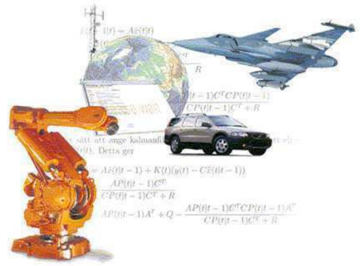
Machine Learning, Lecture 4
Neural Networks (NN) and Introduction to Kernel Methods



Thomas Schön

Division of Automatic Control
Linköping University
Linköping, Sweden.

Email: schon@isy.liu.se,
Phone: 013 - 281373,
Office: House B, Entrance 27.

Outline lecture 4

1. Summary of lecture 3
2. Generalize the linear model to a nonlinear function expansion
3. Training of neural networks
4. Successful examples of NN in real life examples
   - System identification
   - Handwritten digit classification
5. Introducing kernel methods
6. Different ways of constructing kernels

Summary of lecture 3 (I/III)

Investigated one linear **discriminant** (a function that takes an input and assigns it to one of K classes) method in detail (least squares).

Modelled each class as $y_k(x) = w_k^T x + w_{k,0}$ and solved the LS problem, resulting in $\widehat{w} = (X^T X)^{-1} X^T T$.

Showed how probabilistic **generative models** could be built for classification using the strategy,

1. Model $p(x \mid \mathcal{C}_k)$ (a.k.a. class-conditional density)
2. Model $p(\mathcal{C}_k)$
3. Use ML to find the parameters in $p(x \mid \mathcal{C}_k)$ and $p(\mathcal{C}_k)$.
4. Use Bayes' rule to find $p(\mathcal{C}_k \mid x)$

Summary of lecture 3 (II/III)

The "direct" method called **logistic regression** was introduced. Start by stating the model

$$p(\mathcal{C}_1 \mid \phi) = \sigma(w^T \phi) = \frac{1}{1 + e^{-w^T \phi}},$$

which results in a log-likelihood function according to

$$L(w) = -\ln p(T \mid w) = -\sum_{n=1}^{N} \left(t_n \ln(y_n) + (1 - t_n) \ln(1 - y_n) \right),$$

where $y_n = p(\mathcal{C}_1 \mid \phi) = \sigma(w^T \phi)$. Note that this is a nonlinear, **but** concave function of w.

Hence, we can easily find the global minimum using Newton's method (resulting in an algorithm known as IRLS).

Summary of lecture 3 (III/III)

The likelihood function for logistic regression is

$$p(T \mid w) = \prod_{n=1}^{N} \sigma(w^T \phi_n)^{t_n} \left(1 - \sigma(w^T \phi_n)\right)^{1-t_n}$$

Hence, computing the posterior density $p(w \mid T) = \frac{p(T|w)p(w)}{p(T)}$ is intractable and we considered the **Laplace approximation** for solving this.

The Laplace approximation is a simple (local) approximation that is obtained by fitting a Gaussian centered around the (MAP) mode of the distribution.

(Very) brief on evaluating classifiers

Let us define the following concepts,

- **True positives (tp):** Items correctly classified as belonging to the class.
- **True negatives (tn):** Items correctly classified as not belonging to the class.
- **False positives (fp):** Items incorrectly classified as belonging to the class. In other words, a false alarm.
- **False negatives (fn):** Items that are not classified in the correct class, even though they should have. In other words, a missed detection.

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}} \qquad \text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

Two examples of neural networks in use

1. System identification
2. Handwritten digit classification

These examples will provide a glimpse into a few real life applications of models based in nonlinear function expansions (i.e., neural networks) both for regression and classification problems. We provide references for a more complete treatment.

NN example 1 – system identification (I/V)

Neural networks are one of the standard models used in nonlinear system identification.

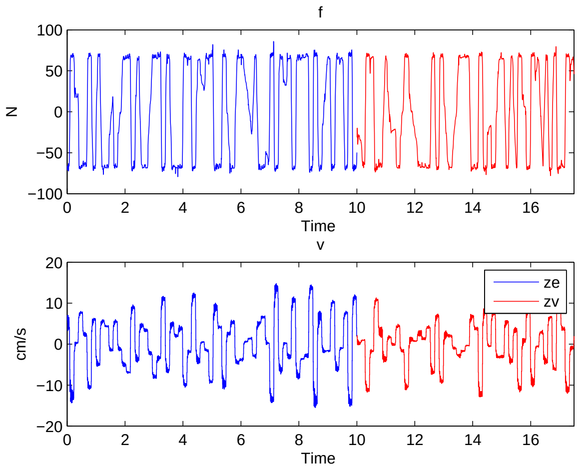
Problem background: The task here is to identify a dynamical model of a Magnetorheological (MR) fluid damper. The MR fluid (typically some kind of oil) will greatly increases its so called apparent viscosity when the fluid is subjected to a magnetic field.

MR fluid dampers are semi-active control devices which are used to reduce vibrations.

Input signal: velocity $v(t)$ [cm/s] of the damper
Output signal: Damping force $f(t)$ [N].

NN example 1 – system identification (II/V)

Have a look at the data



NN example 1 – system identification (III/V)

As usual, we try simple things first, that is a linear model. The best linear model turns out to be an output error (OE) model which gives 51% fit on validation data.

```
LinMod2 = oe(ze, [4 2 1]);  % OE model  y = B/F u + e
```

Try a neural network sigmoidal neural network using 10 hidden units

```
Options = {'MaxIter',50, 'SearchMethod', 'LM'};
Narx1 = nlarx(ze, [2 4 1], 'sigmoidnet',Options{:})
```

This model already gives a 72% fit on validation data.

```
compare(zv, Narx1);
```


NN example 1 – system identification (IV/V)

Using 12 hidden units and only making use of some of the regressors,

```
Sig   = sigmoidnet('NumberOfUnits',12); % create SIGMOIDNET object
Narx5 = nlarx(ze, [2 3 1], Sig, 'NonlinearRegressors', [1 3 4],...
        Options{:});
```

the performance can be increased to a 85% fit on validation data.

Of course, this model need further validation, but the improvement from 51% fit for the best linear model is substantial.

NN example 1 – system identification (V/V)

This example if borrowed from

Wang, J., Sano, A., Chen, T. and Huang. B. **Identification of Hammerstein systems without explicit parameterization of nonlinearity**. *International Journal of Control*, 82(5):937–952, May 2009.

and it is used as one example in illustrating Lennart's toolbox,

`http://www.mathworks.com/products/sysid/demos.html?file=/products/demos/shipping/ident/idnlbbdemo_damper.html`

More about the use of neural networks in system identification can be found in,

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P-Y., Hjalmarsson, H. and Juditsky, A. **Nonlinear black-box modeling in system identification: a unified overview**, *Automatica*, 31(12):1691–1724, December 1995.

NN example 2 – digit classification (I/IV)

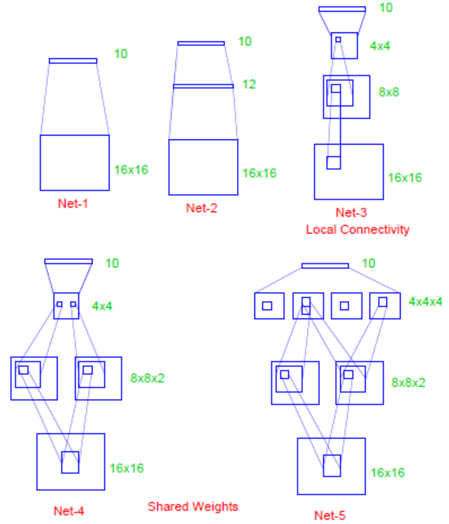
You have tried solving this problem using linear methods before. Let us see what can be done if we generalize to nonlinear function expansions (neural networks) instead.

Let us now investigate 4 nonlinear models and one linear model solving the same task,

- **Net-1:** No hidden layer (equivalent to logistic regression).
- **Net-2:** One hidden layer, 12 hidden units fully connected.
- **Net-3:** Two hidden layers locally connected.
- **Net-4:** Two hidden layers, locally connected with weight sharing.
- **Net-5:** Two hidden layers, locally connected with two levels of weight sharing.

NN example 2 – digit classification (II/IV)

Local connectivity (Net3-Net5) means that each hidden unit is connected only to a small number of units in the layer before. It makes use of the **key** property that nearby pixels are more strongly correlated than distant pixels.



NN example 2 - digit classification (III/IV)

Network Architecture	Links	Weights	Correct
Net-1: 0 hidden layer	2570	2570	80.0%
Net-2: 1 hidden layer network	3214	3214	87.0%
Net-3: 2 hidden layer, locally connected	1226	1226	88.5%
Net-4: 2 hidden layer, constrained network	2266	1132	94.0%
Net-5: 2 hidden layer, constrained network	5194	1060	98.4%

Net-4 and Net-5 are referred to as **convolutional** networks.

This example illustrates that knowledge about the problem at hand can be very useful in order to improve the performance!

NN example 2 – digit classification (IV/IV)

This example if borrowed from Section 11.7 in HTF, who in turn borrowed it from,

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. **Gradient-Based Learning Applied to Document Recognition**, *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

The **best** performance as of today is provided by a **deep neural network** published in last years CVPR (99.77%),

Ciresan, D., Meier, U. and Schmidhuber, J. **Multi-column Deep Neural Networks for Image Classification**, In proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Providence, Rhode Island, USA, June, 2012.

Recall the page mentioned during lecture 1 on this problem,

`http://yann.lecun.com/exdb/mnist/`

There is plenty of **software support** for neural networks, for example MATLAB's neural network toolbox (for general problems) and the system identification toolbox (for sys. id.).

The "deep learning hype"

Even made it into the New York Times in November, 2012,

http://www.nytimes.com/2012/11/24/science/scientists-see-advances-in-deep-learning-a-part-of-artificial-intelligence.html?pagewanted=1&_r=0

Deep learning is a very active topic at NIPS.

A good entry point (tutorial papers, talks, groups, etc.) into the deep learning hype is provided by

http://deeplearning.net/

Interesting topic for a project!

Introducing kernel methods (I/III)

Let us introduce the kernel methods as an **equivalent** formulation of the linear regression problem.

Recall that linear regression models the relationship between a continuous target variable t and a function $\phi(x)$ of the input variable x,

$$t_n = \underbrace{w^T \phi(x_n)}_{y(x_n, w)} + \epsilon_n.$$

From lecture 2 we have that the posterior distribution is given by

$$\begin{aligned} p(w \mid T) &= \mathcal{N}(w \mid m_N, S_N), \\ m_N &= \beta S_N \Phi^T T, \\ S_N &= (\alpha I + \beta \Phi^T \Phi)^{-1}. \end{aligned}$$

Introducing kernel methods (II/III)

Inserting this into $y(x, w) = w^T \phi(x)$ provides the following expression for the predictive mean

$$\begin{aligned} y(x, m_N) &= m_N^T \phi(x) = \phi(x)^T m_N = \beta \phi(x)^T S_N \Phi^T T \\ &= \sum_{n=1}^{N} \underbrace{\beta \phi(x)^T S_N \phi(x_n)}_{k(x, x_n)} t_n = \sum_{n=1}^{N} k(x, x_n) t_n, \end{aligned}$$

where

$$k(x, x') = \beta \phi(x)^T S_N \phi(x')$$

is referred to as the **equivalent kernel**.

This suggests and alternative approach to regression where we instead of introducing a set of basis functions directly make use of a localized kernel.

Introducing kernel methods (III/III)

Kernel methods constitutes a class of algorithms where the training data (or a subset thereof) is kept also during the prediction phase.

Many linear methods can be re-cast into an **equivalent** "dual representation" where the predictions are based on linear combinations of kernel functions (one example provided above).

A general property of kernels is that they are inner products

$$k(x, z) = \psi(x)^T \psi(z)$$

(Linear regression example, $\psi(x) = \beta^{1/2} S_N^{1/2} \phi(x)$)

The above development suggests the following idea. In an algorithm where the input data x enters only in the form of scalar products we can replace this scalar product with another choice of kernel! This is referred to as the **kernel trick**.

Kernel representation of ℓ_2-reg. LS (I/II)

Inserting the solution $\widehat{w} = \Phi^T \widehat{a} = \Phi^T (K + \lambda I)^{-1} T$ into $y(x, w)$ provides the following prediction for a new input x

$$
\begin{aligned}
y(x, \widehat{w}) &= \widehat{w}^T \phi(x) = \widehat{a}^T \Phi \phi(x) = \left(\left((K + \lambda I)^{-1} T \right)^T \Phi \phi(x) \right)^T \\
&= \phi(x)^T \Phi^T (K + \lambda I)^{-1} T \\
&= \phi(x)^T \begin{pmatrix} \phi(x_1) & \phi(x_2) & \cdots & \phi(x_N) \end{pmatrix} (K + \lambda I)^{-1} T \\
&= \begin{pmatrix} k(x, x_1) & k(x, x_2) & \cdots & k(x, x_N) \end{pmatrix} (K + \lambda I)^{-1} T,
\end{aligned}
$$

where we have made use of the definition of a **kernel function**

$$k(x, z) \triangleq \phi(x)^T \phi(z)$$

Hence, the solution to the ℓ_2-regularized least squares problem is expressed in terms of the kernel function $k(x, z)$.

Kernel representation of ℓ_2-reg. LS (II/II)

Note (again) that the prediction at x is given by a linear combination of the target values from the training set (expensive).

Furthermore, we are required to invert an $N \times N$ matrix (compared to an $M \times M$ matrix in the original formulation), where typically $N \gg M$.

Relevant question, **So what is the point?**

The fact that it is expressed only using the kernel function $k(x, z)$ implies that we can work entirely using kernels and avoid introducing basis functions $\phi(x)$. This in turn allows us to implicitly use basis functions of **high, even infinite, dimensions** ($M \to \infty$).

Constructing kernels

1. Chose a feature mapping $\phi(x)$ and then use this to find the corresponding kernel,

$$k(x, z) = \phi(x)^T \phi(z) = \sum_{i=1}^{M} \phi_i(x) \phi_i(z)$$

2. Chose a kernel function directly. In this case it is important to verify that it is in fact a kernel. (we will see two examples of this)
   A function $k(x, z)$ is a kernel iff the Gram matrix K is positive semi-definite for all possible inputs.
3. Form new kernels from simpler kernels.
4. Start from probabilistic generative models.

Techniques for constructing new kernels

Given valid kernels $k_1(x, z)$ and $k_2(x, z)$, the following are also valid kernels

$$
\begin{aligned}
k(x, z) &= c k_1(x, z), & k(x, z) &= f(x) k_1(x, z) f(z), \\
k(x, z) &= q(k_1(x, z)), & k(x, z) &= \exp(k_1(x, z)), \\
k(x, z) &= k_1(x, z) + k_2(x, z), & k(x, z) &= k_1(x, z) k_2(x, z), \\
k(x, z) &= k_3(\phi(x), \phi(z)), & k(x, z) &= x^T A x,
\end{aligned}
$$

where $c > 0$ is a constant, f is a function, q is a polynomial with nonnegative coefficients, $\phi(x)$ is a function from x to $\mathbb{R}^M$, k_3 is a valid kernel in $\mathbb{R}^M$ and $A \succeq 0$.

Example – polynomial kernel

Let us investigate if the polynomial kernel

$$k(x, z) = (x^T z + c)^n, c > 0$$

is a kernel for the special case $n = 2$ and a 2D input space $x = (x_1, x_2)^T$,

$$\begin{aligned} k(x, z) = (x^T z)^2 &= (x_1 z_1 + x_2 z_2 + c)^2 \\ &= x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 + 2cx_1 z_1 + 2cx_2 z_2 + c^2 \\ &= \phi(x)^T \phi(z), \end{aligned}$$

where

$$\phi(x) = \begin{pmatrix} x_1^2 & \sqrt{2} x_1 x_2 & x_2^2 & \sqrt{2c} x_1 & \sqrt{2c} x_2 & c \end{pmatrix}^T$$

Hence, it contains all possible terms (constant, linear and quadratic) up to order 2.

A few concepts to summarize lecture 4

Neural networks: A nonlinear function (as a function expansion) from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector w of parameters.

Backpropagation: Computing the gradients by making use of the chain rule, combined with clever reuse of information that is needed for more than one gradient.

Convolutional neural networks: The hidden units takes their inputs from a small part of the available inputs and all units have the *same* weights (called weight sharing).

Kernel function: A kernel function $k(x, z)$ is defined as an inner product $k(x, z) = \phi(x)^T \phi(z)$, where $\phi(x)$ is a fixed mapping.

Kernel trick: (a.k.a. kernel substitution) In an algorithm where the input data x enters only in the form of scalar products we can replace this scalar product with another choice of kernel.