Neural Networks

1 Preprocessing

1.1 Input Normalization

Linear rescaling to avoid scaling problems. Useful for radial basis function and multilayer networks. Each input treated independently for scaling. Calculate mean and variance of training set $\boldsymbol{x}^{(m)}$ with $m \in [0, M-1]$ and normalize input vectors to $\mu_i = 0$ and $\sigma_i = 1$.

1.2 Feature Extraction

For **speech recognition** use AnaFB, power estimation ($|\cdot|^2$), mel-filterbank (15-30 normalized overlapping triangular filters), cepstrum estimation (IDFT/IDCT of log power spectrum - to decorrelate input features - symmetric), temporal features (combine successive features vectors - delta or delta-delta features), dimension reduction (feature space transformation, e.g. LDA - dimensionality reduction by preserving class discriminatory information - variance of features corresponding to one class is minimized, distance between classes maximized).

1.2.1 LDA

Dataset of M observations of N-dimensional Euclidian variable $\boldsymbol{x}$. Project $\boldsymbol{x}$ to one dimension using a projection vector $\boldsymbol{w}$, such that $y^{(m)} = \boldsymbol{w}^T\boldsymbol{x}^{(m)}$. Cost function $\max_{\boldsymbol{w}} \boldsymbol{w}^T(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$ to separate class means. **Fisher's idea:** Large separation between projected class means while granting small variance within each class:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T\boldsymbol{S}_b\boldsymbol{w}}{\boldsymbol{w}^T\boldsymbol{S}_w\boldsymbol{w}} \quad \Rightarrow \quad \boldsymbol{w}_{\text{opt}} = \boldsymbol{S}_w^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$$

with $\boldsymbol{S}_b$ being between class covariance matrix, $\boldsymbol{S}_w$ within class covariance matrix.

2 Threshold Logic Units - Single Perceptrons

Neural Network is a blackbox/mapping machine/network of functions with N-dim. input $\boldsymbol{x}$ and M-dim. output $\boldsymbol{y}$. A computing element/unit/node has unlimited fan-in and is a primitive function f of N arguments.

2.1 McCulloch-Pitts Neuron

Binary signals transmitted over edges produce binary result, driven by threshold Φ. N exhibitory edges x_i and K inhibitory edges x_j'. Inhibitory edges can inactivate the whole unit if at least one is active. Else it works as a threshold gate. Neuron is activated if $\sum_{k=0}^{N-1} x_k \geq \Phi$.

2.2 Perceptron

With threshold Φ and input weights w_k, so the output is active if $\sum_{k=0}^{N-1} w_k x_k \geq \Phi$. Simple Perceptron and McCulloch-Pitts Unit are equivalent. Geometric interpretation can be used to check if a function can be computed by a Perceptron using a separating line/plane. XOR function with 2 vars can not be solved by one perceptron - not linearly separable ($0 < \Phi$, $w_0 > \Phi$, $w_1 > \Phi$, $w_0 + w_1 < \Phi$). Two sets of A and B with different output values are linearly separable if $\sum_{x_k \in A} x_k w_k \geq \Phi$ and $\sum_{x_k \in B} x_k w_k < \Phi$.
Solve XOR problem using network of perceptrons $(x_0 \wedge \bar{x_1}) \vee (\bar{x_0} \wedge x_1)$. First layer labels the region, output layer decodes the classification.

2.3 Training

2.3.0.1 Supervised learning Weights of the network are initialized randomly. On misclassification network parameters are adjusted. Desired output is always known, thus it is called learning with a teacher. In **reinforcement learning** only input vector is used for the weight adjustment. In **corrective learning** the magnitude of the error together with the input vector are used for the weight correction.
Training rule:

$$\Phi_{\text{new}} = \Phi_{\text{old}} + \Delta\Phi, \qquad w_{\text{new},i} = w_{\text{old},i} + \Delta w_i$$

2.3.0.2 Unsupervised learning For a given input the output is unknown - learning without a teacher.

2.4 Perceptron Learning Algorithm

Transform perceptron into zero-valued threshold by turning Φ into a weight w_N (bias) such that $\sum_{k=0}^{N-1} x_k w_k - \Phi \geq 0$.

$$\begin{aligned}
\textit{start}: &\boldsymbol{w}_0 \text{ is generated randomly} \\
&n = 0 \\
\textit{test}: &\text{Select } \boldsymbol{x} \in P \cup F \text{ randomly} \\
&\text{if } \boldsymbol{x} \in P \text{ and } \boldsymbol{w}_t\boldsymbol{x} > 0 \text{ go to test} \\
&\text{if } \boldsymbol{x} \in P \text{ and } \boldsymbol{w}_t\boldsymbol{x} \leq 0 \text{ go to add} \\
&\text{if } \boldsymbol{x} \in N \text{ and } \boldsymbol{w}_t\boldsymbol{x} < 0 \text{ go to test} \\
&\text{if } \boldsymbol{x} \in N \text{ and } \boldsymbol{w}_t\boldsymbol{x} \geq 0 \text{ go to subtract} \\
\textit{add}: &\text{set } \boldsymbol{w}_{n+1} = \boldsymbol{w}_t + \boldsymbol{x} \text{ and } n = n+1\text{, goto test} \\
\textit{subtract}: &\text{set } \boldsymbol{w}_{n+1} = \boldsymbol{w}_t - \boldsymbol{x} \text{ and } n = n+1\text{, goto test}
\end{aligned}$$

2.5 Fast learning algorithm - delta rule

If $\boldsymbol{x} \in P$ is classified erroneously we have $\boldsymbol{w}_n^T\boldsymbol{x} \leq 0$ so we get the error $\Delta(n) = -\boldsymbol{w}_n^T\boldsymbol{x}$ so that the weight vector gets corrected: $\boldsymbol{w}_{n+1} = \boldsymbol{w}_n \frac{\Delta(n)+\epsilon^+}{\|\boldsymbol{x}\|^2}\boldsymbol{x}$. So that $\boldsymbol{w}_{n+1}^T\boldsymbol{x} = \left(\boldsymbol{w}_n \frac{\Delta(n)+\epsilon^+}{\|\boldsymbol{x}\|^2}\boldsymbol{x}\right)^T \boldsymbol{x} = \epsilon^+ > 0$. ϵ guarantees that the new weight vector barely skips over the border of the region with higher error. For $\boldsymbol{x} \in F$ use ϵ^-. A variant is using $\gamma(\Delta(n) + \epsilon)\boldsymbol{x}$ with γ as learning factor.

2.6 Convergence

If P and N are finite and linearly separable.

$$\begin{aligned}
\cos(\phi) &= \frac{\boldsymbol{w}_{\text{des}}^T\boldsymbol{w}_{n+1}}{\|\boldsymbol{w}_{n+1}\|} \\
&= \frac{\boldsymbol{w}_{\text{des}}^T\boldsymbol{w}_0 + (n+1)\delta}{\sqrt{\|\boldsymbol{w}_0\|^2 + (n+1)}}
\end{aligned}$$

3 Multilayer perceptrons

ANN is a (L, C) tupel with a set of nodes L and a set of directed edges C. $c = (v, l) \in C$ are directed edges from node v to node l. L_{in} input layer subset, L_{hid} hidden layer subset, L_{out} output layer. Set $L_l^{(pre)}$ consists of prior nodes of node l (predecessors). Set $L_l^{(suc)}$ consists of subsequent nodes of node l (successors). Every edge has a weight $w_v^{(l)}$.

3.0.0.3 Generalized neuron Consists of network input function $f_{net}^{(l)}$, activation function $f_{act}^{(l)}$ and network output function $f_{out}^{(l)}$ and three states respectively $y_{net}^{(l)}$, $y_{act}^{(l)}$, $y_{out}^{(l)}$, also external input $x_{ref}^{(l)}$.

3.0.0.4 Properties of multilayer perceptrons A NN with layered architecture doesn't contain cycles. A MP is a feed-fwd NN with strictly layered structure. Normally all units in a layer connected to all other units of the next layer.

3.0.0.5 Weight matrices Given $L_{in} = \{v_0, ..., v_{N-1}\}$ and $L_{out} = \{l_0, ..., l_{M-1}\}$ the connection weights are gathered in a matrix

$$\boldsymbol{W} = \begin{bmatrix} w_{v_0}^{(l_0)} & w_{v_1}^{(l_0)} & \cdots & w_{v_{N-1}}^{(l_0)} \\ w_{v_0}^{(l_1)} & w_{v_1}^{(l_1)} & \cdots & w_{v_{N-1}}^{(l_1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{v_0}^{(l_{M-1})} & w_{v_1}^{(l_{M-1})} & \cdots & w_{v_{N-1}}^{(l_{M-1})} \end{bmatrix} \in \mathbb{R}^{M \times N}$$

so that $\boldsymbol{y}_{net}^{(L_1)} = \boldsymbol{W}\boldsymbol{y}_{out}^{(L_0)}$