Machine Learning: An Introduction

A Module for GenICT II

Herbert Jaeger

Jacobs University Bremen

1

Overview

Session 1: What is "Machine Learning", and what is it not? Examples and non-examples

Session 2: The core of ML: modeling probability distributions

Session 3: Feedforward neural networks: structure

Session 4: Feedforward neural networks: training procedures

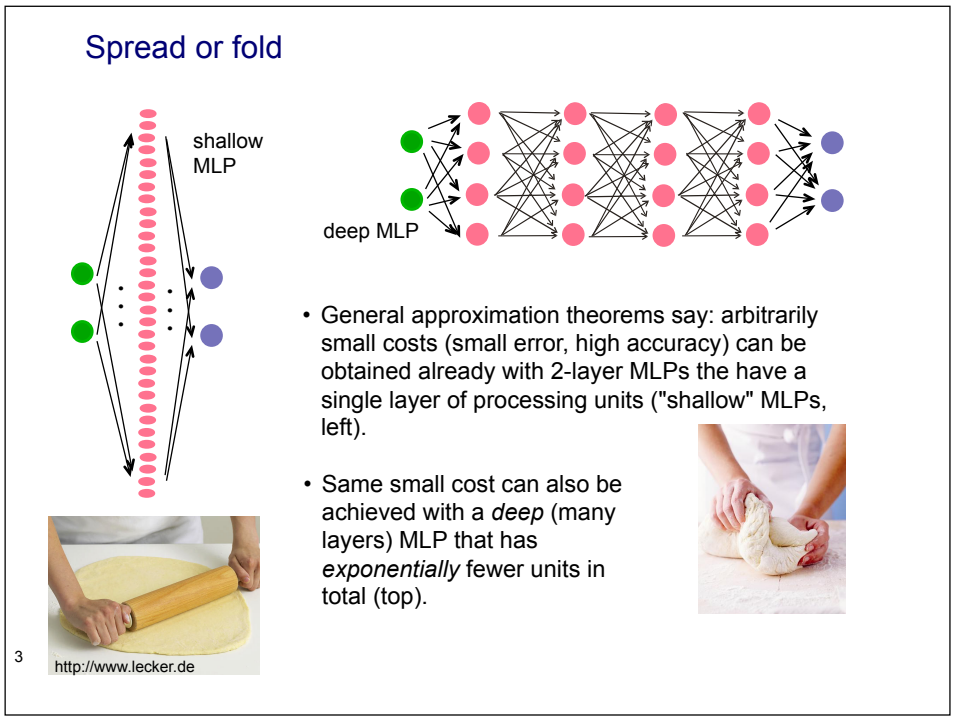
Session 5: Deep Learning: the beauty and power of brute force (striking examples)

Session 6: A bouquet of other ML stuff

2

GenICT 2: Handout 5

A deep impact

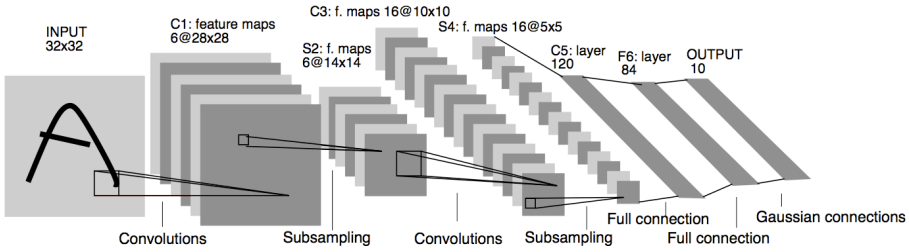


- General approximation theorems say: given a training data set $(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$, a cost function like the MSE_{train}, and an arbitrarily small cost target value $C > 0$, then there exists a 2-layer MLP (that is, 2 layers of adaptive synaptic connections, but only a single layer of hidden units) which achieves this small cost.
- These low-cost (highly accurate) 2-layer MLPs however require an extremely large number of hidden units. The intuitive reason for this: such 2-layer MLPs approximate the performance landscape by modeling every "pixel" in the input space individually with a few dedicated hidden processing units. For high resolution modeling, many pixels are needed; similarly, for high-dimensional input the number of required "pixels" explodes according to the curse of dimensionality.
- A nice property of 2-layer MLPs: Backpropagation algorithm functions well; many of the difficulties listed in the previous session are not an issue.
- Therefore, for a long time (mid-1980's until about 2010), 2-layer MLPs (sometimes 3-layer MLPs) were the only ones used routinely by end-user engineers.
- This implies, reversing the above arguments, that only relatively simple and/or low-accuracy modeling problems could be successfully handled by routine end-users.
- Only very few, highly specialized academic research groups also investigated many-layer MLPs in the years until about 2006.

- It was intuitively evident to every researcher in ML that many-layer MLPs, in principle, should be preferred over 2-layer MLPs when the modeling tasks become complex. There are two inherent advantages of such "deep" MLPs:
    - For a given targetted accuracy, 2-layer MLPs need exponentially more units than "deep" MLPs. In practice this excludes 2-layer MLPs from solving complex modeling tasks.
    - Deep MLPs have intrinsically better properties with respect to overfitting.
- But the difficulties with deep MLPs are enormous and proved insurmountable (except for one or two highly specialized research groups in highly specific modeling tasks) for a long time:
    - Due to the extremly "contorted", irregular geometry of the performance landscape in such deep MLPs, the backprop algorithm would either run dead ("vanishing gradient") on the way or become numerically unstable ("exploding gradient").
    - "Guessing" a good initial model becomes superbly difficult. Again due to the very "rugged" performance landscape, any naïve guess for an initial model would place it next to a local minimum of high cost; "going downhill" in the performance landscape would converge to a very suboptimal solution.
- 2006 was a "turn of the tide" year:
    - Geoffrey Hinton and Ruslan Salakuthdinov published a paper in Science, "Reducing the Dimensionality of Data with Neural Networks", which gave a (very nontrivial and expensive) solution to the initialization problem for backprop-training of many-layer MLPs.
    - Yoshua Bengio and Yann LeCun wrote a long paper (preprints available 2006, final publication 2007), "Scaling Learning Algorithms towards AI", which for the first time gave a clean line of arguments why and in what sense many-layer MLPs are superior to 2-layer MLPs. These arguments were partly analytical, partly intuitive, – but compelling.
    - The term "deep" to denote neural networks with many layers appears to have become firmly established in this year.
- It wasn't coincidental that these landmark papers were written by Hinton, Bengio, and LeCun. Hinton and LeCun are pioneers of the backpropagation algorithm, Bengio is a machine learning superforce, and the three of them operate from close-by locations (Toronto, Montreal, and New York), enabling close contacts and exchanges of group member.
- Today, Hinton has become Distinguished Researcher of Google, LeCun has become Director of a newly created AI Research Institute funded by Facebook, meaning in both cases very large-scale funding to their groups (and reduced academic presence). A deep learning startup called DeepMind, founded by Hinton, has been acquired by Google for about 500 Mio USD.
- Some years after the initial deep learning ignition, Andrew Ng, robotics and ML researcher (and co-founder of Coursera) entered the DL field. After a Google intermezzo he joined Baidu (= "Chinese Google") as Chief Scientist, directing Baidu's Silicon Valley research center.
- Only Bengio still runs free in the wild, wild world of academia, resisting the temptations of big money...

LeNet 1, 2, ...

Figure shows LeNet 5,
from LeCun, Bottou, Bengio, Haffner 1998



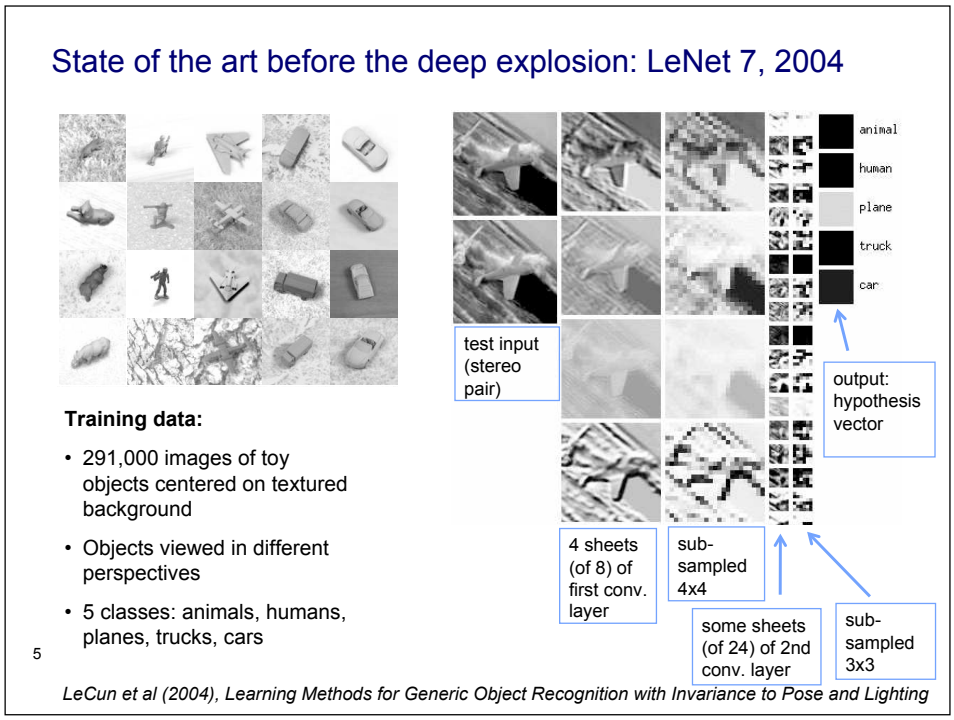
- Yann LeCun, in a solitary way, designed deep NNs throughout the years: LeNet 1, LeNet 2, ... till the present day
- "Convolutional networks": architectures specialized for digit / letter / image recognition tasks
- Early industrial applications in cheque reading
- Deviates from "pure" MLPs in many ways, has different kinds of processing layers that perform feature extraction, spatial shifts, averaging, classical MLP, specialized readout layer

4

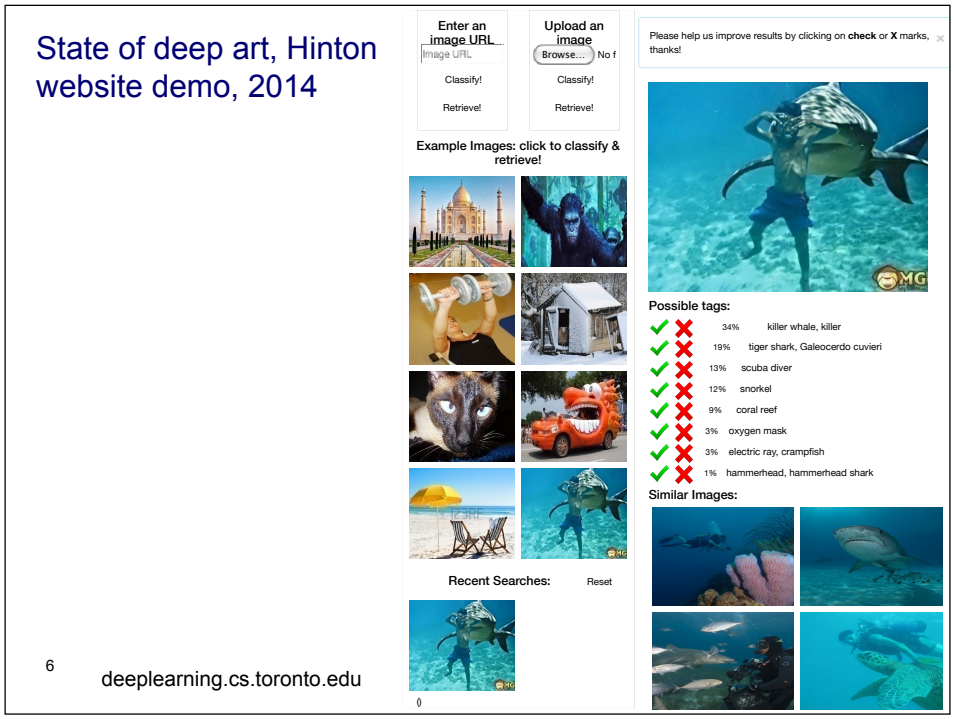
- But still, the entire system is trained by backprop-enabled gradient descent

- In the 1990's and early 2000's, Yann LeCun (Courant Institute of Mathematics, New York) was by and large the only ML researcher who persistently designed many-layer neural networks, christianed LeNet 1, LeNet 2, ...
- LeCun was a pioneer of the original backprop algorithm and refined it in many ways over the years. A role model for tenacity in scientific research.
- The LeNet family of NNs are called *convolutional networks*. This is a particular, highly pre-structured architecture, with a particular dedication to image classification.
- Convolutional networks are characterised by alternating *convolutional* and *subsampling* layers.
- A convolutional layer is "pre-wired" to learn a given small number of local geometric features and detect them *anywhere* in the input image (*translation invariance*).
- For every spatial *feature*, a convolutional layer has a separate "neural sheet" whose processing units represent the local presence of that particular feature. These "neural sheets" are called *maps* in convolutional network parlance. If, for instance, a convolutional layer should represent the three features "local dot pattern", "local diagonal line segment", "local patch of color blue", there would be altogether three maps in that layer.
- Subsampling locally averages over the convolutional-layer activations, reducing the number of units.
- The convolutional and subsampling layers, after training, implement a hierarchical feature extraction cascade. The number of features typically grows along the layer hierarchy, the spatial resolution shrinks.
- These layers are followed by "normal" MLP layers, and enventually by a specialized "readout" layer that transforms the high-level feature representation into the classification output (a hypothesis vector).

- While the architecture of a LeNet has heterogenous types of processing layers, the entire system can still be trained by backprop (if this is done expertly).
- Convolutional networks marked the state of the art in NN based image classification for 15 years and still are a commonly used template for designing today's "deep learning" neural networks in image classification tasks (and other tasks too).



- Training data were difficult (at that time) due to textured background and objects shown in different perspectives. They were easy in the sense that there were only 5 types of objects, and they were all centered in the images and of similar size.
- The first convolutional layer in (this instance of) LeNet 7 had 8 *maps* which computed 8 spatial features (4 shown). These features were "discovered" by the backprop training and therefore do not necessarily correspond to local image characteristics that can be intuitively named by humans.
- The subsampling steps are not trained. In a hardwired way, in these steps the activations of 4 x 4-sized tiles in the convolutional sheets are averaged, giving a reduction of the number of processing units by a factor of 16 (in the first subsampling after the first convolution), and from 3 x 3 tiles in the next subsampling.
- The image on the slide omits a third convolution stage with 100 maps, with no subsampling.
- The output layer has 5 units whose activations correspond to hypothesis strengthes. In the shown example, the unit coding for "plane" lights up = high activity value = high confidence that the input is a plane.



- Go visit the interactive website `deeplearning.cs.toronto.edu` and play with it yourself!
- Task: find tags for input pictures.
- I could not find a publication that describes this particular online demo, so I can only guess it was trained on a large tagged image repository like (parts of) Flickr, using a *deep* convolutional network with many layers (say, 10-20).
- Compared to LeNet 7:
    - Nr of classes (tags) has increased from 5 to thousands
    - Training and testing images are not laboratory-made and standardized, but come "from the wild"
- It's not as perfect as this slide suggests, as you will find out if you upload your own pictures. If your picture contains unconventional objects, they (obviously) won't be tagged in a meaningful way. The trained system does not understand reality but captures "only" the statistics of the training sample. If new test pictures fall outside that statistical range, the network *cannot* come up with appropriate outputs.

... even more stunning ...

7

Training data: 30,000 annotated Flickr pics

deeplearning.cs.toronto.edu

...performing: tagging & commenting...

Results

Tags

- docks
- pontoon
- dockside
- boats
- pontoons

Nearest Caption in the Training Dataset

a group of people on a bridge beside a boat .

Generated Captions

- several people are on a dock in the water .
- a group of people are on a boat in the distance .
- some people on a boat in the harbor .
- a group of people are on the dock on a large boat .
- some people are on a bridge over a boat .

8

deeplearning.cs.toronto.edu

… …

Results

Tags

- bookmarks
- crayon
- advertisements
- stickers
- ipad

Nearest Caption in the Training Dataset

little boy peeks over the side of a large brown box .

Generated Captions

- a boy wearing a blue and orange shirt is sitting in front of a painting .
- a young boy in a blue shirt is painted on a wall outside .
- a young boy is holding a painting of a wall in a red .
- a young boy holding a box of christmas back on the ground is behind him .
- a young boy standing on a red box .

9

deeplearning.cs.toronto.edu

Flickr image annotation experiments: comments

- Task: generate tags and captions for Flickr pics
- Combines two *modalities*: visual and linguistic
- Complex architecture with 2 different interacting network types
  - one of them a 19-layer convolutional network (for image processing functionality),
  - the other a *recurrent neural network* (for text generation)
- Training: still done with backprop

10

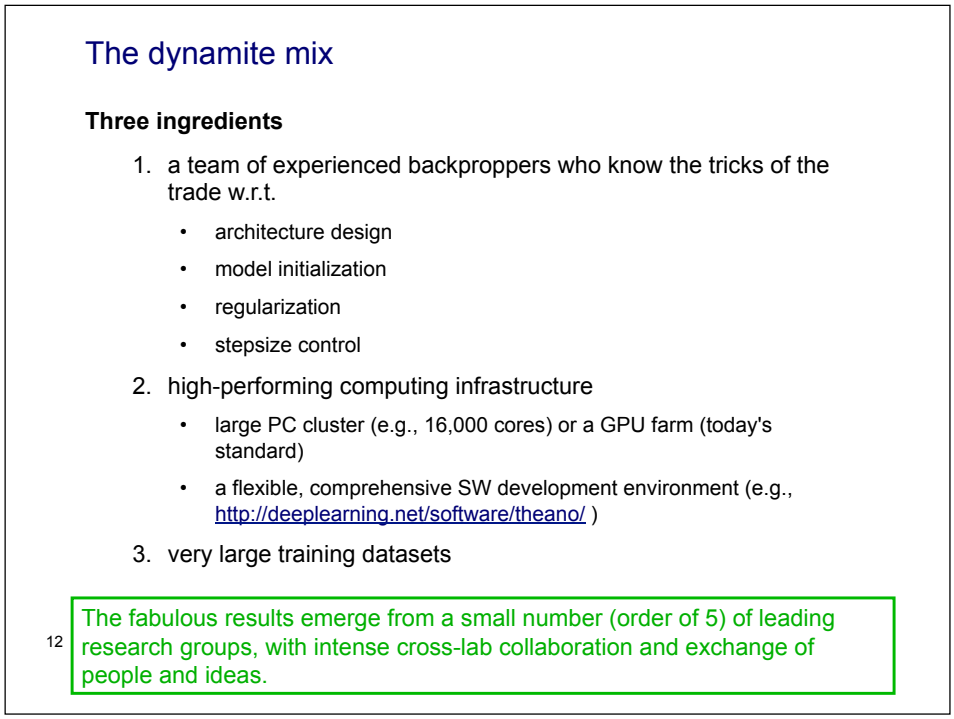
- Another online demo from `deeplearning.cs.toronto.edu` .
- Task is to tag images, and in addition create a 1-sentence descriptive phrase.
- Here I think I spotted a paper that describes this system: R. Kiros, R. Salakhutdinov, R. S. Zemel (Nov 2014), Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models, http://arxiv.org/abs/1411.2539
- If that paper describes what's behind the demo, it was trained on 30,000 annotated Flickr pictures.

- The task of transforming an image into a text string has structural similarities with translation, where one text string in language A is transformed into a text string in language B. The authors re-used many ideas from existing text translation systems.
- The annotation text output can be seen as a time series (text is read/generated in temporal order from left to right). Such *temporal* tasks are typically solved by a variant of neural networks called *recurrent* neural networks (RNNs). In RNNs, connection pathways don't "feed forward" in a single direction, but "cycle back" inside the network. This allows network activation to "reverberate", endowing the network itself with a temporal dynamics. RNNs are in many ways more difficult to train than feedforward networks, but there are variants of backprop that can handle this situation. Technically speaking, RNNs can be re-formulated as feedforward networks of "infinite depth" – time itself creates an incessant sequence of layers.

The deep learning explosion

- Since 2013, and especially since 2014, performance and variety of NN models explodes
- Successes in extremely demanding tasks, for example:
  - online translation
  - read historical hand-written documents
  - "inventing" text, including entire webpages on the html level
  - learning computer programs from example computations, e.g. programs that sort lists
- New results come in too fast for classical journals, use fast-reviewing conferences or arXiv instead
- **Massive** funding through Google, Microsoft, Facebook, and less widely known but almost as large microchip manufacturing companies
- A veritable frenzy in the ML community

11



- Deep Learning currently has an overwhelming coverage in the tech media, outshining all other research directions in ML. Its impact as an advertisment for ML cannot be over-estimated. But in fact, the majority of ML researchers do *not* do deep learning – among other reasons, because they can't (the required resources and specialized expertise are hard to pull together). But there are also other, better reasons why other strands of ML deserve (or need) to be pursued further:
    - DL has almost nothing to say about human/biological learning. Backprop is inherently biologically implausible (even impossible).
    - DL in its current formats is, in a way, too complex and expensive for wide-spread use. The stellar demos of 2014 could only be achieved by singular efforts of singularly experienced experts in superbly funded environments.
    - Deep learning architectures are so complex and individual that a mathematical analysis – in other words, scientific understanding – is out of reach. DL thrives on heuristics (extremely clever and refined!) and brute force computational power.
    - The world of machine learning (and its partner discipline, computational neuroscience) has been producing amazing breakthrough insights, unrelated to backprop or even unrelated to neural networks, at a rate of (my estimate) one revolutionary insight / year since a decade or two. ML in general is a wide, wide field and extremely productive. Don't reduce it to deep learning, stay tuned for surprise marvels of other kinds.

Machine Learning Module in GenICT 2: Handout 4

Learning: hide-and-seek in performance landscapes

Basic set-up

- **Given**: training data sample $D = (\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$
- **Assumption**: there exists a function g such that $\mathbf{y}_i = g(\mathbf{x}_i) + \nu$
- The "noise term" $+\nu$ denotes the circumstance that the function values $\mathbf{y}_i$ in the training data are corrupted by random fluctuations
- **Wanted**: a machine learning model of g, for instance a MLP that approximates g.
- The ML model is a statistical *estimate* of g, often written as $\hat{g}$.
- To get this machinery working, we first need a way to quantify the approximation quality of the estimate $\hat{g}$.

3

Measuring approximation quality, given a training sample

- **Given**: training data sample $D = (\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$, assumption $\mathbf{y}_i = g(\mathbf{x}_i) + \nu$, and an estimate $\hat{g}$.
- **Question**: how should we measure "approximation quality" of $\hat{g}$?
- **Answer**: by a *cost function* (also known as *error function* or *objective function*)
- A widely used cost function: mean square error (MSE).
- If the $\mathbf{y}_i$ are 1-dimensional (just numbers), the MSE incurred by $\hat{g}$ is given by

$$MSE_{\text{train}}(\hat{g}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{g}(\mathbf{x}_i)\right)^2$$

- Because the mean error is computed over the training sample, it is also called the *training error* or *training MSE.*
- If the $\mathbf{y}_i$ are n-dimensional vectors, the MSE_{train} is taken to be the average component-wise MSE.

4

- In Session 2 I said that ML always implies *statistical* modeling. The statistics aspect enters the game in function learning from training input-output records $(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$ through two circumstances:
    1. The training instances have been *randomly sampled*,
    2. the training target output values $\mathbf{y}_i$ contain a *random noise* component.
- Thus, in the background of professional ML modeling, there is always the humming sound of professionally applied statistics. We largely ignore this aspect here.
- In some application scenarios, the function approximation task not only demands to find an approximation $\hat{g}$ for the assumed underlying "true" function g, but furthermore also a model of the noise component ν. In the simplest case, this would be a standard deviation or variance.
- If the $\mathbf{y}_i$ are n-dimensional vectors, the training MSE is concretely computed as

$$MSE_{\text{train}}(\hat{g}) = \frac{1}{Nn}\sum_{i=1}^{N}\sum_{j=1}^{n}\left(\mathbf{y}_i[j] - \hat{g}(\mathbf{x}_i)[j]\right)^2$$

  where $\mathbf{v}[j]$ denotes the j-th component of a vector $\mathbf{v}$.
- A more mathematically elegant notation for the same quantity is

$$MSE_{\text{train}}(\hat{g}) = \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{y}_i - \hat{g}(\mathbf{x}_i)\right\|^2$$

- Besides the mean square error, there are other cost functions in frequent use. The MSE is popular because it leads to relatively simple computations, and because it is statistically appropriate if the noise term ν is *Gaussian* noise (normally distributed noise).

Minimizing the cost by searching in parameter space

- **Given**: training data sample $D = (\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$.
- **Wanted**: an MLP M computing a input-output function $\hat{g}$, such that we get a small

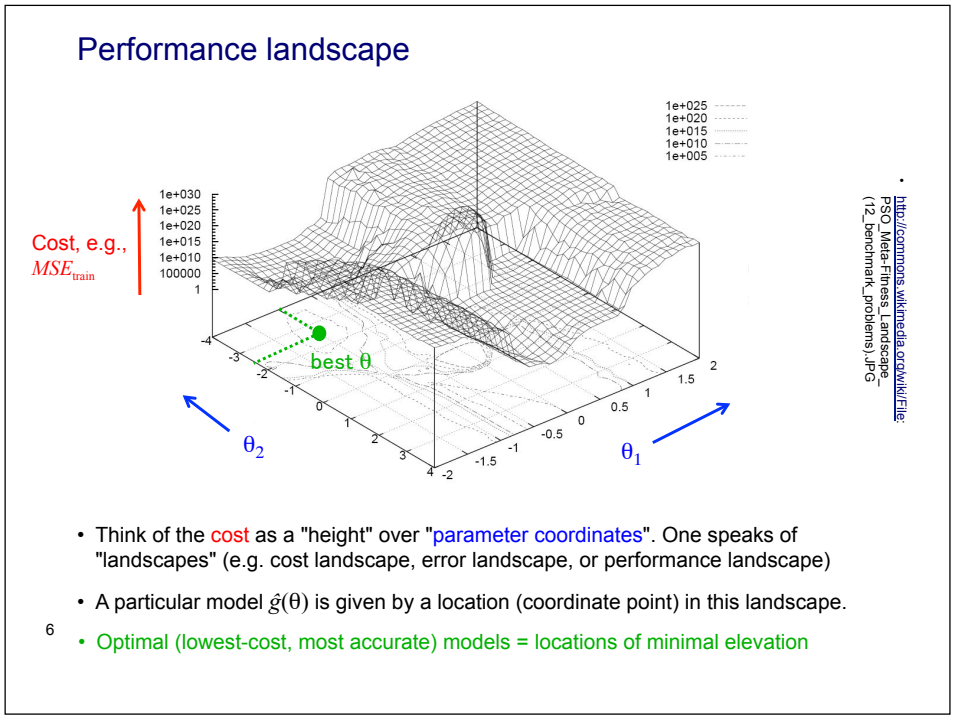
$$MSE_{\text{train}}(\hat{g}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{g}(\mathbf{x}_i)\right)^2$$

- **Routine procedure (and notation)**:
    1. We decide on the structure of the MLP. This means to fix the number and sizes of the weight matrices $\mathbf{W}^{i+1,i}$.

       (only input and output dimension are given by task, number of layers and their sizes must be decided by experimenter)
    2. Call all the matrix elements, lumped together, the *parameter set* $\theta = \{\theta_1, \theta_2, \ldots\}$
    3. The to-be-trained MLP M, and hence $\hat{g}$, then is fully determined by θ. We write $\hat{g}(\theta)$.
    4. "Training" the MLP then just boils down to "find parameters θ such that MSE_{train} becomes minimal".

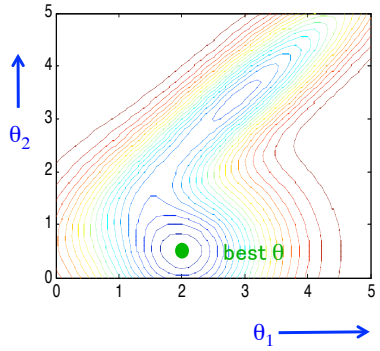
5

- The first step in the routine procedure, "decide on the structure of the MLP", is where the human engineer's experience and judgement enter the game. ML learning algorithms usually do *not* search for the best *type* or *structure* of the model. It is (typically) the humen expert who rubs his/her nose and muses: "well, I think I will use a feedforward neural network (and not some other kind of ML tool) and I will set it up with 3 hidden processing layers made of 100 units each".
- Deciding on the *type* of a ML model is harder than it may seem. While in this lecture I present only one model type (standard MLPs), there exist hundreds or even thousands of different model types, or *learning architectures* as they are also called. Almost every publication in this field proposes a new architecture variant.
- It is indeed in principle possible to also (try to) optimize the model structure (like number and size of layers) by automated search procedures, but these procedures tend to be overly expensive and also not as good as human insight.
- In an MLP, the parameter set θ is the collection of all the connection weight parameters of the network. It's a more convenient notation than writing down all the involved connection matrices $\mathbf{W}^{i,i+1}$. Notation $\hat{g}(\theta)$ is a convenient shorthand for "the MLP that has the parameters θ and tries to approximate g"
- Step 4, "find parameters θ that minimize the training error" is of course the tough part in this enterprise. How could one possibly find the "right" values for a million of such parameters?



- Image on this slide shows a parameter space that has only two parameters. Try to imagine such a cost landscape over 1 Mio parameters...

Performance landscape in a contour plot



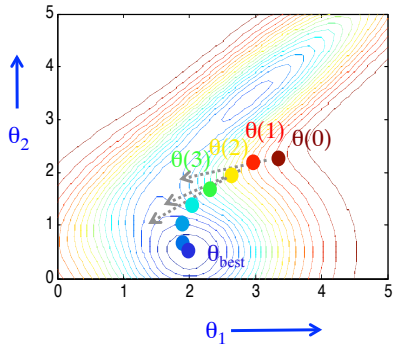
red lines: high cost, bad models

blue lines: low cost, good models

- In a *contour plot* visualization, the cost landscape is represented by elevation lines, like in hiking maps.
- Contour plots of error landscapes are intuitive for explaining the learning process in an MLP.

7

Learning a model by gradient descent optimization

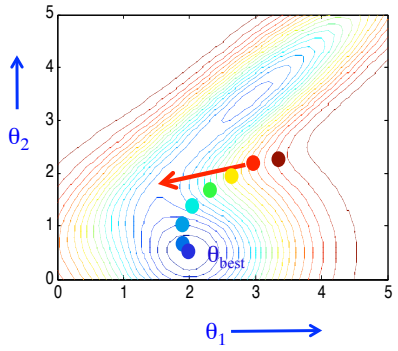


Iterative learning procedure

1. Guess initial (random) model $\theta(0)$.
2. Compute direction where landscape slope goes "downhill" most steeply: the [negative] gradient of the landscape at the location $\theta(0)$.
3. Go a little step in that direction to new location (model) $\theta(1)$.
4. Iterate steps 2 and 3, obtaining a sequence of models $\theta(2)$, $\theta(3)$,…
5. Continue until model sequence $\theta(n\text{-}1)$, $\theta(n)$, $\theta(n+1)$,... settles in a "trough", that is, a *minimum* of the landscape.
6. Stop when this process has "run dead": gradient becomes zero, no further advances. The final model θ_{best} is the result of the learning procedure.

8

The famous backpropagation algorithm



- Obviously crucial: computing the gradient.
- Not a lightweight problem:
  - Gradient depends on cost function
    $$MSE_{\text{train}}(\hat{g}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{g}(\mathbf{x}_i))^2$$
  - cost function depends on all training examples ...
  - ... and on all the (millions...) of network parameters!
- Standard algorithms for computing gradients exist but are too expensive.
- The *backpropagation algorithm* provides a clever computational shortcut.

9

(Re-)discovered in the early 1980's, the backprop algorithm made MLP training feasible. It catapulted neural networks forward, ending a 15-year stalling of NN research.

- For the mathematically inclined ones: formally, the gradient of a 1-dimensional function $f: \mathbb{R}^n \rightarrow \mathbb{R}, f(x_1, ..., x_n) = y$, is the vector made of the partial derivatives of f w.r.t. its arguments:

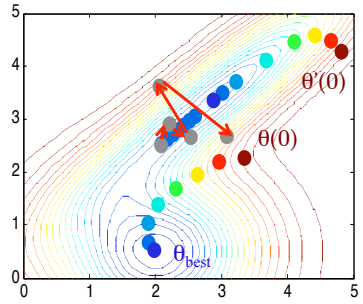
$$\text{grad}(f) = \left(\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}\right)$$

  It can be understood as an "arrow" that points in the direction where f increases most strongly (and the negative gradient, which is used in MLP training, points in the most strongly decreasing direction). The length of the gradient vector signifies the steepness of ascent. If f is rather "flat" at some point $f(\mathbf{x})$, the gradient vector is very short. This is the root cause for two problems. The problem of slow convergence (see slide 10), also known as "vanishing gradient" problem, arises in "rather flat" areas of the performance landscape. Conversely, numerical instabilities may arise in "steep" areas where large gradients make the interations take (too) big adaptation jumps.

- The celebrated backpropagation algorithm (often called just "backprop" or "BP") is not a complete NN training algorithm, but only a subroutine needed to compute the gradients. The entire training procedure involves many more items:
  - architecture selection (nr of layers, size of layers, type of processing units, ...),
  - model initialization (guessing the starting model),
  - programming the entire update loop, including monitoring schemes for the evolution of the cost function and installing termination criteria,
  - a number of control schemes to optimize the choice of adaptation stepsizes, escape routines if the process apparently gets stuck on the way, etc.

- The number of iterations that are needed until a satisfactory degree of convergence toward an optimal model is reached may easily be hundreds – or just a few dozens. It's unpredictable.
- In principle, the gradient is defined with respect to all training data, because the cost function is defined with respect to all training data. Thus, in principle, every incremental parameter update requires to "visit" all training data. This becomes infeasible for large training samples. Then one selects a smaller-sized *subsample* (called *minibatch*) from the complete training data set for each adaptation step. In an extreme version, called *stochastic gradient descent*, one chooses only a single training example per update step. That is computationally cheap but "jittery".
- The gradient descent iterations need some additional control mechanism to unfold stably and swiftly. In particular, the adaptation stepsize ("how far should the adaptation progress in the direction of the current gradient") should be carefully and dynamically adjusted.
- The 2-dimensional contour plots shown on the slides are misleadingly clean and simple. In reality, they are not 2-dimensional but super-high-dimensional, and the landscape is extremely "rugged", with a wild multitude of ridges, peaks, valleys, whirls and troughs. The worst thing is that the steepnesses encountered in such a landscape typically span many orders of magnitude, ranging from almost horizontal planes to almost vertical abysses. The image on slide 6 gives a faint impression.
- Application-oriented end-users of MLP modeling can find many ready-made toolboxes online, allowing them to run an MLP learning "out of the box". However, the results are more often than not quite poor – typically without the end-user being aware of that. The most frequent mistake made by "naïve" end-users is to overfit the training data. They then obtain a trained MLP that fits the training data very nicely (and the end-user believes all is fine as can be), but on new test data in exploitation situations, the model performs poorly.
- Despite these problems, MLP training based on backprop has matured into the arguably most flexible and powerful modeling method available for blackbox modeling today.

Still, not a piece of cake



MLP training can be done by non-experts for simple, small-scale problems.

For large-scale modeling tasks, dedicated expert knowledge is needed.

A zoo of difficulties

- **Local optima**: method finds only the "nearest-by" cost minimum. Globally best model may be missed. Choice of initial model is a real issue!
- Convergence toward optimum may be **extremely slow** ("vanishing gradient" problem)
- **Numerical instabilities** may arise.
- **Overfitting** of training data: the eternal nemesis of ML. Requires clever regularization schemes.
- **Network architecture** needs to be skillfully determined.
- Complex tasks require large models, large training data samples: **long computation times**! (can be weeks on supercomputing clusters)

10

Summary

- Given: training data $D = (\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$, a cost function (like MSE), and an MLP structure (number and sizes of processing layers).
- Training the MLP means to find parameters θ that minimize the cost ("make model fit the training data most accurately").
- Training procedure:
  1. guess initial model $\theta(0)$,
  2. iteratively improve its cost, slightly moving parameters in a "cost downhill" direction in each interation step,
  3. continue until a minimum cost ("trough" in the cost landscape) is reached.
- Critical enabler: the backprop algorithm for computing the "downhill" direction (gradient) cheaply
- Still not trivial: architecture design alternatives, still computationally expensive, local optima, numerical instabilities, model initialization, regularization

11

Gen ICT 2, ML module: handout 3

Artificial neural networks: some history

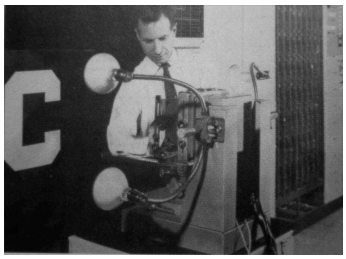
The Perceptron -- forefather of all artificial neural networks (ANNs)

- Developed by the US (neuro-)psychologist Frank Rosenblatt at the Cornell Aeronautical Laboratory
- Most cited early paper: "The Perceptron: a probabilistic model for information storage and organization in the brain", Psych. Review Vol 65 Nr 6, 1958
- Contemporary with beginnings of Artificial Intelligence
- Has all the standard features of the ANNs of today:
  - A specific task (letter image classification)
  - A biologically oriented "architecture" of interconnected, abstracted neural processing elements
  - A probabilistic approach
  - A learning algorithm
  - A mathematical analysis
  - A state-of-the-art computer implementation

3

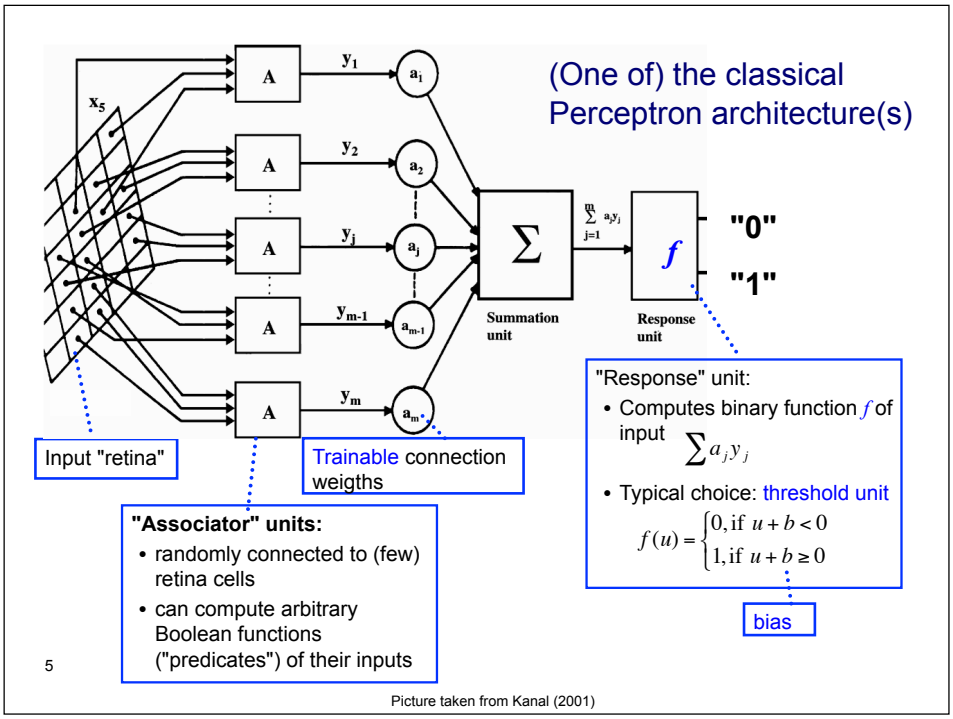
The Perceptron's task: binary pattern classification

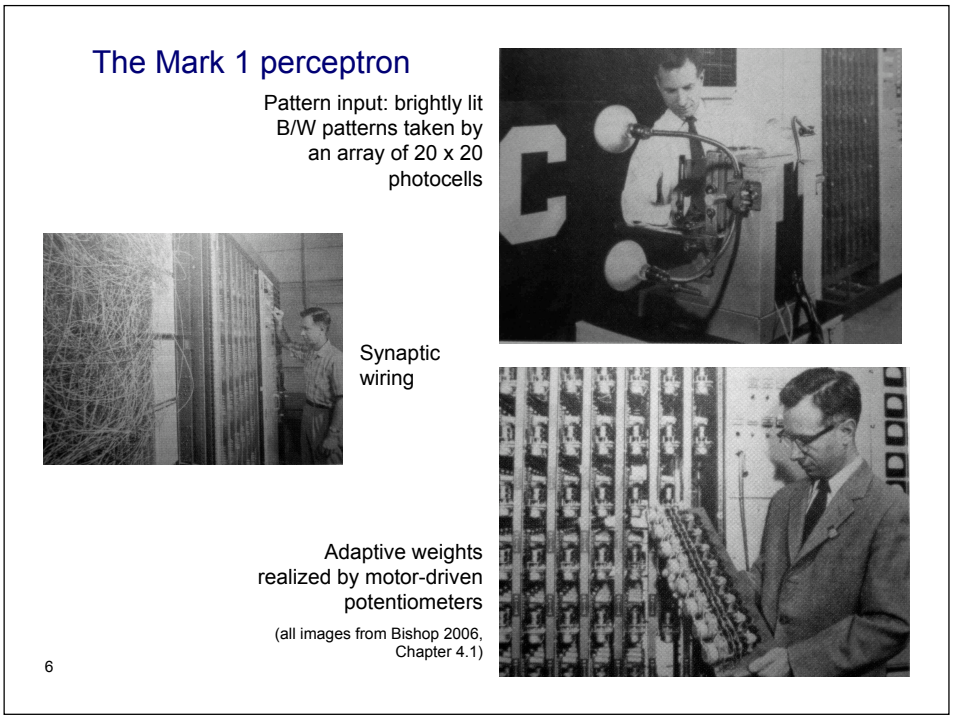
- Same task as we considered before in this lecture!
- Rosenblatt used high-contrast, clean input patterns, for example B&W images of letters A, B, C, ...



- Letters were read by 20 x 20 photocell array, giving 400-dim records
- Rosenblatt was a cognitive neuroscientist. Research motivation was mixture of understanding real brains and building brain-like machines

4





- Rosenblatt developed a series of perceptron implementations whose details differ. Slide 5 shows the schematic of one of them.
- Rosenblatt adopted some design principles that were known/hypothesized from human brains.
- A perceptron mimicks the brain's processing route from the retina to higher brain regions where recognition supposedly takes place.

- Information is passed in one direction from the simulated retina forward through processing layers. Specifically, there are no "feedback" processing paths back from higher processing layers to lower ones. This is the defining criterion for *feedforward* neural networks.
- Information propagates forward through the system through numerous distributed *processing units* which mirror biological neurons (or clusters of neurons, or brain regions).
- Retina "cells" assume real-valued *activation* valued corresponding to light intensities.
- In the perceptron, higher-layer processing units only take binary activation values ("on"/"off", or "excited"/"quiet", or simply 0 / 1)
- Information is propagated along *synaptic connection links*. These connections have different *strengthes*, also called *weights* (think of them as wires that can have different resistances).
- The learning is effected by adapting the connection weights, just like one believes learning in brains is effected.
- The structure of the perceptron was relatively simple (today's neural networks are much more complex). Specifically, there was only one layer of connections with adaptive weights. Today we would call this a *single-layer NN*.
- The first processing layer consisted of processing units which were each randomly connected to a few retina cells and would compute a random binary function of the retina cell's activations. We have learnt to consider these "associator units" as (random) feature extractors.
- In the next processing stage, the activations of the associator units are simply summed. Importantly, before summing these activation values are weighted by the synaptic strengths.
- In the final processing stage, the weighted sum of associator activations is passed through a *binary threshold* function ("response unit" in Rosenblatt's Perceptron parlance).
- Everything in this architecture is hardwired and fixed except the weights of connections between associator units and the summation unit. Those weights are *trainable*.
- First implementations by Rosenblatt on an IBM 704 programmable digital computer at the Cornell Aeronautical Laboratory.
- Rosenblatt went on to build an analog hardware implementation, the Mark 1 Perceptron. His rationale is very interesting:

  "As the number of connections in the network increases, ... the burden on a conventional digital computer soon becomes excessive, and it is anticipated that some of the models now under consideration may require actual construction before their capabilities can be fully explored. [...] The results of these [digital] programs, therefore, should be interpreted as indicating performances which might be expected from an "ideal" ... system, and not necessarily as representative of any particular engineering design. A Mark I perceptron, ... is expected to provide data on the performance of an actual physical system, which should be useful for comparative study." (Rosenblatt 1960)

- Today all practically used NNs are implemented on digital computers, like Rosenblatt's first realization on the IBM 704.
- However, there are current efforts (pushed among others by the large chip-manufacturing companies) to implement NNs on *analog VLSI* (Very Large-Scale

Integrated) microchips, much in agreement with Rosenblatt's rationale to build a non-digital, all-analog-hardware Perceptron.

Perceptron training, overview

Notation. Let a perceptron have k adaptive connection weights. We collect them in a k-dimensional *weight vector* $\mathbf{w}$. Let furthermore the k activation values of the associator units upon presentation of an image record $\mathbf{x}_i$ be assembled in a k-dimensional *activation vector* $\mathbf{x}_i^*$. Then the output of the perceptron is given by

$$y_i = f(\mathbf{w}'\mathbf{x}_i^*).$$

Learning task. Given

- a perceptron with k adaptive connection weights,
- a *training sample* $(\mathbf{x}_i, d_i)_{i=1,\ldots,N}$ of N labelled training instances, where $d_i \in \{0,1\}$,

find connection weights $\mathbf{w}$ such that the perceptron outputs the correct labels, that is

$$d_i = f(\mathbf{w}'\mathbf{x}_i^*) \text{ for all } i = 1,\ldots,N.$$

7

- It is common in mathematical notation (but not universally adopted) to write vectors as boldface lowercase symbols, like $\mathbf{w}$ for a weight vector. Note that by default a vector is assumed to be a column vector. To turn it into a row vector, we have to *transpose* it, for which a common notation is $\mathbf{w}'$. The transpose operation flips the orientation of a vector. Example:

$$\mathbf{w} = \begin{pmatrix} 0.2 \\ 1.0 \\ 0 \\ 3.1 \end{pmatrix}, \quad \mathbf{w}' = \begin{pmatrix} 0.2 & 1.0 & 0 & 3.1 \end{pmatrix}, \quad (\mathbf{w}')' = \begin{pmatrix} 0.2 \\ 1.0 \\ 0 \\ 3.1 \end{pmatrix}$$

- If two vectors $\mathbf{w}$, $\mathbf{x}$ of the same dimension are given, for instance

$$\mathbf{w} = \begin{pmatrix} 0.2 \\ 1.0 \\ 0 \\ 3.1 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} 0 \\ 2.0 \\ 1.0 \\ 30.2 \end{pmatrix},$$

a common operation is to compute the *inner product* of $\mathbf{w}$ and $\mathbf{x}$, written as

$$\mathbf{w}'\mathbf{x} = \begin{pmatrix} 0.2 & 1.0 & 0 & 3.1 \end{pmatrix} \begin{pmatrix} 0 \\ 2.0 \\ 1.0 \\ 30.2 \end{pmatrix} = 0.2*0 + 1.0*2.0 + 0*1.0 + 3.1*30.2.$$

- In our Peceptron, assume we have k associator units. Their output (given an input pattern $\mathbf{x}_i$ presented to the retina) can be written as a k-dimensional vector $\mathbf{x}_i^*$. Let the k connection weights sitting on the synaptic links between the associator units and the summation unit be noted as $\mathbf{w}$. Then the value of the summation unit (upon presentation of pattern $\mathbf{x}_i$) is $\mathbf{w}' \mathbf{x}_i^*$, and the final Perceptron output in the response unit is $f(\mathbf{w}' \mathbf{x}_i^*)$.
- The Perceptron outputs either the number 0 or the number 1. It can only perform a *binary* classification task where it has to distinguish two input types. The class labels are coded to be 0 or 1, for instance "healthy" = 0; "anaemic" = 1.
- The Perceptron did not output a hypothesis vector of class probabilities (like modern NNs do) but merely a strict class *decision*.

The perceptron learning rule

The learning task is solved by the following weight adaptation process:

- Present the training instances $\mathbf{x}_i$ to the perceptron in turn.
- Upon input $\mathbf{x}_i$ in the n-th adaptation step, where the current perceptron weights are $\mathbf{w}(n)$, compare the perceptron's output with the desired output d_i.
- Adapt weights $\mathbf{w}(n-1)$ from previous step to get new weights $\mathbf{w}(n)$ by the following rule:

Case 1: $d_i = f(\mathbf{w}'(n-1)\,\mathbf{x}_i^*)$, that is, correct classification of n-th training record with old weights $\mathbf{w}(n)$. Then, no change: $\mathbf{w}(n) = \mathbf{w}(n-1)$.

Case 2a: $d_i = 1 \neq 0 = f(\mathbf{w}'(n-1)\,\mathbf{x}_i^*)$, false classification of class-1 sample as class-0. Then adapt $\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{x}_i^*$.

Case 2b: $d_i = 0 \neq 1 = f(\mathbf{w}'(n-1)\,\mathbf{x}_i^*)$, false classification of class-0 sample as class-1. Then adapt $\mathbf{w}(n) = \mathbf{w}(n-1) - \mathbf{x}_i^*$.

8

The perceptron learning rule, comments

- If the perceptron is in principle capable of solving the given classification task, the learning rule will converge to a successful set of weights.
- It is an example of *supervised* learning: the correct (desired) outputs are available at training time.
- It is an example of an *iterative* weight adaptation rule. Weights are adapted incrementally, improving (hopefully) the performance at every update step.

9

The hype

A sniplet from the New York Times (*"New Navy Device Learns by Doing",* NYT July 8, 1958), after a press conference held by Rosenblatt at the US Office of Naval Research on July 7, 1958 (cited after Olazaran 1996):

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted.[17]

10

The shattering

1969: Marvin Minsky and Seymour Papert, pioneers of (symbolic) AI, publish *Perceptrons.*
In this book they point out, among other things, that the XOR function cannot be learnt by perceptrons.

The XOR function as a classification task	
input records	**desired output**
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

Consequence: NNs became disreputable and neural network research fell into a sleep for more than a decade.

11

- Hype waves have occurred several times in the history of Artificial Intelligence and Machine Learning. One that I remember well because I was doing my Phd during that phase was the hype surrounding *expert systems* in the late 1980's, early 1990's. Expert system startups popped up like mushrooms and sold expert system development SW at fantasy prices, and graduate programs centered on them were created. Then it became clear that human experts could not easily be replaced by machines and the hype deflated. Today expert system maths and programming

techniques are just one tool among others in the standard repertoire of AI programming, and they aren't called expert systems any more.

- The current screaming madness about *deep learning* may actually be *not* just another hype. A remarkable difference to earlier hypes: it actually functions, comes close to human performance, and it is started to become economically exploited in a large-scale way by Google, Microsoft et al.

1969: Minsky and Papert point out the obvious.

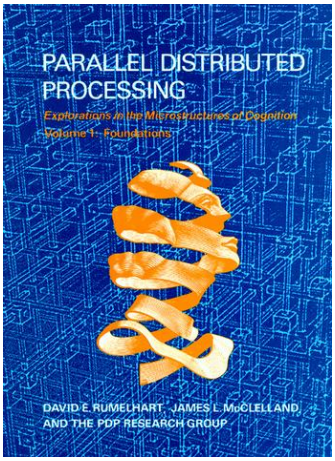
1986, -- 17 years later...

Neural networks reappear with a flourish:

Rumelhart, D.E. and McClelland, J.L. (eds.): *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Vol 1, MIT Press (still in print!)

12

The PDP bible



- An edited collection of (long, foundational, landmark) articles
- One chapter (by Rumelhart, Hinton, Williams) the *backpropagation* learning algorithm for NNs is introduced
- This algorithm unleashes the powers of NNs.

13

The backpropagation algorithm: impact

- Like the Perceptron learning rule, it's an iterative adaptation scheme for neural networks.
- It works for many learning tasks (not only classification), and for a large variety of network architectures.
- In principle, *every* learning task becomes solvable (though computations may become expensive, or numerically difficult). Example: online language translation.
- Soon after the PDP book, the "backprop" + NN grew into a standard method for modeling nonlinear real-world systems. The complexity of tasks that were solved grew slowly over the years.
- Since a few years, achievable performance levels have been exploding. The good-oldfashioned "backprop" + NN combination is now called *deep learning.*
- We will learn about all this in the remainder of this course.

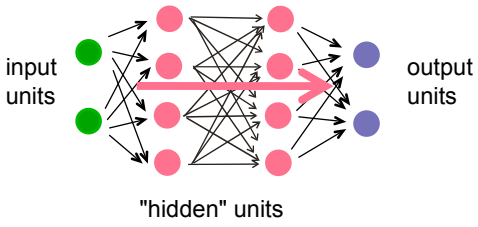
14

- Already before the publication of the PDP book, neural network research had been resumed in an inconspicuous but tenacious manner.
- Partly the renaissance of NN research came from theoretical physics who applied their mathematical methods to models of *associative memory* in neural networks, and was at first unrelated to machine learning.
- There have always been several motivations and subcommunities in NN research, among them the following:
    - Machine learning researchers wanted to create practically useful blackbox models.
    - Theoretical physicists (some employed in neuroscience departments) wanted to understand basic (and hopefully, simple-to-state) principles of information processing in neural systems (example: http://en.wikipedia.org/wiki/Hopfield_network)
    - Neuroscientists proper wanted to understand how, exactly, information is processed in concrete, complex biological brain systems which were modeled at considerable biological detail.
    - Cognitive scientists, linguists and AI researchers wanted abstract models of "conceptual-level", "higher", "intelligent" information processing at the symbolic level (school of *connectionism*, http://en.wikipedia.org/wiki/Connectionism). The PDP bible largely came from that community.
- In this course we deal only with the first strand of work, and within that, only with feedforward neural networks.

Multilayer perceptrons: structure

MLP - outline

- Multilayer perceptrons (MLPs) differ from the original perceptron in 4 important ways:
  1. they have *several* layers of adaptive synaptic connections
  2. all synaptic connections have adaptive weights
  3. the processing units output *continuous* activations
  4. there can be many output units

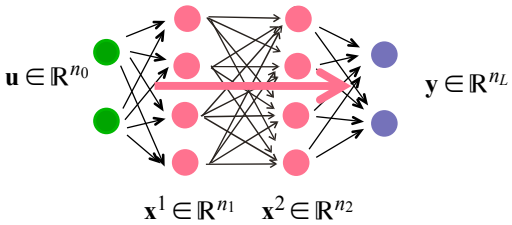


- An MLP is a general, numerical input-to-output transformation device.

16

- Multilayer perceptrons (MLPs) have evolved (quickly) from the original perceptron.
- On the one hand, they are simpler: all processing units are of the same type (in the perceptron, there were several different types)
- On the other hand, they are (**much**) more general and powerful.
- The connection to biology is only a faint echo. MLPs are mainly taken as computational tools for ML applications.
- Other common name: *feedforward neural networks*.

MLP – some general characteristics and notation



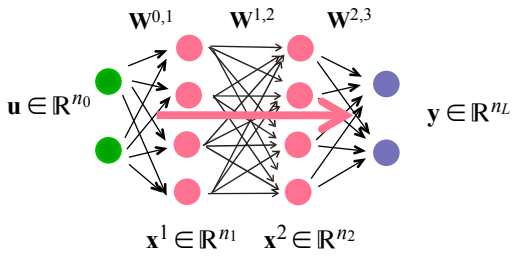
Shown: an L = 3 layer MLP

- n_0 input units read-in a record $\mathbf{u}$ of n_0 numerical (real-valued) numbers.
- The input "activation" values are fed forward through some layers of *hidden* units, until activations arrive at the n_L output units, yielding an n_L-dimensional output vector $\mathbf{y}$.
- It is customary to count layers of connections, not of units. An L-layer MLP has L connection layers, and L-1 hidden layers of units.
- The activations of units hidden layers 1, 2, ... are collected in hidden state vectors $\mathbf{x}^1$, $\mathbf{x}^2$,... of dimensions n_1, n_2, ...

17

- The mathematical symbol $\mathbb{R}^k$ denotes the k-dimensional Euclidean space. $\mathbb{R}^1$ is just the real line, $\mathbb{R}^2$ is the 2-dimensional Cartesion coordinate system, and $\mathbb{R}^3$ is the familiar 3-dimensional "space". For larger dimensions k we lack geometric intuition. Seen as sets, $\mathbb{R}^1$ is the set of all real numbers, $\mathbb{R}^2$ is the set of all pairs of real numbers – or equivalently, of all 2-dimensional real-valued vectors, etc.
- In practically used MLPs, the dimensions of the input layer, the hidden layers, and the output layer vary largely. They can grow quite large or be very small. Examples:
    - MLPs processing image input need as many input units as the images have pixels (times 3 if the image is in color).
    - State-of-the-art MLPs for complex tasks in pattern recognition have hidden layer sizes in the range of a few to many hundreds.
    - In speech recognition networks, output units produce probability hypotheses about which word has been uttered. There must be as many output units as the considered *lexicon* has words. For *continuous large-vocabulary speech recognition*, this may mean 10,000 – 50,000 output units.

MLP – more characteristics and notation

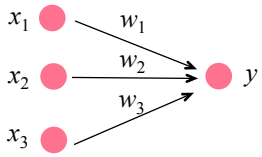


- All connections are *weighted* (and weights can be adapted in learning). Weights are real numbers and can be positive or negative or zero.
- Usually there is *total connectivity* between two layers: if feeding layer has n_i units and receiving layer has n_{i+1} units, there are $n_i\, n_{i+1}$ connection weights between these two layers.
- The weights between layers i and i+1 are collected in a weight matrix $\mathbf{W}^{i+1,i}$ of size $n_{i+1} \times n_i$.
- The first weights $\mathbf{W}^{1,0}$ are sometimes called *input weights,* the last weights $\mathbf{W}^{L,L-1}$ *output weights.*

18

- A weight matrix $\mathbf{W}^{i+1,i}$ of size $n_{i+1} \times n_i$ has n_i columns and n_{i+1} rows. The matrix element sitting in row k and column l is denoted by $\mathbf{W}^{i,i+1}(k, l)$.
- Storing all of the structure information of an MLP means to store all the weight matrices. This may lead to big data chunks. Consider a 3-layer MLP with 1000 units in each layer. This MLP needs 3 * 1000 * 1000 = 3 Mio parameters to be fully specified.

MLP – what happens in a processing unit?



Shown: a processing unit with activation y which receives input from three units with activations x_1, x_2, x_3 along connections with weights w_1, w_2, w_3.

- The activation y of a unit is computed in two steps:
  1. compute the connection-weighted sum of activations of units that feed into the unit,
  2. pass this weighted sum through an *activation function* f:

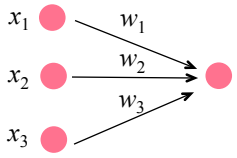
$$y = f(w_1 x_1 + w_2 x_2 + w_3 x_3)$$

- Conveniently and more generally written in matrix notation for an entire layer:

$$\mathbf{x}^{i+1} = f(\mathbf{W}^{i+1,i}\,\mathbf{x}^{i})$$

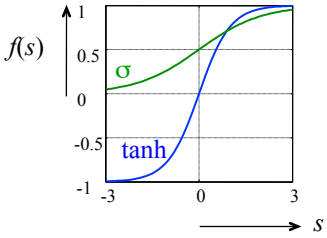
19

The activation function f



$$y = f(w_1 x_1 + w_2 x_2 + w_3 x_3)$$

- In MLPs, the activation function f is traditionally taken to be one of the following two:
  1. the logistic sigmoid σ
     or
  2. the tanh (tangens hyperbolicus)



20

- The full math formulas for these activation functions are

$$\sigma(s) = \frac{1}{1+e^{-s}} \qquad \tanh(s) = \frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$$

- The two are closely related (each is a linear transform of the other), and which one is used is very much a matter of taste.

- The activation functions σ and tanh both have "sigmoid" ("S-shaped") form. Both are symmetric around $s = 0$, and *saturate* for very large negative or positive arguments. The logistic sigmoid has a value range between 0 and +1, the tanh between –1 and +1. One also says that the activation functions *squash* their summed input into the respective value range.
- In the last two years or so, another activation function has been increasingly often used in "deep learning" approaches, the *rectifier* function:

$$r(s) = \begin{cases} s, & \text{if } s > 0 \\ 0, & \text{if } s \le 0 \end{cases}$$

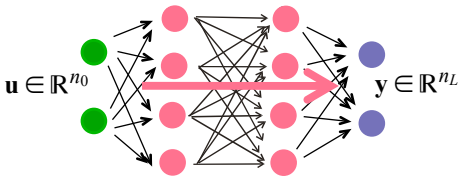
The rectifier function has less pleasing mathematical properties than the logistic sigmoid or the tanh, but it (and its derivative) can be computed much more cheaply. This becomes an important aspect when the MLPs are very large.

- The logistic sigmoid has a biological interpretation. The idea is that the activation (interpreted as firing frequency) of a biological neuron cannot be negative, and is bounded from above when the neuron saturates. The entire biological processing done by a unit (a neuron) is modeled as a two-stage process where the neuron's *potential* (voltage level of the cell body) is determined by the summed activation of all incoming synaptic feeds, and then from this potential the neuron generates a series of electric pulses (it "fires spikes") whose frequency is understood to give the activation x.
- For certain mathematical and practical reasons, the output units often have an activation function that differs from the activation functions of the hidden units. A frequent choice is to drop the activation function entirely for the output layer units; their activations then are just the weighted-summed activations of the last hidden layer.

Multilayer perceptrons: function

- It is important that the activation functions are *nonlinear*. If they were linear, the MLP as a whole would collapse to a linear transformation and could be replaced by an equivalent single-layer perceptron. That is why the activation function is sometimes simply called *the nonlinearity*.

Kneading input records



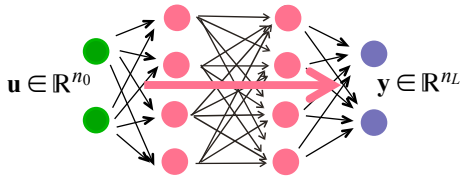
$$\mathbf{x}^{i+1} = f(\mathbf{W}^{i+1,i}\,\mathbf{x}^{i})$$

- An input record **u** is iteratively transformed into a sequence $\mathbf{x}^1, \mathbf{x}^2, \ldots$ of hidden state vectors, and finally into the output **y**.
- Each transformation step can be viewed as an "overlay, fold and squash" operation.
- Very complex $\mathbf{u} \rightarrow \mathbf{y}$ transformations become possible.

http://www.motherearthliving.com/cooking-methods/celebrity-chefs-pizza-dough-recipes.aspx

22

MLPs compute functions – *any* functions



- Any input record $\mathbf{u}$ of the right dimension can be fed into an MLP, yielding an output $\mathbf{y}$.
- Thus, an MLP M simply instantiates a function g_M: $\mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$.
- **Theorem** ("general approximation property"): any function g: $\mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$ can be approximated with arbitrarily small approximation error by some MLP.
- That is, given g and an admissible approximation error level ε, there exists an MLP M such that

$$\text{approx-error}(g, g_M) < \varepsilon.$$

23

- In fact, the general approximation theorem holds even if only two-layer MLPs are admitted. However, approximating complex functions by only two-layer MLPs leads to inordinately large MLPs. In practice, approximating complex functions mandates MLPs with many layers – so called *deep* networks.
- To get an idea what a "very complex function" might be, think of a function that takes 500 x 500 pixel photographic images as input and as output returns a vector made of 10,000 integers, corresponding to how many objects out of a choice of 10,000 object types are visible in the image.
- The general approximation theorem (of which there are many variants and different proofs) requires a formal way to measure the similarity of two functions g_1, g_2. This can be done in various ways. A popular choice is to use the mean squared distance. For the simplest case of one-dimensional functions mapping the unit inverval [0,1] to $\mathbb{R}$, that is g_1, g_2: [0, 1] $\rightarrow$ $\mathbb{R}$, the mean squared distance is

$$\text{approx} - \text{error}(g_1, g_2) = \int_0^1 (g_1(x) - g_2(x))^2 dx$$

Classification as function approximation

- Earlier in this lecture we considered classification tasks.
- For example, inputs were grayscale value vectors coding 15 x 16 pixel images of digits, outputs were hypothesis vectors containing probabilities for the 10 digit classes.
- Solving this classification task means just to find a good function $g: \mathbb{R}^{240} \rightarrow \mathbb{R}^{10}$.
- Classification is just a special case of function approximation.
- Function approximation tasks are often referred to as *regression* tasks.

24

Summary

- Neural networks have a 50-year history and were pioneered by Rosenblatt whose Perceptron aimed a modeling biological image classification.
- In today's ML world, multi-layer perceptrons are the "workhorses" for function approximation tasks.
- MLPs nonlinearly and stagewise transform input records into arbitrarily transfigured outputs. They are general approximators for numerical functions $g: \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$.
- Coming next: How to actually *find* an MLP that approximates a given target function – the *learning* part of this story.

25

Handout 2: Machine learning as modeling

What's that mean, a "model"?

http://www.redicecreations.com/article.php?id=18006

- A (scientific or engineering) **model** is a *formal representation* of some coherent collection of real-life phenomena.
- A **formal representation** can be more or less "formal", e.g.
  - mathematical formulas (physics)
  - simulation algorithms (engineering, computer games)
  - plain-English definitions and descriptions of mechanisms (sociology, psychology).

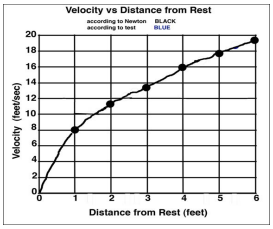
$$F = G\frac{m_1\, m_2}{r^2}$$

3

The classical modeling cycle of science

piece of reality

observation procedures



https://people.rit.edu/andpph/text-newton-right-wrong.html

observation data

detect regularities

$$F = G\frac{m_1\, m_2}{r^2}$$

model

make predictions and test in experiments = novel observation procedures

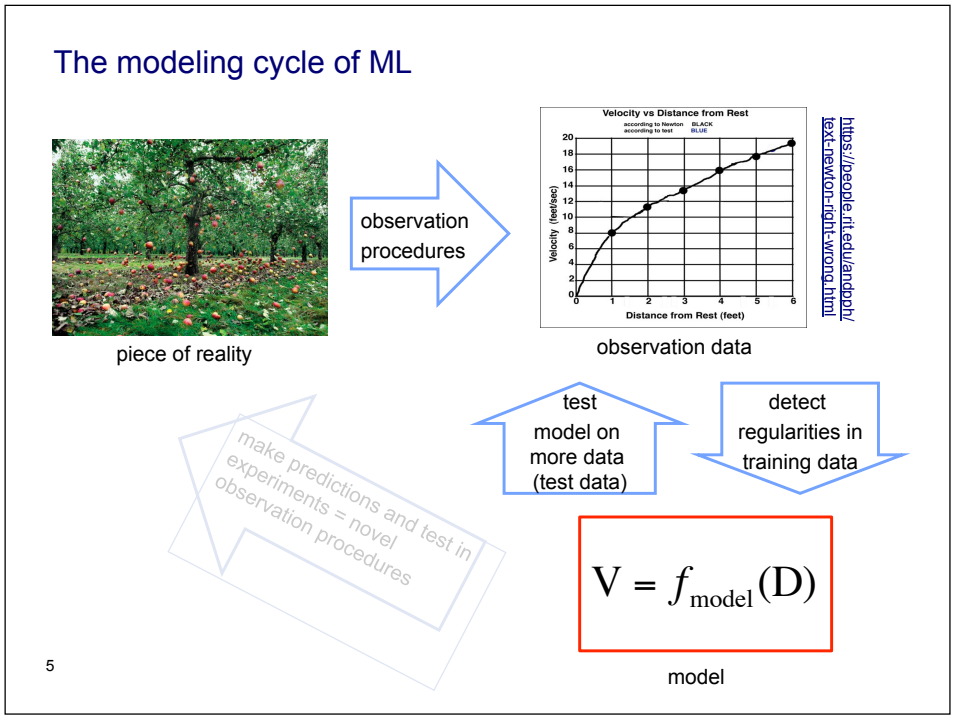
4

- There is an almost dogmatic way of how research in the natural sciences should be carried out:

  1. Delineate a coherent **set of real-life phenomena** (for example: motion of stellar bodies (Newton), mating behavior of ravens (ethology)...).

2. Define **measurement procedures** for some aspects of these phenomena, and give scientific names to these **measurables** (e.g. *mass*, *velocity* or *mating call*).
3. Observe / measure these phenomena under systematically varied conditions, obtaining empirical observation **data**.
4. Generate **hypotheses** about **regularities** in these data ("laws", "symmetries", "rules"...)
5. Formalize these regularities in a suitable **formalism** (or at least, in precise plain English).
6. The formal specification of the regularities is a **model**.
7. While the model should accurately describe regularities in the available data, this does not prove the model is "true". Reality might not be captured by the model even though the currently available data agree with the model. The classical example is the model of classical mechanics which goes back to Newton's era. It agreed with available data for 2 centuries, but then this model was challenged on theoretical grounds (relativity theory) and refined measurement techniques yielded data (at high velocities and/or large interstellar distances) which did not agree with the model of classical mechanics.
8. Expressed in a nutshell: a scientific model of a piece of reality can never be proven correct. It can only be **falsified** by new data.
9. A model should be as succinct (simple) as possible: "Occam's razor" http://en.wikipedia.org/wiki/Occam%27s_razor . It should **reduce** the variability of phenomena/data by distilling underlying mechanisms that give rise to the variability. It should **explain** reality.
10. The standard scientific method uses models to make **predictions** about the outcomes of future **experiments** and tests the model against the outcome of those. If new data from new experiments don't agree with the current model, it has to be modified / extended / refined.

- Finding a "suitable formalism" that is capable to represent the assumed regularities is not always easy. In the case of Newton, first an entire branch of mathematics had to be created – calculus – before the stipulated regularities could be appropriately formalized. Progress in ML very much hinges on developing new mathematical formalisms, too.
- This standard paradigm of the empirical scientific method is widely accepted but has not remained unchallenged. The further one goes away from physics, the more problems arise. In psychology and the social sciences, the standard paradigm is only partially dominating, and the historical sciences function entirely differently.
- Check out http://en.wikipedia.org/wiki/Scientific_method if you want to learn more about the big picture.

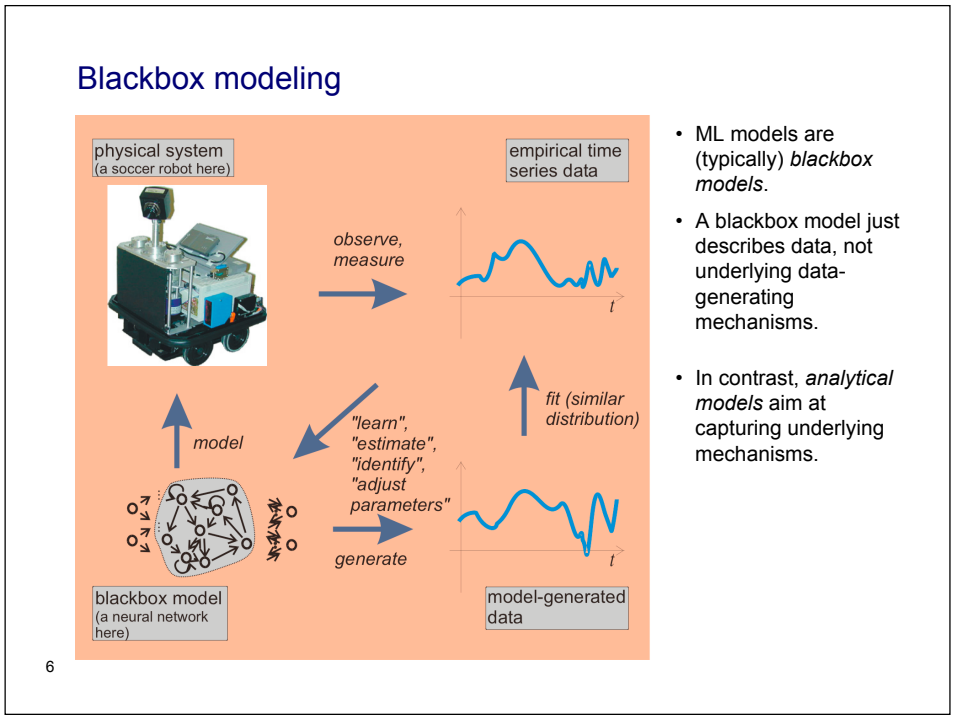


- The models in ML are construed in a different way.
- Like in the scientific method, ML models are designed to capture the variability in observational data, by exploiting regularities / symmetries / redundancies.
- Unlike in the scientfic method, ML models (usually) do not aim at *explaining* reality. All they need to achieve is to **describe** the data. Expressed in false modesty, ML models just have to be good at **data fitting**.
- Because a good ML model doesn't need to *explain* reality, just *describe* data, ML models don't need to be simple or transparent or intelligible. They just must be accurate. Thus, while scientific models are typically expressed in the form of 1-liner mathematical laws, an ML model may take the shape of a computer algorithm that has millions of parameters.
- But even ML models must hold strong in the face of **tests**. But the tests are different from the tests that scientific models are subjected to. In the scientific method, tests are experiments that try to extract novel behavior from reality – that is, a scientific experiment instantiates a *novel observation procedure*. Tests for ML models, in contrast, are carried out on new data that are obtained from reality re-using the *same observation procedure* as were used for the training data.

- Examples:

Scientific method	**Machine learning**
The scientific model of classical mechanics was originally derived from planetary body motion. Later (allegedly) tested against falling apples (still holds) and particles in ring accelerators (fails).	A ML model of planetary motion would extract from celestial observation tables a formula describing the planet's orbits. It would be tested only against further data from planetary motion.
A scientific model of a chemical reaction and its dependance on pressure, temperature and concentrations can be induced from laboratory observations. The model can later be used to design and control chemical manufacturing plants – and a variety of those.	A ML model useful for controlling a chemical manufacturing plant has to be learnt from data from that very plant, and can only be used for that single plant (or clones).
A scientific model of speech utterances is close to impossible to obtain because the mechanism underlying this phenomenon are so vastly complex – involving a brain, the vocal tract, a linguistic community, and sound physics. No insightfully simple-enough "law" seems possible.	A ML model for human speech is trained on speech recordings made from hundreds of speakers. It is then tested against recordings of other speakers. If the variety of the speaker population used for training was large enough, the model will perform well on test data.

- In a nutshell:
  - Scientific models should be small and simple, and they should generalize to phenomena observed in new ways.
  - ML models may be large and complex, and they should generalize to new data obtained in the same way as the training data.



- Blackbox models seem inherently weaker than analytical models. Why bother about them?
    - When the systems one wants to model are very complex, analytical models are practically impossible to get.
    - Analytical modeling requires *insight* from a modeler. Blackbox modeling can be done, in principle, by automated routines: ML learning algorithms.
- Most but not all of ML is about blackbox modeling. Sometimes however even in ML one wants to have *interpretable* models – models that help to *understand* the modelled reality. This usually leads to using logic-based representation formalisms. The model takes the form of a collection of human-readable *facts* and *rules*. Two important approaches in ML that aim in this direction are *fuzzy logic* and *inductive logic programming* (both have Wikipedia entries). We will not consider these in this lecture.

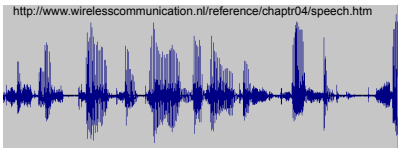
ML models as probability distributions

Randomness everywhere

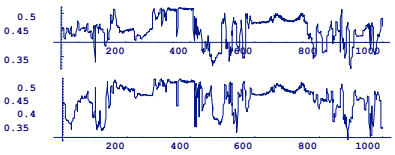
http://ftp.ics.uci.edu/pub/ml-repos/machine-learning-databases/mfeat/mfeat-pix

handwritten, normalized digits "0" and "1"

http://www.wirelesscommunication.nl/reference/chaptr04/speech.htm



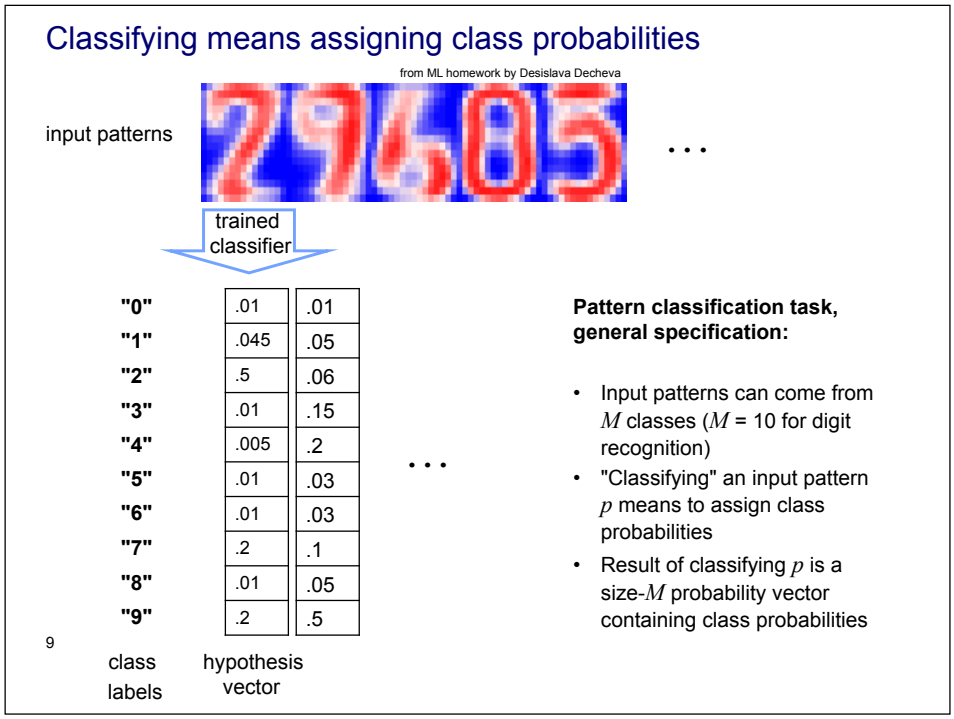
speech signal



robot wheel speed measurements

- Real-life data are always "noisy", have some random components
- Main source of randomness: **ignorance**.
    - Data are generated by the interplay of a very large number of mechanisms, but are observed only by a small number of measurables → loss of information → apparent randomness.
- Practically useful ML models must model the random variability *and* the regularities in the data.

8



- We will concentrate on *pattern classification tasks* from now on. This is the most widely encountered application type of ML, and all basic themes and challenges of ML can be demonstrated with it.
- "Patterns" can be virtually anything:
  - Images
  - Videos
  - Sound recordings
  - Customer profiles
  - Texts
  - etc.
- In a classification task, one assumes that the patterns come from a finite number M of classes. Examples:

Patterns	**Classes**	M
Digit pictures	"0", ..., "9"	10
Single-word recordings	All words from a *lexicon*	10,000 – 100,000
Customer profiles	"creditworthy", "not creditworthy"	2

Measurables, records

Measurable: A measurable is a property of a real-world system that can be assigned *values* through some observation procedure. Examples:

real-world property	possible values	value data type
gender	"male", "female"	symbolic
nr of children	0, 1, 2, 3, ...	integer, "discrete"
velocity	0.0 -- $V_{\max}$	real, "continuous"

Record: A record is a collection of observation values obtained from an *instance* of a system. Think of records as rows in an Excel table. Examples:

instance	measurables taken	record	record type	dimension
Paul	(gender, nr of children)	("male", 2)	mixed symbolic & integer	2
a falling apple	(velocity, weight, distance to ground)	(4.23, 125.4, 1.21)	real	3

10

Samples, labelled samples

Sample: A sample is a collection of records. Think of a sample as an entire Excel table. Example:

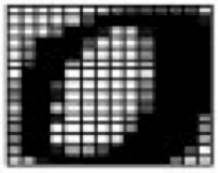
instance	measurable 1: gender	measurable 2: nr of children
Paul	"male"	2
Vivian	"female"	0
Stephen	"male"	1
...	...	...

Labelled sample: A labelled sample is sample where each record in addition has a class label. Example of a sample with $M = 2$ classes "happy" and "unhappy":

instance	gender	nr children	class label
Paul	"male"	2	"happy"
Vivian	"female"	0	"happy"
Stephen	"male"	1	"unhappy"
...	...	...	...

11

Digit pics as records in a sample



a single digit image (instance)

a sample of size 100

- Our digit images are a 15 x 16 pixel images, where each pixel has a grayscale value ranging in the real interval $[0, 1]$.
- There are 240 pixels in total.
- Each pixel is a measurable, with possible values in $[0, 1]$.
- A digit image thus is a real-valued record of dimension 240.
- In math terms, such a record is a real-valued *vector* of dimension 240.
- The possible class labels are "0" ... "9", there are M = 10 classes.
- Taking (preprocessed) photos of handwritten digits from randomly picked letter addresses, and labelling them with the correctly identified classes, gives a labelled sample $(\mathbf{x}_i, d_i)_{i=1,\ldots,100}$, say of size 100.
- The $\mathbf{x}_i$ are vectors of dimension 240.

12

- The collection of records in a samples should be collected "at random". In statistics one speaks of a "random sample". In empirical psychology and opinion research, a great effort goes into assuring that a sample taken for a scientific study is indeed random (and not "biased").
- In formal notation, a labelled sample with N records is written as

$$(\mathbf{x}_i, d_i)_{i=1,\ldots,N}$$

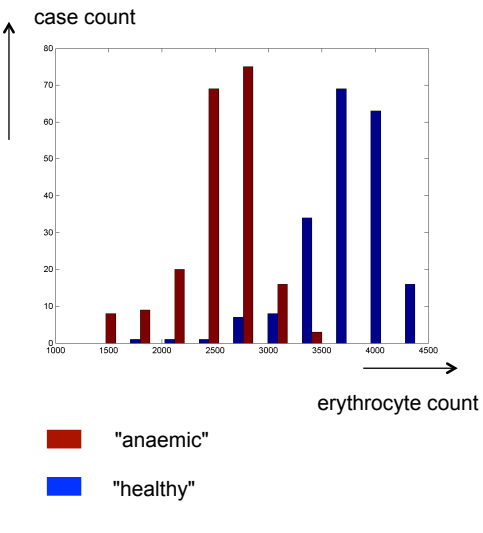
where the $\mathbf{x}_i$ denote the records and the d_i the class labels.

- We use boldface lowercase symbols like $\mathbf{x}$ to denote vectors. I chose letter "d" for the labels in allusion to "***d***esired" output values.

Before we deal with digit classification, let's briefly check out a simpler task

Task: classify patients into "anaemic" vs. "healthy"

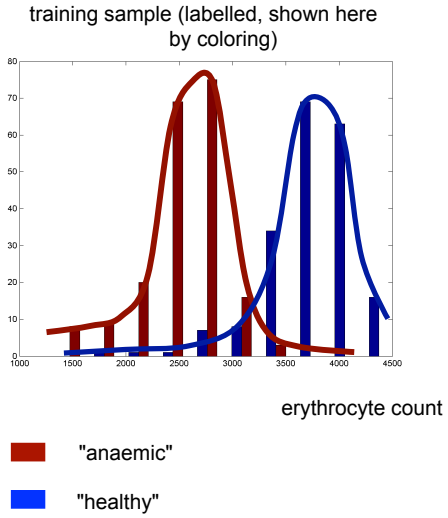
- only measurable: erythrocyte count (1-dimensional records)
- value range of measurable: integers 0 – 10,000
- a labelled sample of 400 patients (instances) $(x_i, d_i)_{i=1,\ldots,400}$
- each x_i is a single number (erythrocyte count), each d_i is either "anaemic" or "healthy"
- labelled sample can be visualized by 2-color histogram
- For visualization, the x_i values were *binned* into bins comprising a value range of 300



13

"Training" a patient classifier

- Step 1: approximate the two histograms by smooth curves
- That's it! model is ready for use.



14

- Step 1, "approximate histogram counts by smooth curves" is the point where ML algorithms must be used. It is not at all straightforward in general to find such nicely fitting but smooth curves!
- There are many ways how such fitting curves can be mathematically represented. Different ML techniques build on different formalisms for representing such curves. Examples of formalisms:
    - The curves can be represented by simple formulas. A very widespread formula is to combine the desired curve by adding weighted Gaussian "bell-shaped" curves. This gives *mixture of Gaussians* representations:

$$f_i(x) = \sum_{j=1}^{K} \alpha_j \frac{1}{\sigma_j \sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_j}{\sigma_j}\right)^2\right)$$

      where x is the measurement value (here we only consider 1-dimensional records), f_i is the function that represents the fitting curve for class i training instances. The curve is made from summing K different Gaussian bell-curves which are centered at the mean values μ_j and have standard deviations (= width of the bell shape) σ_j. These curve components are added up with weights α_j. Usually one requires that the α_j sum to a value of 1. Then the function f_i has an area of 1 underneath it and it becomes a probability density function (pdf). Mixtures of Gaussians have many pleasant mathematical properties, and well-understood learning algorithms for finding best-fitting such $f_i(x)$ given data frequency counts (histograms) are available.
    - The curves can also become realized through complex representations called *neural networks*. We will get to know them better later in this lecture.

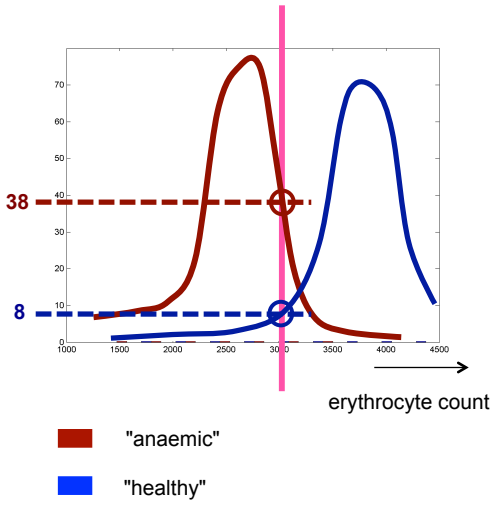
Using the trained patient classifier

- A new patient arrives with an erythrocyte count of 3000.
- Step 1: find the frequency score values in the smooth curves for the two classes (here: **38** and **8**)
- Step 2: Normalize these *class-conditional* scores to unit sum:

  $(38, 8) \rightarrow (38, 8) / (38 + 8)$

  $\approx (.83, .17)$

- $(.83, .17)$ is a probability vector: the result of the classification of the new patient.

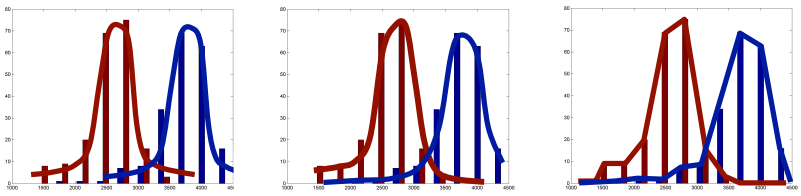


15

- Once the fitting curves (= "the model") are available, they can be exploited on new incoming data records ("test cases") for classification. All that needs to be done to obtain a class hypothesis vector for a new record $\mathbf{x}_{\text{test}}$ is to compute the M class-conditional values $f_1(\mathbf{x}_{\text{test}}), \ldots, f_M(\mathbf{x}_{\text{test}})$, normalize them to sum to 1, and one has obtained the classification hypothesis vector.

Over- and underfitting

- All the "learning" was done in finding the "smoothed" fitting curves. But how should the fitting be done "correctly"?



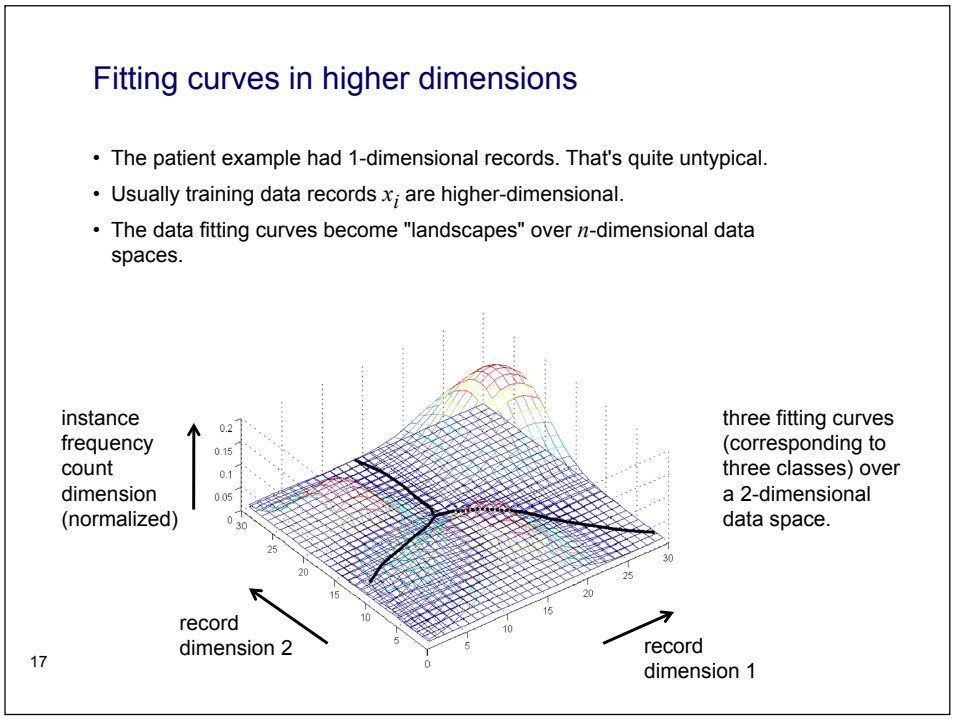
too smooth, too regular and symmetric?
underfitting??

just about right? *but if so, -- why*??

too jittery, too close to training data?
overfitting??

16

- The problem of over- vs. underfitting is the nemesis of ML. In our patient classification example, it doesn't seem too frightening, but when data get more complex, the nastiness of this issue explodes and becomes absolutely critical for the success vs. failure of a ML modeling attempt.
- Another name for the over- vs. underfitting problem is *bias-variance* dilemma. This name comes from a deeper mathematical analysis of the overfitting challenge. In intuitive terms,
    - "bias" means prior assumptions about the smoothness of the fitting curves. If during the learning (= curve fitting) the learning algorithm has been designed to favor smooth fitting curves, these assumptions become moulded into the finally resulting solution – it is *biased* toward smoothness.
    - "Variance" comes from fitting training data scores. The more the learning algorithm lacks smoothness assumptions, the more the fitting curves cling to the exact training data values. If the learning were to be repeated with a new set of training data, an unbiased (unsmoothed) solution would replicate the new random jitter in the new dataset. That is, in repeated triaining runs (using new training data each time), the obtained solutions would be highly varying one from the other.



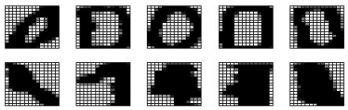
- Unfortunately, one can only visualize fitting curve landscapes for record dimensions 1 and 2. For higher-dimensional data records, our visual imagination is lost. In fact, higher-dimensional "landscapes" have many un-intuitive properties and our low-dimensional intuitions tend to misguide us in important ways.
- Specifically, returning to the digits example, we have no good geometric intuitions about "landscapes" over 240-dimensional data spaces!

The curse of dimensionality

Back to the digits classification task...

Training a classifier as before in the patients example...

1. Bin the x-data.
   - original x-data: $[0,1]^{240}$
   - binning: partition $[0,1]^{240}$ into "voxels" of side length l.
2. Compute histogram (bin count) of training sample.



visualization of our 240-dimensional records

3. **Aaarrrghhh!!!!**
   - There are $(1/l)^{240}$ many bins!
   - Almost all bin counts will be zero!
   - How to estimate density curves from a few 1-counts in a vast space of zero bins???

Curse of dimensionality

For high-dimensional (here dim = 240) training data it is **impossible** to naively estimate densities, because data points are scattered unimaginably thinly (exponentially in dimension) in data space.

19

- "Curse of dimensionality" is indeed a technical term in ML and will be understood by everybody in the field.
- In a nutshell, this curse means: if data records are high-dimensional, the example points will be spread out over the high-dimensional data space unimaginably thinly. Most "voxels" in the abstract data space will not be "hit" by a training data point. One does not get anything resembling a "histogram". Fitting a "density curve" to such highly scattered, essentially isolated data points becomes a real problem.
- This is why machine learning engineers always want from their customers *more* training data. They can never get enough of them!
- Current amazing successes in ML (more about this later) arise, among other reasons, from the circumstance that with the most recent internet and database technologies, very large training datasets have become available. For instance, state-of-the-art ML models of English text are trained on *all of Wikipedia*.
- Still, even with the largest manageable training datasets, the curse of dimensionality is only slightly mitigated, not resolved.

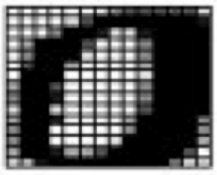
Three ways to fight the curse of dimensionality

1. **Get more training data.**
   - Is not always possible, is expensive, helps a little, and doesn't resolve the root cause of the problem.
2. **Reduce the dimension of data records.**
   - You may think of this as "data preprocessing".
   - Don't use the *raw data* as input for the classifier, but first compress the high-dim raw data records into a low-dim transform (examples next slide).
3. **"Regularize" the model learning.**
   - You may think of this as "implement curve smoothing" (more on slide 22)
   - Regularization at the same time helps against overfitting, and is most often discussed in that context.

20

- Techniques for dimension reduction play a large role in ML.
- They are naturally related to *data compression* methods which are relevant in other fields too.
- Abstractly, reducing high-dimensional data vectors $\mathbf{x}_i$ into low-dimensional data vectors $\mathbf{x}_i$* means to apply a dimension-reducing function F: $\mathbf{x}_i^* = F(\mathbf{x}_i)$.
- The curse of dimensionality and the overfitting problem are related (though not the same thing). The higher-dimensional the data records and the more thinly scattered the training data points, the more one is likely to overfit the data.

Data dimension reduction by feature extraction



- A raw digit record is a vector $\mathbf{x}$ of 240 grayscale values.
- An individual pixel value carries little information about the digit class.
- Idea of preprocessing by *feature extraction*: distil from the 240-dim vectors new quantities ("features") that are instructive about the digit classification.

Examples

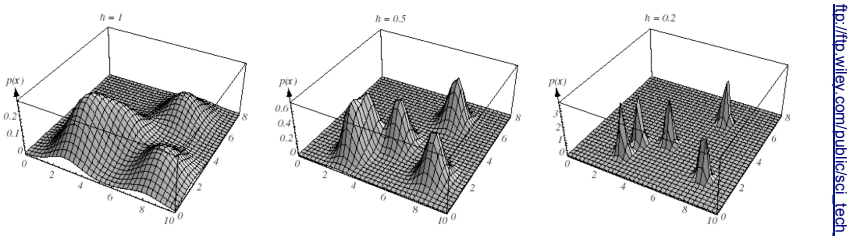
- Feature $F_1(\mathbf{x})$: mean grayscale value of all image pixels
- Feature $F_2(\mathbf{x})$: = 1 if the center of the image is white else = 0
- Feature $F_3(\mathbf{x})$: degree of "overlap" of x with a clean *prototype* instance of a "0".
- Formally a feature is a function F: $\mathbb{R}^{240} \rightarrow \mathbb{R}$.
- Using m features F_i in combination transforms the 240-dim record $\mathbf{x}$ to a merely m-dim *feature vector* $\mathbf{x}^* = (F_1(\mathbf{x}), ..., F_m(\mathbf{x}))$

21

- Transforming high-dim records $\mathbf{x}_i$ into low-dim feature vectors $\mathbf{x}_i$* is a widely used approach to combat the curse of dimensionality.
- The extracted features can be viewed as another, "quality-improved", "insightful", "informative" kind of measurables.
- Many early applications of ML techniques used "hand-designed" features that were informed by human expert insight. In particular this was true for early digit and text recognition software, for speech recognition systems, and object recognition from photos applications.
- Today, effective features are often *discovered* in automated ways without human intervention and their discovery becomes part of the machine learning modeling process.

Regularization

- A vast field, crucial for ML performance, here we only get a glimpse.
- Example: fitting a "probability landscape" over 5 scattered 2-dimensional records:



strong regularization | medium regularization | weak regularization

Duda, Hart & Stork
ftp://ftp.wiley.com/public/sci_tech_med/pattern/)

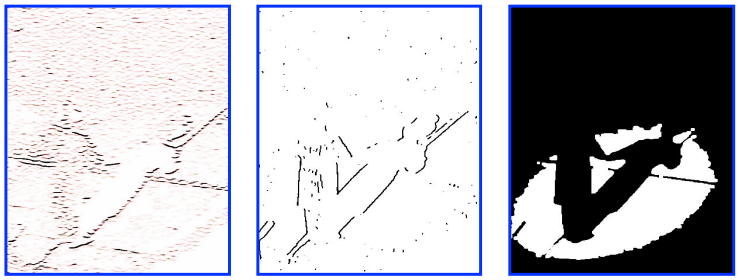
- The strength of regularization can be adjusted continuously.
- Determining the best degree of regularization is typically done by trial-and-error (expensive!)

22

- The importance of regularization can't be over-emphasized.
- Current breakthrough successes of ML hinged on (among other factors) on new clever regularization methods.

Human vs. Machine learning

How machines see



Typical task: determine object of given class (e.g., victory sign "V" or arms of a clock) in image

Processing: extract characteristic "features" (textures, lines, areas, ...) and classify image based on the information

Learning:

- find out what "features" exist and are important and how they should be combined
- learn by going through large numbers of already classified examples

4

- Image recognition is a major application domain of ML
- Example applications:
    - Text recognition (recognizing written letters and words, document reading)
    - Organizing large collections of photos (e.g. Google photo collections for registered users)
    - Robotics: classify navigation obstacles or graspable objects from video input
    - Medical: classify tissue from microscopic slice images as tumor vs. non-tumor
    - Military: detecting and identifying tanks in aerial or satellite photos
- Features come in an unbounded variety, new feature types can always be defined "on the fly" by ML algorithms. Standard types of *low-level* features:
    - Lines and gratings
    - Colors
    - Textures
    - Brightness
    - Orientations (of lines or gratings)
    - Localization (*where* in the image is something)
    - Simple shapes (circles, dots, crosses...)
- Features come in *hierarchies*. From low-level features one may compose, on the next higher level of a feature hierarchy, for instance parts-of-object features, like
    - Hands (part of humans)
    - Eyes (part of faces)
    - Tyres (part of cars)...

Image taken from:
www.addaclevenger.org/show/tophat.jpg

5

How humans see...

How many people are on this picture?

How many people are within 200 meters of this man?

What's this?

How does the hat smell?

Where is all the edge and contour detection gone...?

- Human vision is determined by a large number of factors. It is not a simple, single-purpose "algorithm". Some co-determinants that need to be understood if one wants a full picture of human vision:
    - body growth, brain development
    - motion control
    - exploration, curiosity, play
    - creativity
    - social interaction
    - drill and exercise and rote learning
    - reward and punishment, pleasure and pain
    - evolution
    - dreaming
    - remembering and forgetting
    - ...

- A scientific study of human vision is inherently multidisciplinary. For a full picture one would have to factor in external conditions like
    - the universe, the earth, the atmosphere, water, food, caves (physics, geology, ecology...)
    - body, brain, sensor & motor architecture (biology)
    - physiology and neurophysiology (neuroscience)
    - evolution (evolution theory)
    - other people, living (sociology, psychology)
    - other people, long dead, and their traditions (history)
    - machines, tools, buildings, toys (engineering)
    - words and sentences (linguistics)
    - concepts and meanings (philosophy, cognitive science)
    - letters and books and schools (pedagogics)

Human Information Processing

- As many different "tasks" as there are moments in life
- Information processing and learning: almost same thing ("one cannot step into the same river twice", Heraclites)

7 P. A. Renoir, Le bal au Moulin de la Galette
http://www.lannaronca.it/Renoir%2001%20p.htm

Machine Information Processing

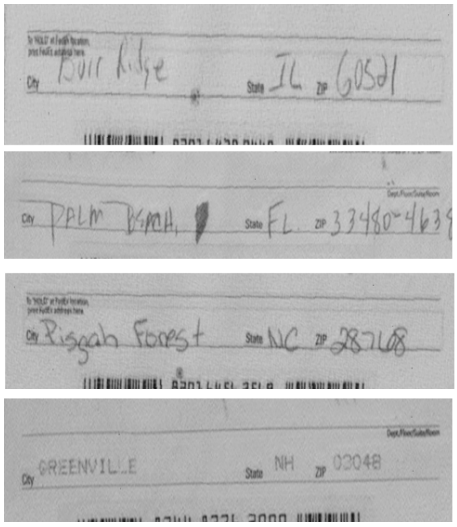
- A few specific tasks, mainly object detection and classification
- Processing and learning are typically separated ("learning phase" vs. "exploitation phase")

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\,\lambda\,\varepsilon(n)\,\mathbf{x}(n)$$

- Human learning and information processing arise from, and affect, the *entire human system* at *all times*: it's just *life*.
- Machine learning is a set of specific data engineering techniques for designing *task-specific* algorithms.
- Machine learning is immensely more constrained than human learning
- On most but not all tasks, machine learning/information processing is still inferior to human performance
- Machines may outperform humans on tasks in certain aspects and under certain conditions:
    - Speed: reading cheques at a rate of hundreds per second
    - Size of input data: finding traces of rare nuclear reactions in Gigabytes of recorded data from physics experiments
    - Accuracy: recognition rates better than human performance if recognizing algorithm could be "trained" on super-humanly vast training data sets
- Machine pattern recognition falls far short of human performance with respect to flexibility, multi-purposeness, context sensitivity, attention control, ... anything that's beyond a clear-cut, single-purpose task setting

A Little ML Zoo: Examples

Image classification: the archetypical ML application

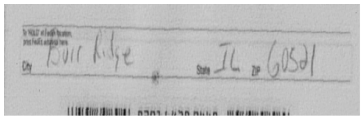


City	State	ZIP
BURR RIDGE	IL	60521
PALM BEACH	FL	33480
RISGAH FOREST	NC	28768
GREENVILLE	NH	03048

courtesy Planet GmbH www.planet.de

9

The *learning* in machine learning



?

City	State	ZIP
BURR RIDGE	IL	60521

- → is a *classifyer* algorithm C. Input: photos of post parcels, output: address in clear text
- C is exceedingly complex. It is a formula that contains millions of parameters.
- Impossible to design C "by hand" or by some mathematical analytical method.
- C has to be *trained*. C has to *learn* how to classify address photos.
- C learns from *training examples:* pairs (photo$_i$, address$_i$) = (x_i, d_i)

Start: create a random initial C_0 with millions of randomly set parameters.

Iterate: present to C_n an address photo x_n. Output will be poor. Adjust parameters slightly such that output comes a little closer to target x_n: get C_{n+1}.

Stop when performance is satisfactory or max learning time reached.

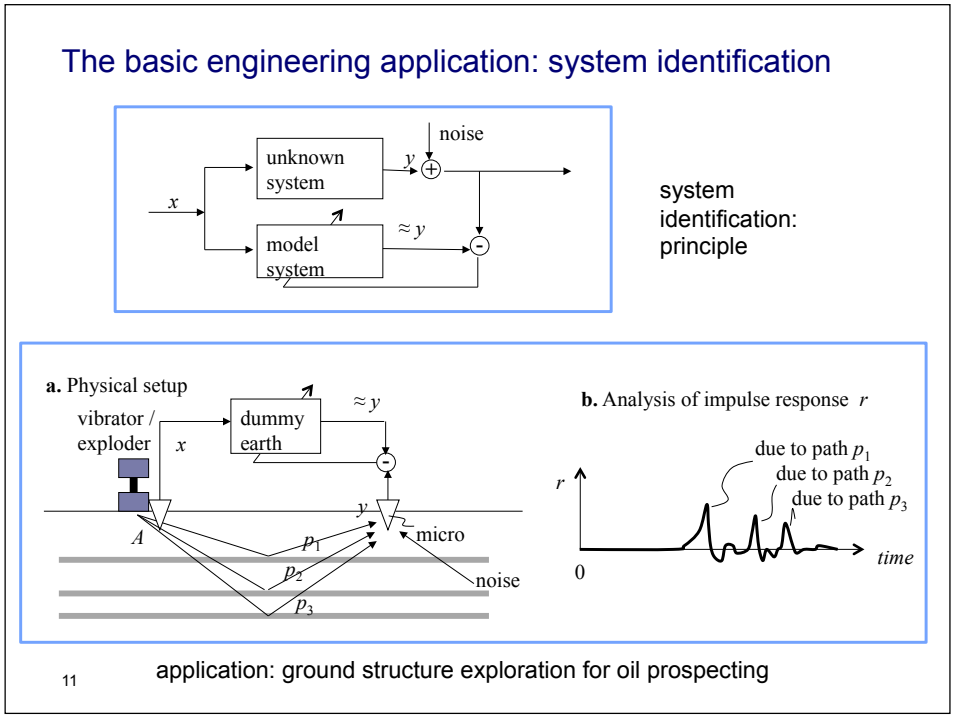
10

- The most widespread kind of ML applications are in *pattern recognition*, also called *pattern classification*.
- A *recognizer* (or *classifier*) is a computational procedure that receives *patterns* as inputs and returns class hypotheses.

- Examples:

Patterns	Classes
Pictures of digits	"1", "2", ..., "0"
Pictures of faces	Person identification, e.g. name
Spoken words (microphone signals)	Recognized words, e.g. "turn left", "stop"
Records of bank account holder data	Creditworthyness judgement, e.g. "will pay back loan with probability > 90%", "will pay back loan with probability < 90%".

- Classification tasks can be arbitrarily complex and require arbitrarily complex and large computer programs (algorithms).
- Training such classifiers requires *labelled training samples* consisting of correctly classified pairs (*pattern*, *class label*).
- The classifier is trained by gradually improving its performance, presenting it with labelled training examples iteratively.
- State-of-the-art handwriting recognizers have millions of parameters, need millions of training examples, and the training time can take some weeks on high-performing computing clusters (e.g. 1000 16-core PCs).
- This is a case of *supervised training*. "Supervised" refers to the fact that *correct* outputs are given to the learning system: its training progress is "supervised". The correct output labels are also sometimes called *teacher* data.



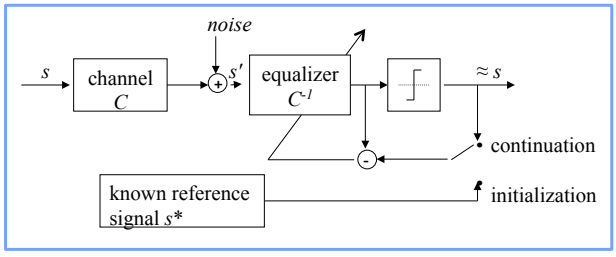
- In engineering and signal processing, the most basic kind of learning task is *system identification*. Some physical system is given which transforms some input signal *x* into some output signal *y*. In system identification one tries to re-create ("learn", "adapt") an artificial system – a computer program – which replicates the

same input-output behavior. This system *model* then can be used for numerous goals. Examples:

- Model the earth atmosphere (for climate change analysis and weather forecast). Input is sunshine energy, output is weather dynamics.
- Model financial markets. Goals are obvious.
- Model a robot body. Input: voltages to motors, output: limb motion. A prerequisite for robot motion control.
- Model geological structures. Input: sound waves, output: sound waves obtained at a distant location. Such sound-in, sound-out models are highly informative about underground structures. A standard tool in geological prospecting.

- In the engineering / signal processing domain, systems are always *temporal* and operate in time. A *signal* is by definition (in engineering) a *time series*.
- In signal processing, such temporal input-output systems are also called *filters* or *transducers*.
- Methods for system identification have been developed a long time before the advent of modern, computer-based machine learning.
- Methods of ML are being adopted slowly by engineers and signal processing people. One reason for their reluctance: classical engineering methods for system identification are *linear* and can be mathematically *fully analysed*. Provable criteria for correctness, accuracy and stability can be given. In contrast, high-performing ML procedures are typically *nonlinear* and their complexity defies mathematical analysis. While they (might) perform better, no mathematical guarantees can be given that/how they will perform in critical application circumstances. But such guarantees are often desirable (think of safety-critical applications like airplane autopilots).
- System identification is also a supervised learning task: the correct system output is available to the learning system at training time.

Channel equalization

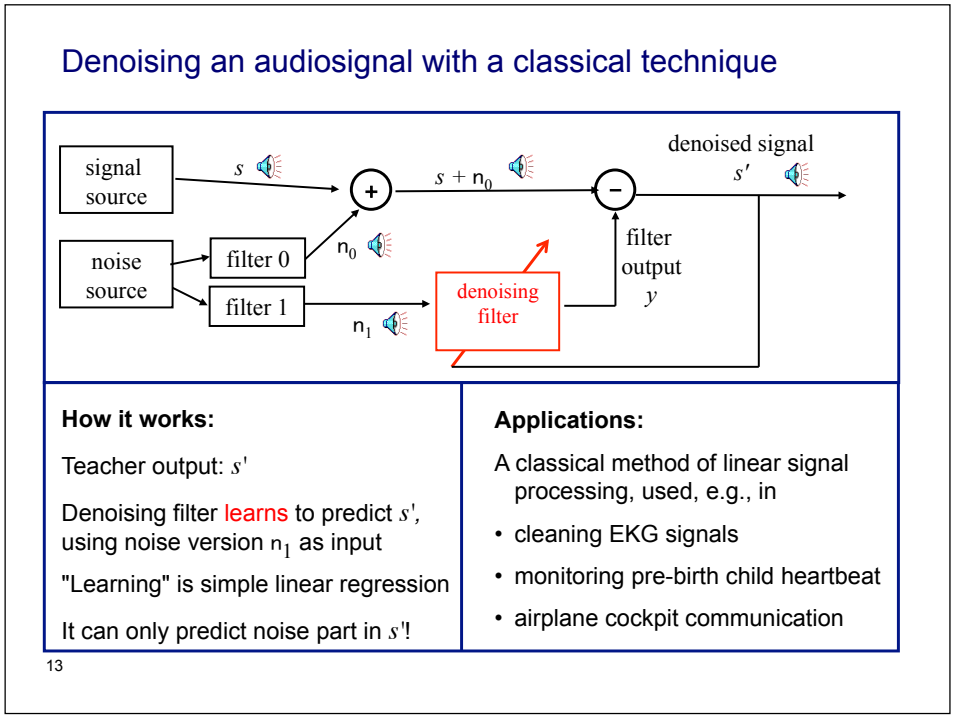


adaptive
channel
equalization

- A crucial component in *every* mobile or stationary communication device!

12

- Channel equalization is a signal processing task of prime practical importance. Scenario: a signal s is transmitted over a channel C, and received as a distorted signal s'. The distortions can be severe. For example, a WLAN signal s is sent through a channel C consisting of the transmitter electronics (which adds some slight nonlinear distortions), the airspace (which adds a truly massive mixture of "echos" resulting from electromagnetic wave reflections), and the receiver electronics. The received s' is a *distorted* version of s (you wouldn't believe how badly it can be messed up). A *channel equalizer* (or channel *inverter*) is a device (= a computer algorithm running on your WLAN receiver) whose task is to transform the distorted s' back to the clean signal s as well as possible.
- Channel equalization "inverts" the channel and is sometimes also called *inverse system identification*.
- One way to train an equalizer is by supervised training: the desired (teacher) output is just the input s to the channel.
- Things get difficult when the channel is quickly changing and the channel inverter has to re-adapt itself equally quickly. This is the situation in mobile phone applications: if you move with your cellphone, the channel (the aerial pathway from the transmitter to your phone) is changing all the time – in fact, quite dramatically. To make things worse, the input signal s is not known – it is the ever-changing, unpredictable phone message that you want to receive. Solution: in short intervals (a few milliseconds) a previously fixed and known reference signal s^* is transmitted and used to re-adapt the equalizer.



- Signal de-noising is another signal processing task of great practical importance.
- If the de-noiser would know the clean signal, denoising would be trivial. Turning this thought a little, training a de-noiser amounts to learning as much as possible about the characteristics of the clean signal. This knowledge can then be used to "subtract" noise (= everything that doesn't fit the clean signal model).

- The de-noising scheme shown on the slide belongs to the classical repertoire of linear signal processing and has been known since at least 60 years. One of the first applications was recording "readable" EKG signals.
- Today much more refined methods are available which use ML methods. Key words are *blind source separation* and *independent component analysis*. The idea is that if you receive a mixture of signals (e.g. speech signal + noise, or speaker1 + speaker2 + speaker3 [the *cocktail party* speech understanding problem], if one may assume that the mixture components are *statistically independent*, then one can de-mix them. Check out http://en.wikipedia.org/wiki/Independent_component_analysis for theory and http://research.ics.aalto.fi/ica/cocktail/cocktail_en.cgi for an audio demo.

More application types, just random highlights

Time series prediction
- financial time series
- local weather development (important for short-term power yield prediction in windmill farms)
- predicting the consequences of action (robot action planning)

System control
(= provide steering input to a system that you want to control)
- steering (or auto-piloting) engines and vehicles
- controlling chemical production plants

Fault monitoring
- monitor power grids or power plants
- monitor any technological device
- driver sleep detection

Temporal pattern generation
- generating motions of robots and game characters

14

Commercially most relevant applications (my guess)

- Customer profiling
- Ad placement optimization
- Financial timeseries prediction
- Control and monitoring of large technological systems (production plants, energy grids, internet)
- Computer games

Societally most relevant applications (my guess)

- Surveillance (communication scanning, face recognition, traffic monitoring)
- Military (autonomous missiles and drones, satellite data interpretation, battlefield robotics)
- Speech and language technology
-

15