Working with Neural Networks

Process:

-problem

-data

-data preprocessing

-network development

-solution

Network Development:

-Data representation

-Network Topology

- error function; activation function

-Network Parameters

-Training

-Validation

Error Functions (Ch. 6, Bishop's book)

- Binary classification: two classes A, B; is x in A or B?
- Multi-class classification: eg., 10 classes (digits)
- Regression: predict the value of f(x)

- Why Sum-Squared-Error? Are there better alternatives?

- Statisticians: under some assumptions about the data there are some "best" error functions for each problem!

- Keep in mind that in practice the assumptions might not be satisfied and another error functions (like SSE) might work best!

Binary Classification

If the output of the network, *output*, is interpreted as probability then for each input pattern **<x, t>** , where the class label t = 0 or 1, it should satisfy:

- **0< *output* <1**
- ***p(t|x)= output*** ^{t}***(1- output)*** $^{1-t}$

It turns out that ML estimates of weights of a MLP with a sigmoid output unit minimize the cross-entropy error function:

$$E=-\Sigma \text{target}_{k*}\log(\text{output}_k) + ((1-\text{target}_k)*\log(1-\text{output}_k))$$

Main Conclusion :
For binary classification problems use logistic output unit and minimize the cross-entropy function!

Multi-class Classification

In case of c classes and 1-of-c coding of the outputs, the error function that should be optimized is also the cross-entropy:

$$E=-\Sigma_{k}\Sigma_{c}\, target_{k^*}\log(output_{k})$$

But the output layer is activated by a <u>softmax</u> activation function:

$$y_k=\exp(a_k)/\,\Sigma_{k'}(\exp(a_{k'}))$$

where a_k denotes the input to the k-th output node.

Main Conclusion :
For multi-class classification problems use <u>softmax activation function</u> and minimize the <u>cross-entropy</u> function!

Regression

Assume that the training set (x, t) is given by:

f(x)+norm(0, σ)

(f(x) is a “deterministic” function).

Then one can show that the ML estimates of weights of a MLP that model the training set correspond to the result of training MLP with the linear output unit with Sum-Squared-Error measure.

Moreover, for each x, the output of the network approximates <t |x> (the assumption about normality of noise can be skipped).

Main Conclusion :
For regression problems use linear outputs and the Sum-Squared-Error function

Error functions in Netlab/other packages

The three error functions are implemented in Netlab and should be specified when providing the type of the output nodes:

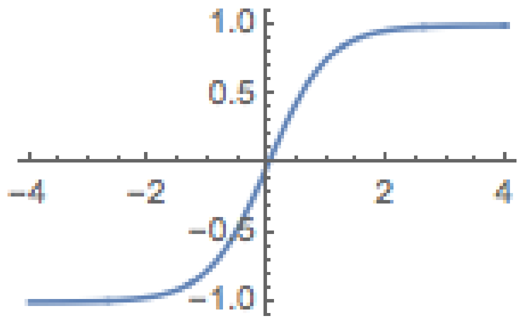
'linear‘ => SSE

'logistic’ => cross-entropy

'softmax‘ => cross-entropy + softmax

One can also add his own error function definitions (and the corresponding gradients).

[Note: in Netlab all hidden nodes use the tanh activation function]



Some links

A statistical view on Linear Regression: Chapter 15 from:

http://www.nrbook.com/a/bookcpdf.php

(my favorite textbook on almost everything)

A very elementary introduction to statistics (and linear regression): Chapter 7 from:

https://www.openintro.org/stat/textbook.php

Data Preparation: example

fuel_consumption = f(

cylinders:	multi-valued discrete
displacement:	continuous
horsepower:	continuous
weight:	continuous (?)
acceleration:	continuous
model year:	multi-valued discrete (?)
model type:	multi-valued discrete
model color:	multi-valued discrete (?)
maker:	multi-valued discrete

)

Input Representation

- Be creative !!!
  - use a single node to represent a single number
  - use several input nodes to represent a single number
  - scale your data
  - ignore some variables
  - introduce new variables
  - use polar system of coordiates

  ……..

Reasons…

- Data representation **depends on the problem**.
- In general NNs work with **continuous (real valued) inputs**. Therefore symbolic attributes are encoded by numbers.
- Attributes of different types may have different ranges of values which affect the training process.
- **Normalization** may be used, like the following one which scales each attribute to assume values between 0 and 1 (or [-1 1]).

$$x = \frac{x - \min}{\max - \min}$$

Alternatives

- Standardization: “center and normalize”:

  x := (x-mean(x))/std(x) (new x has mean=0 and std=1!)

- Use a “thermometer representation”, e.g.:
  any x from [0 10] can be represented by 10 inputs:

  2.7 ➔ [1, 1, 0.7, 0, 0, 0, 0, 0, 0, 0]

- “squeeze” outliers (set lower and upper bounds):

  x = min(x, upperbound); x=max(x,lowerbound)

- Replace **x** by **rank(x)** (the position in sorted version of x)

- Etc., etc.

Discrete Values

- N discrete values can be represented by:
  - Vectors of length n:
    1=> [1, 0, …], 2=>[0, 1, 0, …]
  - Vectors of length log(n) (binary representations)
    1=>[1, 0, 0], 2=>[0, 1, 0], 3=>[1, 1, 0], …
  - A single number, e.g, the relative frequency of observing the given value in the data
  - Colors could be represented by their 3 RGB components
  - use 2 nodes to represent days of a week:

    (sin(k*2*pi/7), cos(k*2*pi/7)), k=0, 1, …, 6

Output representation

- Different strategies for different problems:
  - **binary classification**
  - **multiclass classification**
  - **function approximation**
- Binary classification problems:
  - Two ways of representing outputs:
    - a single output node
    - two output nodes
- **Remark:** sigmoid function never (?) returns 0 or 1

  =>replace 0 by 0.1 and 1 by 0.9

Interpreting the output

A) **if** output >0.5 **then** 1 **else** 0

B) **if** output >0.7 **then** 1
elseif output < 0.3 **then** 0
else “unclassified”

C) **if** output > Threshold **then** 1 **else** 0
(Threshold chosen in such a way that the ratio POS:NEG is consistent with the training set)

Multi-class classification

Cases should be classified into K>2 classes

1. K binary classification problems (K nets)

2. K output nodes (one net)
   - ‘Winner-Takes-All’ rule (max node decides)
   - ‘bit-by-bit' match

3. $\log_2(K)$ output nodes (binary coding)

Function Approximation (regression)

- target values rescaled to:
  [0, 1] or [0.1, 0.9] if logistic sigmoid is used
  [-1, 1] or [-0.9, 0.9] if hyperbolic tangent is used

- output unit could use a linear activation function

- What about unbounded functions
  (e.g. f(x)=1/x)?
  => Use, e.g., y=> 1/(1+y) transformation

In all cases remember to convert outputs to the original scale !!!

Network Topology

- The number of layers and of neurons depend on the specific task. In practice this issue is solved by trial and error.

- Two types of heuristics can be used:
  - start from a large network and successively remove some neurons and links until network performance degrades.
  - **RECOMMENDED:** begin with a small network and introduce new neurons until performance is satisfactory.

Network parameters

- How are the weights initialized?
- How is the learning rate chosen?
- How many hidden layers and how many neurons?
- How many examples in the training set?

Initialization of weights

- In general, initial weights are randomly chosen, with typical values between -1.0 and 1.0 or -0.5 and 0.5.
- If some inputs are much larger than others, random initialization may bias the network to give much more importance to larger inputs. In such a case, weights can be initialized as follows:

$$w_{ji} = \pm \frac{1}{2N} \sum_{i=1,\dots,N} \frac{1}{|x_i|}$$

For weights from the input to the first layer

$$w_{kj} = \pm \frac{1}{2N} \sum_{i=1,\dots,N} \frac{1}{\varphi\left(\sum w_{ji} x_i\right)}$$

For weights from the first to the second layer

Choice of learning rate

- The right value of η depends on the application. Values between 0.1 and 0.9 have been used in many applications.

- Trial-and-error is the best heuristic, e.g,:
  0.01, 0.03, 0.06, 0.1, 0.3, 0.6, ...
  0.01, 0.006, 0.003, 0.001, ...

- Use faster algorithms (whenever possible)!

Size of Training set

- Rule of thumb:
  - the number of training examples should be at least five to ten times the number of weights of the network.
- Other rule:

$$N > \frac{|W|}{(1 - a)}$$

|W|= number of weights

a=expected accuracy

Applications of MLP

Classification, pattern recognition:

- MLP can be applied to non-linearly separable learning tasks:
  - Recognizing printed or handwritten characters
  - Face recognition
  - Scoring loan applications
  - Analysis of sonar data recognize mines

Regression and forecasting:

- MLP can be applied to learn non-linear functions (regression) and in particular functions whose inputs is a sequence of measurements over time (time series).

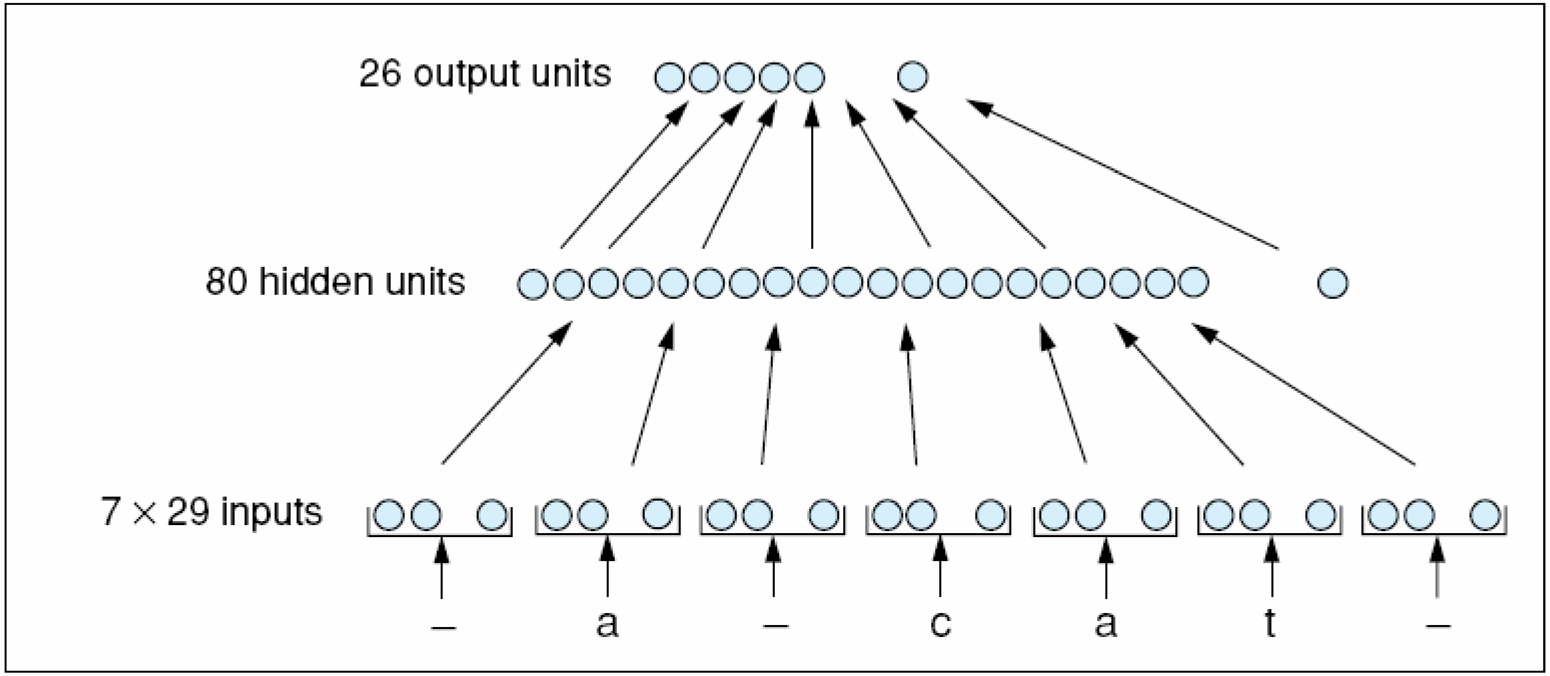
Data Compression and Dimensionality Reduction

NETtalk

(Sejnowski & Rosenberg, 1987)

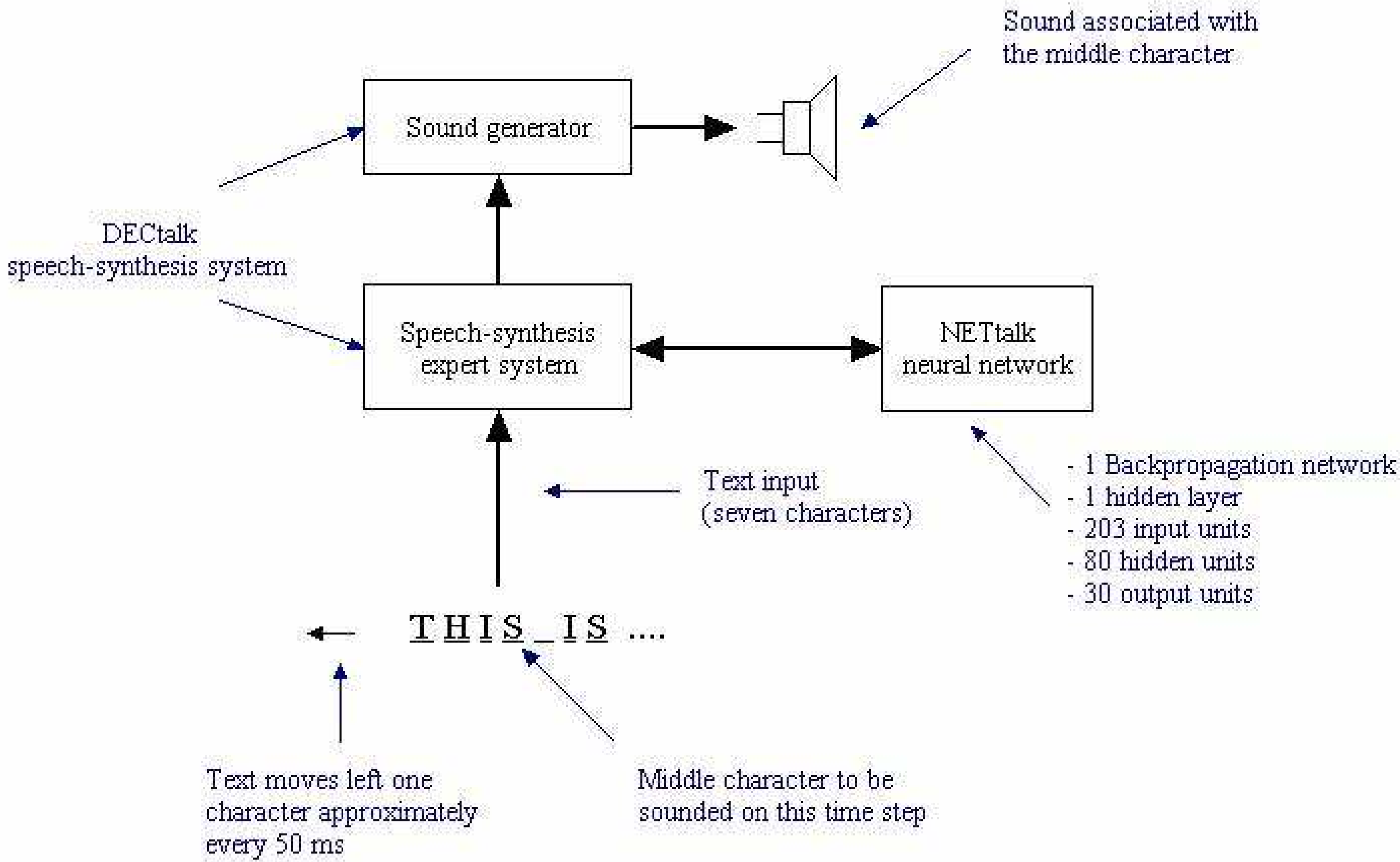
- The task is to **learn to pronounce English text** from examples (text-to-speech)
- Training data is a list of words from a side-by-side English text -phoneme source
- Input: 7 consecutive characters from written text presented in a moving window that scans text
- Output: phoneme code giving the pronunciation of the letter at the center of the input window
- Network topology: 7x29 binary inputs (26 chars + punctuation marks), 80 hidden units and 26 output units (phoneme code). Sigmoid units in hidden and output layer

NETtalk



NETtalk (contd.)

- Training protocol: 95% accuracy on training set after 50 epochs of training by full gradient descent.
  78% accuracy on a test set
- DEC-talk: a rule-based system crafted by experts (a decade of efforts by many linguists)
- Functionality/Accuracy almost the same
- Try: http://cnl.salk.edu/Media/nettalk.mp3


Sound associated with
the middle character
Sound generator
DECtalk
speech-synthesis system
Speech-synthesis
expert system
NETtalk
neural network
- 1 Backpropagation network
- 1 hidden layer
- 203 input units
- 80 hidden units
- 30 output units
Text input
(seven characters)
T H I S _ I S ....
Text moves left one
character approximately
every 50 ms
Middle character to be
sounded on this time step

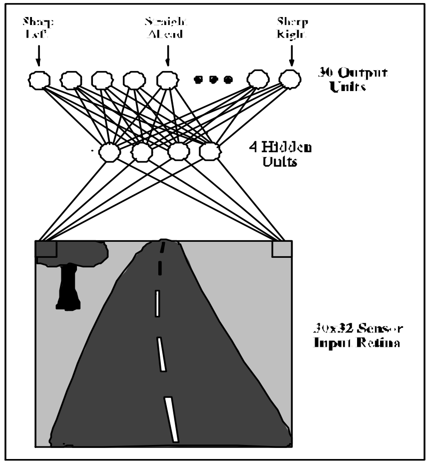
ALVINN:Autonomous Land Vehicle In a Neural Network

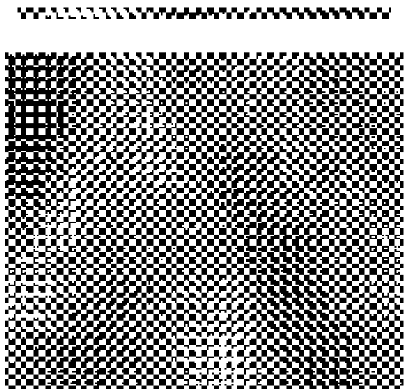
http://www.ri.cmu.edu/projects/project_160.html

30 outputs
for steering

4 hidden
units

30x32 pixels
as inputs





30x32 weights
into one out of
4 hidden units

Fraud Detection with Credit Cards

http://www.fico.com/en/Products/DMApps/Pages/FICO-Falcon-Fraud-Manager.aspx

Right now, leading institutions around the world trust Fair Isaac's Falcon™ Fraud Manager to protect more than 450 million active credit and debit cards. Why? They know they'll receive regular technological advances which will allow them to stay a step ahead of new and emerging fraud types. Protecting 65% of the world's credit card transactions, Falcon detects fraud with pinpoint accuracy via **proven neural network models** and other proprietary predictive technologies. Debit, credit, oil and retail card issuers in numerous marketplaces rely on Falcon to detect and stop fraudulent transactions - and combat identity theft - in real time.

The network: ***several perceptrons trained with Gallant's pocket algorithm with ratchet***