Disclaimer: These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.

1 The Multi-layer Perceptron

1.1 Matlab demos

Matlab tutorials for neural network design:

nnd9sd % Steepest descent
nn9sdq % Steepest descent for quadratic

Character recognition with MLP:

appcr1

Structure of MLP:

![MLP Structure Diagram]

Noise-free input: 26 different letters of size 7×5. Prediction errors:
1.2 An example and notations

Here we will always assume that the activation function is differentiable. This will allow us to optimize the cost function with gradient descent. However, non-differentiable activation functions are getting popular as well.

2 The back-propagation algorithm

2.1 The gradient of the error

The current error:

\[ \epsilon^2 = \epsilon_1^2 + \epsilon_2^2 = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2. \]
More generally:

\[ \epsilon^2 = \sum_{p=1}^{N_L} \epsilon_p^2 = \sum_{p=1}^{N_L} (\hat{y}_p - y_p)^2. \]

We want to calculate

\[ \frac{\partial \epsilon(k)^2}{\partial W_{ij}^l(k)} = ? \]

### 2.2 Notation

- \(W_{ij}^l(k)\): At time step \(k\), the strength of connection from neuron \(j\) on layer \(l-1\) to neuron \(i\) on layer \(l\). \((i = 1, 2, ..., N_l, j = 1, 2, ..., N_{l-1})\)
- \(s_i^l(k)\): The summed input of neuron \(i\) on layer \(l\) before applying the activation function \(f\) at time step \(k\)(\(i = 1, ..., N_l\)).
- \(x^l(k) \in \mathbb{R}^{N_{l-1}}\): The input of layer \(l\) at time step \(k\).
- \(\hat{y}_l^l(k) \in \mathbb{R}^{N_l}\): The output of layer \(l\) at time step \(k\).
- \(N_1, N_2, ..., N_l, ..., N_L\): Number of neurons in layers 1, 2, ..., \(l, ..., L\).

### 2.3 Some observations

\[
x^l = \hat{y}^{l-1} \in \mathbb{R}^{N_{l-1}}
\]

\[
s_i^l = W_{ij}^l \hat{y}^{l-1} = \sum_{j=1}^{N_{l-1}} W_{ij}^l x_j^{l-1} = \sum_{j=1}^{N_{l-1}} W_{ij}^l f(s_j^{l-1})
\]

\[
s_j^{l+1} = \sum_{i=1}^{N_l} W_{ji}^{l+1} f(s_i^l)
\]

### 2.4 The back propagated error

Recall that \(\frac{\partial}{\partial x} f(g(x), h(x)) = \frac{\partial}{\partial y} f(g(x), h(x)) \frac{\partial g(x)}{\partial x} + \frac{\partial}{\partial h} f(g(x), h(x)) \frac{\partial h(x)}{\partial x}\).

Introduce the notation:

\[ \delta_i^l(k) = -\frac{\partial \epsilon^2(k)}{\partial s_i^l(k)} = -\sum_{p=1}^{N_L} \frac{\partial \epsilon_p^2(k)}{\partial s_i^l(k)} \]

where \(i = 1, 2, ..., N_l\).

As a special case, we have that

\[ \delta_i^L(k) = -\sum_{p=1}^{N_L} \frac{\partial (y_p(k) - f(s_p^L(k)))^2}{\partial s_i^L(k)} = 2\epsilon_i(k)f'(s_i^L(k)) \]
Lemma 1. $\delta^l_i(k)$ can be calculated from $\{\delta^{l+1}_i(k), ..., \delta^{L+1}_{N_{L+1}}(k)\}$ using backward recursion.

$$
\delta^l_i(k) = -\sum_{p=1}^{N_L} \frac{\partial \epsilon^2_p}{\partial s^l_i} = \sum_{p=1}^{N_L} \sum_{j=1}^{N_{i+1}} -\frac{\partial \epsilon^2_p}{\partial s^{l+1}_j} \frac{\partial s^{l+1}_j}{\partial s^l_i} \\
= \sum_{j=1}^{N_{i+1}} \sum_{p=1}^{N_L} -\frac{\partial \epsilon^2_p}{\partial s^{l+1}_j} W^{l+1}_{ji} f'(s^l_i)
$$

Therefore,

$$
\delta^l_i(k) = \left( \sum_{j=1}^{N_{i+1}} \delta^{l+1}_j W^{l+1}_{ji}(k) \right) f'(s^l_i(k))
$$

where $\delta^l_i(k)$ is the back propagated error.

Now using that

$$
s^l_i(k) = \sum_{j=1}^{N_{i-1}} W^l_{ij}(k)x^l_j(k)
$$

$$
\frac{\partial \epsilon(k)^2}{\partial W^l_{ij}(k)} = \frac{\partial \epsilon(k)^2}{\partial s^l_i(k)} \frac{\partial s^l_i(k)}{\partial W^l_{ij}(k)} = -\delta^l_i(k)x^l_j(k)
$$

The Back-propagation algorithm:

$$
W^l_{ij}(k + 1) = W^l_{ij}(k) + \mu \delta^l_i(k)x^l_j(k)
$$

In vector form:

$$
W^l_i(k + 1) = W^l_i(k) + \mu \delta^l_i(k)x_i(k).
$$