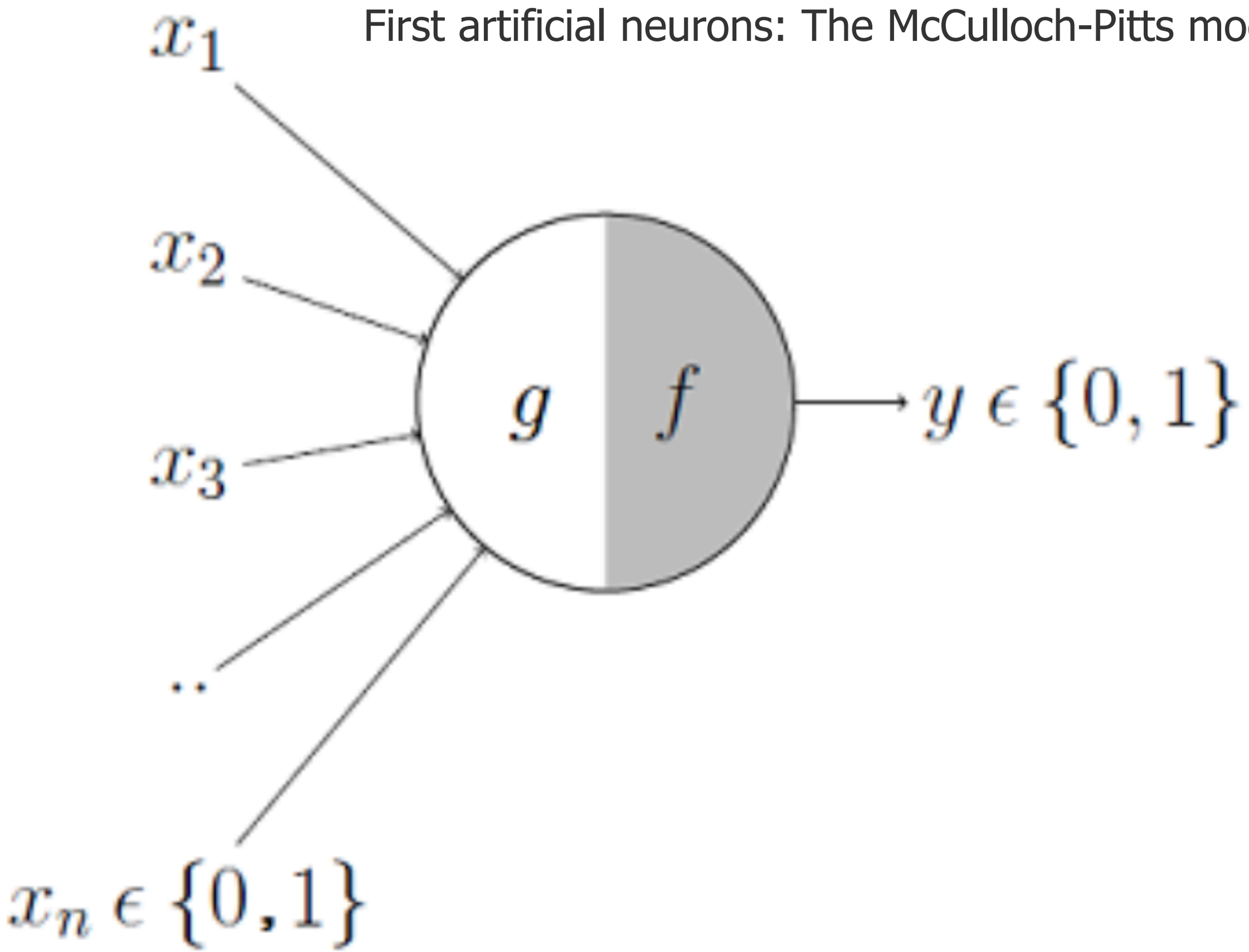





x_1
x_2
x_d
1
a_1
a_2
a_d
b
...
Σ
h
v
??
Σ
h
v
Two linear projections

Two-state binary neurons

First artificial neurons: The McCulloch-Pitts model

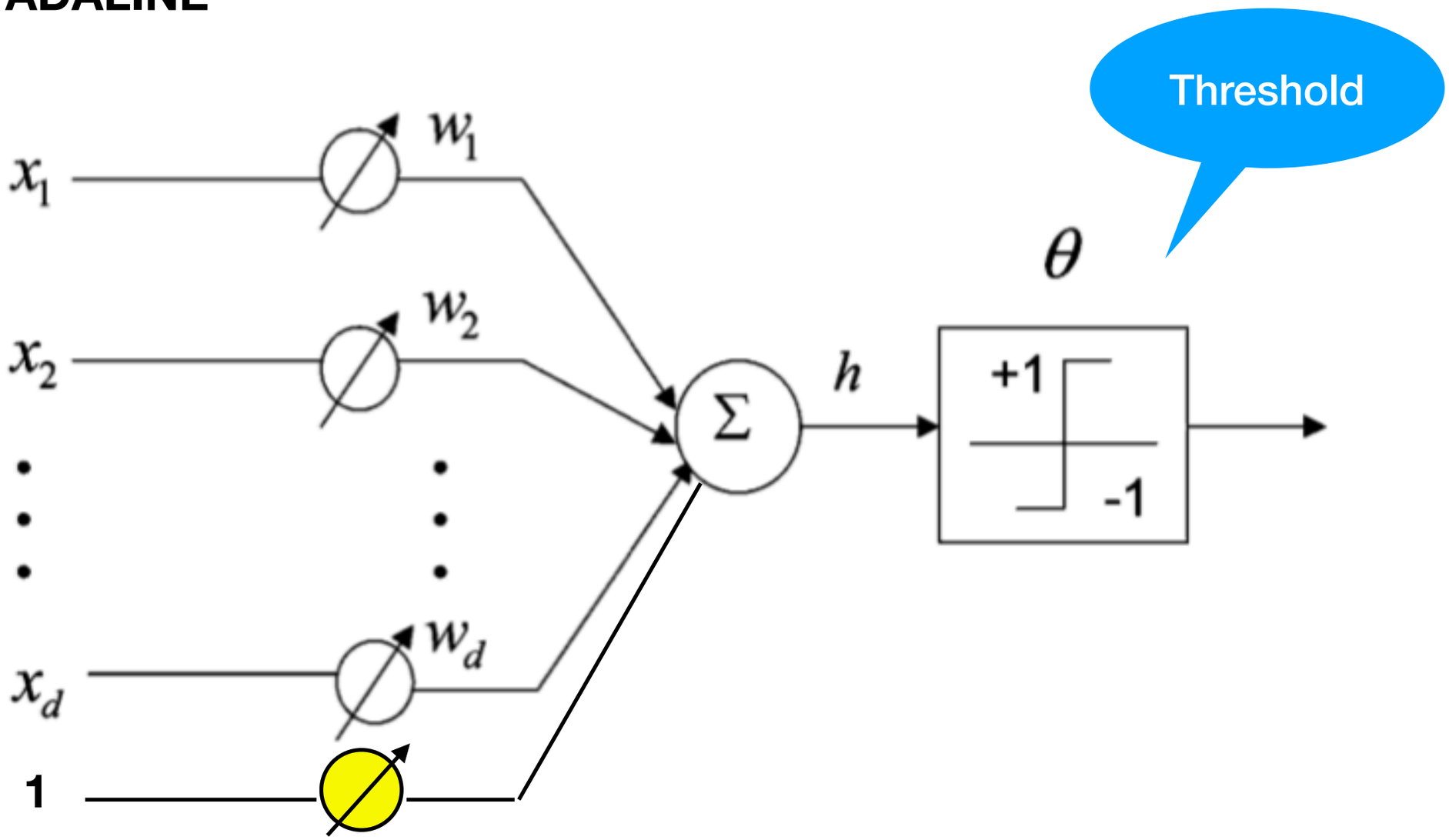


This is where it all began..

$$g(x_1, x_2, x_3, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

Widrow's ADALINE



Adaptive linear element (Adaline).

Nonlinear activation function

Function Approximation Using Generalized Adalines

Jiann-Ming Wu, Zheng-Han Lin, and Pei-Hsun Hsu

***Abstract*—This paper proposes neural organization of generalized adalines (gadalines) for data driven function approximation. By generalizing the threshold function of adalines, we achieve the K-state transfer function of gadalines which responds a unitary vector of K binary values to the projection of a predictor on a receptive field. A generative component that uses the K-state activation of a gadaline to trigger K posterior independent normal variables is employed to emulate stochastic predictor-oriented target generation. The fitness of a generative component to a set of paired data mathematically translates to a mixed integer and linear programming. Since consisting of continuous and discrete variables, the mathematical framework is resolved by a hybrid of the mean field annealing and gradient descent methods. Following the leave-one-out learning strategy, the obtained learning method is extended for optimizing multiple generative components. The learning result leads to parameters of a deterministic gadaline network for function approximation. Numerical simulations further test the proposed learning method with paired data oriented from a variety of target functions. The result shows that the proposed learning method outperforms the MLP and RBF learning methods for data driven function approximation.**

***Index Terms*—Adalines, generative models, mean field annealing, perceptron, postnonlinear projection, potts encoding, supervised learning.**

This paper explores data driven function approximation [5] based on generalized adalines. The novel neural organization is devised by generalizing the threshold function to K-state transfer function, which transforms its input to a K-state activation, represented by elements in $\Xi_K = \{\mathbf{e}_1^K, \ldots, \mathbf{e}_K^K\}$, where $\mathbf{e}_k^K$ is a unitary vector with the kth bit one and the others zero. A K-state transfer function uses K built-in knots to partition its domain to K nonoverlapping intervals so as to represent the exclusive membership of its input to K nonoverlapping intervals by a K-state activation. By replacing the threshold function of an adaline with a K-state transfer function, we have the generalized adaline (gadaline) for constructing novel neural networks.

Internal representations based on K-state activations impact on organizing and learning neural networks for data driven function approximation. Relevant issues are explored by addressing stochastic modeling of predictor-oriented target generation using gadalines. Following the idea, the K-state activation of a gadaline in response to a predictor is employed to trigger one of K independent normal variables or generators to produce an instance in approximating the desired target. The obtained generative component is organized to perform consecutive operations of projecting the predictor on a recep-

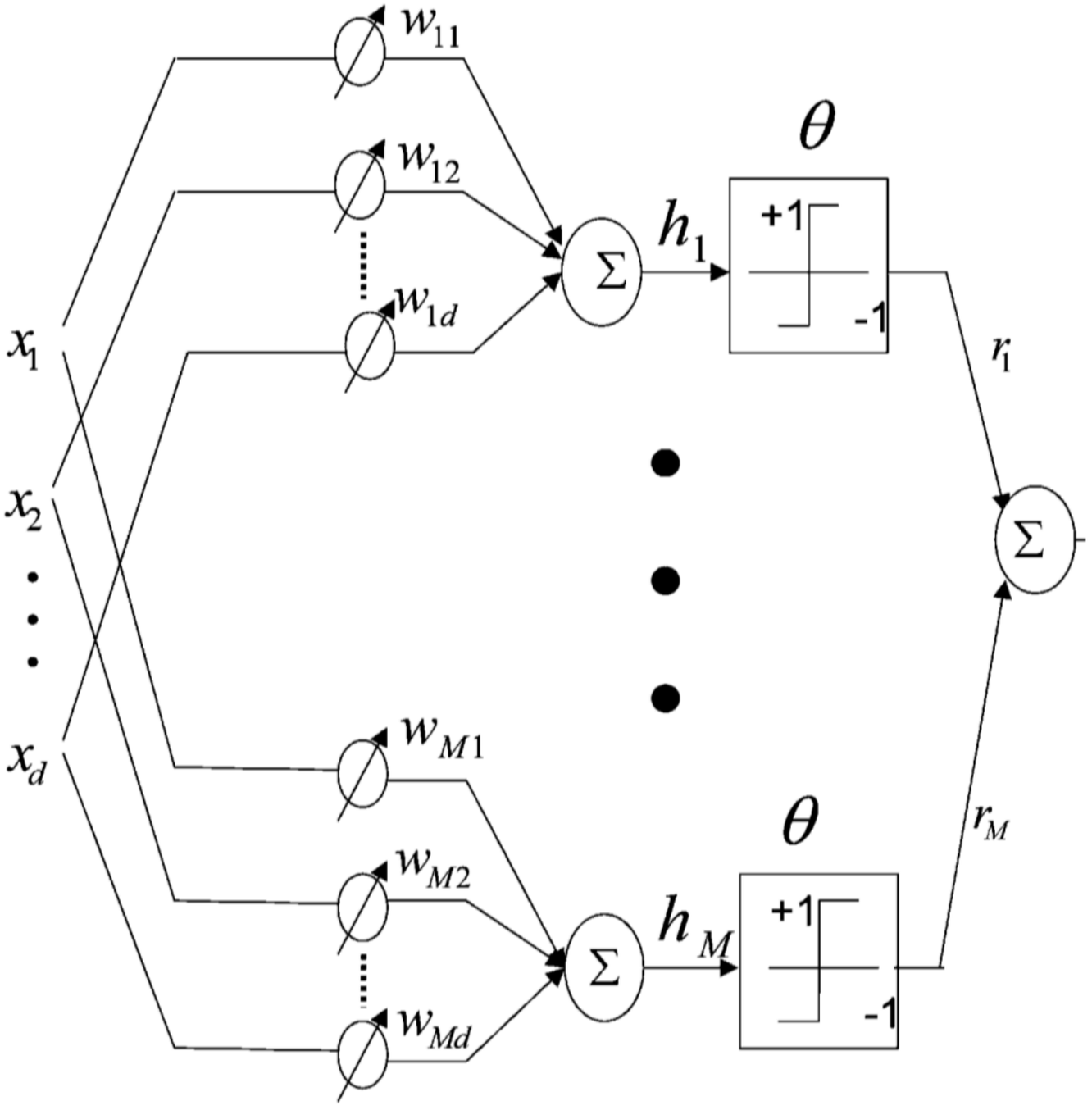
$$h[t] = \mathbf{w}^T \mathbf{x}[t] \tag{1}$$

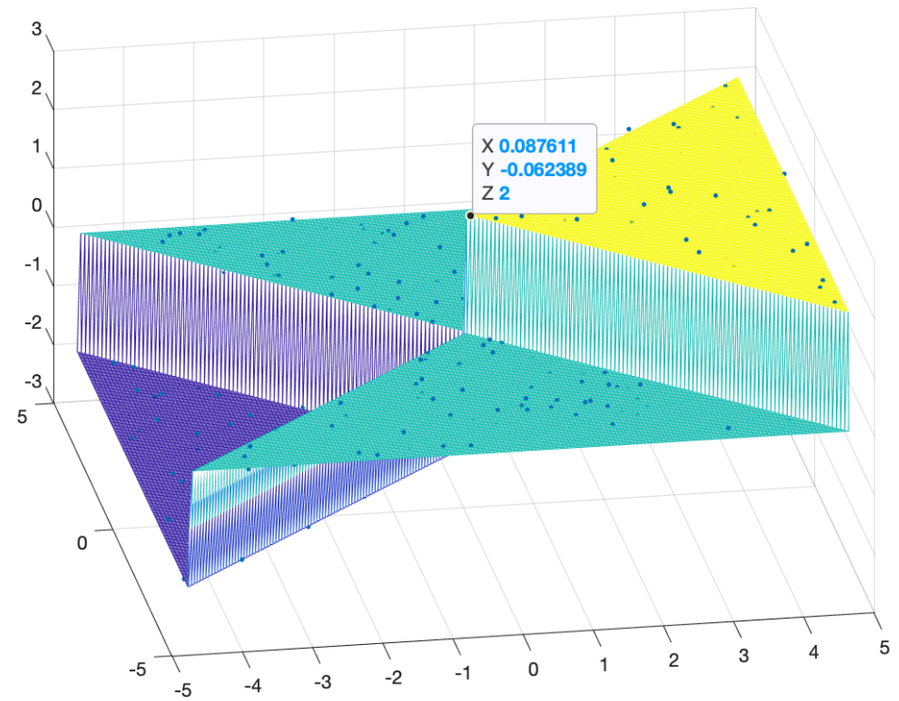
then employ the following threshold function to encode the external field:

$$\theta(h[t]) = \begin{cases} 1, & \text{if } h[t] \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{2}$$

To facilitate our presentations, we set $x_d[t]$ to one for all t so as to represent an arbitrary hyperplane in R^{d-1} by (1) in the following contexts.

Adaline networks





$$sign(x_1 + x_2) + sign(x1 - x2)$$

g. 3. Typical adaline network.

MATLAB R2019a - academic use

HOME | PLOTS | APPS | EDITOR | PUBLISH | VIEW

New Script | New Live Script | New | Open | Find Files | Compare — FILE

Import Data | Save Workspace | New Variable | Open Variable | Clear Workspace — VARIABLE

Favorites | Analyze Code | Run and Time | Clear Commands — CODE

Simulink — SIMULINK

Layout | Preferences | Set Path | Parallel | Add-Ons — ENVIRONMENT

Help — RESOU

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ code2019 ▸ code2006 ▸ Apps ▸ FunAppr ▸ FA2D ▸

Current Folder

Name

- fa2D_printingMGF_MLPot...
- fa2D_RBF.m
- fa3D_MLPotts.asv
- fa3D_MLPotts.m
- findabr.m
- gradient_m6.asv
- gradient_m6.m
- Gradient_MLPotts.m
- hs_err_pid7332.log
- image2data.m
- KState.m
- learn_MLP.m
- lm6.asv
- LM_MLPotts_fun.asv
- LM_MLPotts_fun.m
- MLP_Tool.asv
- MLP_Tool.ctf
- MLP_Tool.exe
- MLP_Tool.fig
- MLP_Tool.m
- mlp_tool_main.c
- mlp_tool_mcc_compone...
- mlpotts_main.c
- mlpotts_mcc_componen...
- MLPotts_tool.ctf
- MLPotts_tool.exe

MLP_Tool.m (Function)

M-file for MLP_Tool.fig

- MLP_Tool(varargin)
- MLP_Tool_OpeningFcn(h...
- MLP_Tool_OutputFcn(hOb...
- edit1_Callback(hObject, e...
- edit1_CreateFcn(hObject, ...

Editor - /Users/apple/Desktop/Jiann-Ming Wu/code2019/code2006/Apps/FunAppr/FA2D/MLP_Tool.m

+11 | call_learning_en.m | remoteparfor.m | plot_subface.m | demo_faceNet_p4_plot.m | MLP_Tool.m

MLP_Tool

MLP learning for
Function Appximation (JM Wu)

Target Function

sign(x1+x2)-sign(x1-x2)

Process

Sampling

Size

300

HP Fit

MLP Fit

M

30

A stochastic threshold function is defined to have a stochastic output in response to an external field h. Let s denote a discrete random variable corresponding to the output of a stochastic threshold function. Since s depends on the external field, the conditional pdf of s to h is assumed proportional to $\exp(\beta hs)$, such as

$$\Pr(s|h) \propto \exp(\beta hs) \qquad s \in \{-1,1\}$$

where β is a positive parameter for modulating randomness. Since the outcome of s is bipolar, by normalization, we have the following conditional pdf:

$$\Pr(s|h) = \frac{\exp(\beta hs)}{\exp(\beta h) + \exp(-\beta h)}.$$

When $h = h[t]$, the expectation of s is expressed as follows:

$$
\begin{aligned}
g(h[t]) &\equiv \langle s | h = h[t] \rangle \\
&= \Pr(s = 1 | h = h[t]) - \Pr(s = -1 | h = h[t]) \\
&= \frac{\exp(\beta h[t]) - \exp(-\beta h[t])}{\exp(\beta h[t]) + \exp(-\beta h[t])} \\
&= \tanh(\beta h[t]). \qquad (3)
\end{aligned}
$$

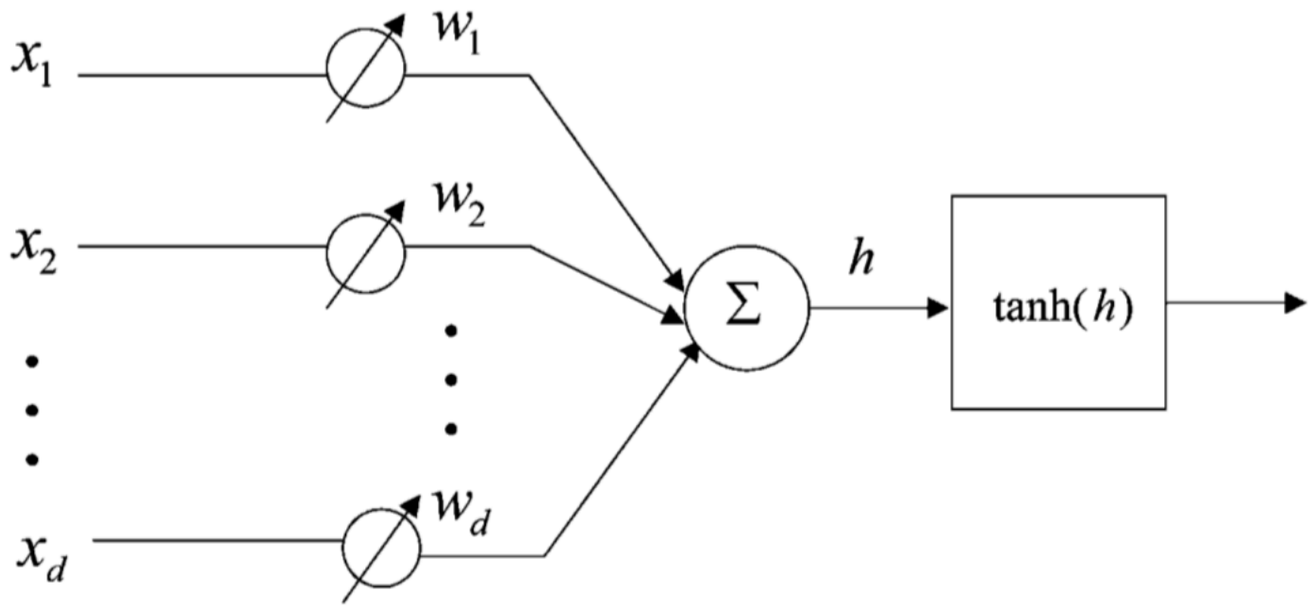


Fig. 2. Perceptron.

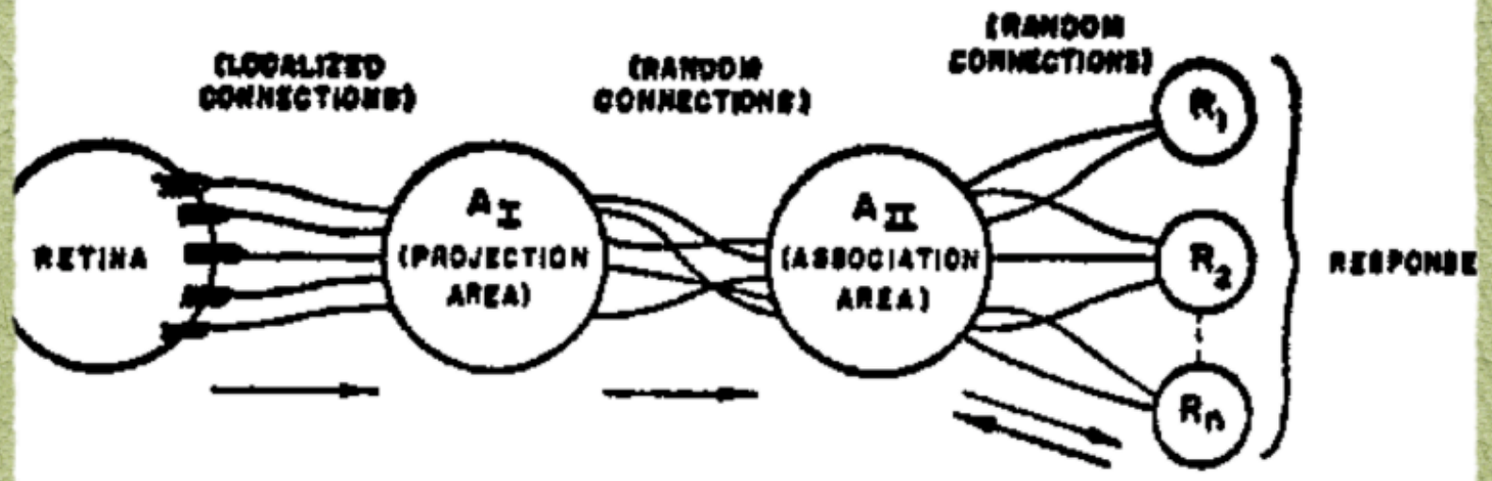


FIG. 1. Organization of a perceptron.

Rosenblatt's perceptrons

Simple and general models

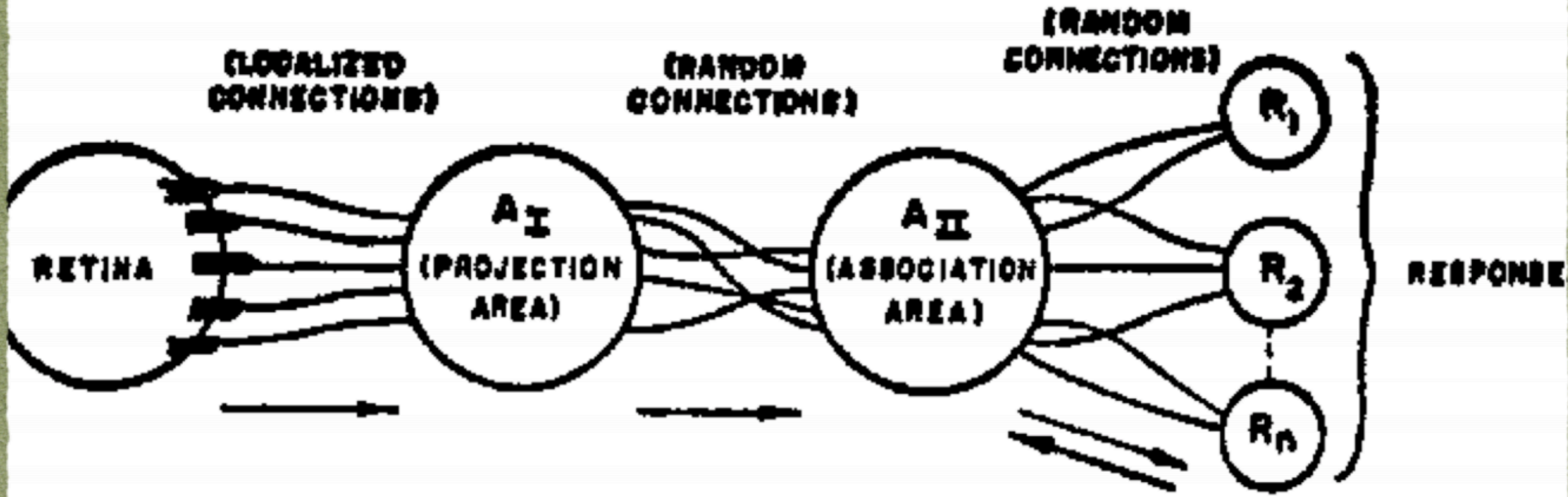


FIG. 1. Organization of a perceptron.

Gradients back-propagating

Letter | Published: 09 October 1986

Learning representations by back-propagating errors

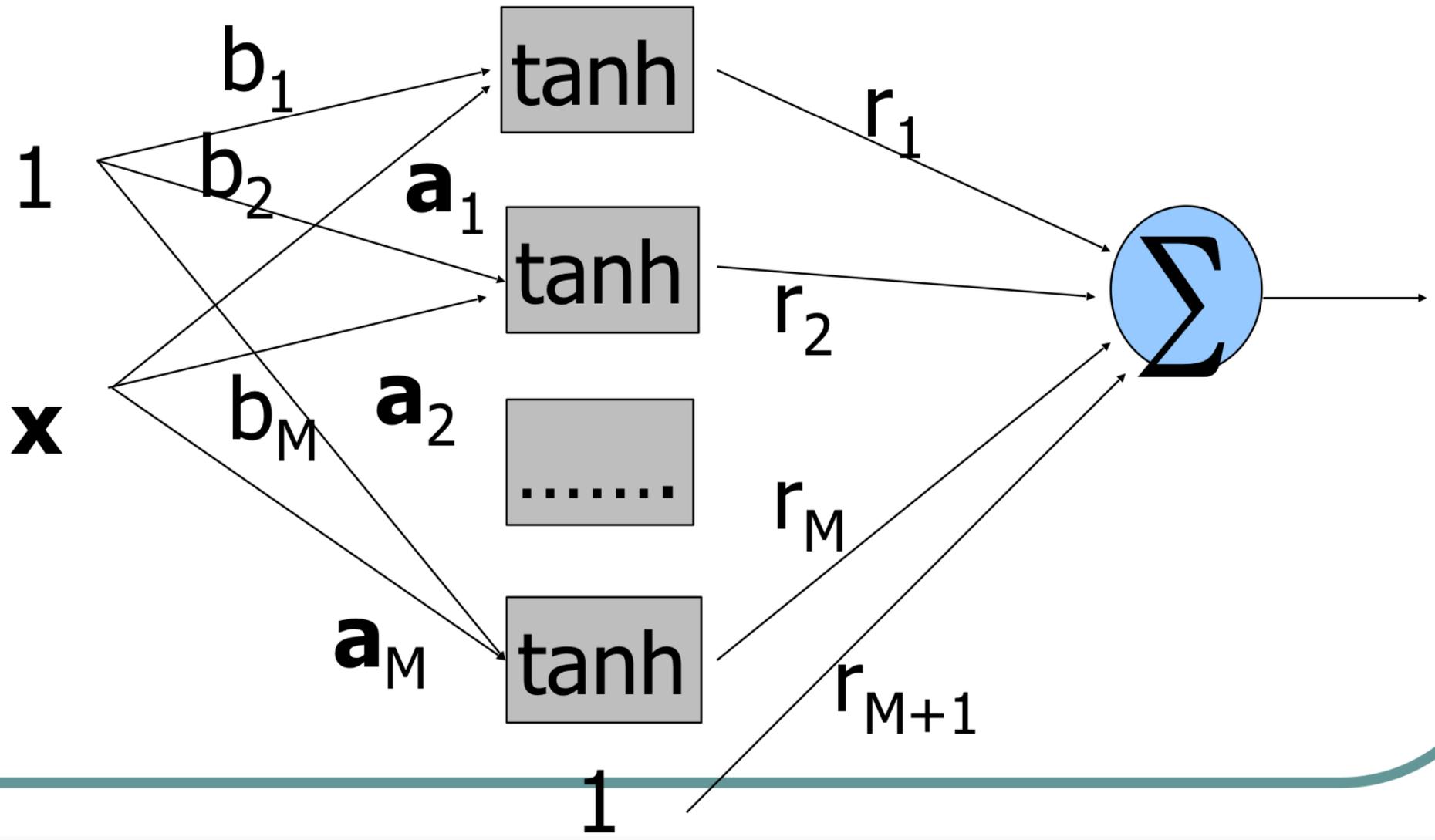
David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536 (1986) | Download Citation ⬇

Abstract

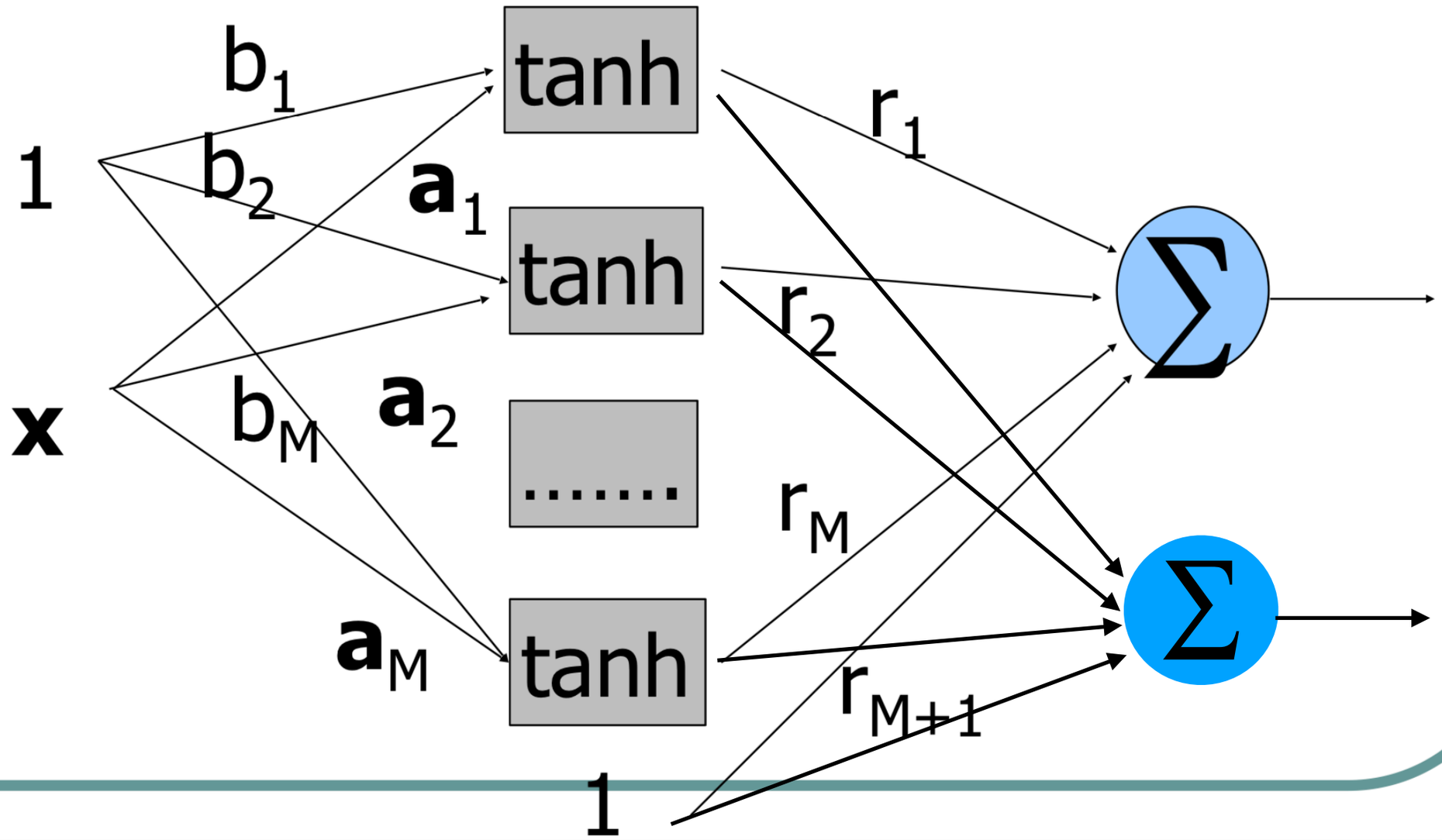
We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

Multilayer perceptrons (MLP) (Rumelhart, 1986)



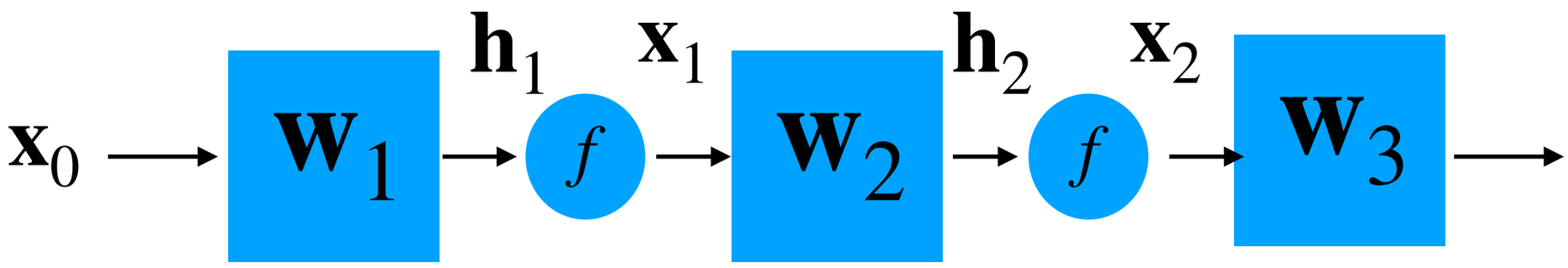
$$y = f(x) = \sum_i r_i tanh(a_i^T x + b_i) + r_0$$

Multiple Inputs Multiple Outputs



A Deep Neural Network with two hidden layers

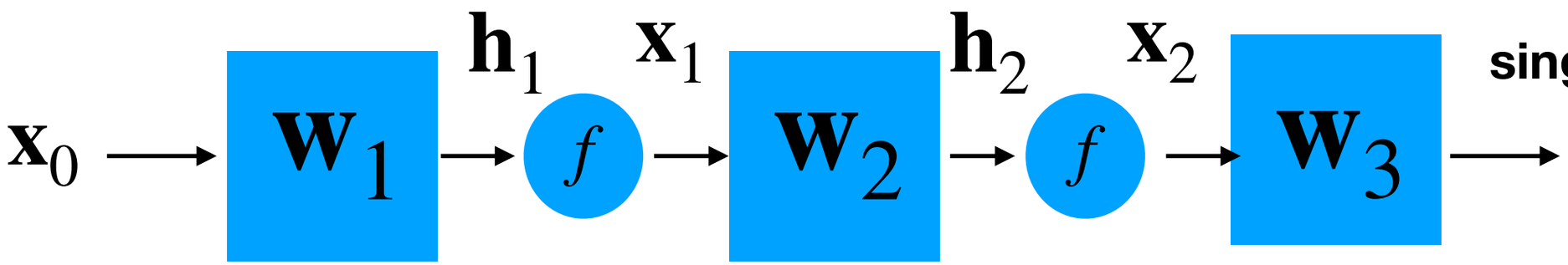
Feedforward propagating



$$\mathbf{h}_1 = \mathbf{w}_1\mathbf{x}_0 \qquad \mathbf{h}_2 = \mathbf{w}_2\mathbf{x}_1 \qquad y = h_3 = \mathbf{w}_3\mathbf{x}_2$$

$$\mathbf{x}_1 = f(\mathbf{h}_1) \qquad \mathbf{x}_2 = f(\mathbf{h}_2)$$

Gradients back-propagation



$$\mathbf{h}_1 = \mathbf{w}_1\mathbf{x}_0 \qquad \mathbf{h}_2 = \mathbf{w}_2\mathbf{x}_1 \qquad y = h_3 = \mathbf{w}_3^T\mathbf{x}_2$$

$$\mathbf{x}_1 = f(\mathbf{h}_1) \qquad \mathbf{x}_2 = f(\mathbf{h}_2)$$

$y = h_3$

$$\frac{d\mathbf{h}_1}{d\mathbf{x}_0} = ? \;(8) \qquad \frac{d\mathbf{h}_2}{d\mathbf{x}_1} = ? \;(4) \qquad \frac{dh_3}{d\mathbf{x}_2} = ?$$

$$\frac{d\mathbf{x}_1}{d\mathbf{h}_1} = ? \;(6) \qquad \frac{d\mathbf{x}_2}{d\mathbf{h}_2} = ? \;(2)$$

$$\frac{dy}{d\mathbf{x}_0} = ? \;(9) \qquad \frac{dy}{d\mathbf{h}_1} = ? \;(7) \qquad \frac{dy}{d\mathbf{x}_1} = ? \;(5) \qquad \frac{dy}{d\mathbf{h}_2} = ? \;(3) \qquad \frac{dy}{d\mathbf{x}_2} = ? \;(1)$$

$$\mathbf{h} = \mathbf{wx}$$

$$\mathbf{h} = [h_1, \ldots, h_m]^T$$

$$\mathbf{x} = [x_1, \ldots, x_n]^T$$

$$W_{m \times n}$$

$$\frac{d\mathbf{h}}{d\mathbf{x}} = \left[\frac{dh_i}{dx_j}\right]_{m \times n} = W_{m \times n}$$

$$\mathbf{x} = f(\mathbf{h}) = tanh(\mathbf{h})$$

$$\mathbf{h} = [h_1, \ldots, h_m]^T$$

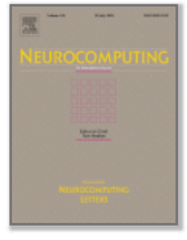
$$\mathbf{x} = [x_1, \ldots, x_n]^T$$

$$\frac{dx_i}{dh_i} = 1 - tanh^2(h_i)$$

```
>> h = sym('h'); diff(tanh(h))

ans =

1 - tanh(h)^2
```

Multi-state Potts Neurons

Neurocomputing
Volume 136, 20 July 2014, Pages 56-70



Annealed cooperative–competitive learning of Mahalanobis-NRBF neural modules for nonlinear and chaotic differential function approximation

Jiann-Ming Wu, Chun-Chang Wu, Ching-Wen Huang

Show more

https://doi.org/10.1016/j.neucom.2014.01.031

Abstract

This work explores annealed cooperative–competitive learning of multiple modules of Mahalanobis normalized radial basis functions (NRBF) with applications to nonlinear function approximation and chaotic differential function approximation. A multilayer neural network is extended to be composed of multiple Mahalanobis-NRBF

LETTER Communicated by Carsten Peterson

Natural Discriminant Analysis Using Interactive Potts Models

Jiann-Ming Wu
jmwu@server.am.ndhu.edu.tw
Department of Applied Mathematics, National Donghwa University, Shoufeng, Hualien 941,Taiwan, Republic of China

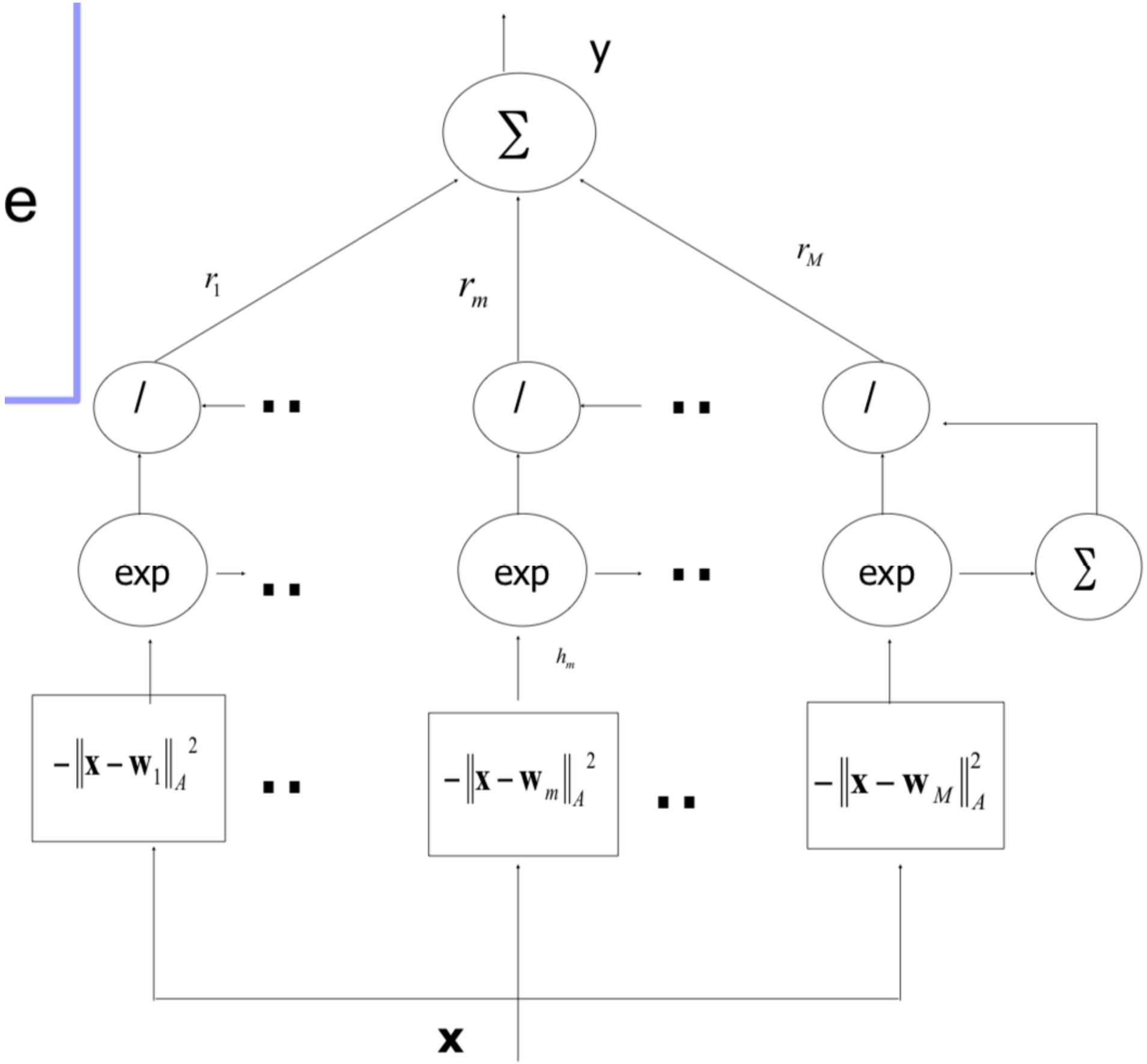
Natural discriminant analysis based on interactive Potts models is developed in this work. A generative model composed of piece-wise multivariate gaussian distributions is used to characterize the input space, exploring the embedded clustering and mixing structures and developing proper internal representations of input parameters. The maximization of a log-likelihood function measuring the fitness of all input parameters to the generative model, and the minimization of a design cost summing up square errors between posterior outputs and desired outputs constitutes a mathematical framework for discriminant analysis. We apply a hybrid of the mean-field annealing and the gradient-descent methods to the optimization of this framework and obtain multiple sets of interactive dynamics, which realize coupled Potts models for discriminant analysis. The new learning process is a whole process of component analysis, clustering analysis, and labeling analysis. Its major improvement compared to the radial basis function and the support vector machine is described by using some artificial examples and a real-world application to breast cancer diagnosis.

LETTER Communicated by Carsten Peterson

Natural Discriminant Analysis Using Interactive Potts Models

Jiann-Ming Wu
jmwu@server.am.ndhu.edu.tw
Department of Applied Mathematics, National Donghwa University, Shoufeng, Hualien 941,Taiwan, Republic of China

Natural discriminant analysis based on interactive Potts models is developed in this work. A generative model composed of piece-wise multivariate gaussian distributions is used to characterize the input space, exploring the embedded clustering and mixing structures and developing proper internal representations of input parameters. The maximization of a log-likelihood function measuring the fitness of all input parameters to the generative model, and the minimization of a design cost summing up square errors between posterior outputs and desired outputs constitutes a mathematical framework for discriminant analysis. We apply a hybrid of the mean-field annealing and the gradient-descent methods to the optimization of this framework and obtain multiple sets of interactive dynamics, which realize coupled Potts models for discriminant analysis. The new learning process is a whole process of component analysis, clustering analysis, and labeling analysis. Its major improvement compared to the radial basis function and the support vector machine is described by using some artificial examples and a real-world application to breast cancer diagnosis.


e
y
Σ
r_1
r_m
r_M
/
/
/
exp
exp
exp
∑
h_m
$-\|\mathbf{x}-\mathbf{w}_1\|_A^2$
$-\|\mathbf{x}-\mathbf{w}_m\|_A^2$
$-\|\mathbf{x}-\mathbf{w}_M\|_A^2$
x

$$h_m \equiv \| \mathbf{x} - \mathbf{w}_m \|_{\mathbf{A}}^2$$
$$= (\mathbf{x} - \mathbf{w}_m)^T \mathbf{A} (\mathbf{x} - \mathbf{w}_m).$$

v_m **denotes the probability of being the state m**

$$\sum_m v_m = 1,$$

$$\mathbf{v} = (v_1, \ldots, v_M)^T \in [0, 1]^M$$

$$v_m \propto \exp(-\beta h_m),$$

$$\sum_m \nu_m = 1,$$

it follows

$$\nu_m \equiv \phi_m(\mathbf{x}) = \frac{\exp(-\beta h_m)}{\sum_j \exp(-\beta h_j)}.$$

Neural Organization

- Adaline
- Perceptron
- Receptive field
- Adaptive filter
- Receptive field
- Linear projection
- Sigmoid function
- Threshold function
- Radial basis function
- Normalization
- Bilinear
- Winner take all
- Relu
- Convolution
- Batch normalization
- Lattice structure
- A generative model

CDFA(Chaitic differential FA) : Mackey-Glass 17

$$\frac{\partial x}{\partial t} = \frac{ax(t-\tau)}{1+x^c(t-\tau)} - bx(t),$$

$$\tau = 17, a = 0.2, c = 10 \text{ and } b = 0.1$$

M. Mackey, L. Glass, Oscillation and chaos in physiological control systems, Science 197 (1977) 287.

Mackey-Glass 30

$$\frac{\partial x}{\partial t} = \frac{ax(t-\tau)}{1+x^c(t-\tau)} - bx(t),$$

$\tau = 30 \quad a = 0.2, c = 10$ and $b = 0.1$

CDFA: Nonlinear delay differential equations

$$\frac{\partial x}{\partial t} = x(t-\tau) - x^3(1-\tau),$$

where the delay τ is set to 1.6.

J.C. Sprott, A simple chaotic delay differential equation, Phys. Lett. A 366 (2007) 397–402.

Chaotic differential function approximation using Multilayer N rural Networks

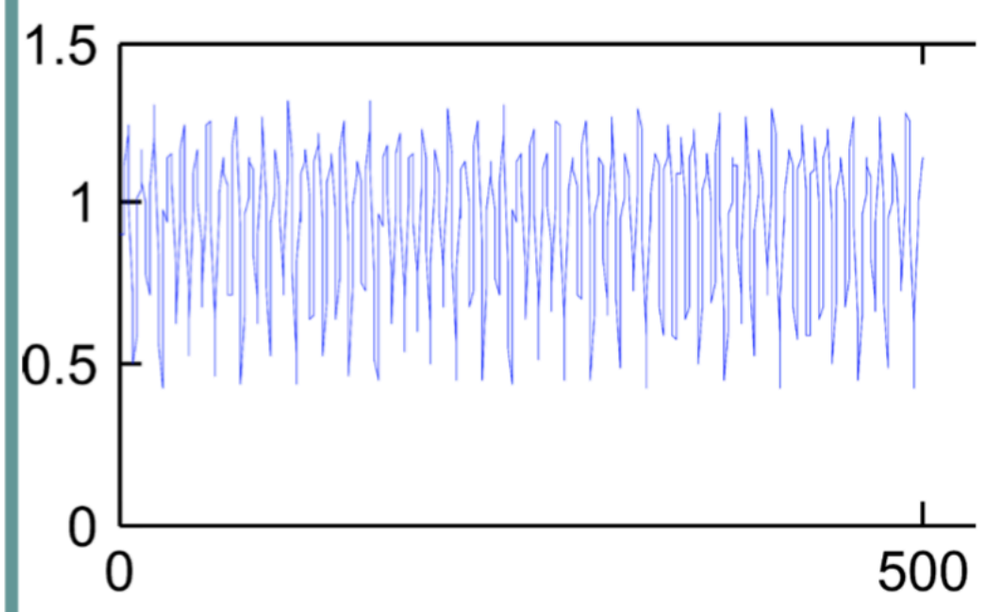
- Cechin, Pechmann, Oliveira, Chaos Solitons Fractal (2008)
- Mirzaee, Chaos Solitons Fractal (2009)
- Moody, Darken, Neural Computation (1989)
- Lin, Horne, Tino, IEEE Trans. Neural Netw. (1996)

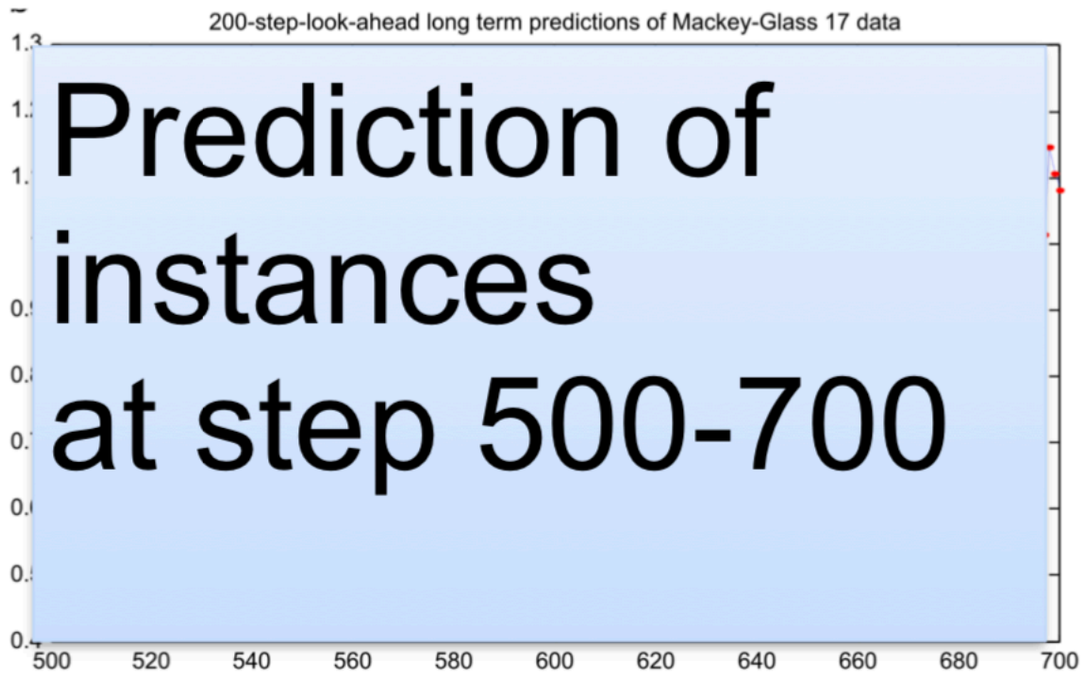
CDFA: goal and methodologies

- Goal
  - Long term look-ahead prediction
- Methodology
  - Data driven approaches
  - Recurrence relation modeling
  - Supervised learning of multilayer neural networks

Data driven long-term prediction

- MG(Mackey–Glass) 17 generated by RK(Runge-Kutta) 4

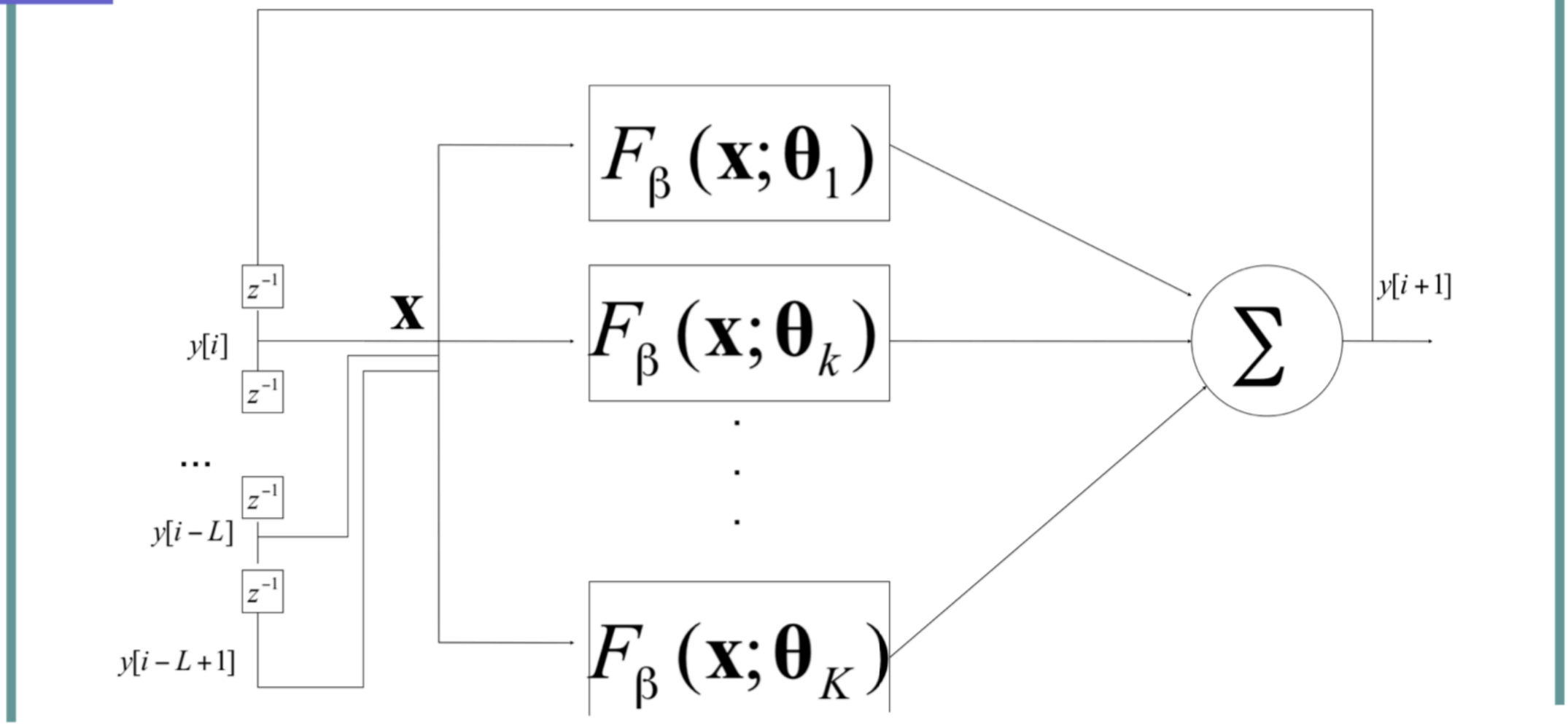




Prediction of instances at step 500-700

200-step-look-ahead prediction.

Nonlinear Recurrent Relation Modeling based on nonlinear function approximation



$$o_t = f(\mathbf{x}_t = (o_{t-L}, o_{t-L+1}, \ldots, o_{t-1})^T),$$

One-dimensional convolution