Predicting match statistics in computer games using machine learning

Artem Vasenin

April 25, 2017

Proforma

Name Artem Vasenin

College Clare College

Title Predicting arbitrary events in competitive computer team games

Examination

Year 2017

Word Count [1]

Project Originator Artem Vasenin (av429)

Project Supervisor Yingzhen Li (yl494)

Original Project Aim

The aim of this project was to develop an algorithm which could predict whether certain events would happen in multiplayer computer games. In most modern multiplayer games, players are provided with a statistic after each match describing their performance in that match and summarising key choices they have made. The project should be able to predict the values of such statistics. The mean squared error of such predictions should be less than half of variance of the variable.

Summary of Work Completed

A graphical model was created which produces a probability distribution for continuous statistics. The first prototype was built using a combination of ad-hoc use of TrueSkill and machine learning functions from scikit-learn python library. The final version was created entirely using TensorFlow and Keras. The system achieved R^2 value of {}on {}variables in the dataset.

Special Difficulties

Other than the typo in success criteria, there were none.

Declaration of Originality

I, Artem Vasenin of Clare College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date: April 25, 2017

[1] pdftotext Disseration.tex −f 7 −l 38 − | wc −w, excluding appendicies and bibliography

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Related Work . . . 1

2 Preparation **3**
- 2.1 Types of variables to predict . . . 3
- 2.2 Requirement Analysis . . . 3
  - 2.2.1 Functional Requirements . . . 3
  - 2.2.2 Non-functional Requirements . . . 3
- 2.3 Starting Point . . . 4
- 2.4 Theoretical Background . . . 4
  - 2.4.1 Rating Algorithms . . . 4
  - 2.4.2 Machine Learning . . . 4
- 2.5 Software Engineering . . . 4
  - 2.5.1 Workflow . . . 4
  - 2.5.2 Version Control . . . 5
  - 2.5.3 Backup . . . 5
- 2.6 Tools used . . . 5
  - 2.6.1 Programming Language and ML Libraries . . . 5
  - 2.6.2 IDE . . . 6

3 Implementation **7**
- 3.1 Data Collection and Pre-processing . . . 7
  - 3.1.1 Collecting the data . . . 7
  - 3.1.2 Pre-processing . . . 7
- 3.2 First Prototype . . . 8
  - 3.2.1 Simple mean and variance . . . 8
  - 3.2.2 Rating of results . . . 8
  - 3.2.3 Analysis of the prototype . . . 9
- 3.3 Final System . . . 9
  - 3.3.1 Neural networks . . . 9
  - 3.3.2 Choosing distributions for variables . . . 9
  - 3.3.3 Player skills . . . 10
  - 3.3.4 Model . . . 10
  - 3.3.5 Prediction . . . 11

4 Evaluation **13**
- 4.1 Preprocessing . . . 13
- 4.2 Train/Validation/Test split . . . 13
  - 4.2.1 Cross-validation . . . 13

5 Conclusions **15**

Bibliography **17**

A Example of a Statistic **19**

B Factor graph example **21**

C Project Proposal **23**

List of Figures

3.1 Distribution of predicted values vs. actual variable distribution 9
3.2 Fitted gaussian and gamma distributions, since gamma distribution has lower χ^2 value, it is a better fit. 10
3.3 An example of an inference model for four players in two teams. 11

B.1 Example of a factor graph 22

Chapter 1

Introduction

1.1 Motivation

Multiplayer computer games are becoming very popular, more than a 100 million people play League of Legends every month [1]. A game's success often rests on how enjoyable it is. Current method of improving player experience is to make sure that all players have equal chance of winning in a game. For that purpose their skill has to be tracked and teams have to be arranged such that they are of equal strength.

In many popular games several roles have to be filled on each team for optimal performance. Current algorithms, such as TrueSkill [2], only track player's overall skill and do not consider what roles the player prefers and how good they are at each one. This often leads teams being composed of players all wanting to play in the same role, which either leads to team underperforming or some players not enjoying the game as much as they could. Moreover, I believe that the events that happen in game are more important to many player's experience than the actual outcome.

To be able to match players better, a system has to be built that will take into account how the game is played and what strategies exist in it. Creating such a system using classical techniques would require a deep knowledge of workings of a game. Unfortunately, I do not have such knowledge of most of the games and obtaining such knowledge although might be fun, will take too long. Furthmore, most such games change their rules every few months to keep players interested, therefore performance of hard coded system would decrease as the time goes on. As a result I decided to use machine learning techniques which would help me create a system which can adapt to different games by itself and also update its judgement as game changes.

1.2 Related Work

Before starting the project, I have looked into papers and articles about predicting events in sports and computer games. I have found out that nothing similar has been done before. Most closely related algorithms (such as [2] and [3]) were made to only predict the outcome of the game, not any related statistic. Outside of academia, other algorithms were made which used machine learning to improve their performance, but again only focusing on the outcome of the game.

Chapter 2

Preparation

2.1 Types of variables to predict

Values included in the game statistic can be rougle separated into two types: class based and regression based.

For example in many games players have the ability to buy items. These items are usually represented as numbers in match statistic, but nearby values usually don't have much in common. Therefore, if we want to predict what items a player will buy, a classification method should be used.

On the other hand something like players score is a value which increases progressively, therefore nearby values have similar significance. In such cases regression techniques should be used.

2.2 Requirement Analysis

The projects aimed to produce a system that would be able to predict match statistics data from players past games.

2.2.1 Functional Requirements

- The system must be able to run on any properly formatted data.
- The predictions should be based on skills of all players in the game, not the player for which predcitions are being made.
- The system should be able to maintain a belief state about a player skill through time and make time-sensitive predictions, not the same prediction for all games.

2.2.2 Non-functional Requirements

- The system should be able to make predictions in under a minute after being trained, on my machine[1].
- The mean squared error of system's predictions should be less than one half of player's standard deviation for that statistic.

[1]Specifications of machine's hardware are given in the project proposal

Regarding the last point: while this is the requirement given in project proposal, I later realised that it contains a typo which make it almost impossible to achieve. "Mean squared error", this should have been "mean absolute error". As it currently stands the systems deviation is squared, while original deviation is not, which means that achieving this requirement gets more difficult the larger the original deviation is.

2.3 Starting Point

At the beginning of the project I had:

- Basic knowledge of artificial intelligence from *Artificial Intelligence I* Part IB course.
- Programming experience in Python (acquired by working on personal projects) and Java (acquired by completing courses in first two years of my study).
- Knowledge of probability and Bayes' theorem from A-Level maths and math courses in the first two years.

During the course of the project I had to gain following qualities:

- Understanding of Bayes' inference and its use in ranking.
- Understanding of various machine learning techniques.
- Familiarity with a chosen ML library (Tensorflow).
- Knowledge of LaTeX and related packages (such as Tikz for graphs) to write-up this dissertation.

2.4 Theoretical Background

2.4.1 Rating Algorithms

2.4.2 Machine Learning

Machine learning allows computers to make predictions on data by learning on past examples and without being explicitly programmed. The aim of my project was to create an algorithm which could work with any game, therefore machine learning was the perfect set of techniques to apply.

Since I decided to use machine learning in my project I had to get some background in the area. Unfortunately, computer science course is in Lent term, therefore I attended a similar course in Engineering department in Michaelmas term, before beginning work on the theory.

2.5 Software Engineering

2.5.1 Workflow

I have decided to use an agile approach of working on this project, since I was not sure exactly how the algorithm should work at the start of the project. The project would be done in two phases:

1. Primary research would be done during the second half of Michaelmas term. A prototype would be developed during Christmas break.

2. At the beginning of Lent term the prototype would be evaluated. In the first two weeks of February the theory would be improved using insights gained during the development of the prototype. In the second two weeks of February the prototype would be refactored to comply with changes in the theory. At the beginning of March the revised implementation would be evaluated.

This approach allowed me to incorporate the knowledge I gained during the evaluation of a prototype into the final version of the algorithm.

2.5.2 Version Control

Git was used for version control. This allowed me to roll-back to a previous version of the project in case a mistake was made. It also allowed me to compare different versions of a specific component. The repository was hosted on GitHub[2] which allowed me to work on the project from multiple machines.

2.5.3 Backup

The code was continuously synchronised to Dropbox[3]. Weekly checkpoints were saved to an external hard drive.

2.6 Tools used

2.6.1 Programming Language and ML Libraries

Machine learning is one of the key elements of this project, therefore I needed a library that would implement key techniques used in ML for me, so that I don't have to write them myself. I have compared a number of libraries (such as Tensorflow[4] and Torch[5]), I was primarily interested in how much funcitonality they provide, their performance and how easy it is to learn them.

I have compared their functionality by completing standard machine learning task, such as creating a neaural network to classify images in the MNIST[6] dataset. Performance was compared by timing how long it would take to achieve 99% accuracy, in most cases it came down whether GPU acceleration was supported. Since I would have to learn how to use the chosen library in detail I also paid attention to the amount of support material available. I took note of quality of documentation and also compared number of questions and answers on StackOverflow.

In the end I have arrived at the conclusion that they all provided required functionality and were sufficiently easy to use. API for most of them were written in different languages. I did not want to learn a new language in addition to learning a library, therefore I decided to use TensorFlow, API for which was written in Python, a language I am most comfortable writing code in. TensorFlow is a low-level library, in which you have to construct each layer of neural network explicitly, I did not need that level of customisation, therefore I used Keras to help me build the neural networks.

[2] github.com
[3] dropbox.com
[4] tensorflow.org
[5] torch.ch
[6] yann.lecun.com/exdb/mnist

To a lesser extent Python was also chosen since it has a package manager[7], which makes it much easier to install additional packages.

2.6.2 IDE

Pycharm was chosen the integrated development environment (IDE) for the project. Pycharm has all of the core features of an IDE, such as syntax checking, autocompletion, running of code, etc. The use of an IDE streamlined the process of development and allowed me to focus on improving the algorithm rather than worrying about language syntax or library functions signatures.

[7] pypi.python.org/pypi/pip

Chapter 3

Implementation

3.1 Data Collection and Pre-processing

3.1.1 Collecting the data

To train and evaluate my algorithm I had to find a complex dataset which would have different types of variables for me to predict. I decided to use game called "Dota 2" for several reasons:

- Summary of each professional game is publicly available and easily accessible through the provided API.
- "Dota 2" is one of the leading E-Sports with hundreds of professional matches each year, which provided me with plenty of data points.
- The game is requires a lot of skill with very little amount of luck involved, which makes it much suitable to prediction.
- The game is team-based and requires a lot of co-operation, which makes it more difficult to predict using classical rating algorithms.

3.1.2 Pre-processing

Cleaning the data The API which provides the summaries sometimes returns incomplete data with some of the values, such as the outcome of the match, missing. I had to find and remove all data points which did not contain complete set of results.

Standardisation Some of the variables in the summary occur over much bigger scales than others. Most machine learning algorithms would not train properly and would prioritise predicting large variables more accurately than smaller ones. I would like to consider all variables with equal priority there I had to normalise the data. To normalise data we can a apply a simple transformation to all data points:

$$x' = \frac{x - \overline{x}}{\sigma} \tag{3.1}$$

Where $\overline{x}$ is the mean of x and σ is its standard deviation

Normalisation During the analysis of data I decided that the data is better modelled using a gamma distribution. Gamma distribution can only represent positive numbers, therefore I could not use standardisation for this dataset. Instead I used normalisation which also rescales the values, but keeps them all positive, it uses the following transformation:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.2}$$

3.2 First Prototype

The main problem in machine learning is frequently generation of correct features to represent the data. In my case I had statistics of more than twenty thousand games. Most players had a few hundred games in the dataset. Statistic for each match is represented as a dictionary of key-value pairs (in JSON[1] format), overall there are more than 200 values per statistic, an example of a statistic can be found in Appendix A. This meant that using raw data for input would be impractical, therefore some kind of aggregation had to be created.

3.2.1 Simple mean and variance

At first I tried using means and standard deviations of players results as features. While this approach made predictions which were better than just predicting player mean, they weren't very good. A variety of machine learning techniques were tried, including: Neural Networks (NN), Support Vector Machines (SVM), Gaussian Processes (GP), Ridge Regression (RR) and Naive Bayes (NB). NN, SVM and NB were used for classification variables, while GP and RR were used for regression. All of the methods had similar performance, which indicated that features generated were not sufficient. To improve the features I first tried putting a limit on how many games were used to calculate player's mean and average, while this improved performance a bit, it was still very poor. Next thing to try was rating players based on their performance in each variable.

3.2.2 Rating of results

The easiest algorithm to use for rating players was TrueSkill, since it offered rating games with multiple players and there was a library[2] for python which provided the functionality. Trueskill only uses the final ranking of the individual at the end of the game, rather than absolute value, therefore some informations is lost. This also meant that I had to do further processing on the data to convert raw statistic into ranks. This was mostly straightforward, one thing to note is that some ranks (such as death count) had to be reversed, since players try to keep those results low rather than make them high.

Such ad-hoc use of TrueSkill meant that one of the assumptions behind the algorithm was not met: the algorithm assumes that all players are competing against each other, but in reality they are separated into two teams. This meant that resulting ratings were very accurate. Player ranks were then used as features for machine learning techniques. This approach generated much better results than simple mean and variance, with best variable achieving R^2 value of 0.47. Although this is much better than previous results, it is still far from required value of 0.75.

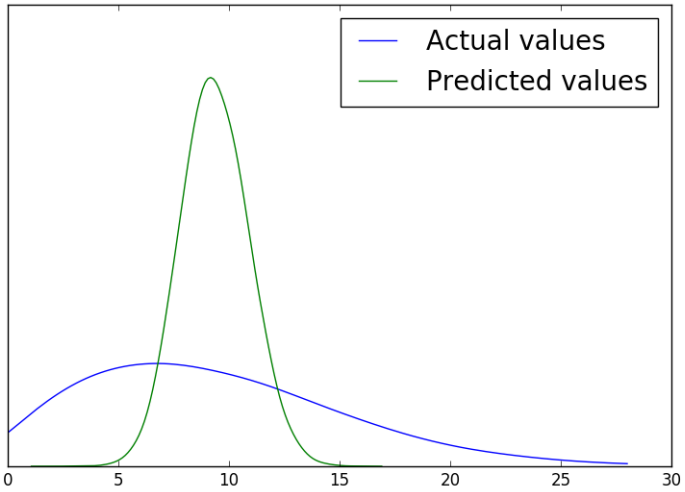

Figure 3.1: Distribution of predicted values vs. actual variable distribution

3.2.3 Analysis of the prototype

During analysis of predictions produced by the prototype, I observed that the range of predictions was much smaller than the actual range for most variables. An example of this can be seen on figure 3.1, where predictions rough range of 5 to 15, while actual variable range is roughly 0 to 30. In addition, most of the machine learning techniques had similar predictive performance. These two facts combined led me to believe that the problem is again with quality of features. Therefore, I decided to create my own graphical model to generate richer features.

3.3 Final System

3.3.1 Neural networks

3.3.2 Choosing distributions for variables

When predicting the values for each variable, rather than predicting the probability for each value independently, it is better to use a distribution to predict the probabilities for all values. To decide which distribution fits each variable best, I used *method of moments* to fit each potential distribution to all observed values of the variable. Then I used *goodness of fit* to decide which of the potential distributions is best.

Method of Moments To fit a distribution we need to estimate its parameters. Parameters for most distributions can be easily expressed in terms of mean and variance of the data. For example for Gamma distribution the equations are:

$$k = \mu^2/\sigma^2 \qquad \theta = \sigma^2/\mu \tag{3.3}$$

[1] json.org
[2] trueskill.org

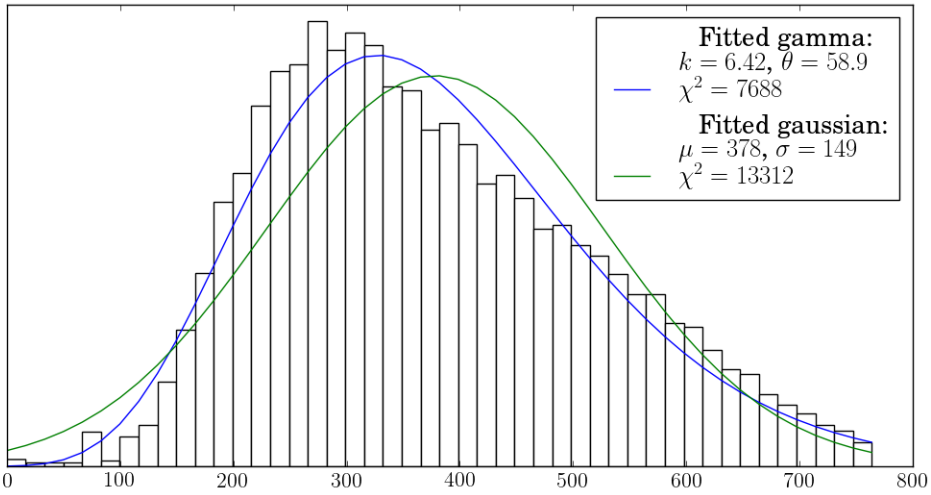

Figure 3.2: Fitted gaussian and gamma distributions, since gamma distribution has lower χ^2 value, it is a better fit.

Goodness of fit To estimate how well the distribution fits the data we can use chi-squared test to measure sum of differences between observed and expected outcome frequencies:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{3.4}$$

Where O_i is the observed frequency and E_i is the expected frequency.

3.3.3 Player skills

Every game is different and trying to pick specific set of skills for each one, would not be practical. Therefore, I decided to make an assumption that players skill in a game can be described by n real numbers. Such n will be different for each game and therefore is a hyper parameter in my model. Since the system can never be sure on the exact skill values for each player, skills of each player are represented as n Gaussian distributions.

3.3.4 Model

Training both the inner model for statistic prediction and model for player skill updates at the same time would greatly increase the data required for processing. It will also raise some difficult problem such as *vanishing gradient*, unfortunately I did not have enough time to solve them properly. Instead I decided to use the technique called *coordinate descent*.

Coordinate descent When we have multiple parameters to optimise, instead of optimising them at the same time we can optimise them alternatively. We optimise each parameter for a certain number of iterations (or until some criteria is achieved) and then move onto the next parameter. Once we optimised each parameter, we can loop back and start again from the first one. We do that until all parameters reach sufficiently stable values.

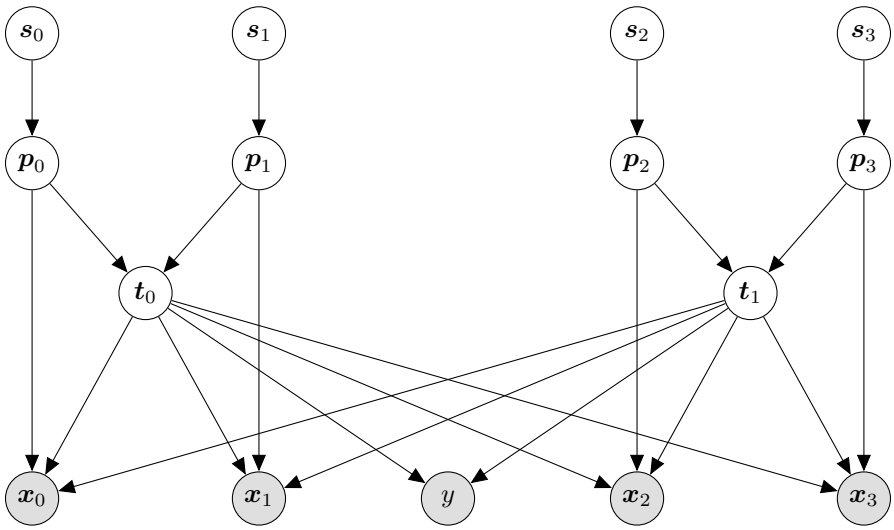

Figure 3.3: An example of an inference model for four players in two teams.

3.3.5 Prediction

To predict the values for a particular match-up we just need to take the set of skills corresponding to players in the match-up and feed them into the top of the model.

Chapter 4

Evaluation

4.1 Preprocessing

When neural networks have been trained, we need process every match in chronological order to update the player skills. To make evaluation quicker we can create a dictionary which contains match id as keys and skills of the players taking part in that match. We need to store the player skills before processing the match and updating them to accurately reflect real world scenario.

4.2 Train/Validation/Test split

Most important rule when evaluating machine learning systems is that the algorithm has to be finalised before starting to test. Therefore right at the start we can split our data into two parts: *train* and *test*. *Train* set will be used to train the algorithm and adjust hyper-parameters. *Test* set will be used for evaluation of the algorithm.

To adjust hyper-parameters fairly we need to reflect evaluation procedure during our training. For that purpose the train set is further split into two parts: *train* and *validation*.

4.2.1 Cross-validation

If we reuse the same part for validation, we run the risk of over-fitting our algorithm to that particular part of the dataset. To combat that, we can use cross-validation: we run train/validation multiple times, each time picking new set to use for validation. To ensure complete coverage of the train-set we can use *k-fold cross-validation*:

1. Divide the training set into k random subsets.
2. Pick a subset that has not been validated on, use it for validation and the rest for training. Record the result.
3. Repeat step two until all subsets have been used for validation.
4. Average the results.

Chapter 5

Conclusions

Bibliography

[1] Forbes. *Riot Games Reveals 'League of Legends' Has 100 Million Monthly Players*. 2016. URL: forbes.com/sites/insertcoin/2016/09/13/riot-games-reveals-league-of-legends-has-100-million-monthly-players/#274b74155aa8 (visited on 04/16/2017).

[2] Ralf Herbrich, Tom Minka, and Thore Graepel. "TrueSkill(TM): A Bayesian Skill Rating System". In: MIT Press, Jan. 2007, pp. 569–576. URL: https://www.microsoft.com/en-us/research/publication/trueskilltm-a-bayesian-skill-rating-system/.

[3] Ruby C. Weng and Chih-Jen Lin. "A Bayesian Approximation Method for Online Ranking". In: *J. Mach. Learn. Res.* 12 (Feb. 2011), pp. 267–300. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.1953057.

Appendix A

Example of a Statistic

Some parts of this statistic are repetetive and take up a lot of space. In such parts, all but the first set were hidden, indicated by comments in <>.


```
{
   "result":{
      "players":[
         {
            "account_id":87382579,
            "player_slot":0,
            "hero_id":60,
            "item_0":239,
            "item_1":92,
            "item_2":46,
            "item_3":108,
            "item_4":29,
            "item_5":6,
            "backpack_0":0,
            "backpack_1":0,
            "backpack_2":0,
            "kills":3,
            "deaths":7,
            "assists":17,
            "leaver_status":0,
            "last_hits":72,
            "denies":2,
            "gold_per_min":254,
            "xp_per_min":298,
            "level":14,
            "hero_damage":5682,
            "tower_damage":277,
            "hero_healing":170,
            "gold":73,
            "gold_spent":9220,
            "scaled_hero_damage":0,
            "scaled_tower_damage":0,
            "scaled_hero_healing":0,
            "ability_upgrades":[
               {
                  "ability":5275,
                  "time":886,
```

```
                    "level":1
                },
                <There is one set per level and players can advance up to
                    level 25>
            ]
        },
        <There are nine more sets of player data>
    ],
    "radiant_win":false,
    "duration":2368,
    "pre_game_duration":90,
    "start_time":1471142574,
    "match_id":2569610900,
    "match_seq_num":2244488581,
    "tower_status_radiant":1540,
    "tower_status_dire":1956,
    "barracks_status_radiant":3,
    "barracks_status_dire":63,
    "cluster":113,
    "first_blood_time":89,
    "lobby_type":1,
    "human_players":10,
    "leagueid":4664,
    "positive_votes":38429,
    "negative_votes":2503,
    "game_mode":2,
    "flags":1,
    "engine":1,
    "radiant_score":27,
    "dire_score":30,
    "radiant_team_id":2512249,
    "radiant_name":"Digital Chaos",
    "radiant_logo":6927801062027479 75,
    "radiant_team_complete":1,
    "dire_team_id":1836806,
    "dire_name":"the wings gaming",
    "dire_logo":3527707085973443 69,
    "dire_team_complete":1,
    "radiant_captain":87382579,
    "dire_captain":111114687,
    "picks_bans":[
        {
            "is_pick":false,
            "hero_id":79,
            "team":1,
            "order":0
        },
        <There are about 20 sets in this list, detailing the drafting
            phase>
    ]
  }
}
```

Appendix B

Factor graph example

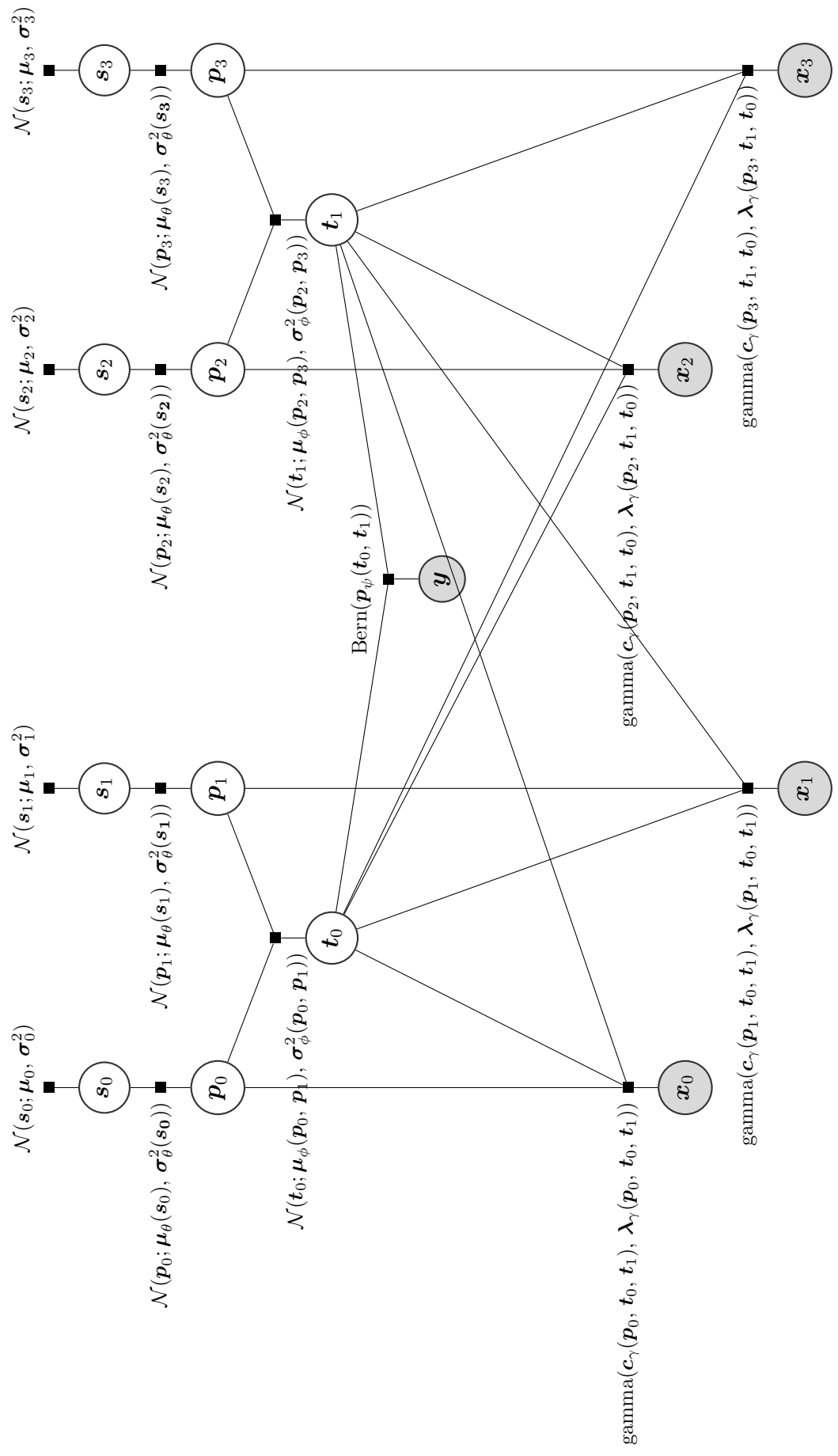

Figure B.1: Example of a factor graph

Appendix C

Project Proposal