Statistical Machine Learning (BE4M33SSU)
Lecture 9.

Czech Technical University in Prague

- Hopfield networks: asynchronous dynamics and energy minimisation
- Hopfield networks: weight learning
- Graphical models and energy minimisation
- Submodular minimisation, equivalence to MinCut-MaxFlow

9.1 Hopfield networks

Hopfield (1982): Consider a fully connected network of n binary valued neurons

$$y_i = \mathrm{sign}\Big(\sum_{j \neq i} w_{ij}\, y_j - b_i\Big)$$

Assumptions:

- symmetric weights, i.e. $w_{ij} = w_{ji}$, $\forall i, j$,
- no neuron has a connection to itself, i.e $w_{ii} = 0$, $\forall i$.

Asynchronous dynamics:

Only one neuron is updated at a time. E.g. by picking them at random or in some pre-specified order.

Q: Will the network forever cycle through its state space if started in some particular state?

9.1 Hopfield networks

Energy: Each state $\boldsymbol{y} \in \{-1,1\}^n$ of the network is characterised by a real number called energy

$$E(\boldsymbol{y}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\, y_i\, y_j + \sum_{i=1}^{n} b_i\, y_i = -\frac{1}{2}\langle \boldsymbol{y}, \boldsymbol{W}\boldsymbol{y}\rangle + \langle \boldsymbol{b}, \boldsymbol{y}\rangle,$$

where $\boldsymbol{W}$ denotes the matrix of weights (symmetric, zero diagonal elements) and $\boldsymbol{b}$ denotes the vector of thresholds.

Theorem: *A Hopfield network with n units and asynchronous dynamics, which starts from any given network state, eventually reaches a stable state at a local minimum of the energy function.*

Proof: Consider the update of a neuron, assume it is unit k, i.e. $\boldsymbol{y'} = (y_1, \ldots, y'_k, \ldots, y_n)$:

$$y'_k = \mathrm{sign}\Big(\sum_{j \neq i} w_{ij}\, y_j - b_i\Big) \neq y_k$$

Denoting the activation by a_k, we consequently have $y'_k a_k > 0$ and $y_k a_k < 0$.

Considering all affected terms in the energy, we have

9.1 Hopfield networks

$$E(\boldsymbol{y}) - E(\boldsymbol{y'}) = -(y_k - y'_k)\Big[\sum_{j=1}^{n} w_{kj} y_j - b_k\Big] > 0$$

This shows that the energy is reduced each time the state of a unit is altered. The assertion follows, because the state space of the network is finite. □

Hopfield networks can be used as *auto-associative memory* for storing binary patterns!

Q: Given a set of patterns $\boldsymbol{y}^\ell$, $\ell = 1, \ldots, m$ which we want to store, how shall we choose the weights $\boldsymbol{W}$ and thresholds $\boldsymbol{b}$?

A1: Hebbian learning:

$$w_{ij} = \frac{1}{m} \sum_{\ell=1}^{m} y_i^\ell y_j^\ell \text{ for } i \neq j \text{ and } b_i = -\frac{1}{m} \sum_{\ell=1}^{m} y_i^\ell$$

A2: Perceptron learning: cycle through $\ell = 1, \ldots, m$ and $k = 1, \ldots, n$. If for some ℓ, k

$$y_k^\ell \neq \operatorname{sign}\Big(\sum_{j \neq k} w_{kj}\, y_j^\ell - b_k\Big),$$

update $w_{kj} \to w_{kj} + y_k^\ell y_j^\ell$ and $b_k \to b_k - y_k^\ell$.

9.1 Hopfield networks

How many binary patterns can be stored in a network with n units? On average $2n$ random patterns.

So far considered fix-points and learning conditions - local minima of the energy.

Critical questions:

- Are there polynomial time algorithms for computing global minima of the energy of a Hopfield network? No, the task is NP-complete in general.
- Are there learning algorithms s.t. the patterns are stored as global minima? No, not in general.

9.2 Graphical models (structured output predictors)

Structured output predictors

- Graph (V, E) and label alphabet K
- A labelling $\boldsymbol{y}\colon V \to K$ assigns to each node $i \in V$ a label $y_i \in K$
- Measurements: a feature x_i for each node $i \in V$
- Predictor

$$\boldsymbol{y}^* = \arg\min_{\boldsymbol{y}} \Big[\sum_{ij \in E} g_{ij}(y_i, y_j) + \sum_{i \in V} q_i(y_i, x_i) \Big]$$

where g_{ij} and q_i are functions associated with the edges and nodes of the graph.

Remarks

- Such energy minimisation problems are also called (Min,+)-problems,
- The class of (Min,+)-problems is NP-complete (MaxClique)
- There are tractable subclasses of (Min,+)-problems.
  - (Min,+)-problems are solvable in polynomial time if the graph (V, E) is acyclic
  - (Min,+)-problems are solvable in polynomial time for submodular functions
- There are efficient approximation algorithms for (Min,+)-problems

9.2 Graphical models (structured output predictors)

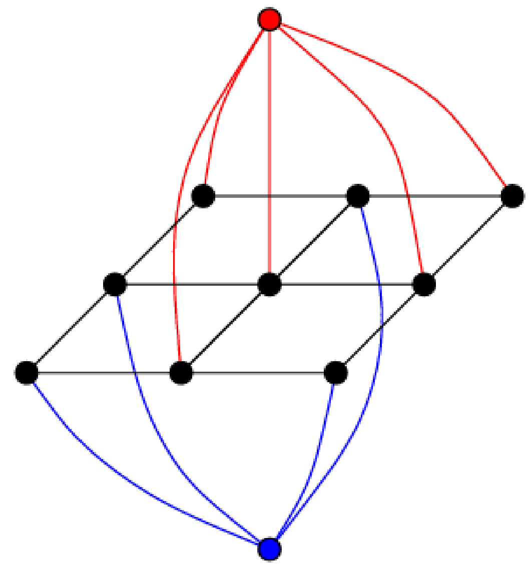
A tractable subclass of (Min,+)-problems for $|K| = 2$

- w.l.o.g. $K = \{0, 1\}$, $y_i = 0, 1$ and $g_{ij}(y_i, y_j) = \alpha_{ij}|y_i - y_j|$

$$\begin{aligned}\boldsymbol{y}^* &= \arg\min_{\boldsymbol{y}} \Big[\sum_{ij \in E} \alpha_{ij}|y_i - y_j| + \sum_{i \in V} q_i y_i \Big] \\ &= \arg\min_{\boldsymbol{y}} \Big[\sum_{ij \in E} \alpha_{ij}|y_i - y_j| + \sum_{i \in V_+} q_i y_i + \sum_{i \in V_-} |q_i|(1 - y_i) \Big]\end{aligned}$$

where $V_+ = \{i \in V \mid q_i \geqslant 0\}$, $V_- = V \setminus V_+$.

This is a **MinCut-problem**!



9.2 Graphical models (structured output predictors)

MinCut problems

- Let (V, E, w) be an undirected, weighted graph, where $w\colon E \to \mathbb{R}$.
- $s, t \in R$ two fixed vertices (called source and target)
- (s,t)-cut: Partition of vertices $V = V_1 \cup V_2$ such that $s \in V_1$, $t \in V_2$
- Cost of an (s,t)-cut

$$C(V_1, V_2) = \sum_{i \in V_1} \sum_{j \in V_2} w_{ij}$$

- MinCut: Find an (s,t)-cut with minimal cost

Can be expressed as an integer optimisation task by assigning to each vertex $i \in V$ a binary variable $y_i = 0, 1$

Each MinCut-problem with non-negative edge weights is equivalent to a linear optimisation problem. Its dual is a **MaxFlow-problem**

9.2 Graphical models (structured output predictors)

MaxFlow problems

- Let (V, E, w) be an undirected, weighted graph, where $w \colon E \to \mathbb{R}_+$.
- $s, t \in V$ two fixed vertices (called source and target). Fix an orientation for each edge.
- (s,t)-Flow: a map $f \colon E \to \mathbb{R}$ with convention $f_{ij} = -f_{ji}$ such that $\forall i \neq s, t$

$$\sum_{j:(j,i)\in E} f_{ji} + \sum_{j:(i,j)\in E} f_{ij} = 0$$

- Feasible flow: $0 \le f_{si} \le w_{si}$, $0 \le f_{it} \le w_{it}$ and $|f_{ij}| \le w_{ij}$.
- Value of a feasible (s,t)-flow f:

$$V(f) = \sum_{i:(s,i)\in E} f_{si} = \sum_{j:(j,t)\in E} f_{jt}$$

- MaxFlow problem: find a feasible flow with maximal value.
- MaxFlow problems can be solved in polynomial time.