**Overall Script ~ 35 – 40 min.**

**Introduction Slide: 1 minute. (1)**

So first of all, can everybody clearly see the screen, and am I audible in the back as well? Okay then, let’s get started. Hello everybody and welcome to this special day for me. My name is Martin de Quincey, and this is the final presentation of my master’s thesis, on which I have been working on for the past year. If you have any short questions during the presentation, feel free to raise your hand. The more in-depth questions, we can discuss after the presentation. So, the name of the thesis and the presentation is ``structure learning in high-dimensional time series data”.

**Outline: 1 minute. (2)**

So, what will we be discussing in these 30 minutes. First, I will explain what is structure learning on a more conceptual level and provide an application where structure learning is useful. Then, we will be formalizing the concepts more rigorously.

Then, we will discuss some of the methodologies that we have developed during this project with some small examples that showcase the inner workings of the algorithms. Then, we will be applying these methods on both simulated and real-life data and we will compare the weaknesses and strengths of our proposed methods.

Lastly, we will list our conclusions and provide directions for future work.

**Introduction: Sprinkler Example: 3 minutes. (5)**

Okay, so let us get started. So, the overall goal of this thesis was to develop methodologies to learn the *structure* of some observational dataset consisting of time series data. Time series data means that we have measured variables over some time segment, and there is most likely dependency between these variables over time. We assume that this time series data has been generated by some graphical model, and we would like to recover the structure of this graphical model. Now, what is such a graphical model?

To understand the notion of a graphical model, let us consider this toy example on five variables. Suppose we have gathered measurements twice a day for a full year of these five variables to gain deeper insights into how the seasonality affects the slipperiness of your pavement. For this, we have gathered data of these five variables; the seasonality, the rainfall, whether your sprinkler installation was on, the wetness of your pavement, and the slipperiness of your pavement. We now would like to learn from this observational data or time series, the structure of the graphical model. We create this structure by drawing an arc from variable *x* to variable *y* if the past of *x* is useful in predicting *y*.

From real-life, we are already able to infer this graph structure. For example, we know that the season greatly influences the sprinkler’s uptime, as you water your garden more in the Summer than in the Winter. Furthermore, as turning on the sprinklers does not change the season, we know the arc should go from season to sprinkler. Additionally, the seasonality will also affect the rainfall. This already yields this structure of the upper part of the graphical model. Furthermore, both the rain and the sprinkler will make the pavement wet. Note that although the season influences the wetness of the pavement, we have already fully captured this relation through the rain and sprinkler variable. Lastly, the wetness of the pavement influences the slipperiness of the pavement.

Another crucial aspect of this thesis is to enforce that the inferred structure is *acyclic*, meaning it must not contain cycles. In other words, there does not exist a path of directed edges such that we can start and end in the same node.

One of the main reasons to enforce acyclicity is to make sure that the graphical model is easy to interpret with less cluttering arcs, and also to reduce the search space of possible structures. So, from a high-level perspective, to learn such an acyclic graphical model from time series data, that is the goal of this master’s thesis.

**Motivation: Root Cause Analysis: 3 minutes. (8)**

Now, let us consider a setting in which such a learned structure can be very helpful. Consider a complex system, for example a car, consisting of seven interacting components. We can accompany each component with a sensor, which can be used to verify whether the component is broken. This yields time series data, which we can use to learn the structure.

To highlight the usefulness of having a learned structure, suppose that something breaks down in the car, and the sensors accompanying the exhaust, real axle, and engine now produce anomalous data. It is reasonable to assume that only one component has broken, and that this causes the other two components to also exhibit unusual behavior. So, the question is, where is the *root cause* located? Does anybody here dare to make a guess?

It is in fact impossible to locate the root cause if you have no idea of the structure of the variables, how the variables interact. So, suppose we have learned using our time series data that *this* is the correct structure that captures how the components in the car interact. Note that I am not an automotive student, so this structure might not be a sensible structure, but for the sake of the argument, assume it is. The battery influences the central unit and the engine, the central unit then influences the gearbox, who in its turn affects the engine and two axles. Now, suppose this is a rear wheel drive car, such that the engine and the gearbox influence the rear axle, which lastly interacts with the exhaust. So, if we now know this structure, does anybody have an idea of which component is broken down?

It is quite likely that some part of the engine has broken down, and because of this structure, it could very well be that this malfunction has caused the two other dependent components to exhibit unusual behavior. So, therefore, we see that we can use such a learned structure to perform a root cause analysis on a complex system. We do not need to tear the whole machine apart to locate the root cause, and we can use a principled approach to locate the root cause rather than trying some components arbitrarily. Here we also see why acyclicity is useful. If we had cycles, then it would be not be possible to pinpoint where the root cause in a cycle might be.

**Formal Problem Setting 1 1 minute. (9)**

Now, let us make this notion of structure learning more formal. How will we be learning this structure? For this, we first assume that we have a data matrix X, consisting of *T* real-valued measurements of our *p* variables. To learn such a model, we make some assumptions that yield the following graphical model, which is called a Vector AutoRegressive model of order 1.

Here, X\_t corresponds to the measurements at time step t, and *W* here is the important coefficient matrix that encapsulates the structure of the graphical model. Furthermore, we assume that epsilon here is some random Gaussian noise, which captures the error in our estimates. Then, we assume that X\_t is some linear combination of the values of the previous timestep, where these linear relations are described in this coefficient matrix W.

These assumptions are also listed here below. We see that X\_t only does not depend on all past values but only on the previous time step t – 1. Furthermore, we assume that the relation between X\_t and X\_{t-1} is linear, and these linear relations are captured in this coefficient matrix W.

**Formal Problem Setting 2: 1 minute (10).**

So, let us explain the correspondence between this coefficient matrix W and the structure of the graphical model. From the previous slide, we saw that this W captures the linear relations between the measurements at time step t and the measurements at time step t – 1.

Now, the coefficient wij is equal to zero if and only if the arc (i, j) is contained in the structure. Therefore, the so-called support, or non-zero entries of W, characterize the structure of the graphical model. To explain this using an example, consider the graphical model of the first example about our front lawn. We see that five arcs are contained in the structure, which correspond to five non-zero entries in the corresponding coefficient matrix W. These non-zero entries can be any non-zero value, but for simplicity, we have chosen the value 1.

Now, when we say that the coefficient matrix W is acyclic, we technically mean that the graphical model corresponding to this coefficient matrix W is acyclic.

**Formal Problem Setting 3: 1 minute (11)**

So, let us return to this model that we have assumed to have generated our data X. Given such a data matrix X that we assume has been generated by this model, the goal is to find the acyclic coefficient matrix W that was most likely to have generated this data matrix X. We can find this coefficient matrix by minimizing the training mean squared error, which captures how far our prediction X\_{t-1}W is off the true value X\_t on average.

Note that this minimization problem is NP-hard, which means that this problem is so difficult to solve that we cannot efficiently solve this problem exactly for say more than twenty nodes.

**Orderings 2 minutes. (13)**

Now, what makes this problem so difficult is that the inferred structure must be acyclic. A useful decomposition for this is that when the matrix is acyclic, then there must exist an ordering of the variables such that all arcs or coefficients only appear in an upper triangular matrix U, meaning that all entries below the diagonal are zero. Given a specific ordering P, we can easily find a suitable acyclic matrix W using Ordinary Least Squares for example.

This also highlights one of the aspects that I find intriguing about this problem, estimating the coefficients is not difficult at all; it is determining which coefficients to estimate is the most difficult problem here.

To see what I mean which such an ordering, suppose we have this coefficient matrix W on four variables. An entry (i, j) is one if and only if there is an arc from i to j. Now, since this graph is acyclic, we can reorder the variables as follows, such that all arcs go from left to right, and when we rearrange the coefficients in W accordingly, we see that all coefficients are in the upper triangular part.

So, given such an ordering P, we can easily estimate the complete upper triangular U, or equivalently, all arcs that go from left to right in this ordering.

**Methodologies 1 minute (14)**

Now that we have discussed this notion of an ordering, I will be discussing three types of approaches with you during the presentation. The first type of approach relies on these orderings, which is why we have called them order-based methods. The second type of approach is called an iterative approach, where we will not use these orderings, but rather add one arc per iteration to our structure. Lastly, we will be discussing the NOTEARS method, which was developed by other researchers which we will use as a benchmark.

**Random Walk: 1 minute (15)**

So, for these orderings, we would find the most likely acyclic W exactly by simple trying out all possible orderings. However, for *p* variables, there are a total of *p!* orderings. Even for just 10 variables, there are millions of orderings possible, meaning that we cannot use such an exhaustive approach in high dimensions.

Therefore, we can decide not to try *all* orderings, but only a subset of all orderings. We might not find the best acyclic matrix, but hopefully we will be able to find a decent acyclic matrix in a reasonable amount of time. To decide which orderings we try, we can only swap to variables in the ordering.

**Regular Random Walk: 2 minute (17)**

To see what I mean with such a random walk, consider an example on three variables. Now, for three variables, there are only six orderings, so we can easily visualize this. The subscript of W denotes the ordering of the structure. For example, from (1, 2, 3) here, we can swap the first two variables, or the last two variables, or the first and last variable, which would yield these three transition possibilities. We see that we from each ordering, we can either traverse along the boundary of the hexagon or straight through it using these double pointed arcs.

Now, how such a random walk works is that we start at some initial ordering, estimate W and check this mean squared error to see how well we can use W for prediction. We remember the best W. Now, we simply traverse this search space for a set amount of iterations, after which we return the best matrix we have found. For three variables, it is more sensible to just try these six orderings and return the most suitable matrix. However, for ten variables, it is inefficient to try out millions of orderings, and then it is better to travel through this search space for a suitable coefficient matrix.

**Greedy Random Walk: 1 Minute (18)**

However, such a random walk, as the name suggests, is random and unguided. We have no idea if we are getting closer to a suitable matrix, we simply try out some orderings at random. To improve on this, we can also use this mean squared error to assess how suitable this coefficient matrix is. We again use the same transition rules as in the random walk, but we can only transition to another matrix if that matrix achieves a strictly lower mean squared error, meaning we only transition to another coefficient matrix if it is an improvement.

**Greedy Random Walk Example: 2 Minute (21)**

So, let us consider the same example, but now I have added the mean squared error of the ordering next to it. The lower, the better. So, we can quickly see that W(1, 2, 3) is the best performing matrix here, but how can we efficiently search the set of orderings for it? If we start at W(2, 1, 3) here, we see that we can transition to W(1, 2, 3), as it achieves a better MSE, and to W(2, 3, 1), but not to W(3, 1, 2), as that one achieves a poorer MSE. From W(3, 2, 1), which achieves the poorest MSE< we can transition to all adjacent orderings. If we now draw all transition arcs, we see that we get this greedy random walk. Especially for large *p*, such a transition rule will help us to travel the space of orderings more efficiently.

However, there is one problem here. We see that W(1, 2, 3) is the best performing matrix. However, can we always reach this matrix using this greedy random walk? No, from W(2, 3, 1), we cannot escape this local optimum, and therefore we cannot always reach the most optimal matrix.

These two order-based approaches are two extremes, one is completely unguided and transitions to all adjacent orderings, and the other is greedy and only transitions to improving orderings. We can also use a middle ground, for example where we sometimes transition to a poorer ordering to escape local optima.

**Orthogonal Matching Pursuit: 1 Minute (22)**

The second type of method is an iterative method, called Orthogonal Matching Pursuit, where we start with an empty matrix W or an empty structure. At each iteration, we will check which unincluded arc has the largest correlation with the current residual. If adding this edge creates a cycle, we exclude it, and if not, we add it to our matrix and re-estimate the residuals. We keep on doing this until we have again reached a stopping criterion.

**OMP Example: 2 Minutes (24)**

Let us again see a simple example in three dimensions. We start with an empty graph, so W contains only zeros, and the C matrix here contains all correlations with the current residuals. We first see that the top-left entry corresponds to the largest correlation, so we estimate this entry in W and re-estimate the residuals. We see that only the correlations in the corresponding column change. Now, the bottom right entry has the largest correlation with the residual, so we estimate this coefficient in W. We now estimate the middle coefficient, as it attains the largest correlation.

Now, the largest correlation is the top center entry, so we now add this coefficient to W, and we see that re estimating W here also changes the middle coefficient, as the full second column of W needs to be re estimated. We also draw this arc in the structure. Now, we see that the middle left coefficient achieves the largest correlation. However, as this would introduce a cycle, we do not include it, so the coefficient is equal to zero. We now continue adding the largest correlation as long as no cycle is introduced. Now, the bottom-left entry here would introduce a cycle of length 3, so we exclude this arc as well. The next arc would also introduce a cycle, so in the end we only add this final arc, which yields us the structure in black, corresponding to the coefficient matrix in the left here.

An important remark here is that we have added these arcs in a specific order, from what we considered to be most important to least important.

**Selecting a suitable number of arcs: 1 Minute (25)**

So, we see that such an iterative approach construct W with one arc per iteration. If we add the most important arc first, we see that we will get diminishing returns, meaning that the gain in training MSE decreases as we add more arcs. This raises the question when we should stop adding arcs, as do not yield sufficient gain anymore?

**Selecting a suitable number of arcs 2: 1 Minute (26)**

To see what I mean, consider this plot, where we used Orthogonal Matching Pursuit to estimate a matrix on ten variables, meaning that the complete acyclic matrix contains 55 arcs. We see that in the beginning, the decrease in training MSE is very significant, with quite steep decreases in predictive performance per arc that we introduce. However, we also see the diminishing returns here, from say 25 arcs, the predictive performance barely increases anymore. So, we could say that after we have iteratively added 25 arcs, the gain in predictive performance is not worth the added complexity of the structure. However, how can we select such a suitable number of arcs?

**LOOCV: 1 Minute (27)**

For this, we propose a regular leave-one-out cross-validation approach. Note that this is a very naïve approach and would most likely not work because of the dependence in our time series data, but let us naively try and see whether it works. Let W\_k represent the matrix after k iterations, so after we have added k arcs. We estimate our matrix W minus t by using all the data except for the tth data pair. We then compute the predictive performance on the sample that we have *not* trained on. We do this for all *T – 1* adjacent time steps, exclude it for training and then compute the predictive performance on left out sample. The average is then the leave-one-out cross-validation score.

Then, we pick the matrix W\_k that achieves the lowest leave-one-out cross-validation score, which so that k corresponds to the suitable number of arcs.

**LOOCV2: 1 Minute (28)**

If we now look at the cross-validation score. We see that after k = 25, the leave-one-out cross-validation score increases, meaning that these arcs cause a poorer leave-one-out cross-validation score. So, this is very interesting. Although this leave-one-out cross-validation seemed like it was a bad idea in time series setting, it seems suitable in our scenario for selecting a suitable number of arcs.

**NOTEARS: 1 Minute (29)**

As a last method, let us consider NOTEARS, which was proposed in 2018. They used an interesting continuous optimization procedure, where they used an interesting function h, that is equal to zero if and only if its argument W is acyclic. This function is an infinite sum, where for each *k*, this component is a weighted sum of all cyclic of length *k* in W. Therefore, if there are no cycles of length *k* for any value of *k*, then h(W) = 0, meaning that W corresponds to an acyclic structure.

Then, how they solved it is by minimizing the mean squared error, but thereby also taking into account the acyclicity through this h function. They add a penalty multiplied by this h function to penalize solutions that are cyclic. By iteratively increasing the penalty parameter, they eventually end up with a local optimum where W is also acyclic.

**Results: 1 Minute (30)**

Now that we have discussed three different types of approaches, let us compare the methods. To do this, we will vary the number of variables between five and fifty, corresponding to low- and high-dimensional time series. Then, for each *p*, we will generate ten acyclic ground truth coefficient matrices with 3 arcs per variable on average. Then, we will generate ten data matrices using these ten coefficient matrices using the generative Vector AutoRegressive model of order 1 which we described before, for a total of 1000 time steps. We then use all three described methods to estimate W, and compare their performance.

**SHD: 1 Minute (31)**

We will first be investigating the Structural Hamming Distance between the estimated W and the ground truth W, which can be computed as the number of missing arcs plus the number of incorrect arcs, plus the number of reversed arcs. The lower the SHD, the better. When we plot our three methods as a function of *p*, we see that NOTEARS in orange performs the best, but the iterative method Orthogonal Matching Pursuit in green remains quite close, also for large values of *p*. The order-based method, the greedy random walk, seems quite okay in the beginning, but drops in performance quite for larger values of *p*. One crucial remark that is not visible here is that NOTEARS is quite slow. In fact, for *p = 50*, one estimation took about a full hour, whereas it only took two or three seconds for Orthogonal Matching Pursuit. Therefore, the iterative approach was almost 1000 times faster, and returns a solution only slightly worse.

**Expected Excess Risk: 1 Minute (32)**

For the expected excess risk, which can be seen as a difference in predictive performance between the estimated matrix W and the ground truth W, we see the following. NOTEARS seems to be the best for all method again, but Orthogonal Matching Pursuit is again quite close. The order-based method is quite close for small values of *p*, but increases more steeply for larger values of *p*. We think that the greedy random walk scales quite poorly for large values of *p* because of this exponential increase with the number of different orderings. Even for ten variables, we had millions of orderings, but for p = 50, we have so many more orderings the performance does not scale well.

**Recovering Causal Pathways using Structure Learning: 1 Minute (33)**

So, let us also investigate a real-life example. I really like this example because it is something different from mathematics that could just have popped up in a biology textbook. What we see here is how eleven different proteins and phospholipids interact in a human immune cell. Now, what you see here is a structure that biologists themselves have agreed upon as some sort of ground truth on how these proteins and phospholipids interact.

We can also use our devised algorithms to learn such a structure from an observational dataset. The authors who published this dataset have gathered measurements of how these eleven variables in orange interact. They used known stimulatory and inhibitory reagents and measured the concentration of these variables in the immune cells to investigate their interactions.

**Results: 1 Minute (34)**

Instead of relying on the biologists ground truth, we can also use our methods next to NOTEARS to see if our methods achieve a similar performance on the state of the art. Four other methods we have developed are also shown in this table. We see that all methods recover six to eight out of the 20 arcs, and the structural hamming distance is between 20 and 22, all quite close. Furthermore, the training mean squared error are all similar, close to five million. So, we see that our methods are just as capable as NOTEARS of recovering these causal pathways in this immune cell dataset.

**Conclusion: 1 Minute (35)**

So, in this presentation, I have introduced the concept of structure learning to you, where we want to learn the interactions between our measured variables in time series data. We have proposed several types of approaches who are competitive with the state-of-the-art. We have first proposed the order-based methods, who seemed to be competitive mostly in low-dimensional time series. As the number of orderings grows exponentially, their relative performance drops as the number of variables increases. The iterative method Orthogonal Matching Pursuit was competitive mostly in high-dimensional time series. Furthermore, this method is also very fast, approximately 1,000 times faster than NOTEARS, which is a huge computational gain. Furthermore, we have also investigated using the naïve leave-one-out cross-validation technique for Orthogonal Matching Pursuit to select a suitable number of arcs, which was quite effective, interestingly.

**Future Work: 2 Minute (37)**

So, lastly, what would be some interesting directions for future work on this thesis? First of all, we think that some significant gains can be made by expanding the model assumptions. Recall that we ended up with a Vector Autoregressive model of order 1 using our assumptions. However, such a model is quite limited, with only linear relations between X\_t and the previous time step. We could expand the model by allowing non-linear relations, or more intricate relations that rely on time steps further in the past.

Secondly, although we have some interesting findings for selecting a suitable number of arcs for the iterative procedure, we have not investigated finding a principled procedure for selecting a suitable number of arcs in the order-based methods. It could be very useful to also have such a procedure there as well.

Lastly, it would be nice to have some statistical guarantees relating to our methods. All our results are experimental in nature, showing the performance on some simulated or real-life dataset. However, it would be very interesting from a mathematical point of view to have some theoretical guarantees as well, which could for example state that with some probability, we recover the correct arcs in the model as long as we have a large enough sample size.

So, this concludes the presentation, thank you very much for listening! Are there any questions?