# Cognitive Neuroscience II

Prof. Dr. Andreas Wendemuth

Lehrstuhl Kognitive Systeme

Institut für Elektronik, Signalverarbeitung und Kommunikationstechnik

Fakultät für Elektrotechnik und Informationstechnik
Otto-von-Guericke Universität Magdeburg

http://iesk.et.uni-magdeburg.de/ko/

# Lecture 10 (Concludes Chapter 8)

- Function approximation
- Stochastic Learning
- Kullback-Leibler-Measure
- Expectation-Maximization (EM)

# Function approximation

- Match function h(s) which is not of the type of activation function
- Mathematically, expand h(s) into a set of known functions which act as a *function basis*
- Examples: Taylor expansion, Fourier expansion
- We are left with learning weights of the functional prototypes

# Network structure

Is

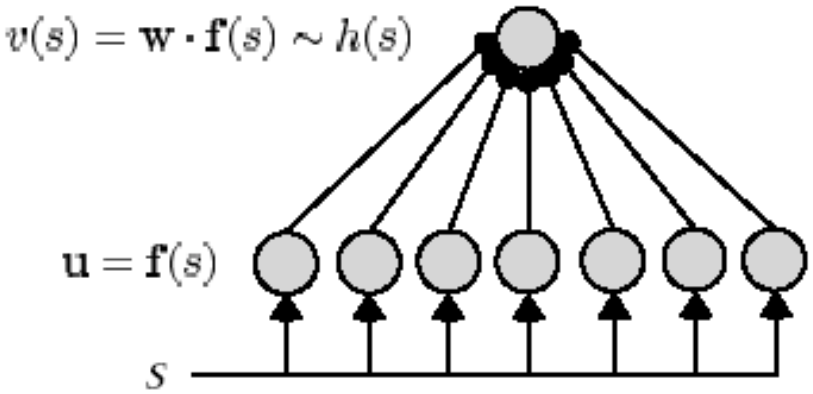


Figure 8.13: A network for representing functions. The value of an input variable $s$ is encoded by the activity of a population of neurons with tuning curves $\mathbf{f}(s)$. This activity drives an output neuron through a vector of weights $\mathbf{w}$ to create an output activity $v$ that approximates the function $h(s)$.

# Learning the weights

- Approximation: $v(s) = \mathbf{w} \cdot \mathbf{u} = \mathbf{w} \cdot \mathbf{f}(s) = \sum_{b=1}^{N} w_b f_b(s)\,.$
- Cost: $E = \frac{1}{2N_S} \sum_{m=1}^{N_S} \left(h(s^m) - v(s^m)\right)^2 = \frac{1}{2} \left\langle \left(h(s) - \mathbf{w} \cdot \mathbf{f}(s)\right)^2 \right\rangle .$
- Hebbian rule: $\mathbf{w} = \langle \mathbf{f}(s) h(s) \rangle / \alpha .$

  or indeed other known types of rules.
- Can be interpreted as input tuning curves f(s) for generating a suitable (overcomplete) basis.

# Example for function approx.:

- Learning a sine:

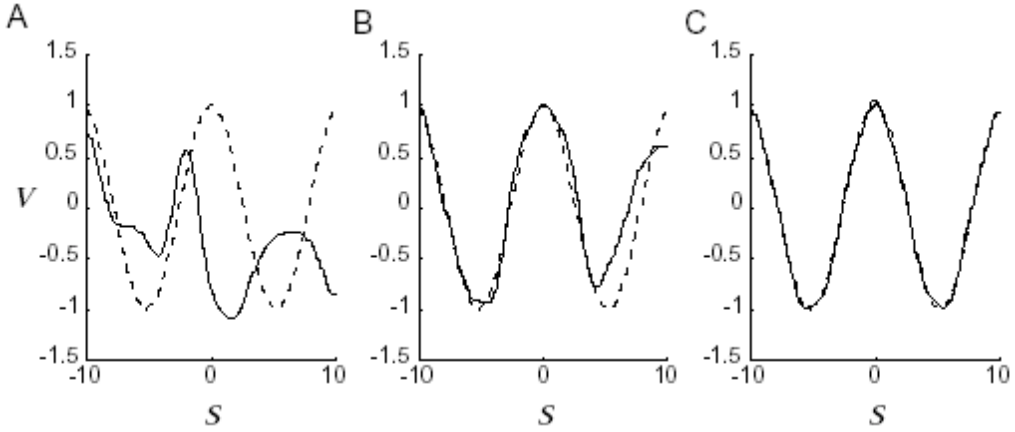


Figure 8.14: Eleven input neurons with Gaussian tuning curves drive an output neuron to approximate a sine function. The input tuning curves are $f_b(s) = \exp[-0.5(s - s_b)^2]$ with $s_b = -10, -8, -6, \ldots, 8, 10$. A delta plasticity rule was used to set the weights. Sample points were chosen randomly in the range between -10 and 10. The firing rate of the output neuron is plotted as a solid curve and the sinusoidal target function as a dashed curve. A) The firing rate of the output neuron when random weights in the range between -1 and 1 were used. B) The output firing rate after weight modification using the delta rule for 20 sample points. C) The output firing rate after weight modification using the delta rule for 100 sample points.

# Ex 8: Function Approximation

- In the above example (learning a sine), what happens outside of the displayed range?
- Can neurons learn a periodic function on an unlimited time range?
- What is an overcomplete basis?
- If a neural network does a function approximation with an overcomplete basis, what happens? Is this harmful?

# Stochastic Learning

- Central to Stochastic Learning is the concept of matching probability distributions.
- We consider the input-output relation (to be realized by a network) as *drawn* from a distribution P[**v**|**u**].
- The true network realization, however, depends on the actual weights and realizes P[**v**|**u, W**].
- Obviously, the task is to choose the weights such that P[**v**|**u, W**] and P[**v**|**u**] match as best as possible.

# Probabilistic Rules

- Recall from the Boltzmann machine that a neuron *a* is selected at random. Then its state is changed *stochastically* according to:
- Gibb‘s sampling: The neuron with total input activation $I_a(u)$ is active ($v_a=1$) with *probability*

$$P(v_a = 1) = \frac{1}{1 + \exp(-I_a(u))}$$

# Markov Chains

- Obviously, the update rule only depends on the *present* state of the input activation. The dynamics is *memory-less*.
- This defines a Markov Process, the corresponding train of states are a *Markov Chain*.
- Since neuron selection and update rule are stochastic, the dynamics does not converge to a fixed point $\mathbf{v}_\infty$, but to a limit distribution $P_\infty[\mathbf{v}]$.

# Limit distribution

- The limit distribution for the vector (!) **v** is given by $P(\mathbf{v}) = \frac{\exp(-\mathbf{I} * \mathbf{v})}{Z}$

with the partition function (Z= Zustandsumme)

$$Z = \int_{\mathbf{v}} \exp(-\mathbf{I} * \mathbf{v}) d\mathbf{v} \quad \text{or} \quad Z = \sum_{\mathbf{v}} \exp(-\mathbf{I} * \mathbf{v})$$

# Likelihood

- Sample the vector **v** component-wise, independently for each component $v_a$. Then the probability for a particular output **v** is given by the *likelihood*

$$Q(\mathbf{v}) = \prod_{a=1}^{N} Q(v_a) = \prod_{a=1}^{N} P(v_a = 1)^{v_a} P(v_a = 0)^{1-v_a}$$

or

$$Q(\mathbf{v}) = \frac{P(v_a = 1)^{N(v_a=1)}}{(1 - P(v_a = 1))^{N(v_a=1)-N}}$$

# Log-Likelihood

- Is given by

$$\log Q(\mathbf{v}) = N(v_a = 1) \log P(v_a = 1) + (N - N(v_a = 1)) \log(1 - P(v_a = 1))$$

or $\log Q(\mathbf{v}) =$

$$N(v_a = 1) \log \frac{1}{1 + \exp(-I_a(u))}$$

$$+ (N - N(v_a = 1)) \log \frac{\exp(-I_a(u))}{1 + \exp(-I_a(u))}$$

or

$$\log Q(\mathbf{v}) = -N \log(1 + \exp(-I_a(u))) - I_a(u)(N - N(v_a = 1))$$

# …

- Or

$$\frac{1}{N}\log Q(\mathbf{v}) = -\log(1+\exp(-I_a(u))) - I_a(u)\,p(v_a = 0)$$

- If only recurrent weights, $I_a(u) = \mathbf{M} * \mathbf{v}_a$ and

$$\frac{1}{N}\log Q(\mathbf{v}) = -\log(1+\exp(-\mathbf{M} * \mathbf{v}_a)) - \mathbf{M} * \mathbf{v}_a\, p(v_a = 0)$$

# Analysis

- Derivative

$$\frac{1}{N}\frac{d\log Q(\mathbf{v})}{d\mathbf{M}} = \mathbf{v}\frac{\exp(-\mathbf{M}*\mathbf{v})}{1+\exp(-\mathbf{M}*\mathbf{v})} - \mathbf{v}p(v_a = 0)$$

or

$$\frac{1}{N}\frac{d\log Q(\mathbf{v})}{d\mathbf{M}} = \mathbf{v}P(v_a = 0) - \mathbf{v}p(v_a = 0)$$

- Watch closely!: At maximum, posterior = prior probability.
- Derivative is in **v**-direction, so with gradient descent, weights are updated Hebbian-style.

**Beispiel 1.1 (Likelihood von Studenten):**
Ein Vorlesung über Computational Neuroscience wird im Mittel von 30% Frauen und 70% Männern gehört. Da es sich um eine diskrete Verteilung mit nur 2 Ausgängen handelt, reicht ein Parameter, z.B. $\theta = P(\text{Hörer=männlich}) = 0.7$, um die Verteilung zu beschreiben.

Uns liegt nun eine Stichprobe von 20 Hörerinnen und 10 Hörern vor. Es ist mit dem „wahren“ $\theta$

$$L(\theta|X) = (0.3)^{20} \cdot (0.7)^{10}$$

bzw.

$$L_{\log}(\theta|X) = 20\log(0.3) + 10\log(0.7) = -27.65.$$

Unter Maximum-Likelihood würden wir aufgrund unserer Stichprobe allerdings optimieren und folgendes $\theta^*$ mit Hilfe von Ableitungen finden:

$$\theta^* = \max_\theta L_{\log}(\theta|X) = \max_\theta(20\log(1-\theta) + 10\log(\theta))$$
$$0 = \frac{d}{d\,\theta} L_{\log}(\theta|X) = -20\frac{1}{1-\theta^*} + 10\frac{1}{\theta^*}$$
$$\theta^* = 1/3$$

Dies entspricht dem Ergebnis, welches ohne Rechnung direkt aus dem Anteil der Männer (1/3) in der Stichprobe abgelesen werden kann.

Die maximale Likelihood $L^*$ wird damit *aufgrund der Stichprobe !*

$$L^*_{\log}(\theta^*|X) = 20\log(2/3) + 10\log(1/3) = -19.09$$

und dies ist in der Tat größer als die Likelihood gemäß der wahren Verteilung. □

Im obigen Beispiel wurde die wahre Verteilung mit der ML-Methode nicht korrekt ermittelt. Grund dafür ist natürlich, dass die Stichprobe deutlich andere Geschlechterverhältnisse hat als die wahre Verteilung. Wir erkennen daran, dass für die Anwendung von ML eine repräsentative Stichprobe vorliegen muss.

# Ex 9 - Likelihood

- In a US election, 60% Republicans and 40% Democrats were elected.
- If this is known, what is the Log-Likelihood of an election result of a city, where 5000 citizens voted Republicans and the same number of citizens voted Democrats?
- With this city‘s election result, maximize the likelihood with respect to the distribution of votes which now is regarded as *unknown*. What is the optimal distribution of votes? What is the best Likelihood? Why is there a difference to the Likelihood we computed above? What is the best possible likelihood at all?

# Kullback-Leibler-Statistics, Expectation-Maximization (EM)

- See handout

# Matching Distributions?

- After having leart W, conditional distribution

$$P[\mathbf{v}|\mathbf{u};\mathbf{W}] = \frac{\exp(-E(\mathbf{u},\mathbf{v}))}{Z(\mathbf{u})} \quad \text{with} \quad Z(\mathbf{u}) = \sum_{\mathbf{v}} \exp(-E(\mathbf{u},\mathbf{v}))$$

where $E(\mathbf{u},\mathbf{v}) = -\mathbf{v}\cdot\mathbf{W}\cdot\mathbf{u}$.

- The Kullback-Leibler distance is

$$D_{\mathrm{KL}}(P[\mathbf{v}|\mathbf{u}], P[\mathbf{v}|\mathbf{u};\mathbf{W}]) = \sum_{\mathbf{v}} P[\mathbf{v}|\mathbf{u}] \ln\left(\frac{P[\mathbf{v}|\mathbf{u}]}{P[\mathbf{v}|\mathbf{u};\mathbf{W}]}\right)$$

$$= -\sum_{\mathbf{v}} P[\mathbf{v}|\mathbf{u}] \ln\left(P[\mathbf{v}|\mathbf{u};\mathbf{W}]\right) + K,$$

...

- Or, after selecting examples from P

$$\langle D_{\mathrm{KL}}(P[\mathbf{v}|\mathbf{u}],\, P[\mathbf{v}|\mathbf{u};\mathbf{W}])\rangle = -\frac{1}{N_{\mathrm{S}}}\sum_{m=1}^{N_{\mathrm{S}}} \ln\left(P[\mathbf{v}^m|\mathbf{u}^m;\mathbf{W}]\right) + \langle K\rangle$$

- This is equivalent to maximizing likelihood.

# Ex 10

- Regard Ex 9 again. In the two situations given, now maximize the Kullback-Leibler-Statistics.
- Which results are obtained?

# Learning rule

- Boltzmann machine:

$$\frac{\partial \ln P[\mathbf{v}^m|\mathbf{u}^m;\mathbf{W}]}{\partial W_{ab}} = \frac{\partial}{\partial W_{ab}}\left(-E(\mathbf{u}^m,\mathbf{v}^m) - \ln Z(\mathbf{u}^m)\right)$$

$$= v_a^m u_b^m - \sum_{\mathbf{v}} P[\mathbf{v}|\mathbf{u}^m;\mathbf{W}] v_a u_b^m .$$

- Or with the generated output:

$$W_{ab} \rightarrow W_{ab} + \epsilon_w \left(v_a^m u_b^m - v_a(\mathbf{u}^m) u_b^m\right) .$$

# Resumé of Chapter 8 (Lectures 6-10)

- Plasticity and Learning / Hebb‘s rule
- Unsupervised Learning: Hebb, Covariance Rule, PCA, Oja Rule, Subtractive Normalization:
  - Example: Ocular Dominance
- Feature learning, Self-Organizing Maps
- Supervised Learning: Perceptron, robust Perceptron
  - linear Separability, capacity
  - delta-Rule (Adaline, Adatron)
- Function Approximation, Stochastic Learning, Likelihood, Kullback-Leibler-Distance, EM