

COMP4302/COMP5322, Lecture 3
NEURAL NETWORKS

ADALINE.
**Evaluating Performance of
Learning Algorithms.**

ADALINE - Outline

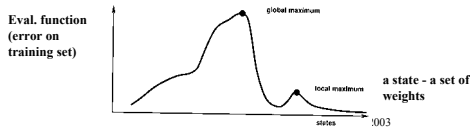
- **ADALINE's architecture**
- **Investigation of ADALINE's boundaries**
- **LMS algorithm**
 - steepest descent
 - LMS rule - derivation
 - generalization to multiple neuron linear nets
 - error space
 - learning rate
 - example
- **Capabilities and limitations**
- **History**

ADALINE Network

- **ADALINE:**
ADaptive LInear Neuron or
ADaptive LInear Element - when NNs became less popular :-)
- **Similar to the perceptron, the only difference is?**

ADALINE's Learning as a Search

- **Supervised learning:** $\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_n, t_n\}$
- The task can be seen as a search problem in the weight space:
 - Start from a random position (defined by the initial weights) and find a set of weights that minimizes the error on the given training set
- **Initial state:** a random set of weights
- **Goal state:** a set of weights that minimizes the error on the training set
- **Evaluation function (performance index):** an error function
- **Operators:** how to move from one state to the other; defined by the learning algorithm



7

Performance Index: Mean Square Error

- ADALINEs use the **Widrow-Hoff algorithm** or **Least Mean Square (LMS) algorithm** to adjust the weights of the linear network in order to minimize the mean square error
 - **Error** - difference between the target and actual network output
 - **mean square error**

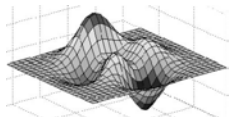
$$mse = \frac{1}{n} \sum_{k=1}^n e(k)^2 = \frac{1}{n} \sum_{k=1}^n (t(k) - a(k))^2$$

COMP4302/5322 Neural Networks, w3, s2 2003

8

Error Landscape in Weight Space

- Total error is a function of the weights
- In general, there are many local minima and only one global minimum
- Ideally, we would like to find the global minimum (i.e. the optimal solution)



Try Matlab demos - 2d and 3d graphics!

- The error space of the linear networks (ADALINE's) is a parabola (in 1d: one weight vs. error) or a paraboloid (in high dimension) and it has only one minimum



COMP4302/5322 Neural Networks, w3, s2 2003

9

How Does LMS Minimize the Error?

- Takes steps downhill
 - not guaranteed to find the global minimum except in the (glorious) situation where there is only one global minimum, that is the case for the ADALINEs!
 - All downward paths take you to the minimum. Therefore, LMS starting from anywhere will find the global minimum
- It is a hill climbing algorithm
 - moves around trying to find the lowest peak
 - keeps track only of the current state
 - do not look ahead beyond the immediate neighbors of the state
 - like climbing down in thick fog with amnesia
- Moves down as fast as possible
 - i.e. moves in the direction that makes the largest reduction in error
 - how is this direction called?

COMP4302/5322 Neural Networks, w3, s2 2003

10

Steepest Descent

- The direction of the *steepest descent* is called *gradient* and can be computed
- A function *increases* most rapidly when the direction of the movement is *in the direction of the gradient*
- A function *decreases* most rapidly when the direction of movement is *in the direction of the negative of the gradient*
- Hence, we want to adjust the weights so that the change moves the system down the error surface in the direction of the locally steepest descent, given by the negative of the gradient
- We want to minimize the total mean square error (over all examples). Widrow and Hoff *estimate* this error by using the square error after each iteration (i.e. this is an *approximate steepest descent*)

COMP4302/5322 Neural Networks, w3, s2 2003

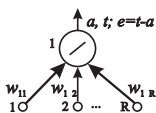
11

LMS Algorithm - Derivation

- Steepest gradient descent rule for change of the weights:

$$\Delta w_{i,j} = w_{i,j}(k+1) - w_{i,j}(k) = -\alpha \frac{\partial e^2(k)}{\partial w_{i,j}}, \quad j=1,2,\dots,R \quad k - \text{iteration number}$$

$$\begin{aligned} \frac{\partial e^2(k)}{\partial w_{i,j}} &= 2e(k) \frac{\partial e(k)}{\partial w_{i,j}} = 2e(k) \frac{\partial [t(k) - a(k)]}{\partial w_{i,j}} = \\ &= 2e(k) \frac{\partial [t(k) - (\sum_{j=1}^R w_{i,j} p_j(k) + b)]}{\partial w_{i,j}} = -2e(k) p_j(k) \end{aligned}$$



$$\begin{aligned} \Delta w_{i,j} &= -\alpha(-2e(k)p_j(k)) = 2\alpha e(k)p_j(k) = \eta e(k)p_j(k) \\ w_{i,j}(k+1) &= w_{i,j}(k) + \eta e(k)p_j(k) \end{aligned}$$

- Similarly, for the bias:

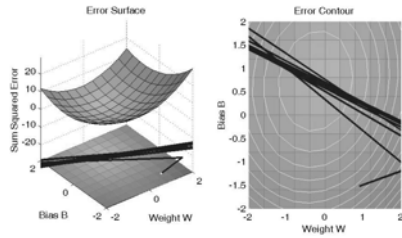
$$b_j(k+1) = b_j(k) + \eta e(k)$$

COMP4302/5322 Neural Networks, w3, s2 2003

12

Learning Rate

- **Too big**
 - the system will oscillate as the correction will be too large and will overshoot the target
- Try demolin?!



COMP4302/5322 Neural Networks, w3, s2 2003

16

Learning Rate

- **Too small**
 - the system will take a long time to converge
- **A constant**
 - may never converge to an unchanging value but will oscillate around the best solution
- **If it gradually drops toward zero**
 - eventually the weights will cease to change, even if the errors are not completely corrected.
 - Typically used: $\eta = \text{constant} / n$ n - number learning trials

COMP4302/5322 Neural Networks, w3, s2 2003

17

Apple/Banana Example

Training set:

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$$

Banana

Apple

Learning rate: $\eta = 0.4$

Stopping criteria: $\text{mse} < 0.3$

COMP4302/5322 Neural Networks, w3, s2 2003

18

Apple/Banana Example Iteration One

Learning rate: $\eta = 0.4$

First iteration - p_1 (banana):

$$a(0) = \mathbf{W}(0)\mathbf{p}(0) = \mathbf{W}(0)\mathbf{p}_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = 0$$

$$e(0) = t(0) - a(0) = t_1 - a(0) = -1 - 0 = -1$$

$$\mathbf{W}(1) = \mathbf{W}(0) + \eta e(0)\mathbf{p}^T(0) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} + 0.4(-1) \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix}$$

Apple/Banana Example - Iteration Two

Second iteration - p_2 (apple):

$$a(1) = \mathbf{W}(1)\mathbf{p}(1) = \mathbf{W}(1)\mathbf{p}_2 = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4$$

$$e(1) = t(1) - a(1) = t_2 - a(1) = 1 - (-0.4) = 1.4$$

$$\mathbf{W}(2) = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} + (0.4)(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix}$$

End of epoch 1, check the stopping criteria

Apple/Banana Example – Check Stopping Criteria

$$p_1, e_1 = t_1 - a_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

$$p_2, e_2 = t_2 - a_2 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = 1.28$$

$$mse = \frac{(-0.64)^2 + 1.28^2}{2} = 1.03 > 0.3$$

Stopping criteria is not satisfied => continue with epoch 2

Apple/Banana Example – Next Epochs

Third iteration - p_1 (banana):

$$a(2) = W(2)p(2) = W(2)p_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

$$e(2) = t(2) - a(2) = t_1 - a(2) = -1 - (-0.64) = -0.36$$

$$W(3) = [1.104 \ 0.016 \ -0.016]$$

...

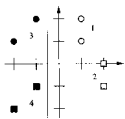
If we continue this procedure, the algorithm converges to:

$$W(\infty) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

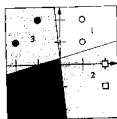
The decision boundary produced by the perceptron was:

$$W = [-1.5 \ -1 \ 0.5]$$

The Four-Class Example



perceptron



ADALINE

- This boundary is different than the one produced by the perceptron
- the perceptron stops as soon as the patterns are correctly classified, even though some patterns may be close to the boundary
 - LMS algorithm minimizes the mean square error => it tries to move the boundary as far as possible from the reference patterns =>the boundary falls half way between the patterns

Linear Networks - Capability and Limitations

- Both ADALINE and perceptron suffer from the same inherent limitation - can only solve linearly separable problems
- LMS, however, is more powerful than the perceptron's learning rule:
 - Perceptron's rule is guaranteed to converge to a solution that correctly categorizes the training patterns but the resulting network can be sensitive to noise as patterns often lie close to the decision boundary
 - LMS minimizes mean square error and therefore tries to move the decision boundary as far from the training patterns as possible
 - In other words, if the patterns are not linearly separable, i.e. the perfect solution does not exist, an ADALINE will find the best solution possible by minimizing the error (given the learning rate is small enough)

Linear Networks - Capability and Limitations – cont.

- Widrow-Hoff rule is a stable, reliable technique, insensitive to choice of parameters - one of these algorithms that “want to work” :-)
- LMS has more practical use than the perceptron; especially useful in digital signal processing : noise cancellation, echo cancellation (most long distance phone lines use it)

Things to Explore

- 1. A linear network with more constraints than free parameters (overdetermined system):
 - a linear neuron with 1 input => 1 weight and 1 bias to adjust
 - 4 training patterns (1-element input, 1-element output)
 $wp_1 + b = t_1$
 $wp_2 + b = t_2$
 $wp_3 + b = t_3$
 $wp_4 + b = t_4$
 - What will happen? Try demolin4!

Things to Explore – cont.

- 2. A linear network with more free parameters than constraints (underdetermined system):
 - a linear neuron with 1 input => 1 weight and 1 bias to adjust
 - 1 training pattern (1 element input, 1 element output)
 - What will happen? Try demolin5!

Perceptron and ADALINE - History

- 1943 - Warren McCulloch and Walter Pitts introduced one of the first artificial neurons
 - weighted sum of input signals is compared to a threshold to determine the output; when the sum is greater or equal to the threshold, the output is 1; else - 0
 - show that these neurons could compute any arithmetic function
 - e.g. basic Boolean functions can be represented with appropriate weights and biases: AND: $w_1=1, w_2=1, b=1.5$; OR: $w_1=1, w_2=1, t=0.5$; NOT: $w=-1, b=-0.5$
 - these units can be used to build an NN to compute any Boolean function
 - unlike biological neurons, the parameters of the network had to be designed, as no training method was available
 - the connection between biology and digital computers generated a lot of interest!

Perceptron and ADALINE – History (cont.1)

- 1958 - Frank Rosenblatt developed the perceptrons
 - the neurons in them are similar to those of McCulloch and Pitts
 - key contribution: introduction of learning rule for training perceptrons to solve pattern recognition problems
 - proved that the rule will always converge to correct weights, if such weights exist
 - learning: simple and automatic
 - perceptrons show great success for such a simple model
 - could even learn when initialized with random weights ...
- 1960 - Bernard Widrow and his student Marcian Hoff introduced the ADALINE and its learning rule which they called the LMS algorithm
- great deal of interest in NN research

Perceptron and ADALINE – History (cont.2)

- 1969 - Marvin Minsky and Seymour Papert - book “Perceptrons”
 - widely publicized the limitations of the perceptrons
 - demonstrated that the perceptrons were not capable of implementing certain elementary functions - XOR
 - provided detailed analysis of the capabilities and limitations of perceptrons
 - Rosenblatt, Widrow and Hoff were aware of these limitations and proposed new networks that would overcome them. But they were not able to successfully modify their learning algorithms to train these more complex nets
 - mortal blow in the area; the majority of scientific community walked away from the field of neural networks...
- Mid 60s -Widrow stopped working on NNs (because he was not able to adapt the rule to multilayer perceptrons) and begun to work on adaptive signal processing. He returned to NNs in the 80s and begun research on the use of NN in adaptive control using temporal backpropagation, a descendant of his original LMS...

Evaluating Performance of Learning Algorithms - Outline

- Evaluating what has been learnt
 - Error rate
 - Training and testing
 - Single holdout estimation, repeated holdout
 - Cross-validation

Evaluation: the Key to Success

- There are many learning methods (neural and non-neural)
- But to determine which methods to use on a particular problem we need a systematic ways to evaluate and compare them
- Is the performance on the training data a good indicator of the performance on the future data? Why?

Evaluation: the Key to Success

- There are many learning methods (neural and non-neural)
- But to determine which methods to use on a particular problem we need a systematic ways to evaluate and compare them
- Performance on the training data is not a good indicator of performance on the future data
- Simple solution that can be used
 - Split data into training and testing set
 - *Holdout* procedure – the method of splitting the original data into training and test set
- Problem: labeled data is usually limited
 - => more sophisticated techniques need to be used, e.g. special holdout procedure for small datasets

Overfitting (Overtraining)

- The error on the training set is very small but when a new data is presented to the classifier, the error is high
 - => the learning algorithm has memorized the training examples but has not learned to generalize to new situations!
- When does overfitting occur in general?
 - noise in data
 - too small training set - cannot produce a representative sample of the target function
- Next week - more about overfitting and how to prevent it!

Holdout Procedure

- Simple way to evaluate performance
 - Split data into training and testing set
 - *Holdout* procedure – the method of splitting the original data into training and test set
 - Usually: 1/3 for testing, 2/3 for training
- Training set: set of examples used to build the classifier
- Test set: set of independent examples that have played no part in the formation of the classifier
 - Assumption: both training and test data are representative samples of the underlying problem

Error and Success Rate

- Natural performance measure for classification problems: *success rate (accuracy)* and *error rate*
 - *Success*: instance's class is predicted correctly
 - *Error*: instance's class is predicted incorrectly
 - *Error rate*: proportion of errors made over the whole set of instances
 - *Success rate (accuracy)*: proportion of correctly classified instances over the whole set of instances
- Accuracy and error rate can be evaluated on training and testing set
 - Overly optimistic on training set
- Other performance measures – recall, precision, F1; cost sensitive measures

A Note on Parameter Tuning

- It is important that the test data is not used *in any way* to create the classifier
- Some learning methods operate in two stages:
 - Stage 1: build the basic structure
 - Stage 2: optimize parameter settings
- The test data can *not* be used for parameter tuning!
- Proper procedure uses three sets: training data, validation data, and test data
 - Validation set is used to optimize parameters
 - Examples:
 - neural networks – validation set is used as a stopping criterion (to prevent overtraining)

Making the Most of the Data

- Once the evaluation is complete, all the data can be used to build the final classifier
 - i.e. the validation and test data can be bundled back into the training data to produce a classifier for actual use
 - But the error rates must not be quoted based on this data!
- Generally:
 - The larger the training data, the better the classifier
 - The larger the test data, the more accurate the error estimate
- Dilemma - ideally we want to use as much of the data as possible for:
 - training to get a good classifier
 - testing to get a good error estimate

Stratification

- The holdout method reserves a certain amount for testing and uses the remainder for training
- Problem: the examples in the training set might not be representative
 - Example: all examples with a certain class are missing in the training set => the classifier cannot learn to predict this class
- Solution: *stratified holdout* – uses *stratification*
 - Ensures that each class is represented with approximately equal proportions in both data sets
 - It is well worth trying but provides only a primitive safeguard against uneven representation in training and test set

Repeated Holdout Method

- Holdout estimate can be made more reliable by repeating the process with different sub-samples
 - In each iteration, a certain proportion (e.g. 2/3) is randomly selected for training (possibly with stratification)
 - The error rates on the different iterations are averaged to yield an overall error rate
- This is called *repeated holdout method*
- Still not optimum – the different test sets overlap
 - Can we prevent overlapping?

Cross-Validation

- Avoids overlapping test sets
 - Step 1: data is split into k subsets of equal size
 - Step 2: each subset in turn is used for testing and the remainder for training
 - Step 3: the error estimates are averaged to yield an overall error estimate
- This is called *k-fold cross-validation*
- Often the subsets are stratified before the cross-validation is performed

10-Fold Cross-Validation

- Standard method for evaluation: 10-fold cross-validation or 10-fold stratified cross-validation
- Why ten?
 - Extensive tests on different data, with different learning methods, have shown that 10 is the best choice to get an accurate estimate
 - There is also some theoretical evidence for this
- Was also shown that the use of stratification improves the results slightly
- Note: neither the stratification, nor the division into 10 folds has to be exact
 - 10 approximately equal sets, in each of which the class values are represented in approximately the right proportion

More on Cross-Validation

- **Even better: repeated stratified cross-validation**
 - e.g. 10-fold cross-validation is repeated 10 times and results are averaged
 - Reduces the effect of random variation in choosing the folds
- **Leave-one-out cross-validation**
 - n -fold cross validation, where n is the number of examples in the data set
 - How many times do we need to build the classifier?
 - **Advantages:**
 - The greatest possible amount of data is used for training => increases the chance that the classifier is an accurate one
 - Deterministic procedure – no random sampling is involved (no point to repeating it 10 times or at all – the same results will be obtained)
 - **Disadvantage:**
 - high computational cost => useful for small data sets

COMP4302/5322 Neural Networks, w3, s2 2003

43
