# Classification and Multilayer Perceptron Neural Networks

Paavo Nieminen

Department of Mathematical Information Technology
University of Jyväskylä

Data Mining Course (TIES445), Lecture 10; Feb 20, 2012

# Outline

1. Automatic Classification of Objects
   - Classification Tasks
   - How to Evaluate Classification Performance
   - Some Classification Methods
2. Basic Idea of Artificial Neural Networks (ANN)
   - Inspiration from Biological Neural Cells
   - Multilayered Perceptron (MLP)
   - Other Neural Architectures
3. Training of a Neural Network, and Use as a Classifier
   - How to Encode Data for an ANN
   - How Good or Bad Is a Neural Network
   - Backpropagation Training
   - An Implementation Example

# Outline

1. Automatic Classification of Objects
   - Classification Tasks
   - How to Evaluate Classification Performance
   - Some Classification Methods
2. Basic Idea of Artificial Neural Networks (ANN)
   - Inspiration from Biological Neural Cells
   - Multilayered Perceptron (MLP)
   - Other Neural Architectures
3. Training of a Neural Network, and Use as a Classifier
   - How to Encode Data for an ANN
   - How Good or Bad Is a Neural Network
   - Backpropagation Training
   - An Implementation Example

# What is Classification? Why Automatic Tools?

"Classification" is to search an answer to a question like these:

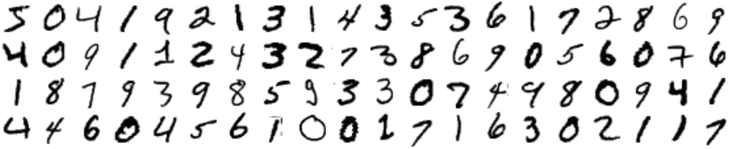
- "Does this song sound more like Abba or The Beatles?"
- "Is this a picture of an Apple, a Banana or a Cayenne Pepper?"
- "Does this picture represent the letter A, B, or C, D, E, F, . . . ?"

Classification here means the identification of the class or category of an object (Here, we assume predefined and known classes, which separates this definition from that of clustering).

- This kind of classification is easy for a person with enough experience about the objects and classes.
- But human is a slow classifier. For classifying, say 100 000 objects, we'd like a computer to do this automatically to avoid spending years of manpower on the job.

# Practical Example: OCR

What is this 80-digit number:



(first 80 digits from the MNIST data `http://yann.lecun.com/exdb/mnist/`)

Optical character recognition, OCR:

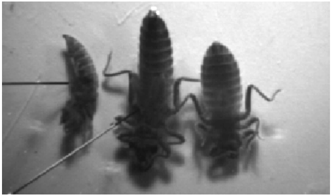
- Digitization of books, forms, postal ZIP codes, etc. . . .
- Much less manual work!
- Can be challenging; consider the different handwritings of persons.

This is only one example among many where automatic classification can be helpful . . .

# Other examples: biomonitoring, papermaking

Recent research topics at our Department:

- Taxa identification of benthic macroinvertebrates (bugs of the lake bottom) to aid biomonitoring.
- Detection and classification of faults in paper coming out from the paper machine.



These are examples where classification is applied to digital images, but naturally they can be used for any kind of data.

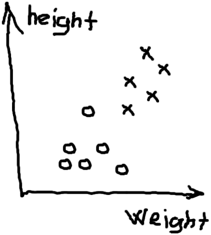
# Goal: Classifier is a System that Learns / Remembers

The goal is to create an automatic classifier that can classify objects that it has not yet seen (but they are expected to belong in one of the classes for which the classifier is built, for example characters in OCR, or taxa in biomonitoring).

- Classifier has to have some sort of a “memory”.
- A *training set* / reference set of data is inputted to the classifier before we let it do its job. We need to know the correct classes (called labels) in the training set.
- We need algorithms that “learn” and “remember”.
- Testing and comparing algorithms is done using a *test set*, for which the labels are known but which has no overlapping individuals with the training set.
- Many algorithms also use a *validation set* (essentially part of the labeled training set) to manage its learning process.

# Features and Feature Vectors

Naturally, every object must be represented as numbers before it can be input to a computer algorithm. These numbers, called *features*, are some measurements of the real-world objects to be classified. The measurements are gathered into a sequence of numbers called a *feature vector* whose dimension is the number of measurements made. Example in $\mathbb{R}^2$:



(Idea of “jockeys”(o) and “basketball players”(x), borrowed from Sonka et al.: “Image Processing, Analysis, and Machine Vision”)

# Classification Methods: Overview

Several methods exist for automatic classification. Some categories:

- Nearest-Neighbor
- Artificial Neural Networks
- Support Vector Machines
- Decision Trees
- Combination Methods (incl. “boosting” types and committee machines)
- ...

Which method is suitable depends on the task. How to evaluate and compare the performance of different classification algorithms/systems?

# Confusion Matrix, Two Classes

Classifier between two classes (for example a medical diagnosis of positive/negative for a certain disease) is called a *binary classifier*. A simple way to examine its classification performance is to collect the number of right and wrong guesses it makes (for the test set objects) into a 2-by-2 *confusion matrix*:

| | | Truth: positive | Truth: negative |
|---|---|---|---|
| Predicted as | positive | TP | FP |
| Predicted as | negative | FN | TN |

TP = number of true positives, FP = false positives, FN = false negatives, TN = true negatives. Total number of test set samples = (TP+FP+FN+TN). General “accuracy” = (TP+TN) / (TP+FP+FN+TN). “Sensitivity” = TP / (TP+FN). “Specificity” = TN / (TN+FP).

When I get diagnosed, I prefer the doctor has high sensitivity (not to miss any disease that needs further care). As a tax payer, I’d like publically funded doctors to have high specificity since false positives need further examinations which have a cost but turn out to be unnecessary.

# Confusion Matrix, Multiple Classes

A similar matrix can be made for a *multi-class classifier*. The numbers of right and wrong classifications are gathered in their respective slots in the matrix:

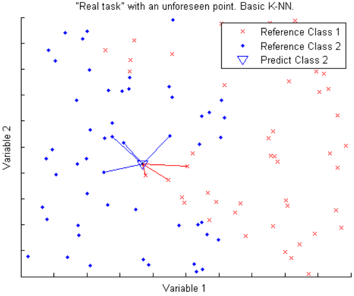
| | | Truth: | Truth: | Truth: | Truth: |
|---|---|---|---|---|---|
| | | Class 1 | Class 2 | Class 3 | Class 4 |
| Predicted as | Class 1 | **right** | wrong 2⟶1 | wrong 3⟶1 | wrong 4⟶1 |
| Predicted as | Class 2 | wrong 1⟶2 | **right** | wrong | wrong |
| Predicted as | Class 3 | wrong 1⟶3 | wrong | **right** | wrong |
| Predicted as | Class 4 | wrong 1⟶4 | wrong | wrong | **right** |

Correct classifications are found on the diagonal. From the other elements, one can see which classes get mixed up most easily.

Accuracy = sum of diagonal / sum of all elements. Accuracy within one class = diagonal value from a column / sum of all elements in one column.

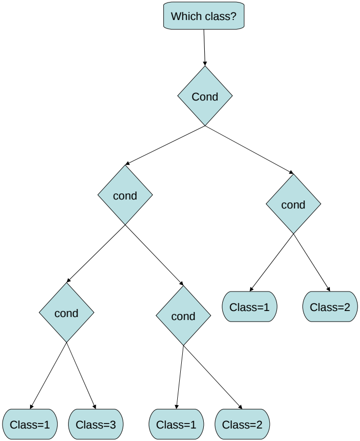
# Basic Idea of K-NN Classification

- “Memory” is a reference set of feature vectors with known labels. (No need for learning).
- Purpose is to classify a new, unforeseen feature vector.
- Choose K closest neighbors from the reference set (Ex. $K = 7$)
- Predict that the new object belongs to the class in which the majority of its closest neighbors belong to.



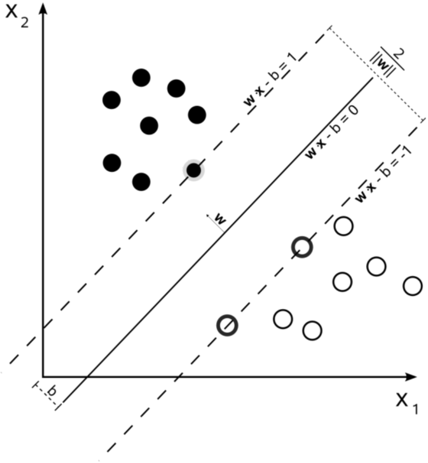
# Basic Idea of Decision Trees

- “Memory” is a tree of rules about the features. (Need an algorithm to learn rules).
- Begin applying the rules to an unforeseen feature vector from the root node.
- Upon arriving to a leaf node, predict that the new object belongs to the class represented by that leaf.



# Basic Idea of Support Vector Machines

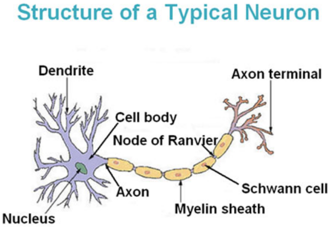
- “Memory” is the linear class boundaries in a high-dimensional space (“kernel trick” applied to transform original features, need an algorithm to learn boundaries)
- More detail on Tomi’s dedicated lecture about SVMs.



# Outline

1. Automatic Classification of Objects
   - Classification Tasks
   - How to Evaluate Classification Performance
   - Some Classification Methods
2. Basic Idea of Artificial Neural Networks (ANN)
   - Inspiration from Biological Neural Cells
   - Multilayered Perceptron (MLP)
   - Other Neural Architectures
3. Training of a Neural Network, and Use as a Classifier
   - How to Encode Data for an ANN
   - How Good or Bad Is a Neural Network
   - Backpropagation Training
   - An Implementation Example

# A Neuron

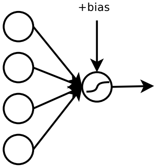


`http://commons.wikimedia.org/wiki/Image:Neuron.jpg`

- A biological neuron receives electrochemical signals from many sources; when the excitation is high enough, the neuron fires, passing on the signal. Neurons are connected via synapses.
- Since the 1950's this analogy has been used as the basis of a family of mathematical models.

Accuracy of this model? Well . . . See for example
`http://www.eurekalert.org/pub_releases/2010-11/sumc-nim111510.php`
(a research announcement from Stanford a couple of years ago)

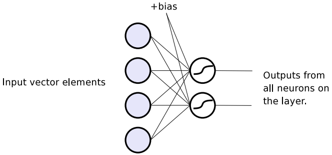
# An Artificial Neuron, and the Activation Function



- An artificial neuron combines its input signals for example by a weighted sum. The output is one numerical value, computed from a so called activation function, modeling approximately a “firing” of a coarsely modeled biological neuron.
- Mathematically, for example, $o = f(b + \sum_{i=1}^{n} w_i(\mathbf{a})_i)$, where $o$ is the output, $(\mathbf{a})_i$ are $n$ inputs, $w_i$ are summation weights, and $f$ is the activation function; $b$ is a bias term that determines how much activation induces firing.

→ Demo: Let us use Matlab or Octave to plot activation functions (linear, logistic sigmoid, hyperbolic tangent, step function)

# A Layer of Multiple Neurons



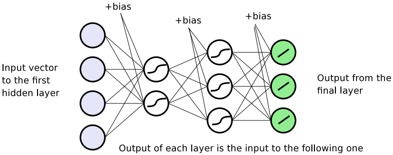
+bias

Input vector elements

Outputs from all neurons on the layer.

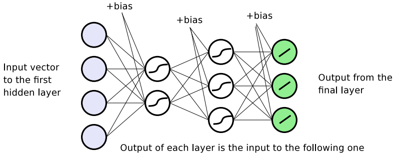
- Let us put many artificial neurons next to each other. Each neuron has its own synaptic weight coefficients, and an activation function is operating in each neuron. The input and the output are both numerical vectors.
- Mathematically, for example, $(\mathbf{o}^l)_j = f_j^l(b_j^l + \sum_{i=1}^{n_{l-1}} w_{j,i}^l (\mathbf{o}^{l-1})_i)$, where $\mathbf{o}^l$ is the output vector, $(\mathbf{o}^{l-1})_i$ are $n_{l-1}$ inputs, $w_{j,i}^l$ is the weight of the $i$:th input in the $j$:th neuron, $f_j^l$ is the activation function and $b_j^l$ the bias of the $j$:th neuron. The indices $l-1$ for the input vector and $l$ for the output vector anticipate the next step, where layers are interconnected.

# By combining layers, we get an MLP


+bias
+bias
+bias
Input vector to the first hidden layer
Output from the final layer
Output of each layer is the input to the following one

- Layers of different sizes can be chained.
- The output of each layer is computed using the same mechanism
- The output from layer $l$ is fed forward to layer $l + 1$ (when there are no feed-back loops; things get slightly more complex if there are; such constructions are called recurrent neural networks).
- With a bit of tweaking, the whole formulation with weights, biases, and functions, can be written in a layer-wise matrix form; each layer has a coefficient matrix and a "function matrix" …

# Output of an MLP in Layer-Wise Matrix Form



A compact notation for a simple MLP is

$$\mathbf{o}^0 = \mathbf{x}, \quad \mathbf{o}^l = \mathcal{F}^l(\mathbf{W}^l \hat{\mathbf{o}}^{(l-1)}) \quad \text{for} \quad l = 1, \ldots, L. \qquad (1)$$

Here $\mathbf{x}$ is the input vector. We set it as the ”output of the zeroth layer”. The special hat notation $\hat{\mathbf{o}}^{(l-1)}$ represents an operation where a number 1 is prepended to a vector, increasing its dimension; this way the bias terms of layer $l$ can be written as the first column of matrix $\mathbf{W}^l$. The notation $\mathcal{F}^l$ means that the activation function is applied to all components of a vector (it is a “diagonal function matrix”).

# Remarks About the Simple MLP

- The logistic sigmoid or the hyperbolic tangent are common choices for the activation function. These functions are differentiable.
- Usually you would use the same activation function on all layers.
- On the last layer, one can leave out the activation (i.e., “have linear activation”). Then the final layer outputs won’t be confined to the range $[0, 1]$ (logsig) or $[-1, 1]$ (tanh).
- But these, and many other variations, are to be decided according to the task at hand.

# Applications of Neural Networks

- MLP is a universal approximator: It can be shown that any continuous function can be approximated arbitrarily well by putting together a big enough hidden layer with suitable coefficient values. (But the proof of existence gives no construction advice for building such a net!)
- Such an approximator could obviously be a good tool for approximating an unknown function.
- Other applications include classification, denoising, control of machines . . .
- See literature using the keyword ”Neural network” or pick for starters something like http://en.wikipedia.org/wiki/Artificial_neural_network

# Other Neural Architectures

- Here we dealt with a simple artificial feed-forward neural network with fully connected weights. There are many variations (better for some tasks).
- Connections can be omitted, they can go further down than the next layer, weight coefficients can be shared among connections, . . .
- Feed-back loops, with delay, are possible. (Recurrent networks, well-suited for time-series forecasting and signal processing.)
- Radial basis function networks (RBF), Support Vector Machines (SVM)
- Self-Organizing Maps (SOM), Deep autoencoders, . . .
- Again, there is quite a lot of information available by searching with the above keywords.

# Outline

1. Automatic Classification of Objects
   - Classification Tasks
   - How to Evaluate Classification Performance
   - Some Classification Methods
2. Basic Idea of Artificial Neural Networks (ANN)
   - Inspiration from Biological Neural Cells
   - Multilayered Perceptron (MLP)
   - Other Neural Architectures
3. Training of a Neural Network, and Use as a Classifier
   - How to Encode Data for an ANN
   - How Good or Bad Is a Neural Network
   - Backpropagation Training
   - An Implementation Example

# Everything Needs to Be in Vectors

- An MLP operates on numerical vectors, so everything needs to be coded as ordered n-tuples of floating point numbers.
- The numbers are called “features”.
- Multiple features can be catenated into a feature vector (input to the ANN).

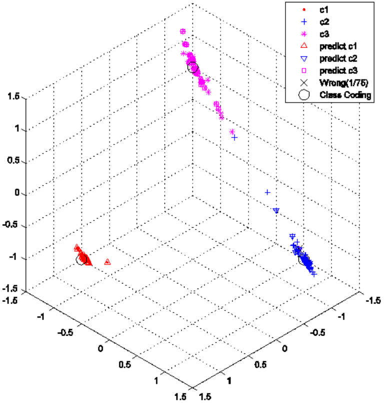
# Encoding of Nominal Features

- When a feature is essentially the name of a category, it is usually coded as an integer number (as in: 1=Finnish, 2=Swedish, 3=Norwegian)
- BUT in the case of an MLP classifier, we should convert these to binary vectors, for example: (1 -1 -1) = Finnish, (-1 1 -1) = Swedish, (-1 -1 1) = Norwegian
- The same should be done with the outputs of the classification task (i.e., names of the classes).
- Reason: Nominal entries lack a natural order, so the distance of their encoding should be the same between each two classes.

# Encoding of Classes (outputs)

Classes can also be encoded as binary vectors, for example:
$(1, -1, -1)$ = apple,
$(-1, 1, -1)$ = banana,
$(-1, -1, 1)$ = cayenne pepper.

- MLP maps the data from feature space to classification output space.
- Prediction can be selected as the “prototype”/encoding vector that is closest to the output.
- Distance from the ideal prototype reflects the uncertainty.



# Cost function: How bad is this MLP?

- Let us evaluate how bad a certain MLP is!
- If we can minimize the badness, we get a good MLP.
- Suitable cost function is for example the mean squared error $J(\{\mathbf{W}\}) = \frac{1}{2N}\sum_{i=1}^{N} ||\mathcal{N}(\mathbf{x}_i) - \mathbf{y}_i||^2$ where $\mathcal{N}(\mathbf{x}_i)$ is the output for a training vector $\mathbf{x}_i$ and $\mathbf{y}_i$ is the known truth about where the ideal classifier would place $\mathbf{x}_i$.

# How is the cost function affected by each weight coefficient

Let us write the error induced by the $i$:th training-target vector pair as $\mathbf{e}_i = \mathcal{N}(\{\mathbf{W}^l\})(\mathbf{x}_i) - \mathbf{y}_i$. It turns out that the gradient of the mean squared cost function can be computed iteratively in a layer-wise matrix form by using the backpropagation of error:

$$\nabla_{\mathbf{W}^l} J(\{\mathbf{W}^l\}) = \frac{1}{N} \sum_{i=1}^{N} \xi_i^l \, [\hat{\mathbf{o}}_i^{(l-1)}]^T,$$

where (assuming no activation on layer $L$)

$$\begin{aligned} \xi_i^L &= \mathbf{e}_i, & (2) \\ \xi_i^l &= \mathrm{Diag}\{(\mathcal{F}^l)'\,(\mathbf{W}^l\,\hat{\mathbf{o}}_i^{(l-1)})\}\,(\mathbf{W}_1^{(l+1)})^T\,\xi_i^{(l+1)}. & (3) \end{aligned}$$

In the equation $\mathbf{W}_1^{(l+1)}$ denotes a matrix that you get by removing the first column (containing bias terms) of $\mathbf{W}^{(l+1)}$.

# Let us Go Downhill

Now we have a cost function that tells how bad some set of weights are for the training data. We are also able to compute the gradient of this function. The simplest way to minimize badness, thereby making the MLP work better is the steepest descent method:

- On every iteration, we compute the gradient and take a small step in the opposite direction. If the step size is small enough, a local minimum is bound to be found. We recognize such when a bottom (small gradient) is reached.

If the step size is too big, we find nothing useful; if it is too small, it takes a long time to get anywhere. Nevertheless, steepest descent works, and it is commonly used.

# Other Training Methods and Formulations

- On these slides we have a differentiable cost function and the simplest possible training method that uses gradient information.
- The field of mathematics known as “Optimization” continuingly develops better methods for minimizing a function.
- Some methods applicable for MLP training are conjugate gradient methods, Broyden-Fletcher-Goldfarb-Shanno (BFGS), Levenberg-Marquardt (LM) . . .
- Also genetic algorithms have been used with success.
- The cost can be formulated differently, including non-smooth functions (that are at least in some points non-differentiable). We have methods for those, too!
- So. . . learn more by searching with the keyword ”optimization”

# Example Programs

→ Let us look at some code. (Made with Matlab; can be run also in Octave)

- Even though we have Matlab available for use at the university, I tend to prefer Octave because Matlab requires pretty stupid stupid silly connections with the license server, and that has brought just problems, problems, most of the time. And the last time I tried, my home Linux refused to co-operate with Matlab, invoking security policy issues; very inconvenient indeed.
- Don't let this personal opinion affect your decisions, though. Matlab is a beautiful tool when the license server issues don't fry your head.

# Summary

This was a short mini-introduction to the idea of classification, overview of some methods and in specific an artificial neural network (ANN) of the kind that is commonly called a multilayer perceptron (MLP). Key points:

- Classification needs training, validation, and test sets of known (labeled) input-output pairs.
- Comparison of classifiers can be done using confusion matrices.
- ANNs are one of many methods for classification.
- “Inspired by nature”, but in practice just matrix computations and sigmoidal functions.
- A few dozen lines of Matlab code is enough to create a working MLP. (When using C, Java, or such, you’ll need a library for matrix computations, but the MLP code itself remains small.)
- Gradient-based optimization is suitable for training.