K-Class Classification Problem
Let us denote the k-th class by $C_k$, with $n_k$ exemplars or training samples, forming the sets $T_k$ for $k = 1, \ldots, K$:

$$T_k = \{x^i_k, d^i_k\} \mid p = 1, \ldots, n_k$$

The complete training set is $T = \bigcup_{k=1}^{K} T_k$. The desired output of the network for an input of class $k$ is $1$ for output unit $k$ and $0$ for all other output units:

$$d^i = (0,0,\ldots,0,1,0,\ldots,0)$$

with a $1$ at the $k$-th position if the sample is in class $k$.

K-Class Classification Problem
However, due to the sigmoid output function, the net input to the output units would have to be $\approx$ or $\approx$ to generate outputs 0 or 1, respectively. Because of the shallow slope of the sigmoid function at extreme net inputs, even approaching these values would be very slow. To avoid this problem, it is advisable to use desired outputs $\varepsilon$ and $(1 - \varepsilon)$ instead of 0 and 1, respectively. Typical values for $\varepsilon$ range between 0.01 and 0.1. For $\varepsilon = 0.1$, desired output vectors would look like this:

$$d^i = (0.1,0.1,\ldots,0.1,0.1,\ldots,0.1)$$

NN Application Design
Now that we got some insight into the theory of backpropagation networks, how can we design networks for particular applications?

Designing NNs is basically an engineering task. As we discussed before, for example, there is no formula that would allow you to determine the optimal number of hidden units in a BPN for a given task.

Data Representation
• Most networks process information in the form of input pattern vectors.
• These networks produce output pattern vectors that are interpreted by the embedding application.
• All networks process one of two types of signal components: analog (continuously variable) signals or discrete (quantized) signals.
• In both cases, signals have a finite amplitude; their amplitude has a minimum and a maximum value.

NN Application Design
We should not “punish” more extreme values, though. To avoid punishment, we can define $\ell_{pj}$ as follows:

1. If $d_{pj} = (1 - \varepsilon)$ and $o_{pj} \geq d_{pj}$ then $\ell_{pj} = 0$.
2. If $d_{pj} = \varepsilon$ and $o_{pj} \leq d_{pj}$ then $\ell_{pj} = 0$.
3. Otherwise, $\ell_{pj} = o_{pj} - d_{pj}$

We need to address the following issues for a successful application design:

• Choosing an appropriate data representation
• Performing an exemplar analysis
• Training the network and evaluating its performance

We are now going to look into each of these topics.

October 5, 2010 Neural Networks Lecture 9: Applying Backpropagation
The main question is:
How can we appropriately capture these signals and represent them as pattern vectors that we can feed into the network?

We should aim for a data representation scheme that maximizes the ability of the network to detect (and respond to) relevant features in the input pattern.

Relevant features are those that enable the network to generate the desired output pattern.

Similarly, we also need to define a set of desired outputs that the network can actually produce. Often, a “natural” representation of the output data turns out to be impossible for the network to produce.

We are going to consider internal representation and external interpretation issues as well as specific methods for creating appropriate representations.

As we said before, in all network types, the amplitude of input signals and internal signals is limited:
- analog networks: values usually between 0 and 1
- binary networks: only values 0 and 1 allowed
- bipolar networks: only values –1 and 1 allowed

Without this limitation, patterns with large amplitudes would dominate the network’s behavior.

A disproportionately large input signal can activate a neuron even if the relevant connection weight is very small.

From the perspective of the embedding application, we are concerned with the interpretation of input and output signals.

These signals constitute the interface between the embedding application and its NN component.

Often, these signals only become meaningful when we define an external interpretation for them.

This is analogous to biological neural systems: The same signal becomes completely different meaning when it is interpreted by different brain areas (motor cortex, visual cortex etc.).

Without any interpretation, we can only use standard methods to define the difference (or similarity) between signals.

For example, for binary patterns x and y, we could:
- ... treat them as binary numbers and compute their difference as | x – y |
- ... treat them as vectors and use the cosine of the angle between them as a measure of similarity
- ... count the numbers of digits that we would have to flip in order to transform x into y (Hamming distance)
External Interpretation Issues
Example: Two binary patterns $x$ and $y$:

$$x = 0001000011110100110001100110010011110$$

$$y = 10000100010100100010001000100011110$$

These patterns seem to be very different from each other. However, given their external interpretation...

...$x$ and $y$ actually represent the same thing.

Creating Data Representations
The patterns that can be represented by an ANN most easily are binary patterns.

Even analog networks "like" to receive and produce binary patterns — we can simply round values $< 0.5$ to $0$ and values $\geq 0.5$ to $1$.

To create a binary input vector, we can simply list all features that are relevant to the current task.

Each component of our binary vector indicates whether one particular feature is present (1) or absent (0).

Creating Data Representations
With regard to output patterns, most binary-data applications perform classification of their inputs.

The output of such a network indicates to which class of patterns the current input belongs.

Usually, each output neuron is associated with one class of patterns.

As you already know, for any input, only one output neuron should be active (1) and the others inactive (0), indicating the class of the current input.

Creating Data Representations
Tertiary (and n-ary) patterns can cause more problems than binary patterns when we want to format them for an ANN.

For example, imagine the tic-tac-toe game.

Each square of the board is in one of three different states:

- occupied by an X,
- occupied by an O,
- empty

Creating Data Representations
Let us now assume that we want to develop a network that plays tic-tac-toe.

This network is supposed to receive the current game configuration as its input.

Its output is the position where the network wants to place its next symbol (X or O).

Obviously, it is impossible to represent the state of each square by a single binary value.
Creating Data Representations

Possible solution:

• Use multiple binary inputs to represent non-binary states.
• Treat each feature in the pattern as an individual subpattern.
• Represent each subpattern with as many positions (units) in the pattern vector as there are possible states for the feature.
• Then concatenate all subpatterns into one long pattern vector.

Example:

• X is represented by the subpattern 100
• O is represented by the subpattern 010
• <empty> is represented by the subpattern 001
• The squares of the game board are enumerated as follows:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Then consider the following board configuration:

```
X X
O O X
```

It would be represented by the following binary string:

```
100 100 001 010 010 100 001 001 010
```

Consequently, our network would need a layer of 27 input units.

Creating Data Representations

And what would the output layer look like?

Well, applying the same principle as for the input, we would use nine units to represent the 9-ary output possibilities.

Considering the same enumeration scheme:

```
1 2 3
4 5 6
7 8 9
```

Our output layer would have nine neurons, one for each position.

To place a symbol in a particular square, the corresponding neuron, and no other neuron, would fire (1).

But...

Would it not lead to a smaller, simpler network if we used a shorter encoding of the non-binary states?

We do not need 3-digit strings such as 100, 010, and 001, to represent X, O, and the empty square, respectively.

We can achieve a unique representation with 2-digit strings such as 10, 01, and 00.

Similarly, instead of nine output units, four would suffice, using the following output patterns to indicate a square:

```
0000 0001 0010
0100 0101 0110
1000 1001 1010
```
Creating Data Representations

The problem with such representations is that the meaning of the output of one neuron depends on the output of other neurons.

This means that each neuron does not represent (detect) a certain feature, but groups of neurons do.

In general, such functions are much more difficult to learn.

Such networks usually need more hidden neurons and longer training, and their ability to generalize is weaker than for the one-neuron-per-feature-value networks.