1 Bayes-Theorie

$p(w|f) = \frac{p(f|w)p(w)}{p(f)}$

2 Aktivierungsfunktionen

The activation functions should have the following properties:

- continous
- bounded
- monotonically increasing
- differentiable

2.1 Step Function

$$\varphi(x) = \begin{cases} 1 \text{ if} x > 0 \\ 0 \text{ if} x \leq 0 \end{cases}$$

Derivative is always 0

2.2 Linear Function

$$\varphi(x) = x$$

Linear functins alone can only solve linear separable problems but can be used in a combination node for function approximation problems.

2.3 Logistic / Sigmoid Function

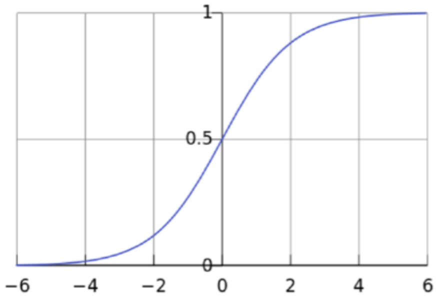
$$\varphi(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$
$$\frac{\delta\varphi(x)}{\varphi(x)} = \varphi(x)(1 - \varphi(x))$$

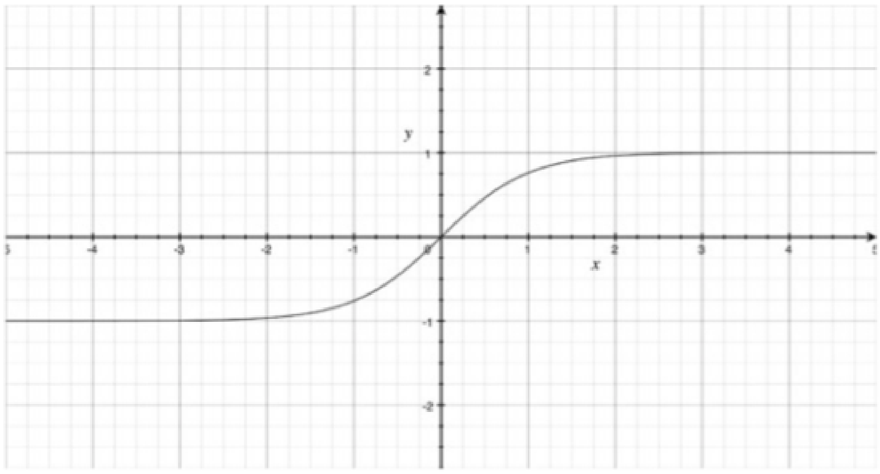
Good for internal nodes, bad for outpur nodes.

2.4 Hyperbolic Tangent function

$$\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$\frac{\delta\sigma(x)}{\sigma(x)} = 1 - \tanh^2(x) = 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2}$$

If the input has a mean of 0 then so will the output





2.5 Softmax Function

$$\varphi(x_j) = \frac{e^{x_j}}{\sum_k e^{x_k}}$$

$$\frac{\delta\varphi(x_j)}{\varphi(x_j)} = \varphi(x_j) - \varphi(x_j)^2 = \varphi(x_j)(1-\varphi(x_j))$$

Outputs an a posteriori probability $p(c|x)$ and is good for classification tasks.

2.6 Rectified Linear Unit

$$\varphi(x) = max(0, x)$$

$$\varphi' = \begin{cases} 1\, if\, x > 0 \\ 0\, if\, x \leq 0 \end{cases}$$

Can result in sparse networks

3 Fehlerfunktionen

- $E_{MSE}(w) = \frac{1}{2} \sum\limits_{x \in X} \sum\limits_{k} (t_k^x - o_k^x)^2$

- Mean-Squared-Error = $\frac{1}{N} * SSE$
- $E_{CE}(w) = -\sum_{x \in X} \sum_{k} [t_k^x * log(o_k^x) + (1 - t_k^x) * log(1 - o_k^x)]$

4 Perzeptron Lernalgorithmus

$w_i^{t+1} = w_i^t + \eta(t_x - o_x)x_i$

5 Backpropagation

$w = w - \eta \nabla_w E(x, w)$ mit $\nabla_w E = \frac{\partial E}{\partial o} \frac{\partial o}{\partial \sigma} \frac{\partial \sigma}{\partial w}$

6 Hopfield

$$x_j = \sum_{i; i \neq j} u_i T_{ij}$$

$$u_i = g(x_j) = \begin{cases} 1 \; if x \geq 0 \\ -1 \; otherwise \end{cases}$$

$$E = -\frac{1}{2} \sum_j \sum_{i; i \neq j} u_i u_j T_{ij}$$

$$C \approx 0.15N$$

$$\frac{N}{4 ln N} < C < \frac{N}{2 ln N}$$

The energy function assigns a numerical value to each possible state of the system (**Lyapunov Function**.

7 Boltzmann-Maschinen

$$z_i = b_i + \sum_j s_j w_{ij}$$

$$p(s_i = 1) = \frac{1}{1 + e^{-z_i}}$$

$$E = -\sum_{i<j} w_{ij} s_i s_j - \sum_i b_i s_i$$

$$p(v) = \frac{e^{-E(v)}}{\sum_u e^{-E(u)}}$$

b_i: bias
s_j: state

w_{ij}: weight between state j and i
Simulated Annealing:

$$p(s_i = 1) = \frac{1}{1 + e^{-\frac{z_i}{T}}}$$

8 RESTRICTED-BOLTZMANN-MASCHINEN

$$E(V,H) = -\sum_{i=1}^{m}\sum_{j=1}^{F} W_{ij} h_j v_i - \sum_{i=1}^{m} v_i a_i - \sum_{j=1}^{F} h_j b_j$$

$$p(h_j = 1|V) = \sigma(b_j + \sum_{j=1}^{m} W_{ij} v_i$$

$$p(v_i = 1|H) = \sigma(a_j + \sum_{j=1}^{F} W_{ij} h_j$$

$$\sigma = \frac{1}{1 + e^{-x}}$$

9 REINFORCEMENT LEARNING

Bellmanngleichung:

$$Q^{\pi}(s,a) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...|s,a]$$

Recursively:

$$Q^{\pi}(s,a) = \mathbb{E}_{s'}[r_{t+1} + \gamma Q^{\pi}(s^{'}, a^{'})|s,a]$$

Optimal value function:

$$Q^{*}(s,a) = \mathbb{E}_{s'}[r_{t+1} + \gamma Q^{\pi}(s^{'}, a^{'})|s,a]$$

Value iteration solve the Ballman Equation:

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}[r_{t+1} + \gamma Q_i(s^{'}, a^{'})|s,a]$$

Objective function by *mean-squared error* in Q-values:

$$L(w) = \mathbb{E}[(r + \gamma max_{a'} Q(s^{'}, a^{'}, w^{'} - Q(s,a,w))^2]$$

Q-learning gradient:

$$\frac{\delta L(w)}{\delta w} = \mathbb{E}[(r + \gamma max_{a'} Q(s^{'}, a^{'}, w^{'} - Q(s,a,w))\frac{\delta Q(s,a,w)}{\delta w}]$$

General TD-learning update rule:

$$Q(s_t, a_t) + = learning_rate \cdot (td_target - Q(s_t, a_t))$$

TD Target for SARSA:

$$R_{t+1} + discount_factor \cdot Q(s_{t+1}, a_{t+1})$$

TD Target for Q-Learning:

$$R_{t+1} + discount_factor \cdot maxQ(s_{t+1}, a_{t+1})$$

SARSA

$$Q(s_t, a_t) += \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

10 Generalisierung

$<\epsilon_{\check{D}est}> = <\epsilon_{train}> + 2 \cdot \sigma^2 \frac{p}{n}$ mit Varianz σ, Parameteranzahl p und Anzahl an Trainingsbeispielen n

11 Normalisierung

- Max-Min (Rescaling): $x' = \frac{x - min(x)}{max(x) - min(x)}$
- Standardisierung: $x' = \frac{x - \bar{x}}{\sigma}$
- Skalierung auf Einheitslänge: $x' = \frac{x}{||x||}$
- lückenhafte Daten: Null filling - Smoothing

12 Regularisierung

- L1 Norm: $||w||_{L1} = \sum_j |w_j|$
- L2 Norm: $||w||_{L2} = \sum_j w_j^2$
- KL-Divergenz:

$$KL(p||\widehat{p}_j) = p\log(\frac{p}{\widehat{p}_j}) + (1-p)\log(\frac{1-p}{1-\widehat{p}_j})$$

- Cross-Entropy: $E_{CE}(w) = -\sum_{x \in X} \sum_k [t_k^x * log(o_k^x) + (1 - t_k^x) * log(1 - o_k^x)]$
- Edit-Distance:
- Dropout:
- Meiosis: Idea: adding of hidden unis depends on the "uncertainty" of the network. The mean and varianz is learned.

$$w_{ij}^* = \mu(w_{ij}) + \sigma(w_{ij})\phi(0,1)$$

Start with one hidden unit and split unit if

$$\frac{\sum_i \sigma_{ij}}{\sum_i \mu_{ij}} > 1.0 \, and \, \frac{\sum_k \sigma_{ik}}{\sum_k \mu_{ik}} > 1.0$$

13 Adaptive Lernratenanpassung

- AdaGrad: $w_t = w_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} L(x, w_{t-1})$ mit der Diagonalmatrix G_t, die die Beträge des Gradienten enthält und ϵ: Smoothingterm, um Division durch 0 zu verhindern
- $w_t = w_{t-1} - \frac{RMS[\Delta w]_{t-1}}{RMS[g]_t} g_t$, wobei $RMS[\Delta w]_t$ der „root mean squared error" $\sqrt{E[\Delta w^2]_t + \epsilon}$ ist.
- RMSProp: $w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$

14 Updates für Backprop

- Momentum-Term: $\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}(t)} + \alpha * \Delta w_{ij}(t-1)$
- QuickProp:

$$\Delta w(t) = \frac{s(t)}{s(t-1) - s(t)} \cdot \Delta w(t-1)$$

- WeightElimination:

$$E = MSE + \lambda \sum_{i,j} \frac{w_{i,j}^2}{1 + w_{i,j}^2}$$

15 Autoencoders

Durchschnittliche Aktivierung von Sparse Autoencoder:

$$\widehat{p_j} = \frac{1}{|N|} \sum_{x \in X} w_j x$$

force this average activation to be p ≈ 0.2 by adding the Kullback–Leibler divergence $KL(p||\widehat{p_j})$ to the error function

16 Shared Weights bei TDNNs

1. $w_j^{t_1} = w_j^{t_2} \Rightarrow \Delta w_j^{t_1} = \Delta w_j^{t_2}$
2. Berechne $\frac{\partial E}{\partial w_j^{t_1}}$ und $\frac{\partial E}{\partial w_j^{t_2}}$
3. $\Delta w_j^{t_1} = \Delta w_j^{t_2} = -\eta \left(\frac{\partial E}{\partial w_j^{t_1}} + \frac{\partial E}{\partial w_j^{t_2}} \right)$

17 GEWICHTSUPDATE MIT cos

(Franzini)

$$\cos\Theta = \frac{\sum_{i,j}\left(\Delta w_{ij}(t-1)\cdot\Delta w_{ij}(t)\right)}{\sqrt{\sum_{i,j}\left(\Delta w_{ij}(t-1)\right)^2\cdot\sum_{i,j}\left(\Delta w_{ij}(t)\right)^2}}$$

$\cos\Theta = 1 \quad (\Theta = 0^o, 360^o, ...)$ $\qquad$ $\cos\Theta = -1 \quad (\Theta = 180^o)$

=> ε größer machen $\qquad$ => ε kleiner machen

$$\boxed{\varepsilon(t) = \varepsilon(t-1)\cdot \text{const}\cdot\frac{\cos\Theta+1}{2}}$$

18 LVQ

18.1 LVQ1

$c = argmin_i\{||x - m_i||\}$: Index des prototype vectors
Learning rules:

- $m_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$: x und m_c selbe Klasse
- $m_c(t+1) = m_c(t) - \alpha(t)[x(t) - m_c(t)]$: x und m_c unterschiedliche Klasse
- $m_c(t+1) = m_i(t)$: für $i \neq c$

18.2 LVQ2

Die Klassifizierung ist die selbe wie bei LVQ1. Das updaten ist anders:

- m_i und m_j sind die nähsten Nachbarn von x und werden simulaten geupdated.
- x muss in ein "window" um m_i und m_j fallen.
- d_i und d_j sind die Distanzen (z.B. euklidien) zwischen x und m_i und m_j
- $min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > s$ where $s = \frac{1-w}{1+w}$ (recommended window: 0.2 to 0.3)

Ergänzungen: m_i ist der Gewinner (falsche klasse) m_j zweiter gewinner, richtige klasse -> dann updaten α die Lernrate wird kleiner und kleiner