1 Biological Motivation

Why should someone try and learn dependencies between entities? We will try to answer this question by a couple of biological examples:

- **Splice Junctions**: Splice junction can be represented by a PSSM. After we learn the dependencies between different positions in the sequence we will first, understand the biology better, and second, be able to predict previously unknown splice variants based on our junction predictions.

- **Proteins**: There are known structures, like $\alpha$-helix, in which each 7 amino acids complete 2 cycles, and in these cases we will search for dependencies in constant distances between amino acids. Learning dependencies in $\alpha$-helix is complicated, because the alphabet is large - more than 400 parameters to estimate. Many samples are needed for the parameters estimation.

2 Problem Definition

Intuitively, given data $D$, we need to build “the best graph” $G$ in some sense. But what is “the best graph”? Suppose, $D$ was sampled from a real distribution $P^*$. Then we want the distribution $P_{\text{learned}}$, which is represented by the graph $G$, to be a good approximation of the real distribution. But in real life we rarely know the real distribution $P^*$. One solution is dividing $D$ to train set and test set. In such a way the test set will help us approximating $P^*$.

The likelihood of the graph $G$ and its parameters $\Theta$ is:

$$l(G, \Theta : D) = \sum_m \log P(x[m] : G, \Theta)$$

We built the graph by maximizing its likelihood. For testing the graph we should look at the following expression:

$$l(G, \Theta : D_{\text{test}}) = |D|E_{P_{\text{test}}}(\log P(x; G, \Theta))$$

If the test set is big enough - we will get a good approximation of $P^*$. Note: here we used the following approximation of expectation:

$$E[f(x)] \approx \frac{1}{M} \sum_{m=1}^{M} f(x_m)$$

To summarize, our goal is to learn a graph that represents a distribution $P_{\text{learned}}$ close as possible to the desired distribution $P^*$.

3 Two approaches for building a graph $G$

3.1 Minimum Description Length

The intuition comes from Information theory: if we want to maximize the likelihood, how many bits do we need to code any sample? The answer from the previous lesson is $\log_2 P(x)$, but the problem is that we don’t know the real
distribution $P$. Let’s define graph and parameters likelihood:

$$-l(G, \Theta : D) = \{\text{number of bits to send D if G and } \Theta \text{ are known}\}$$

and penalty for coding complexity ($DL = \text{description length}$):

$$DL(G) + DL(\Theta) = \{\text{number of bits to describe G, } \Theta\}$$

Now we need to think how to code the graph and the parameters. Instead of maximizing the likelihood only, we will add the penalty for complexity and solve a bit different problem:

$$\min_{G, \Theta} DL(G) + DL(\Theta) - l(\Theta, G : D) = \max_{G, \Theta} l(\Theta, G : D) - (DL(G) + DL(\Theta))$$

We still want to maximize the likelihood, but now we set constraints on $G$ and $\Theta$ in order to stay with compact representation. The meaning is that it will cost us to make the model more complex. The most important factor is $DL(\Theta)$ - for each edge in the graph, we will add many parameters. The likelihood grows linearly with number of samples: $l = M \cdot E_p(\log(P_x))$, while $DL(\Theta)$ grows in $\log M$ (the number of bits required to describe the model is $|G| \cdot \frac{1}{2} \log M$, where $|G|$ is the number of parameters). The inter-play will be between these two terms, the penalty will stop us from adding complexity based on train data.

### 3.2 Online Coding

#### 3.2.1 Problem definition

The sender and the receiver agree on some initial distribution $P_0$. The sender sends message $x[1]$, coded based on $P_0$. The sender and the receiver update the distribution to $P_1$. Afterwards, the sender sends a new sample $x[2]$, coded based on current distribution $P_1$, and they both update current distribution to $P_2$, etc. The goal is to keep coding as compact as possible.

#### 3.2.2 Score

In this case, what is actually maximized is:

$$- \sum_m \log P(x[m] | x[1], \ldots, x[m-1])$$

This can also be seen as simulation of generalization error: each new sample is handled as test case. Recall the calculation of probability for a new sample given all the samples till now:

$$P(x[m] | x[1], \ldots, x[m-1]) = \text{Bayesian integration} = \int_\Theta P(x[m] | \Theta)P(\Theta | x[1], \ldots, x[m-1])d\Theta$$

On the other hand:

$$- \sum_m \log P(x[m] | x[1], \ldots, x[m-1]) = - \log \prod_k P(x[k] | x[1], \ldots, x[k-1])$$

Chain Rule

$$= - \log P(x[1], \ldots, x[m]) = - \log (\text{Marginal Likelihood})$$

We can conclude from this, that the samples order doesn’t matter. So what is the difference from Batch approach? First, we don’t handle directly parameter coding. Second, there is direct estimation of generalization error.

Number of bits in online coding (the score we would like to minimize): $- \log P(x[1], \ldots, x[m] | G)$.

If we were Bayesian, we would like to maximize $P(G | x[1], \ldots, x[m]) \sim P(G)P(x[1], \ldots, x[m] | G)$, while $P(G)$ is the prior for graph $G$. If we want to incorporate the graph complexity in this score - we can add $DL(G) = - \log P(G)$ to number of bits in online coding.
### 3.2.3 Example: Single Coin

Marginal likelihood:

\[
P(x[1], \ldots, x[m])_{\text{Bayesian}} = \int_{\Theta=0}^{1} P(x[1], \ldots, x[m] \mid \Theta) P(\Theta) \, d\Theta
\]

\[
= \int_{0}^{1} \Theta^{N_H} (1 - \Theta)^{N_T} \, d\Theta
\]

\[
= \frac{1}{M+1} \left( \frac{M}{N_H} \right)^{-1} \approx 2^{H(N_H/M)} - \frac{1}{2} \log M
\]

### 3.3 Intermediate Summary

We discussed two scores for probabilistic models, which take structure into account:

1. **MLE with a penalty on structure complexity**
2. **Marginal likelihood with a prior on the graph structure, \(G\)**

The last example shows that for a uniform prior the two scores are asymptotically similar. This is true also for multinomial distributions with a Dirichlet prior (See Exercise 5 question 4). Practically, when the number of data samples is small, the Bayesian approach is better than MDL.

### 4 Structure Learning

Once we defined a score function for a Bayesian network with a graph \(G\) and parameters \(\Theta\) we would like to search for the model which maximizes this score. This task is not trivial since the number of possible graphs is exponential in the number of the random variables of the model. So how do we search for the best model?

We will describe a search procedure that uses MDL score. The same arguments are valid for the Online method when choosing an appropriate prior distribution for the network.

#### 4.1 Structured Representation of the Score Function

In order to cope with the search task we shall rewrite the log-likelihood of a Bayesian network:

\[
l(G, \Theta : D) = \sum_m \log P(x|m) | G, \Theta
\]

\[
= \sum_m \sum_i \log P(x_i|m) | \text{pa}_i[m] : \Theta_i
\]

\[
= \sum_i \left( \sum_m \log P(x_i|m) | \text{pa}_i[m] : \Theta_i \right)
\]

where \(\Theta_i\) is the set of parameters associated with the conditional distribution of \(X_i\) given his parents. When these parameters are not shared between the families we can write the last term as a sum of independent local-likelihood terms:

\[
l(G, \Theta : D) = \sum_i l_{X_i|\text{pa}_i}^\Theta (\Theta_i : D)
\]

taking the maximum, we have

\[
\max_{\Theta_i} l_{X_i|\text{pa}_i}^\Theta (\Theta_i : D) = \sum_i \max_{\Theta_i} l_{X_i|\text{pa}_i}^\Theta (\Theta_i : D)
\]
Recal that the score function is the log-likelihood minus the description length of the model, $DL(\Theta)$. The description length is also a sum of local terms, since for each variable $X_i$ we need $|\Theta_i| \cdot \log M$ bits to describe its conditional distribution. Using the last two observations we can rewrite the score as a sum of local weights:

$$DL(G, D) = \sum_i w_{X_i | Pa_G}$$

(1)

Thus our score has the nice property in which a local change in the structure is bound to a local change in one of the local terms. This is true also for a Bayesian score when choosing an appropriate prior distribution.

### 4.2 Searching the best Structure

The task of searching the graph $G$ which has an optimal score according to Equation (1) defines the following combinatorial problem:

*Suppose we are given a set of random variables $X$ and a set of weights:*

$$\{ w_{X | U} : X \in X, U \subset X, X \not\in U \}$$

*find a DAG $G$ that maximizes*

$$\sum_i w_{X_i | Pa_G}$$

This problem is NP hard. Intuitively, this is a consequence of the global constraint imposed on the graph — we require that the graph would not contain any cycles. As we saw in a previous lesson, when we constrain the structure of the graph to be a tree, the problem is tractable. The difference is that a tree structure impose a local constraint on the graph. Namely, each node should have a single parent. So, can we come up with an easier problem if we impose an additional constraint on the DAG? Suppose for example that we constrain the in-degree of every edge to be smaller than two. Unfortunately this problem is still NP hard. Limiting the out-degree would not help either.

A constraint that renders the problem into a tractable one is to impose an order relation on the variables and a constant number of maximum parents. The solution for such an optimization task is to use a greedy algorithm that chooses the parents of $X_i$ from $X_1, \ldots, X_{i-1}$ in polynomial time. However, there are $n!$ possible orderings. So, how can we choose a good one? In some cases the domain of our model dictates a natural order - for example in case of temporal sequences.

In the general case we can not assume an ordering. Therefore, we resort to heuristic search algorithms.

### 4.3 Greedy Search

In each step of a greedy search we test the effect of some local operations such as addition, deletion or reversal of edges and we choose the operation that results in the largest score improvement. This is a reasonable idea because a small change in the structure results in a small change in the score.

Since there are more graphs than orderings of the random variables, we can take a possible approach of searching in the ordering space. We need to define an operator of switching orders and calculate the optimal score of the optimal graph with a specific order. Note that if we switch a pair of variables in the order we need to consider only the score from $X_1$ up to the children of the couple that was switched. Even with this relaxation, calculating the score in each space requires a considerable amount of computation.

### 4.4 Escaping Local Maxima

A greedy search stops when a local maximum is reached. In order to avoid this, we can use one of the following strategies:

- Perturbations in the search steps
- Random restarts of the initial search points
Using a Tabu List

The intuition of this interesting strategy arises from optimization of continuous functions. If we had a continuous score function we could escape a local maximum by moving down in a spiral course until we reach another point from which we could climb into another hill.

In this method we first search for a local maximum using a greedy DFS search. When we reach the local maximum we perform a variant of BFS in which a queue of the last $K$ nodes is stored. In every search step we will examine the best alternative which is not included in the $K$ nodes that were visited previously. The search is terminated if there was no improvement in a significant number of steps. The list of $K$ nodes is termed a Tabu list.

This strategy is not efficient since we need to store graph structures in the queue. Therefore we can use another variant in which we store the last $K$ operators. In the next search steps the operation of these operators is not overridden. For example if we added a specific edge, we will not allow to remove it in the next $K$ steps. The Tabu list in this variant covers a large territory in the graph space.

The second variant is a good strategy for avoiding local maximum in cases that a score function is “wrinkeled” or forms a plateau around the local maximum. Empirically using a Tabu list results in a dramatic improvement compared to stochastic search.