Technical Report

Hasso Plattner Institute

Chair for Artificial Intelligence and Sustainability

TrueSkill Beyond Gaussians

Authors:
Cedric Lorenz
Nick Bessin
Nina Burdorf
Till Zemann
Isabel Kurth
Lucas Kerschke
Constantin Kühne
Paul Ermler

Submitted to:
Prof. Dr. Ralf Herbrich
Nicolas Alder

March 5, 2025

Abstract

Factor graphs are a widely used framework for Bayesian inference. Use cases include recommendation systems and ranking systems such as TrueSkill [HMG07]. For these use cases, messages and marginals are usually represented as Gaussian distributions. However, often messages are not truly Gaussian, and the real messages are approximated with a Gaussian. This is computationally efficient and has the advantage that all messages stay within the exponential family, allowing closed-form solutions for many typical operations. The disadvantage is a potentially decreased approximation quality due to approximation. Thus, the goal of the project was to develop more expressive messages and marginal representations for factor graphs.

To address this limitation, we explore different distribution representations that go beyond Gaussians and enable the use of multimodal probability distributions in factor graphs. To this end, we first look at different discrete (histogram-like) distribution representations and subsequently introduce a framework for message representations using mixtures of Gaussians (MoGs). We apply the results to TrueSkill and test the different approaches on various real-world and synthetic datasets. We show that using MoG achieves higher prediction accuracies for specific datasets, but it is outperformed by the classic TrueSkill implementation on all real-world datasets. Lastly, we extend the framework from the previous master project [Ada+24] and thesis [Som24] to build multilayer perceptrons with factor graphs to support the MoG message framework. However, more work is needed to make the machine learning framework more robust and scale to larger datasets.

Our source code written, mainly in Julia, is available on GitHub.

Nomenclature

ATP Association of Tennis Professionals

BNN Bayesian Neural Network

CDF Cumulative Density Function

CNN Convolutional Neural Network

DBD Derivation-Based-Density

DNN Deep Neural Network

EBD Equal Bucket-Density

EBW Equal Bucket-Width

EM Expectation Maximization

$FGNN$ Factor Graph Neural Networks

GM Graphical Model

KL Kullback-Leibler

KLD Kullback-Leibler Divergence

LBP Loopy Belief Propagation

LLM Large Language Model

LoL League of Legends

MAP Maximum A Posteriori

MLE Maximum Likelihood Estimator

MLP Multilayer Perceptron

$MOBA$ Multiplayer Online Battle Arena

MoG Mixture of Gaussians

MoU Mixture of Uniforms

PDF Probability Density Function

SMA Shifted Mean Approach

Contents

Chapter 1

Introduction

1.1 Motivation

Accurate ranking systems are essential in competitive environments, ranging from traditional two-player games such as chess to modern multiplayer online games. Traditional systems, such as Elo [ES78], offer a simple but effective approach in one-on-one settings. However, these models struggle with fundamental challenges such as modeling skill uncertainty, and they can only be provided with workarounds, such as bootstrapping. They have additional difficulties with handling multiplayer games [Gli99]. Addressing these limitations, probabilistic ranking models, such as TrueSkill [HMG07], take advantage of Bayesian inference to provide a more flexible and uncertainty-aware estimation of player skills.

The core of TrueSkill is the factor graph framework [KFL01]. Factor graphs allow efficient message-passing algorithms, enabling scalable computations of skill distributions. The TrueSkill framework models player skills as Gaussian probability distributions rather than deterministic values, allowing continuous updates of the expected skill and uncertainty based on match outcomes. In practice, the probability distribution can be decomposed so that the posterior skill of a player can be taken as the prior skill for that player's next match. This sequential learning approach is particularly advantageous in modern ranking scenarios, where skill estimates must be continuously refined with new observations [MCZ18].

TrueSkill operates under the assumption that player skills and other latent variables follow a Gaussian distribution, and therefore all messages exchanged within the factor graph are required to be Gaussian. In TrueSkill, this leads to approximations of the greater-than factor (see [HMG07; Her24]. Moreover, this is not a problem specific to TrueSkill and the greater-than factor. Looking at recommendation systems [SHG09] and Bayesian neural networks expressed as factor graphs [Som24], finding good Gaussian approximations can become problematic for the product factor.

In order to address these challenges, we explore alternative message representations within TrueSkill and factor graphs more broadly. By extending TrueSkill to support non-Gaussian message representations, we aim to increase the approximation quality of latent marginals and skill posteriors, which could increase the accuracy of match outcome predictions and subsequently enhance the match quality of ranking systems.

Contributions In this work, we investigate different distribution representations for factor graphs and our contributions are summarized as follows. First, we explore the use of mixtures of uniforms as discrete representations and analyze their limitations. Second, we propose a framework for multimodal message representation in factor graphs using mixtures of Gaussians and more accurate message approximations through expectation maximization. Third, we introduce a framework for constructing multilayer perceptrons with mixtures of Gaussians, including derivations for the prod-

uct and leaky ReLU factors. Finally, we empirically evaluate our approaches on both synthetic and real-world datasets to demonstrate their practical applicability.

Overview This report is structured as follows: In Chapter 2, we provide an overview of factor graphs, the TrueSkill framework, and different types of distributions. In Chapter 3, we propose mixtures of uniforms as our first alternative distribution representations. Next, we investigate mixtures of Gaussians in Chapter 4. In Chapter 5, we define factors for a Bayesian neural network with factor graphs. We compare the different distribution representations, test them on different datasets, and discuss the findings in Chapter 6. Finally, in Chapter 7, we summarize the report and give an outlook into future work.

1.2 Related Work

Matchmaking in gameplay is the process of pairing participants to play together – competitively, cooperatively, or as a combination [HJM16]. The core premise is that pairing the right players significantly influences the overall perception of enjoyment for the played match. For example, connecting two opponents with a drastic skill difference level in competitive multiplayer online games may lead to frustration or boredom. The less-skilled player, with an extremely low probability of winning, may perceive the game as pointless. In contrast, the highly skilled player might not find the game sufficiently challenging or engaging [Yuk24]. In most cases, matchmaking is made based on the skill level of the players. Rating systems form the backbone of these matchmaking processes [VJS17], aiming to assess the performance of individuals or teams on a standardized scale. This ensures that competitors can be ranked according to their expected skill level at any given time. Additionally, a well-designed rating system should go beyond simple ranking by offering an estimate of the relative differences in strength between competitors, regardless of how strength is defined [ES78]. Rating systems should be able to estimate player skills quickly and accurately. For instance, they should not measure a player's skill solely based on how often they compete; instead, they must account for other dynamic factors such as their skill uncertainty, win rate, and opponent's strength to provide a fair and accurate assessment.

An alternative to skill-based matchmaking and rating is role-based matchmaking. Players are assigned different roles based on their performance, and a degree determines how much they belong to that role. These prototypical role configurations vary depending on the game and may be defined by experts [JJD11]. For example, in team-based games, players may be categorized into offensive or defensive roles based on their game interactions [SZ04]. In the gaming community, further approaches have been developed to include more metrics, such as personal interests, to reach social connections through gaming [HJM16]. Matchmaking approaches like those above focus on incorporating additional player preferences while deprioritizing skill classification [JJD11].

One of the first common skill ranking systems was the Harkness rating system [Har67] used in chess. The system aimed at allowing players to track their individual performances in addition to simple wins and losses [Sfe11]. Therefore, all players' median tournament performance is considered [US nd]. However, using the Harkness system as the backbone of a rating system introduced a bottleneck: strong players could receive low ratings if they competed primarily against lower-rated opponents. To overcome this, the Elo system [ES78] was introduced. Here, the expected outcome of a match is calculated based on the difference in ratings between two players. If the expected and actual outcomes differ, ratings are updated accordingly. The Elo system provides a method for estimating skill levels from paired comparison data [Dav88]. Assuming that skill differences lie on a continuous scale (following Thurstone's case V model), i.e. via Gaussian skill distribution instead of just wins and losses, a special continuous case using a logistic function corresponds to the Bradley–Terry model [HMG07]. The main difference between the Elo system and the Bradley-Terry model lies in its distributional applicability, as the latter can only be used for bimodal outcomes, i.e., win or loss of a game [Gli99].

While using the Elo rating system is effective for one-on-one competitions, Elo assumes that player skills are static. It does not model uncertainty in skill estimation. Consequently, the player's rating

can change significantly based on a single match result. On the other hand, its main advantage is its simplicity: Elo scores can easily be calculated by hand for a small set of players. However, it cannot rank new players with few or few games played. At least thirty games are required to determine a player's ranking with reasonable confidence [ES78].

To address some of the limitations implied by the original Elo system, Mark Glickman introduced the rating system Glicko [Gli99], which incorporates the concept of rating deviation. Rating deviations quantify the confidence in a player's skill estimate: a low deviation indicates that the rating is reliable, while a high deviation suggests more uncertainty. Unlike Elo, where all updates are treated equally, Glicko dynamically adjusts ratings based on performance and confidence in previous ratings. Glicko-2 [Gli22] further refines the approach by introducing volatility. A volatility parameter accounts for fluctuations in a player's rating. While a low volatility value indicates a constant, a high value signifies an unstable performance over time [Gli22].

Despite improvements, challenges introduced by team-based multiplayer online games have not been resolved by Glicko-2. There are multiple concerns: on the one hand, there is the need to handle team-based performance while inferring individual player skills; on the other, more than two players or teams compete against each other. This leads to the game outcome being a permutation of players or teams instead of simply a winning and a losing side [HMG07]. In addition, to fully capture the game dynamics, the rating system must account for draws, partial play, and skill transfer over games while learning efficiently from few match outcomes [Her24].

To address these issues, TrueSkill [HMG07] was developed. In TrueSkill, Bayesian principles are applied using factor graphs, the sum-product algorithm, and approximate message passing to propagate beliefs about player and team skills efficiently (see Section 2.4). TrueSkill Through Time is an extension of TrueSkill. Time series of player skills are inferred, and past and future predictions are considered. In addition, the estimation of player-specific draw margins is introduced [Dan+07]. TrueSkill 2 extends the original TrueSkill implementation by considering further player information, such as the squad membership or the tendency to quit the game. The system leads to improved accuracy of skill ratings in the online shooter game Halo 5 over TrueSkill [MCZ18].

Building on the extensions introduced in TrueSkill 2, examining the underlying computational framework that makes these innovations possible is instructive. In particular, factor graphs enable efficient and flexible probabilistic inference in complex, multi-player environments. Factor graphs are bipartite graphical models representing how a global function, typically a joint probability distribution, can be factorized into a product of local functions. This representation lends itself to efficient inference via message-passing algorithms such as the sum-product algorithm [Fre+97]. Originally introduced to streamline decoding in error-correcting codes [CW24], factor graphs have since been applied to diverse fields including computer vision [SCI09], robotics (e.g., SLAM problems) [DK+17], and statistical signal processing [Loe+07].

In practice, factor graphs provide a scalable framework for modeling complex dependencies. For instance, in online gaming, they can efficiently propagate uncertainties about individual player skills, team dynamics, and even temporal changes in performance.

Transitioning from the traditional role of factor graphs in probabilistic inference, it is worth noting that similar principles have inspired a novel class of neural network architectures known as factor graph neural networks (FGNNs) [ZWL20]. In FGNNs, message passing is executed via neural modules, often multilayer perceptrons (MLPs), that update node embeddings based on local interactions. This approach combines the interpretability and modularity of factor graphs with the non-linear processing power of deep learning.

FGNNs have been successfully applied in various domains where data exhibits intrinsic structured dependencies. In computer vision, for instance, they have been used for tasks like image segmentation [ZJ09] and object recognition [He+18], where spatial relationships between pixels or regions are crucial. More recently, FGNNs have been leveraged in biomedical research where interpretability is paramount. For example, by directly encoding biological knowledge such as Gene Ontology into the factor graph and using attention mechanisms to capture hierarchical interactions among genes, these models have achieved superior performance in predicting clinical outcomes and facilitating

gene set enrichment analysis [MZ19].

While FGNNs provide a structured approach to capturing intricate dependencies in data, alternative probabilistic models like Gaussian mixture models (GMMs) have also proven effective in modeling complex, multimodal distributions. GMMs represent a probability density function as a weighted sum of several Gaussian components, thereby offering a richer depiction of data variability. For example, Reynolds et al. (2000) utilized GMMs to enhance speaker verification by better capturing the variability in speech features [RQD00]. Furthermore, Qu et al. (2020) introduced a GMM-based anomaly detection method for hyperspectral images that extracts anomaly pixels using prescribed GMM parameter ranges, fuses the results with a weighting approach, and refines the final detection map via a guided filter, demonstrating superior performance on four datasets [Qu+20].

Chapter 2

Preliminaries

2.1 Factor Graph

Factor graphs are a class of bipartite graphical models that represent the factorization of a function, commonly used in probabilistic modeling, signal processing, and machine learning. They offer a structured way to visualize and compute complex distributions by breaking them into smaller, more manageable components [KFL01]. Factor graphs have gained significant importance in fields such as robotics [DK+17] and computer vision [SCI09]. Suppose $n, m \in \mathbb{N}^+$, $i \in \{1, ..., n\}$ and $j \in \{1, ..., m\}$. A factor graph is a bipartite graph consisting of a set of factor nodes $F = \bigcup_i \{f_i\}$ and a set of variable nodes $X = \bigcup_j \{x_j\}$ that subdivide a joint probability distribution $p(x_1, ..., x_m)$ into factors. The variable nodes represent the random variables in the model, and the factor nodes represent functions that define relationships between subsets of variables.

Edges in the graph connect factor nodes to variable nodes, signifying that the function (or factor) depends on those particular variables. Mathematically, a factor graph represents the factorization of a function over a set of variables. Let $X = \{x_i\}_{i \in \mathbb{N}}$ be a collection of variables (indexed by a finite set $N = 1, 2, 3, ..., n$. If A is a non-empty subset of N, then X_A is the subset of X indexed by A ($X_a \subseteq X$) [Fre+97]. Let Q e a collection of subsets of N. The global function g can be written as the product of local functions with the subsets as arguments, i.e.,

$$f(X) = \prod_{A \in Q} f_A(X_A). \tag{2.1}$$

Consider a joint distribution (global function) that factors into several local functions as follows:

$$p(x_1, x_2, x_3, x_4) = f_1(x_1, x_2) f_2(x_2, x_3) f_3(x_3, x_4) \tag{2.2}$$

The corresponding factor graph consists of variable nodes x_1, x_2, x_3, and x_4 and factor nodes f_1, f_2 and f_3. Edges connecting x_1 and x_2 to f_1, x_2 and x_3 to f_2, and x_3 and x_4 to f_3.

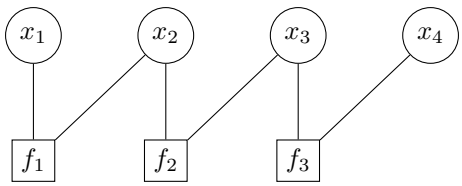


Figure 2.1: Factor graph example

In many situations, we are interested in the marginal function $p(x_i)$. We can get an expression for calculating the marginal function by separating it into independent terms:

$$\begin{aligned} p(x_1) &= \int_{x_2}\int_{x_3}\int_{x_4} f_1(x_1,x_2)f_2(x_2,x_3)f_3(x_3,x_4)) \\ &= \int_{x_2} f_1(x_1,x_2)\left(\int_{x_3} f_2(x_2,x_3)\left(\int_{x_4} f_3(x_3,x_4)\right)\right) \end{aligned} \tag{2.3}$$

To compute the marginal more efficiently, we can use message-passing algorithms, such as the sum-product algorithm.

2.2 Sum-Product Algorithm

The objective of the sum-product algorithm is to efficiently compute the marginal distribution of each variable x_j by "marginalizing out" all other variables $x_{j'}$ with $j' \neq j$. This is achieved by passing messages through a factor graph in a forward and backward fashion [Fre+97]. Suppose $x_j \in X$, $f_i \in F$, a message from a factor f_i to a variable x_j is denoted as $m_{f_i \to x_j}(x_j)$, while a message from a variable x_j to a factor f_i is written as $m_{x_j \to f_i}(x_j)$. The algorithm leverages the distributive law to iteratively update the marginal probabilities and the messages exchanged between factors and variables, ensuring an efficient computation of the desired distributions.

The message-passing equations [KFL01] for continuous random variables are the following:

Sum-Product Message-Passing Algorithm

Suppose $x_j \in X$, $f_i \in F$, $\mathrm{Ne}(x_j) \subseteq F$ is the neighborhood of x_j and $\mathrm{Ne}(f) \subseteq X$ is the neighborhood of $f \in F$.

Marginals:

$$p(x_j) = \prod_{f \in \mathrm{Ne}(x_j)} m_{f \to x_j}(x_j) \tag{2.4}$$

Message from factor f_i to variable x_j with $x_1, ..., x_k \in \mathrm{Ne}(f_i) \setminus \{x_j\}$:

$$m_{f_i \to x_j}(x_j) = \int_{x_1} \cdots \int_{x_k} f(x_1, \cdots, x_k, x_j) \prod_{i=1}^{k} m_{x_i \to f}(x_i)\, \mathrm{d}x_1 \cdots \mathrm{d}x_k \tag{2.5}$$

Message from variable x_j to factor f_i:

$$m_{x_j \to f_i}(x_j) = \prod_{f' \in \mathrm{Ne}(x_j)\setminus\{f_i\}} m_{f' \to x_j}(x_j) \tag{2.6}$$

These equations hold for discrete random variables as well, except that the integrals must be replaced by sums.

The sum-product algorithm ensures exact solutions for factor graphs without cycles. However, when cycles are present in the graph, the message-passing process may continue eternally without reaching a stable solution. This scenario is known as loopy belief propagation (LBP), often used as an approximation method. LBP utilizes the same message-passing equations iteratively but without guaranteed convergence to the actual marginal distributions. One common approach to improve the convergence of LBP is damping, where the new message update is a weighted combination of the previous message and the newly computed message [Hes02]. Another approach is scheduling, which determines the order in which messages are updated to facilitate better convergence [EMK12].

2.3 Moment Matching and KL Divergence

Moment Matching for a Univariate Gaussian Sometimes, exact messages from a factor to variables are not closed under the distribution family we use, for example, for Gaussians as a representative of the exponential family. To address this, we will use an essential property of the exponential family: minimizing the Kullback-Leibler (KL) divergence (KLD) is equivalent to moment matching [Her05]. This technique is also known as the method of moments or assumed-density filtering [Min13]. For a univariate Gaussian approximation $q(x;\theta) = \mathcal{N}(x;\mu,\sigma^2)$, we derive that minimizing the $\mathrm{KL}(p \mid q)$ divergence to the target distribution p leads to setting the approximate mean and variance by using the first and second moments, see Appendix A.3. This guarantees that the approximation retains important statistical properties of the original distribution (i.e., mean and variance for a Gaussian) while remaining computationally efficient.

Approximate Message Passing Moment matching can be applied to marginal distributions, but messages must be passed between factors and variables when performing inference in factor graphs. If a factor transformation does not preserve the distribution family (e.g., a product of Gaussian random variables does not necessarily yield another Gaussian), we approximate messages using moment matching. This allows efficient updates while maintaining a tractable form for subsequent computations in the factor graph.

However, for some factors, it can be shown that minimizing the forward KL divergence would result in increasing variance. This indicates that the entropy of that distribution, which is a function of the variance for a Gaussian distribution, would increase after an update. In this case, approximate message passing might not converge, and we need to minimize different divergence metrics that guarantee convergence via approximations with decreasing variance after updating.

In this work, we investigate the reverse KL divergence as an alternative that fulfills this property and which we can thus use for specific factors, such as the product factor (see Subsection 5.2.1). It yields a mode-seeking solution, i.e., an approximation that fits a subset of high-density modes. This can help break symmetry, i.e. when we want to avoid having a multimodal belief overweight in a neural network.

2.4 TrueSkill

TrueSkill [HMG07] is a Bayesian skill estimation system developed by Herbrich et al. for skill-based matchmaking and ranking in multiplayer games. It was introduced as an alternative to the Elo rating system [ES78], which is commonly used in chess and other competitive games. TrueSkill is designed to address some of the Elo system's limitations, particularly enabling the ranking of games with more than two players or teams. Figure 2.2 shows the TrueSkill factor graph for a two-player game. The prior factor $\mathcal{N}(s_i;\mu_i,\sigma_i^2)$ stores the current player skill estimate as a Normal distribution. In the next step, a player's performance p_i is calculated as a Normal distribution around the player's skill s_i. The standard deviation β of this Normal distribution determines how many more skill points a player needs to have an 80% probability of winning the game [Mos10]. Next, the player skills within a team are summed up to calculate the team's performance. In the case of a two-player game, where each team consists of only one player, the team performance equals the player performance. In the end, the pairwise team performance differences are compared. Here, it only matters whether the game outcome was a draw or whether one of the teams won. This information is observed in the difference node, shown in gray in Figure 2.2. The result is backpropagated through the entire graph to update the player's skills.

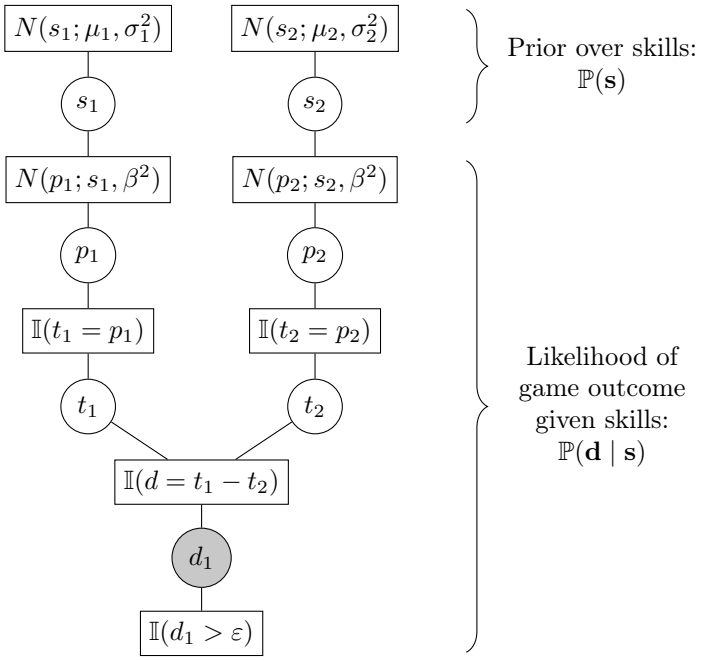


Figure 2.2: TrueSkill factor graph example for a two-player game. Observed variables are shaded gray. During an iteration of the TrueSkill algorithm, a forward pass (top-to-bottom) calculates the likelihood of the game outcome (as the product of the performance, weighted sum, and greater-than factors), and a backward pass (bottom-to-top) calculates the posterior skill distribution $\mathbb{P}(\mathbf{s} \mid \mathbf{d})$. In the case of multiple teams, the backward pass requires a slightly more complicated message schedule, see Herbrich et al. [HMG07].

For message passing with Gaussian distributions, the marginals are stored using their natural parameters. This is because operations such as the multiplication of two Gaussian distributions for the message updates can be rewritten using fewer multiplication and addition operations, making it more energy-efficient. In the following subsections, we will outline the different factors that makeup TrueSkill.

2.4.1 Prior Factor

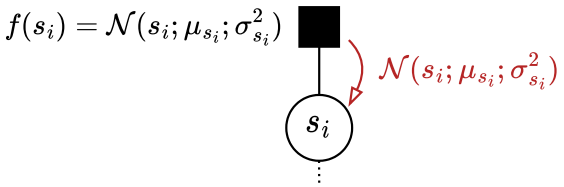


Figure 2.3: Prior factor

The prior factor (see Figure 2.3), often referred to as the Gaussian factor in the context of TrueSkill, defines the initial belief about a player's skill level before the game has been played. It serves as a prior distribution, modeled as a Gaussian, that represents our uncertainty about a player's actual skill. Initially, all players are assigned the same prior. After a game, the prior is updated to represent the new marginal distribution of the player's skill after the game.

The message from the prior factor to the player's skill s_i is as follows:

$$m_{f \to s}(s) = \mathcal{N}(s; \mu_s, \sigma_s^2) \tag{2.7}$$

2.4.2 Gaussian Mean Factor

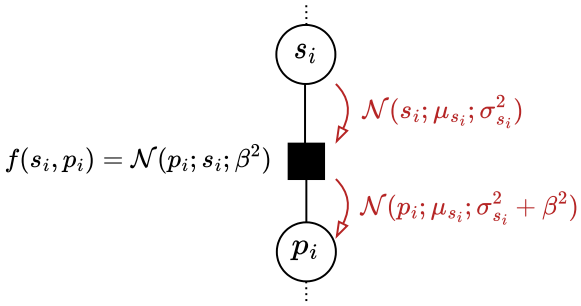


Figure 2.4: Gaussian mean factor

The Gaussian mean factor (see Figure 2.4) maps a player's skill s_i to their performance p_i by increasing the variance, allowing for uncertainty in performance. Adding variance to the skill makes it possible to model a player's performance as random fluctuation around the skill on a game-by-game basis. On the other hand, the skill itself changes more slowly over time. In practice, players might play better at certain times of the day or focus more than usual because they are close to a rank promotion. All of these influences are modeled in a simplified way by using a Normal distribution around the player's skill.

The Normal distribution around the player's skill has a predefined standard deviation β. From a technical point of view, β can be used to slowly decrease the variance of player's skill estimates during the "training" process, i.e., forward and backward passes through the TrueSkill graph using a time-ordered dataset or data stream of match outcomes. A large β results in a small variance reduction of the player's skill distribution, and vice versa for a small β.

The Gaussian mean factor is defined as:

$$f(s_i, p_i) = \mathcal{N}(p_i; s_i, \beta^2) \tag{2.8}$$

with β^2 as a hyperparameter, as discussed. Given that the incoming message from s_i to the factor is Gaussian with mean μ_{s_i} and variance $\sigma^2_{s_i}$, the message from the Gaussian mean factor to the player's performance p_i is defined via the message-passing equations in Equation 2.5:

$$m_{f \to p_i}(p_i) = \int_{-\infty}^{\infty} \underbrace{\mathcal{N}(p_i; s_i, \beta^2)}_{f(s_i, p_i)} \cdot \underbrace{\mathcal{N}(s_i; \mu_{s_i}, \sigma^2_{s_i})}_{m_{s_i \to f}(s_i)} \, \mathrm{d}s_i = \mathcal{N}(p_i; \mu_{s_i}, \sigma^2_{s_i} + \beta^2) \tag{2.9}$$

This convolution of two Gaussians yields another Gaussian [HMG07]. We provide a full derivation of the message $m_{f \to p_i}(p_i)$ in Appendix A.4. The resulting factor-to-variable message is:

$$m_{f \to p_i}(p_i) = \mathcal{N}(p_i; \mu_{s_i}, \sigma^2_{s_i} + \beta^2) \tag{2.10}$$

2.4.3 Weighted Sum Factor

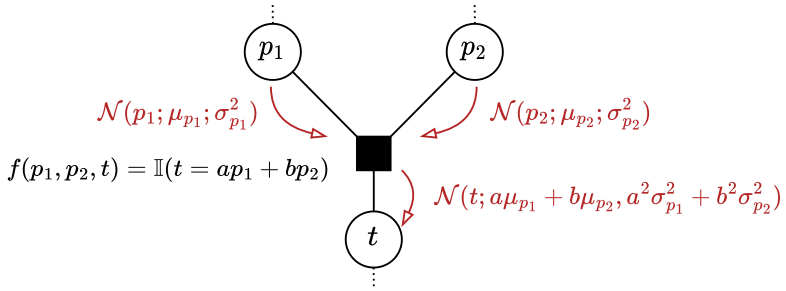


Figure 2.5: Weighted sum factor for two-player performances

The weighted sum factor (see Figure 2.5) maps a set of weighted inputs $p_1, ..., p_n$ to the output t. It is formally defined as:

$$f(p_1, p_2, \ldots, p_n, t) = \delta\left(t - \sum_{i=1}^{n} w_i \cdot p_i\right) \tag{2.11}$$

where w_i is a weight associated with input p_i, and n is the total number of input variables contributing to the weighted sum. The weights depend on the specific use case. To sum up a team's performance in the TrueSkill graph, all weights w_i are one. But they could, for instance, also be set to the fraction of the total game time a player participated in the game. The weighted sum factor is also used to calculate the team performance differences.

The message to t is:

$$m_{f \to t}(t) = \mathcal{N}\left(t; \sum_{i=1}^{n} w_i \cdot \mu_i, \sum_{i=1}^{n} w_i^2 \cdot \sigma_i^2\right) \tag{2.12}$$

The message to a variable p is:

$$m_{f \to p_j}(p_j) = \mathcal{N}\left(p_j; \frac{1}{w_j} \cdot \left(\mu_t - \sum_{i \neq j} w_i \cdot \mu_i\right), \frac{1}{w_j^2} \cdot \left(\sigma_t^2 + \sum_{i \neq j} w_i^2 \cdot \sigma_i^2\right)\right) \tag{2.13}$$

2.4.4 Greater-Than (Truncation) Factor

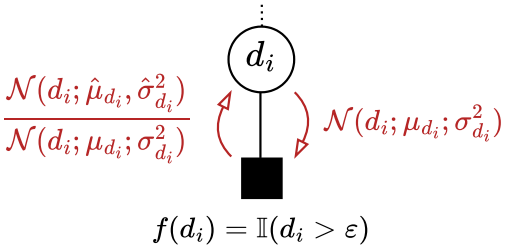


Figure 2.6: Greater than factor with its connected team performance difference variable d_i.

As input for the greater-than factor (aka truncation factor, see Figure 2.7), we have the performance difference d_i, which is modeled as a Gaussian. Using the observed game outcome, the greater-than factor "cuts" away all probability mass that contradicts the observation. This means that if team t_1 wins against team t_2, the performance difference $d = t_1 - t_2$ between teams has to be positive for the observed game. In other words, we can keep only the part of the difference distribution for positive differences $d > 0$:

$$f(d) = \mathbb{I}(d > 0) \tag{2.14}$$

Mathematically, this is equivalent to multiplying the distribution with a step function that has its step at the threshold $\varepsilon = 0$. This yields the truncated (Gaussian) distribution:

$$\mathbb{P}(d) \propto \mathbb{I}(d > 0)\,\mathcal{N}(d; \mu_d, \sigma_d^2) \tag{2.15}$$

This truncated distribution has to be normalized. The distribution can be written explicitly as:

$$\mathbb{P}(d) = \frac{\mathbb{I}(d > \varepsilon)\,\mathcal{N}(d; \mu_d, \sigma_d^2)}{\int_0^\infty \mathcal{N}(d; \mu_d, \sigma_d^2)\,\mathrm{d}d} = \frac{\mathbb{I}(d > 0)\,\mathcal{N}(d; \mu_d, \sigma_d^2)}{1 - \Phi\left(\frac{\varepsilon - \mu_d}{\sigma_d}\right)} = \frac{\mathbb{I}(d > 0)\,\mathcal{N}(d; \mu_d, \sigma_d^2)}{1 - \Phi\left(-\frac{\mu_d}{\sigma_d}\right)} \tag{2.16}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function (CDF), and $F(d; \mu_d, \sigma_d) = \Phi(\frac{d-\mu_d}{\sigma_d})$ is the CDF of the distribution $\mathcal{N}(d; \mu_d, \sigma_d^2)$. Because there is no probability mass below ε, the CDF of the truncated distribution integrates to the probability mass that is contained above ε. Mathematically, we have $\int_{-\infty}^{\infty} F(d; \mu_d, \sigma_d)\,\mathrm{d}d = 1 - F(\varepsilon; \mu_d, \sigma_d) = 1 - \Phi\left(\frac{\varepsilon - \mu_d}{\sigma_d}\right)$, which evaluates to $1 - \Phi\left(-\frac{\mu_d}{\sigma_d}\right)$ for $\varepsilon = 0$.

Continuing to the messages, we start with the factor-to-variable message. The true message is $m_{f \to d}(d) = \mathbb{I}(d > 0)$. However, since this indicator is not Gaussian, we replace it with a moment-matched (see Appendix A.3) Gaussian approximation. The approximated factor-to-variable message $\hat{m}_{f \to d}(d) = \mathcal{N}(d; \hat{\mu}_d, \hat{\sigma}_d^2)$ is chosen so that when we multiplying it with the incoming message $\mathcal{N}(d, \mu_d, \sigma_d^2)$, the resulting Gaussian matches the truncated Gaussian's first two moments.

Using moment matching, the resulting message is:

$$\hat{m}_{f \to d_i}(d_i) = \frac{\hat{p}(d_i)}{m_{d_i \to f(d_i)}} = \frac{\mathcal{N}(d_i; \hat{\mu}_{d_i}, \hat{\sigma}_{d_i}^2)}{\mathcal{N}(d_i; \mu_{d_i}, \sigma_{d_i}^2)} = \mathcal{N}\left(d_i;\, \hat{\mu}_{d_i}^*, \hat{\sigma}_{d_i}^{*2}\right) \tag{2.17}$$

$$\text{with} \quad \hat{\sigma}_{d_i}^{*2} = \left(\frac{1}{\sigma_{d_i}^2\left(1 - \omega\left(-\frac{\mu_{d_i}}{\sigma_{d_i}}\right)\right)} - \frac{1}{\sigma_{d_i}^2}\right)^{-1} \tag{2.18}$$

$$\hat{\mu}_{d_i}^* = \hat{\sigma}_{d_i}^{*2}\left(\frac{\mu_{d_i} + \sigma_{d_i}\,\nu\left(-\frac{\mu_{d_i}}{\sigma_{d_i}}\right)}{\sigma_{d_i}^2\left(1 - \omega\left(-\frac{\mu_{d_i}}{\sigma_{d_i}}\right)\right)} - \frac{\mu_{d_i}}{\sigma_{d_i}^2}\right). \tag{2.19}$$

with $\nu(t) = \frac{\mathcal{N}(t;0,1)}{1 - F(t;0,1)} = \frac{\mathcal{N}(t;0,1)}{1 - \Phi(t)}$ and $\omega(t) = \nu(t)[\nu(t) + t]$. The true and approximated message and marginal are visualized in Figure 2.7. We refer to Herbrich [Her24; HMG07] for the derivation of the moment-matched message.

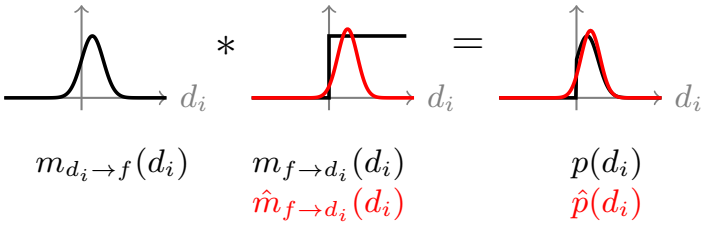


Figure 2.7: Greater-than factor, showing the true messages in black and moment-matched Gaussian approximation in red. The approximation for the factor-to-variable message is computed by using the approximated marginal as $\hat{m}_{f \to d_i}(d_i) = \frac{\hat{p}(d_i)}{m_{d_i \to f(d_i)}} = \frac{\mathcal{N}(d_i; \hat{\mu}_{d_i}, \hat{\sigma}^2_{d_i})}{\mathcal{N}(d_i; \mu_{d_i}, \sigma^2_{d_i})}$.

2.5 Distributions

All messages and marginals in a factor graph are represented by probability distributions. The following sections introduce the preliminaries for the various representations we explored throughout the project.

2.5.1 Uniform Distribution

The uniform distribution is an unimodal continuous probability distribution that models events equally likely to occur. Let $f : \mathbb{R} \to \mathbb{R}$ be a function. We define f as a uniform density written as

$$f(x) = \mathcal{U}(x; a, b) \quad \text{for all } x \in \mathbb{R},$$

where $a \in \mathbb{R}$ and $b \in \mathbb{R}$. $\mathcal{U}(x; a, b)$ denotes the uniform distribution with left interval boundary a and right interval boundary b, defined by the probability density function:

$$\mathcal{U}(x; a, b) = \begin{cases} \frac{1}{b-a} & \text{, if } a \leq x \leq b \\ 0 & \text{, otherwise.} \end{cases} . \tag{2.20}$$

The uniform density fulfills the following properties:

$$\forall x \in \mathbb{R} : f(x) \geq 0 \tag{2.21}$$

$$\int_{-\infty}^{\infty} f(x) dx = 1. \tag{2.22}$$

2.5.2 Mixture of Uniforms

Combining several adjacent uniform distributions makes it possible to approximate any continuous function via discretization. A Mixture of Uniforms (MoU) is a weighted sum of M uniform densities given by the equation:

$$p(x) = \sum_{i=1}^{M} w_i * \mathcal{U}(x; a_i, b_i)$$

where w_i are the mixture weights, and $\mathcal{U}(x; a_i, b_i)$ is the density at x for the uniform distribution i. Each weight w_i satisfies $w_i \geq 0$ and $\sum_{i=1}^{M} w_i = 1$ [GAB19].

2.5.3 Gaussian Distribution

Instead of discretizing messages in a factor graph, we can approximate them with a parameterized class of functions. A Gaussian distribution, also known as a normal distribution, is defined by its probability density function where μ is the mean or expectation of the distribution (and also its median and mode), σ is the standard deviation, and σ^2 is the variance.

We define f as a Gaussian density written as

$$f(x) = \mathcal{N}(x; \mu, \sigma^2) \quad \text{for all } x \in \mathbb{R},$$

where $\mu \in \mathbb{R}$ and $\sigma > 0$. $\mathcal{N}(x; \mu, \sigma^2)$ denotes the normal distribution with mean μ and variance σ^2, defined by the probability density function:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.23}$$

Multiplication of Gaussian Densities Let $\mu_1, \mu_2 \in \mathbb{R}$ and $\sigma_1, \sigma_2 > 0$. Then for all $x \in \mathbb{R}$ we have

$$\mathcal{N}(x; \mu_1, \sigma_1^2)\mathcal{N}(x; \mu_2, \sigma_2^2) = \mathcal{N}(x; \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}, \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}) \tag{2.24}$$

Addition of Gaussian Random Variables Let $\mu_1, \mu_2 \in \mathbb{R}$ and $\sigma_1, \sigma_2 > 0$. Let $X \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $Y \sim \mathcal{N}(\mu_2, \sigma_2^2)$. Then for constants $a, b \in \mathbb{R}$, the linear combination $Z = aX + bY$ is also normally distributed, given by:

$$Z \sim \mathcal{N}(a\mu_1 + b\mu_2, a^2\sigma_1^2 + b^2\sigma_2^2) \tag{2.25}$$

Scaling of a Gaussian Density Let $\mu \in \mathbb{R}$ and $\sigma > 0$. Then for all $x, c \in \mathbb{R}$ we have

$$c \cdot \mathcal{N}(x; \mu, \sigma^2) = \mathcal{N}(x; \mu \cdot c, \sigma^2 \cdot c^2) \tag{2.26}$$

and analogously for division:

$$\frac{1}{c} \cdot \mathcal{N}(x; \mu, \sigma^2) = \mathcal{N}(x; \frac{\mu}{c}, \frac{\sigma^2}{c^2}) \tag{2.27}$$

2.5.4 Mixture of Gaussians

Gaussian distributions are always unimodal. To enable factor graphs with multimodal distributions, we also consider Mixtures of Gaussian. A Mixture of Gaussians (MoG) is a weighted sum of M component Gaussian densities given by the equation:

$$p(x) = \sum_{i=1}^{M} w_i \mathcal{N}(x; \mu_i, \sigma_i^2),$$

where w_i are the mixture weights, and $\mathcal{N}(x; \mu_i, \sigma_i^2)$ are the component Gaussian densities. Each weight w_i satisfies $w_i \geq 0$ and $\sum_{i=1}^{M} w_i = 1$.

For MoGs, the result of a multiplication and the addition of the random variables distributed after MoGs is again a MoG. However, for two MoGs with N and M components, the result has $N \cdot M$ components.

Multiplication of Mixture of Gaussians Let $p(x)$ and $q(x)$ be two MoGs defined as:

$$p(x) = \sum_{i=1}^{M} w_i \mathcal{N}(x; \mu_i, \sigma_i^2) \quad \text{and} \quad q(x) = \sum_{j=1}^{N} v_j \mathcal{N}(x; \nu_j, \tau_j^2) \tag{2.28}$$

The product of $p(x)$ and $q(x)$ results in another mixture model, where each pair of Gaussian components from $p(x)$ and $q(x)$ is multiplied. The resultant mixture model is:

$$p(x) \cdot q(x) = \sum_{i=1}^{M} \sum_{j=1}^{N} w_{ij} \mathcal{N}(x; \mu_{ij}, \sigma_{ij}^2) = \sum_{i=1}^{M} \sum_{j=1}^{N} w_{ij} \mathcal{N}(x; \mu_i, \sigma_i^2) \mathcal{N}(x; \nu_j, \tau_j^2), \tag{2.29}$$

where μ_{ij} and σ_{ij}^2 are calculated as:

$$\mu_{ij} = \frac{\tau_j^2 \mu_i + \sigma_i^2 \nu_j}{\sigma_i^2 + \tau_j^2} \tag{2.30}$$

$$\sigma_{ij}^2 = \frac{\sigma_i^2 \tau_j^2}{\sigma_i^2 + \tau_j^2} \tag{2.31}$$

analogously to (2.24). The new weight w_{ij} for each resulting Gaussian component is:

$$w_{ij} = w_i v_j \frac{1}{\sqrt{2\pi(\sigma_i^2 + \tau_j^2)}} \exp\left(-\frac{(\mu_i - \nu_j)^2}{2(\sigma_i^2 + \tau_j^2)} \right) \tag{2.32}$$

as shown in [SFH]. The new mixture model has more components, specifically $N \cdot M$.

Addition of Mixture of Gaussians Let $p(x)$ and $q(x)$ be two MoGs as defined above (2.28) Let $X \sim p(x)$ and $Y \sim q(x)$ then $Z \sim aX + bY$. Then for constants $a, b \in \mathbb{R}$, the linear combination $Z = aX + bY$ is also a MoGs, given by:

$$Z \sim \sum_{i=1}^{M} \sum_{j=1}^{N} w_i v_j \mathcal{N}(x; a \cdot \mu_i + b \cdot \nu_j, a^2 \cdot \sigma_i^2 + b^2 \cdot \tau_j^2) \tag{2.33}$$

The new mixture model again increases the number of components to $N \cdot M$.

Scaling of Mixture of Gaussians Scaling a MoG by a scalar $c \in \mathbb{R}$ is simply scaling the individual components as in (2.5.3) without changing the weights.

2.6 Runalls Algorithm

To use MoGs in Factor Graphs, we need them closed under multiplication. The product and sum of two MoGs is again a MoG as explained in Equation 2.28. As shown in Equation 2.5.4, it is also easy to scale them. However, the number of components would increase exponentially if we continuously multiply MoGs with more than one component. Several approximation algorithms reduce the number of components of a MoG. A comprehensive comparison can be found in [Cro+11]. We re-implemented Runalls' algorithm.

Runnalls' is a greedy algorithm that minimizes an upper bound on the increase in the Kullback Leibler divergence between the original and the reduced mixture [Run07].

For merging components [Run07] gives the following equations:

$$w_{\text{merged}} = w_i + w_j \tag{2.34}$$

$$\mu_{\text{merged}} = \frac{1}{w_{\text{merged}}} \cdot (w_i \mu_i + w_j \mu_j) \tag{2.35}$$

$$\sigma^2_{\text{merged}} = \frac{1}{w_{\text{merged}}} \cdot (w_i \sigma_i^2 + w_j \sigma_j^2 + \frac{w_i w_j}{w_{\text{merged}}} (\mu_i - \mu_j)^2) \tag{2.36}$$

Algorithm 1 Runnalls' algorithm [Cro+11]

1. Set the current mixture to the full mixture.
2. The cost of merging components i and j in the current mixture is the upper bound on the increase in the KL divergence [Run07],

   $$c_{i,j} = \frac{1}{2} \left((w_i + w_j) \log |\sigma_{i,j}| - w_i \log |\sigma_i^2| \right) - \frac{1}{2} w_j \log |\sigma_j^2|$$

   where $\sigma^2_{i,j}$ corresponds to Equation 2.36 for merging only components i and j. Merge the components of the current mixture having the lowest $c_{i,j}$ using Equation 2.34-Equation 2.36 and set the current mixture to the result.
3. If the current mixture has the desired number of components, quit. Otherwise, go back to step 2.

For reducing the number of components of two MoGs with N and M components each to C after performing an addition or multiplication, the runtime of this algorithm $(N \cdot M - C) \cdot N^2 M^2$.

2.7 Expectation Maximization

To fit a mixture of Gaussians to a given set of one-dimensional data samples, we maximize the (log) likelihood of the samples iteratively via the expectation-maximization (EM) algorithm [Moo96]. Its two alternating steps are:

Expectation (E)-Step: Compute posterior responsibilities:

$$\gamma_{ik} = \frac{w_k \, \mathcal{N}(x_i \mid \mu_k, \sigma_k^2)}{\sum_{j=1}^{K} w_j \, \mathcal{N}(x_i \mid \mu_j, \sigma_j^2)} \tag{2.37}$$

Maximization (M)-Step: Update parameters:

$$w_k^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} \gamma_{ik} \tag{2.38}$$

$$\mu_k^{(t+1)} = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{\sum_{i=1}^{N} \gamma_{ik}} \tag{2.39}$$

$$\sigma_k^{2(t+1)} = \frac{\sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k^{(t+1)})^2}{\sum_{i=1}^{N} \gamma_{ik}} \tag{2.40}$$

Chapter 3

Mixture of Uniforms Approach

In the original TrueSkill, messages, and marginals are initialized as Gaussians. This simplifying assumption works well and is computationally efficient but has the obvious drawback that the messages are inherently limited in their expressiveness. For example, it is impossible to represent multimodal or asymmetric distributions accurately. In an attempt to remove the Gaussian constraint, we explore MoUs since they can theoretically approximate the shape of any distribution with a resolution that is high enough.

3.1 Initialization Techniques

Naturally, MoUs have their own set of complications. For example, while Gaussians can be represented by only two parameters (mean and variance), we must save $2n$ parameters for a MoU with n uniforms, also called buckets. Each parameter represents either the lower or upper bound for one of the buckets. The bucket thresholds $a(i)$ and $b(i)$ in MoUs can be set using different approaches, which impact computational costs and complexity during further processing steps. In the following, we will outline the techniques applied in the scope of this master project to initialize the bucket thresholds.

3.1.1 Equal Bucket-Width Representation

An equal bucket-width (EBW) MoU is a MoU in which, for each of two adjacent thresholds $t(i)$ and $t(i+1)$, we find a constant width:

$$k = |t(i+1) - t(i)| \tag{3.1}$$

Our implementation requires the specification of start and end thresholds $t(1)$ and $t(n)$. The amount of buckets in a MoU will then be:

$$n = \left\lceil \frac{|t(n) - t(1)|}{k} \right\rceil \tag{3.2}$$

While the advantage of this approach lies in the simplicity of combining two distributions, it leads to one clear disadvantage: If we want to cover an extensive range while maintaining a high resolution (low bucket width), we end up with a large number of bucket thresholds n, leading to high computational costs. Further, since every bucket has an equal resolution, it is impossible to accommodate areas with a high frequency of change.

3.1.2 Equal Bucket-Density Representation

While the method of equal bucket-width MoU (see Subsection 3.1.1) is a discretization of the x-axis of a CDF into n buckets, it is also possible to discretize the y-axis, leading to different bucket widths, but equal bucket densities. An equal bucket-density (EBD) MoU $p(x)$ is a MoU in which for each of the two adjacent thresholds $t(i), t(i+1)$ and n buckets, it holds:

$$\int_{t(i)}^{t(i+1)} p(x)dx = \frac{1}{n} \tag{3.3}$$

Evidently, it follows $\int_{t(1)}^{t(n)} p(x)dx = 1$.

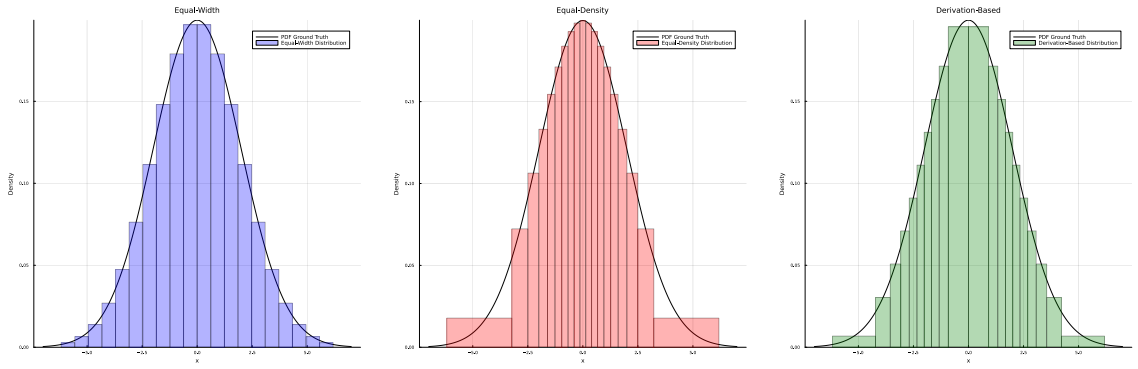
As a result, areas with a higher point density will receive a higher resolution on the x-axis and a smaller bucket width. In contrast, low-density areas will be summarized into larger buckets. This is advantageous if areas with higher density are seen as more important and require higher approximation quality. As a result, this setup can no longer guarantee the perfect alignment of buckets of two distinct MoUs. Therefore, operations have to take the case of half-overlapping buckets into account.

3.1.3 Derivation-Based Bucket-Density Representation

While the equal bucket-density approach works well for areas with high point density, it degrades in areas where density is low, for example, in the tails of distributions. One approach to mitigate this issue is to use derivation-based bucket densities (DBD) to discretize the distribution. We refer to this approach simply as derivation-based. Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function and $|f'(x)|$ be the absolute derivation of $f(x)$. For all buckets $[t(i), t(i+1)]$ of the MoU, it shall hold:

$$\int_{t(i)}^{t(i+1)} |f'(x)|dx = \frac{1}{n} \tag{3.4}$$

Therefore, each threshold must be placed such that the CDF of $|f'(x)|$ is discretized into equal parts on the y-axis as seen in Figure 3.1. As a result, the buckets will have larger widths in areas where the CDF did not increase much, which means that $|f'(x)|$ must have had a low value. This, in turn, signifies a small slope in $f(x)$, resulting in fewer buckets where the slope is low and more buckets where the slope is high. Overall, we allocate more resolution to areas with more changes, leading to an efficient distribution of buckets.



(a) Equal width (b) Equal density (c) Derivation-based

Figure 3.1: Comparison of initialization methods (num_quantiles = 20)

3.2 Multiplication Techniques

In TrueSkill, multiplication is one of the most important operations for performing skill updates (see Section 2.4). While multiplication is straightforward when dealing with two Gaussian distributions (see Subsection 2.5.3), other distributions need different algorithms. The goal is to find a balance between accuracy and computational efficiency. When dealing with MoUs, multiplication is especially challenging due to the bucket thresholds, which may inherently differ between the two distributions being multiplied. Therefore, the following subsections introduce the necessary techniques for applying multiplication based on MoUs. This lays the foundation for the application of MoUs in TrueSkill.

3.2.1 Overlap Handling

Overlap handling is the general approach used as the backbone for all more refined techniques in this section. Instead of only considering multiplication, this approach also applies to addition. ($\diamond \in \{+, *\}$). By iterating over the thresholds of both distributions p_1 and p_2 from start to end, we need to handle three distinct cases.

Full Alignment If the bucket thresholds of the two MoU operands align, we can keep the existing thresholds for the buckets and calculate the new weights as follows: Let p_1 and p_2 be two equal bucket-width MoUs. If we enforce $t_1(1) = t_2(1)$, $t_1(n) = t_2(n)$ and $k_1 = k_2$ such that all buckets align, any operation $\diamond \in \{+, *\}$ is the element-wise operation between the corresponding weights $w_1(i)$ and $w_2(i)$ of p_1 and p_2:

$$w_{new}(i) = w_1(i) \diamond w_2(i) \tag{3.5}$$

$$p_1(x) \diamond p_2(x) = \sum_{i=1}^{k} w_{new}(i) * \mathcal{U}(x; t_1(i), t_1(i)) \tag{3.6}$$

No Overlap In this case, we assume the non-existent bucket's weight to be zero and apply the operation to the existing bucket's weight $w(i)$, such that $w_{new} = w(i) \diamond 0$.

Partial overlap Assume a bucket of p_1 has thresholds $t_1(i)$ and $t_1(i+1)$ and a bucket of p_2 has thresholds $t_2(j)$ and $t_2(j+1)$. If these buckets overlap, there must then be a start point a with $a \geq t_1(i)$ and $a \geq t_2(j)$ and an end point b with $b \leq t_1(i+1)$ and $b \leq t_2(j+1)$ such that $[a, b]$ is the overlapping area. We therefore create a new bucket with thresholds a, b, and a weight

$$w_{new} = \left(\frac{w_1}{t_1(i+1) - t_1(i)} \diamond \frac{w_2}{t_2(j+1) - t_2(j)}\right) * (b - a) \tag{3.7}$$

Here, we normalize w_1 and w_2 with their respective initial bucket width and multiply with the overlapping bucket width $b - a$.

3.2.2 Slope Calculation

In the MoU approach, each bucket represents a uniform distribution. In this representation, the information about the local slope of the distribution gets lost. This is especially the case for MoUs with a low amount of buckets. However, when multiplying overlapping buckets, this information could be helpful in better approximating the product of the densities. To incorporate the information about the slope of the probability distribution, we approximate slope information for each bucket. We achieve this by calculating the slope of a straight line that runs through the centers of the neighboring buckets at the density's height in the corresponding bucket (see black line in Figure 3.2). The following formula calculates the slope m:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.8}$$

y_2 is the height of the right bucket, y_1 is the height of the left bucket, and x_2 and x_1 are the x-coordinates of the center of the right and left bucket. This line is then shifted to the center point of the bucket to which we want to add the slope. It is stored from the start threshold of the bucket to the end threshold (see red line in Figure 3.2).

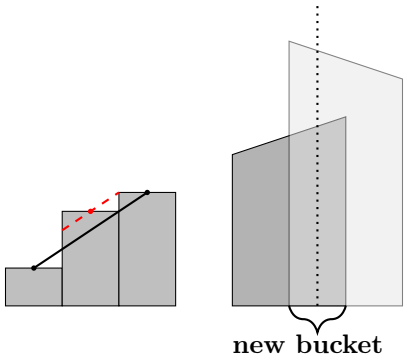


Figure 3.2: Calculation of the slope (left) and multiplication (right).

Using the slope approach, we work with estimated point densities when multiplying two probability distributions with buckets. First, the intersection of the two slopes with the center point of the new overlapping bucket is calculated (intersection of the slopes with the dotted black line in Figure 3.2). Second, we multiply these two values to determine the height of the new bucket. Lastly, we apply the slope approach again to get a distribution with slopes.

3.2.3 Shifted Mean Multiplication

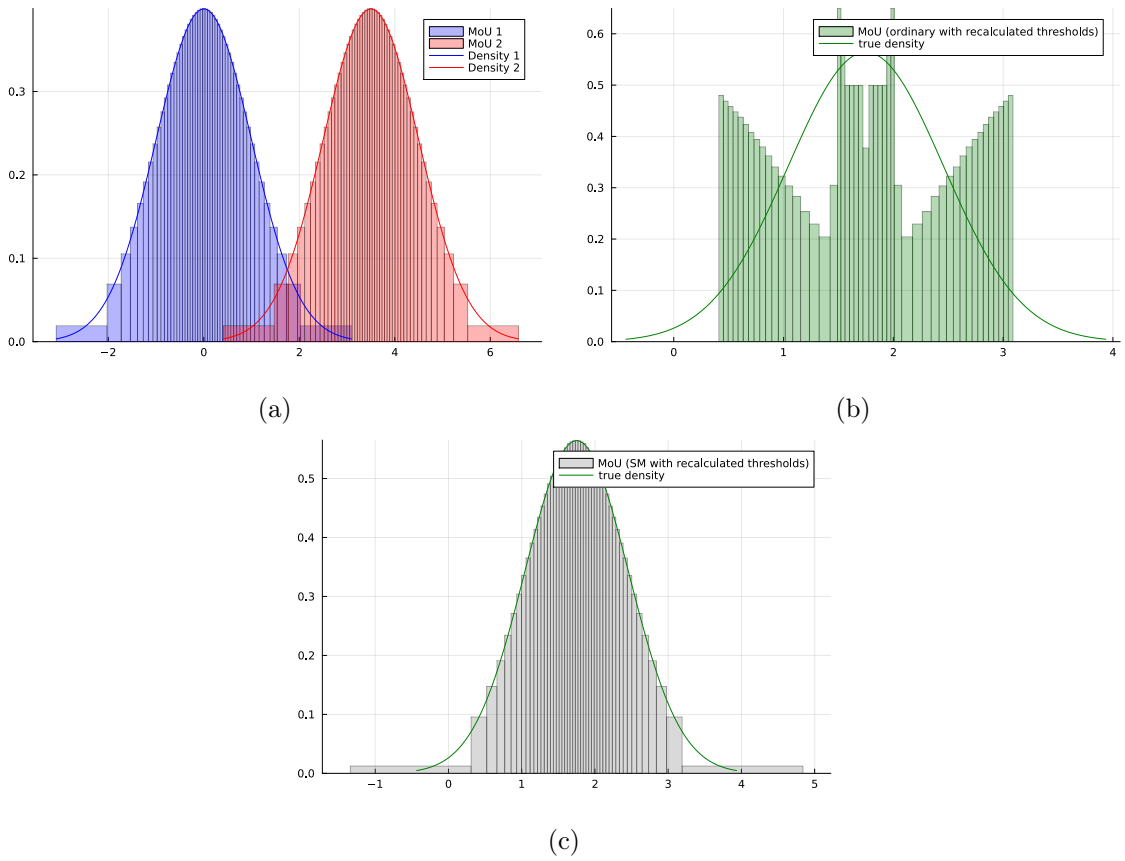
Since MoUs represent approximations for target distributions, e.g., Gaussian distributions, they possess inherent issues with respect to different operations, such as the multiplication of two MoUs. The biggest one is that the multiplications are only well-defined in the range where both distributions are non-zero. This often results in the deterioration of the resulting MoU. Compared to the actual PDF, the resulting MoU may show severe deterioration depending on the overlap between the original MoUs.

Another approach to solving this problem is the Shifted Mean Approach (SMA). The SMA ensures that all thresholds from both original MoUs are shifted such that for μ_1 and μ_2, it holds that $\mu_1 = \mu_2 = 0$. Subsequently, the ordinary overlap handling approach is applied. Afterward, the resulting MoU is obtained by shifting the newly calculated thresholds by:

$$\Delta = \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \tag{3.9}$$

(see Equation 2.5.3), which represents the mean of the resulting MoU.

By doing so, the SMA provides a more robust approach than the general overlap handling approach, effectively addressing the deterioration issue that arises in small overlap ranges. However, this approach is not feasible when the original MoUs do not approximate a Gaussian distribution. Due to the formula above, the shift of MoUs may not accurately reflect the true resulting mean.



(a) (b)

(c)

Figure 3.3: A multiplication operation of two Gaussian distributions (a) approximated as MoUs with number of buckets = 50. MoU 1: $\mu_1 = 0, \sigma_1^2 = 1$ & MoU 2: $\mu_2 = 3.5, \sigma_2^2 = 1$. (b) shows the resulting MoU based on the ordinary overlap handling approach (see Subsection 3.2.1). (c) depicts the resulting MoU based on the shifted mean approach. Either MoU has been recalculated to match the original number of buckets (= 50).

Chapter 4

Mixture of Gaussians Approach

Using MoGs instead of Gaussians in factor graph methods such as TrueSkill can have multiple advantages. Firstly, we can assume more complex and multimodal priors. This makes sense in cases such as skill estimation, where a player might have different skill levels depending on certain conditions, such as form on the day. Additionally, we can employ MoGs to better approximate calculations, such as the truncation step in TrueSkill (see Subsection 2.4.4). However, the added complexity of MoGs also introduces challenges. Since factor graphs are highly dependent on multiplication and division operations, we run into multiple problems. First of all, the multiplication of two MoGs having n and m components results in a MoG with $n \times m$ components. One way to keep the runtime complexity bounded is to reduce the number of components after each multiplication. This is why we apply the Runalls' algorithm after each step, as seen in Section 2.6. The MoGs are then closed under multiplication and addition, with the trade-off that the results will be approximations. Second, the division of two MoGs is not clearly defined when obtaining marginal distributions. As an alternative, we can multiply all incoming messages. However, this is computationally inefficient, so we introduce segment trees and prefix and postfix arrays to reduce the runtime in Section 4.2.

4.1 Mixture of Gaussians Factors

This section describes the details for implementing TrueSkill with mixtures of Gaussians as seen in Figure 2.2. This means that we need to adapt all factors to use MoGs. While in the prior factor, the underlying distribution is simply replaced by a MoG, we need to modify the other factors more.

4.1.1 Gaussian Mean Factor

The Gaussian mean factor for MoGs extends the regular Gaussian mean factor as seen in Equation 2.9 to cases where a player's skill s_i is modeled using a MoG instead of a single Gaussian distribution. This allows for multimodal uncertainty in skill estimation and captures diverse performance patterns across game conditions.

Forward Message MoG Mean Factor

As with the unimodal case, the factor maps skill s_i to performance p_i. The message to p_i is:

$$m_{f \to p_i}(p_i) = \sum_k w_k \mathcal{N}(p_i; \mu_k, \sigma_k^2 + \beta^2),$$

where $\{w_k, \mu_k, \sigma_k^2\}$ are the weights, means, and variances of the Gaussian component k in the skill distribution. Each component's variance is increased by β^2, just as in the unimodal case, ensuring uncertainty propagates uniformly over all components of the MoG.

The backward message follows the same principle, substituting p_i with s_i.

4.1.2 Weighted Sum Factor

In contrast to the weighted sum factor for Gaussians in Subsection 2.4.3, the summation of two MoGs is done component-wise. This means that each Gaussian component in one mixture is paired with every Gaussian component of the other mixture. Given two independent MoG distributions:

$$p(x) = \sum_i w_i \mathcal{N}(x \mid \mu_i, \sigma_i^2), \quad p(y) = \sum_j w'_j \mathcal{N}(y \mid \mu'_j, \sigma'^2_j), \tag{4.1}$$

where w_i and w'_j are the respective component weights, the weighted sum transformation:

$$Z = aX + bY \tag{4.2}$$

is performed by applying the sum operation to each component pair in the mixtures. Since the sum of two independent Gaussian-distributed variables remains Gaussian:

$$aX + bY \sim \mathcal{N}(a\mu_i + b\mu'_j, a^2\sigma_i^2 + b^2\sigma'^2_j), \tag{4.3}$$

the resulting Gaussian mixture for Z is:

$$p(Z) = \sum_j \sum_i w_i w'_j \mathcal{N}(z_i; a\mu_i + b\mu'_j, a^2\sigma_i^2 + b^2\sigma'^2_j). \tag{4.4}$$

This means that every Gaussian component in X is combined with every Gaussian component in Y to form a new component in Z. The weight of each resulting component is the product of the original weights, ensuring the overall mixture maintains proper probability mass allocation.

Forward Message MoG Weighted Sum Factor

The message to t is:

$$m_{f \to t}(t) = \sum_j \sum_i w_i w'_j \mathcal{N}(t_{ij}; a\mu_i + b\mu'_j, a^2\sigma_i^2 + b^2\sigma'^2_j)$$

The backward message can be calculated analogously by applying Equation 2.13 component-wise.

4.1.3 Greater-Than Factor

There are different ways to implement the greater-than factor for MoGs. We have explored two possibilities:

Component-wise Truncation Approximation For an MoG with density function:

$$p(x) = \sum_i w_i \mathcal{N}(x \mid \mu_i, \sigma_i^2) \tag{4.5}$$

truncation at threshold T is applied component-wise to each Gaussian in the mixture. That is, each component $\mathcal{N}(x \mid \mu_i, \sigma_i^2)$ is transformed into a truncated Gaussian:

$$p^*(x) = \sum_i w_i' \mathcal{N}^+(x \mid \mu_i^*, (\sigma_i^*)^2) \tag{4.6}$$

The new weight w_i' of each component is the old weight multiplied by the survival function (i.e., one minus the cumulative density function) of the original Gaussian at T. This way, components past the threshold retain their relative probability mass, while components with much probability mass below the threshold only retain their mass past the threshold.

MoG Component-wise Truncation

Formally, for each component $\mathcal{N}(X \mid \mu_i, \sigma_i^2)$, the updated mean, variance, and weights are:

$$\mu_i^* = \mu_i + \sigma_i \cdot v(\alpha_i) \tag{4.7}$$
$$(\sigma_i^*)^2 = \sigma_i^2 \cdot (1 - w(\alpha_i)) \tag{4.8}$$
$$w_i* = w_i \cdot (1 - \Phi(\alpha_i)) \tag{4.9}$$

where $\{w_i, \mu_i, \sigma_i^2\}$ are the weights, means, and variances of the Gaussian component i in the skill distribution. Each component's variance is increased by β^2, just as in the unimodal case, ensuring uncertainty propagates uniformly over all components of the MoG.

Here, $\alpha_i = \frac{T - \mu_i}{\sigma_i}$ is the standardized threshold, $\Phi(\alpha)$ is the cumulative density function (CDF) of the standard normal distribution, while $v(\alpha)$ and $w(\alpha)$ are correction terms derived from the truncated Gaussian moments as seen in Subsection 2.4.4. After computing these truncated components, the final Gaussian mixture is re-normalized to maintain proper probability mass.

Overall, this procedure produces three cases for each component: First, a component can lie much to the right of T, in which case nothing changes. Second, a component can lie much to the left of T. If this happens, the component is virtually eliminated when rescaling if the weight is almost zero and other components have a much higher weight. Lastly, a component can lie such that T splits it into two parts. The left-most part is once again discarded, while the right part is subject to the approximation method above. This approach minimizes the component-wise KL divergence through moment matching as seen in Section 2.3. However, there might be better approximations when approximating the entire MoG instead of making component-wise approximations. Furthermore, the case exists that all components lie to the left of the threshold, eliminating all components. Even though we have several ideas for resolving the issue, such as only considering the right-most distribution, a definitive solution is still unclear.

Expectation Maximization As a second variant, we can use the Expectation Maximization (EM) algorithm [Moo96] to obtain a higher-quality approximation. This comes at the cost of having a slower algorithm because it has to iteratively optimize the likelihood of samples $\{x_i\}_{i=1}^N$

under the MoG parameters. In practice, we first get 1,000 samples x_i from the truncated MoG by using rejection sampling to keep only samples that satisfy $x_i > t$ with the threshold $t = 0$.

In the first step, we apply rejection sampling (see Algorithm 2) from the truncated MoG. Given a MoG with K components:

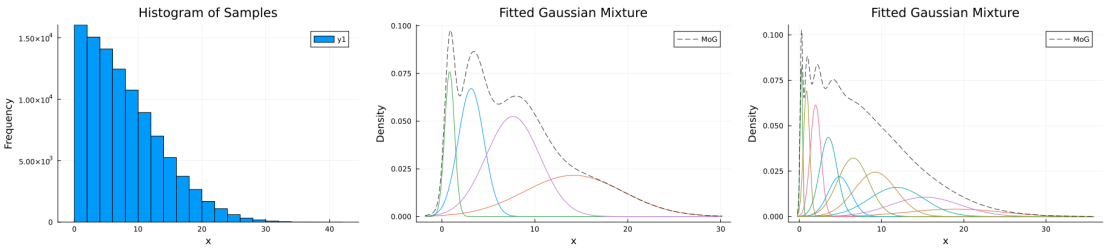
$$p(x) = \sum_{k=1}^{K} w_k \, \mathcal{N}(x \mid \mu_k, \sigma_k^2), \tag{4.10}$$

we use rejection sampling to generate a sample x_i from the truncated distribution that enforces the constraint $x_i > t$:

Algorithm 2 Rejection sampling from a truncated mixture of Gaussians

1. Initialize an empty set $S = \emptyset$ to store the samples.
2. While $|S| < N$:
   1. Draw component index $k \sim \text{Categorical}(w_1, \ldots, w_K)$.
   2. Sample $x_i \sim \mathcal{N}(\mu_k, \sigma_k^2)$.
   3. If $x_i > t$, update $S \leftarrow S \cup \{x_i\}$.
3. Return the set of samples S.

Given N truncated samples $\{x_i\}_{i=1}^{N}$, we fit a new MoG $q(x)$ via the expectation-maximization (EM) algorithm. We initialize the Mixture of Gaussians by slightly perturbing the means of the components from the input distribution by a small $\epsilon \sim \mathcal{N}(0, 0.01)$ to break the symmetry. This is because our implementation also works with MoGs where all means are initialized to the same value and slightly perturbed, so the EM algorithm can update the distribution assignments differently. The weights and variances are taken directly from the input distribution. Then we run the expectation-maximization algorithm as described in Section 2.7 for 10 iterations, yielding a maximum likelihood estimator (MLE)-based approximation $q(x) \approx p(x \mid x > t)$. As an additional stopping criterion, we perform early stopping when the likelihood does not change by more than a tolerance of $1e-6$. The results for an MoG of four and ten components can be seen in Figure 4.1.



(a) Samples of the true truncated MoG marginal

(b) EM approximation of the truncated MoG using four components

(c) EM approximation of the truncated MoG using ten components

Figure 4.1: Truncated MoG marginal approximations with expectation maximization

EM-Based Truncation Approximation

Input: Number of EM iterations k (we set it to 10 in our experiments), log-likelihood convergence threshold ($\varepsilon = 1e-6$).

1. Generate samples $\{x_i\}$ from the truncated MoG using rejection sampling (only keep $x_i > t$).
2. Slightly perturb the means from the input MoG to break symmetry and use the input weights/variances as a starting point.
3. Run k iterations (or until log-likelihood convergence) of EM to fit a new MoG $q(x)$ that maximizes the likelihood of the truncated samples.

Result: A MoG $q(x) \approx p(x \mid x > t)$ that approximates the truncated MoG more accurately than component-wise truncation.

4.2 Message Computing

When all messages in a factor graph belong to the exponential family or other considerably easy-to-work-with parametric families, one can divide out one distribution from the product of many distributions. This allows each variable's outgoing message to be computed as the marginal divided by the incoming message. However, if we chose to represent messages as a MoG (see Subsection 2.5.4) or any other family of distributions that do not have an exact multiplicative inverse in the same family, then performing a division of the form:

$$p_{\setminus i}(x) = \frac{\prod_{j=1}^{N} p_j(x)}{p_i(x)} \tag{4.11}$$

can be intractable or result in a distribution not part of the chosen distribution family.

Concretely for factor graphs, when using Gaussians, if $m_{f \to x}(x)$ denotes the outgoing message from factor f to variable x, and $m_{x \to f}(x)$ represents the incoming message from x to f, the outgoing message is calculated by:

$$m_{f \to x}(x) = \frac{p_x(x)}{m_{x \to f}(x)}$$

where $p_x(x)$ is the marginal of x. We derivate this equality in Appendix A.7.

For MoGs to calculate an outgoing message of a variable, instead of storing the product of all incoming messages and dividing it by one incoming message, we propose to compute the product of all incoming messages, excluding that message for every outgoing message, which avoids division. However, this formulation requires an implementation that allows for the quick computation of products of many distributions to maintain computational efficiency. This will be especially important for Bayesian Neural Networks (introduced in Chapter 5), where many neurons in a layer and large batch sizes can result in factors connected to many variables.

4.2.1 Segment Tree

The naive approach to compute the outgoing message from a factor to a variable x would involve computing the product of all incoming messages to that factor, excluding the message from x, for each outgoing message, leading to a runtime of $O(n^2)$. To improve this, we propose efficiently managing these computations by using a segment tree (see Figure 4.2). A segment tree allows for

efficient aggregation (e.g., product, sum), update operations in $O(log(n))$ time, and $O(n)$ space complexity, which we can leverage when computing products of messages for subsets of variables efficiently.

Implementation We implement the segment tree as an array with a total size of $2n$, where n is the input data size. In this array, the internal nodes occupy the first half from index 1 to $n-1$ (for 1-based indexing), with the root at index 1. The data nodes are stored from index n to $2n-1$ in the second half. Our implementation does not need point or range updates since we always instantiate new trees in a (neural network) forward pass. During instantiation, a parent node i is created with the operation applied to its children $2i$ and $2i+1$.

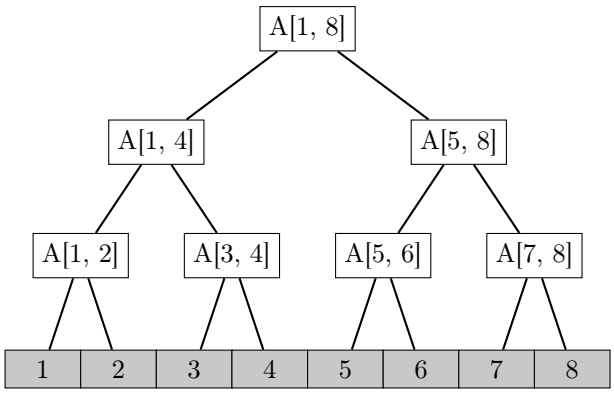


Figure 4.2: Example segment tree for a data array of size 8. Gray leaf nodes represent the data entries. Root and intermediate nodes are labeled with the range of the input array that they cover.

Queries A range query is computed by using two pointers, left and right. The left pointer starts from the range start, and the right pointer from the range end. From these indices, we traverse the segment tree according to the following rules:

- If the left pointer points to a right node, include its value in the result and move to its right neighbor (increment by one).
- If the right pointer points to a left node, include its value in the result and move to its left neighbor (decrement by one).

At each iteration, both pointers move to their respective parent nodes by dividing the pointer indices by two. This continues until the left pointer surpasses the right pointer, at which point the entire range is included in the result. This traversal is bounded by the number of layers of the tree. In the worst case, we move up to the root node to query the entire data array range. This results in the time complexity of $O(log(n))$. Figure 4.3 shows an example of a range query using a segment tree.

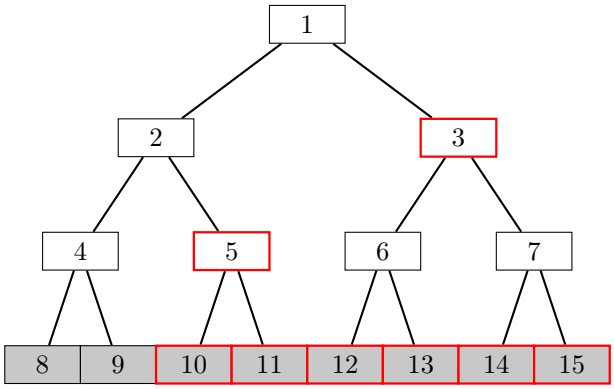


Figure 4.3: Example range query for the interval [10, 15] in node indices, corresponding to the interval [3,8] in data array indices (see Figure 4.2). The gray data nodes with red outlines indicate the queried range, and the white internal nodes with red outlines indicate the included values (e.g., multiplied) to calculate the result.

We also implement a query that can apply the operation to all data entries except a single one, which is useful to compute a division in the factor graph by using $O(log(n))$ multiplications. This is implemented by using one range query if the queried index i is the start or end and otherwise using two range queries $[1, i-1]$ and $[i+1, n]$.

4.2.2 Prefix and Postfix Products

Another possible workaround for computing the product of N distributions except for one distribution without explicit division is to keep track of a prefix and postfix product. These can be calculated in a single left-to-right and right-to-left pass, respectively. The prefix and postfix products are defined such that the prefix array at index i is equal to the product of all distributions up to i, and the postfix array stores the product of all distributions after i:

$$\begin{aligned} \text{prefix[i]} &= \prod_{j=1}^{i-1} p_j(x) \text{ for } i = 1, \ldots, n+1 \\ \text{postfix[i]} &= \prod_{j=i+1}^{n} p_j(x) \text{ for } i = 0, \ldots, n \end{aligned} \tag{4.12}$$

with the base cases prefix[1] = 1 and postfix[n] = 1, which is the neutral element for multiplication. Note that these formulas assume the arrays to be indexed starting from 1, following Julia convention. Then, to get the product of all distributions except the i-th distribution, we can compute:

$$p_{\setminus i}(x) = \text{prefix[i]} \times \text{postfix[i]} \tag{4.13}$$

For n distributions, building the two arrays (prefix and postfix products) takes $2n \in O(n)$ time, and evaluating any partial product takes $O(1)$ time, which is even faster than segment trees. The space complexity for both arrays is $2n$, the same as our segment tree implementation. Updating a distribution at index i in the array is inefficient. It takes $O(n)$ time because all values in the prefix sum starting from index i and in the postfix sum up to index i must be recomputed. An example is outlined in Figure 4.4.

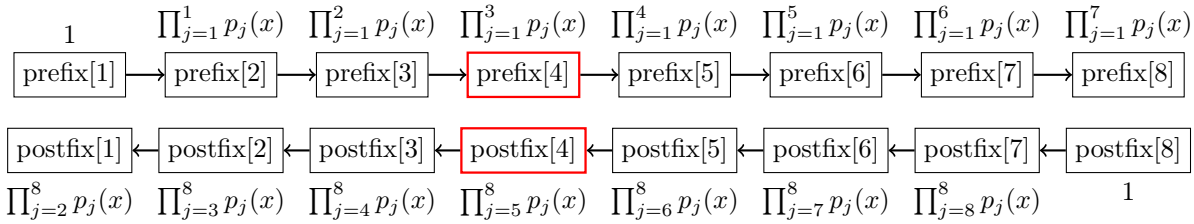


Figure 4.4: Example lookup at index $i = 4$ in the prefix and postfix product arrays. The queried result $p_{\setminus 4}(x) = \frac{\prod_{j=1}^{8} p_j(x)}{p_4(x)}$ is retrieved in $O(1)$ time by simply multiplying the prefix and postfix product distributions at the queried index. Our implementation thus computes $p_{\setminus 4}(x) = \text{prefix}[4] \times \text{postfix}[4] = \left(\prod_{j=1}^{3} p_j(x)\right) \times \left(\prod_{j=5}^{8} p_j(x)\right)$.

Chapter 5

Bayesian Neural Networks as Factor Graphs

TrueSkill is just one example of leveraging the capabilities of factor graphs. To expand possible application areas, we research how factor graphs, in combination with mixtures of Gaussians, can be used to implement Bayesian neural nets. This work is based on [Som24] but adapts it to the use of Gaussian mixtures. In the following subsections, we first lay out the motivation. Next, we introduce the factors that are essential to simple neural nets but have not been implemented within the scope of TrueSkill so far. Finally, we show how to construct a simple multilayer perceptron.

5.1 Motivation

Point Weight Estimates The weights of deep neural networks (DNNs), such as multi-layer perceptrons (MLPs) [Ros62; RHW86], cnvolutional neural networks (CNNs) [Lec+98; KSH17] or transformers [Vas+23], are typically represented as a matrix or tensor of fixed floating point numbers (or integers/ bits for quantized networks). This point estimate of the optimal weights w^* can be seen as the optimization result of a maximum-likelihood objective:

$$\begin{aligned} w^*_{\text{MLE}} &= \arg\max_w \mathbb{P}(X \mid w) \\ &= \arg\max_w \prod_{i=1}^{N} \mathbb{P}(x_i \mid w) \\ &= \arg\max_w \sum_{i=1}^{N} \log \mathbb{P}(x_i \mid w) \end{aligned} \tag{5.1}$$

Another typical formulation of the problem is the maximum a posteriori (MAP) estimate, where we include a prior over the weights for regularization. Using Bayes rule, the posterior over weights is:

$$\begin{aligned} \mathbb{P}(w \mid X) &= \frac{\mathbb{P}(X \mid w)\,\mathbb{P}(w)}{\mathbb{P}(X)} \\ &\propto \mathbb{P}(X \mid w)\,\mathbb{P}(w) \end{aligned} \tag{5.2}$$

Thus, the optimal MAP point estimate of model weights w^* is:

$$\begin{aligned} w^*_{\text{MAP}} &= \arg\max_w \mathbb{P}(w \mid X) \\ &= \arg\max_w \mathbb{P}(X \mid w)\, \mathbb{P}(w) \\ &= \arg\max_w \left(\prod_{i=1}^{N} \mathbb{P}(x_i \mid w) \right) + \mathbb{P}(w) \\ &= \arg\max_w \left(\sum_{i=1}^{N} \log \mathbb{P}(x_i \mid w) \right) + \log \mathbb{P}(w) \end{aligned} \tag{5.3}$$

In practice, the MLE and MAP parameter point estimates are usually computed via (stochastic) gradient-based methods [Rud17]. To predict a value for a new sample x, we would then use the point estimate w^* to calculate:

$$\hat{y} = f(x; w^*) \tag{5.4}$$

where f is the DNN parameterized by w^*.

Point Predictions and Miscalibration A pitfall of this method is that we only get a point prediction and no uncertainty estimate. One might argue that the model's confidence could be used as a surrogate uncertainty estimate. For example, for a classification network, this would mean using the score from the predicted argmax class as our prediction uncertainty. However, neural networks are usually overconfident in their prediction, meaning they are miscalibrated [Guo+17]. In other words, their predicted probability score of a class does not correspond to the probability of that class occurring given a data point x. There are ways to calibrate neural networks. However, there is a trade-off between prediction accuracy and calibration [Min+21], which means that practitioners worried about calibration must select a model on the Pareto front.

Underspecification A last motivation that we introduce is that neural networks are underspecified [Mur23], in the sense that the training data does not uniquely determine a single solution of model parameters. In modern overparameterized DNNs, many distinct sets of weights often yield similar good training losses. Yet, these models can find different generalizations (functions) and thus behave differently on unseen data. There has also been work to use this fact to extract more robust models. Some interesting findings include that the effective dimensionality of neural networks is correlated to the generalization ability, i.e., test loss. The effective dimensionality counts the number of well-determined parameters (if the loss surface depending on the model weights is flat, the parameters are not well determined as they can move in different directions without changing the loss) [Mur23]. Garipov et al. [Gar+18] also find that two sets of trained weights can be connected via simple curves that achieve near-constant training and test errors across the mode-connecting curves. Then, they use models on the curve to form well-performing ensembles. Another work exploiting this insight is the sharpness-aware minimization [For+21] framework by Foret et al. that finds flat regions within the loss landscape. This results in better generalization ability of the trained models.

Bayesian Neural Networks (BNNs) BNNs address the point prediction, miscalibration, and underspecification limitations of neural networks by marginalizing over the posterior:

$$\mathbb{P}(y \mid x, X) = \int_{-\infty}^{\infty} \mathbb{P}(y \mid x, w)\, \mathbb{P}(w \mid X)\, \mathrm{d}w. \tag{5.5}$$

which means that they average over all possible model predictions multiplied by the probability of that model under the dataset X. The prediction is a distribution, contrasting the point estimates that a model with a fixed parameter set w^* would yield. This predictive distribution can also quantify epistemic (model) uncertainty, making it attractive for topics like active learning [Erl20] or deployment in safety-critical applications. Furthermore, the overconfidence in one prediction

(e.g., a MAP estimate) is reduced by averaging over all possible models, which leads to better calibration [SHH25] compared to state-of-the-art optimizers like AdamW [LH19] or IVON [She+24]. The expression for calculating the posterior in Equation 5.5 lends itself well to be expressed in a factorized form. We exploit this fact and model the weights and intermediate results as parametric distributions by applying the factor graph framework to BNNs.

5.2 Factor Graph Setup

Based on the TrueSkill Section 2.4, this section describes the additional building blocks necessary to implement a multi-layer perceptron using a factor graph framework with Gaussian mixtures. In the end, we put it together into a simple MLP.

5.2.1 Product Factor

The product factor encodes the hard constraint $z = a \cdot b$, a non-linear transformation of the random variables a and b. For Gaussian inputs $a \sim \mathcal{N}(\cdot; \mu_a, \sigma_a^2), b \sim \mathcal{N}(\cdot; \mu_b, \sigma_b^2)$, the product of the two random variables is not a Gaussian distribution (see Figure 5.1). Thus, it needs to be approximated by a parametric distribution to be able to continue the calculations for subsequent factors. We approximate the product factor using either a Gaussian density or a Gaussian mixture density. As an optimization criterion, we will disregard the forward KL and follow previous work by optimizing for a mode-seeking solution to break symmetry using the reverse KL, which is a special case of the α-divergences family [Ama07], which we discuss in more detail in Appendix A.1.

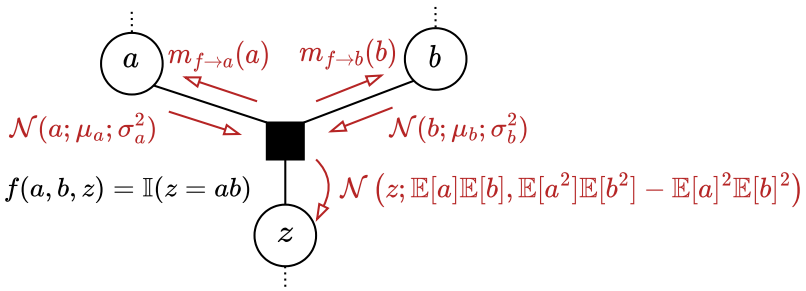


Figure 5.1: Product factor using a Dirac delta distribution to model the constraint $\mathbb{I}(z = ab) \equiv \delta(z - ab)$. See Appendix A.2 for more detail on the Dirac delta. The factor-to-variable messages $m_{f\to a}(a)$ and $m_{f\to b}(b)$ for the backward pass are presented in Equation 5.14. All factor-to-variable messages are approximated using the distribution family of choice, in our case the exponential family.

Forward Message The forward message is defined as:

$$\begin{aligned} m_{f\to z}(z) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \delta(z - ab)\, m_{a\to f}(a)\, m_{b\to f}(b)\, \mathrm{d}a\, \mathrm{d}b \\ &= \lim_{s^2\to 0} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathcal{N}(z - ab, 0, s^2)\, \mathcal{N}(a; \mu_a, \sigma_a^2)\, \mathcal{N}(b; \mu_b, \sigma_b^2)\, \mathrm{d}a\, \mathrm{d}b \end{aligned} \tag{5.6}$$

We approximate the forward message as a Gaussian to make its approximation tractable. A mathematically sound way to compute the optimal parameters of the Gaussian approximation is to perform moment matching on the marginal $\mathbb{P}(z)$, which minimizes the forward KL-divergence between our approximation and $\mathbb{P}(z)$. This guarantees an optimal marginal approximation (for the selected criterion, i.e., the forward KL), which is important for robust and interpretable results. The reason for this is that decisions are always performed on marginals. Performing moment matching

directly on the message can lead to a good approximation in practice, but, in theory, entails that we have no guarantees for the approximation quality of the marginal. Approximating the message directly via moment matching can lead to arbitrarily bad marginal approximations. This pitfall limits the applicability to domains where an interpretable result or certain approximation guarantees are required, for example, for safety in self-driving cars and medicine. Suppose we follow the proper marginal approximation approach. In that case, we can use the fact that the marginal of a variable can be computed as the product of its incoming and outgoing messages:

$$\mathbb{P}(z) = m_{z \to f}(z)\, m_{f \to z}(z) \tag{5.7}$$

This lets us use the Gaussian approximation $\hat{p}(z)$ to retrieve the forward message:

$$m_{f \to z}(z) = \frac{\hat{p}(z)}{m_{z \to f}(z)} \tag{5.8}$$

The denominator $m_{z \to f}(z) = \mathcal{N}(z; \mu_z, \sigma_z^2)$ is the updated distribution of z during the backward pass through the factor graph. First, we initialize it as a uniform or Gaussian distribution with infinite variance. If the subgraph connected to z contains observed variables, they will influence the marginal of z and messages from z to the product factor.

For computing the forward message $m_{f \to z}(z)$, the marginals $\mathcal{N}(a; \mu_a, \sigma_a^2)$ and $\mathcal{N}(b; \mu_b, \sigma_b^2)$ are given as two independent Gaussians. This only holds as long as we do not know the value of z, then they become dependent. This dependency can be easily verified by d-separation [GVP13]: For a converging connection (one "output" node depends on two "input" nodes), the two input nodes are only independent as long as the output is unobserved. This dependence is explicitly modeled in the factor as $\mathbb{I}(z = ab)$. Unfortunately, a message approximation via moment matching on the marginal leads to a term that includes a product of dependent Gaussians. Deriving a satisfactory approximation for this approach is still open for future work. In this report, we instead perform moment matching on the message directly, which has no guarantees but works well in practice. This approach yields the same approximation as in Stern et al. [SHG09]. Next, we present a derivation.

Forward Message Derivation For our derivation, we make the strong assumption of a fully factorized posterior, or in other words, that the two Gaussians a and b are independent. This assumption is usually called the mean-field assumption or mean-field variational inference [BKM17; Mur23; Bis06]. The idea is to average over the free statistics (parameters) to approximate the complex distribution. This yields a more straightforward and now tractable problem. With the independence assumption, we can perform moment matching on the message from the product factor $f(a, b, z) = \delta(z - ab)$ to z directly as discussed:

$$\mathbb{E}[z] = \mathbb{E}[a\, b] = \mathbb{E}[a]\, \mathbb{E}[b] \tag{5.9}$$

Now we can decompose the variance into its first and second moment $\mathbb{V}[a] = \mathbb{E}[a^2] - \mathbb{E}[a]^2$. We can rearrange it, so that $\mathbb{E}[a^2] = \mathbb{V}[a] + \mathbb{E}[a]^2$. Using the variance decomposition and the independence assumption between a and b, we can derive the variance of the forward message:

$$\begin{aligned} \mathbb{V}[z] &= \mathbb{E}[z^2] - \mathbb{E}[z]^2 \\ &= \mathbb{E}[a^2]\, \mathbb{E}[b^2] - (\mathbb{E}[a]\, \mathbb{E}[b])^2 \\ &= \mathbb{E}[a^2]\, \mathbb{E}[b^2] - \mathbb{E}[a]^2\, \mathbb{E}[b]^2 \end{aligned} \tag{5.10}$$

Plugging the matched moments into our Gaussian density approximation, we retrieve the forward message $m_{f \to z}(z)$ that is presented in Ster et al. [SHG09] and visualized exemplarily in Figure 5.2:

Product Factor - Forward Message (Gaussian Approximation)

$$m_{f \to z}(z) = \mathcal{N}\left(z; \mathbb{E}[a]\, \mathbb{E}[b], \mathbb{E}[a^2]\, \mathbb{E}[b^2] - \mathbb{E}[a]^2\, \mathbb{E}[b]^2\right) \tag{5.11}$$

Product Factor - Forward Message (MoG component-wise approximation)

$$m_{f\to z}(z) = \sum_{i,j} w_i w'_j \, \mathcal{N}(z_{ij}; \mathbb{E}[a_i]\,\mathbb{E}[b_j], \mathbb{E}[a_i^2]\,\mathbb{E}[b_j^2] - \mathbb{E}[a_i]^2\,\mathbb{E}[b_j]^2) \tag{5.12}$$

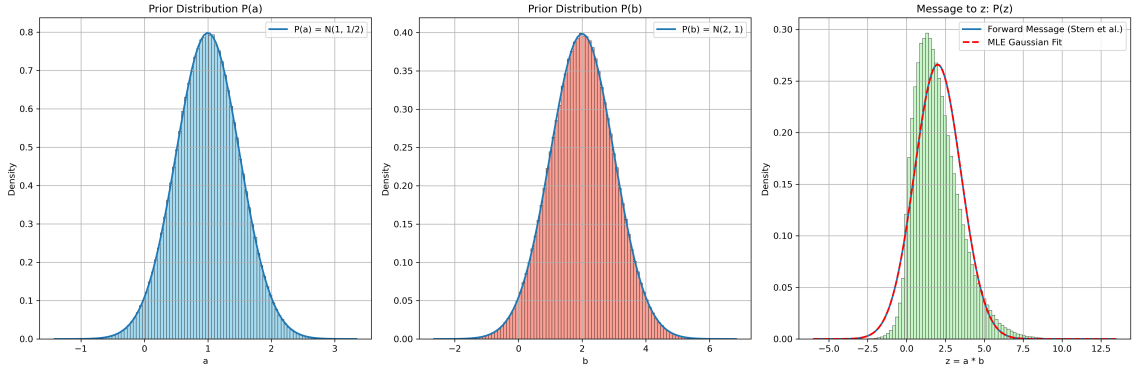


Figure 5.2: Product factor example for Gaussian priors with $\mu_a = 1, \sigma_a^2 = \frac{1}{2}$ and $\mu_b = 2, \sigma_b^2 = 1$. Each subplot shows a histogram of 1M samples from $\mathbb{P}(a), \mathbb{P}(b)$ and $m_{f\to z}(z)$ respectively. For the forward message $m_{f\to z}(z)$ shown in the third subplot, we added a maximum likelihood estimate as the dashed red line and a Gaussian approximation derived by moment matching the message (5.11), which was originally presented by Stern et al. [SHG09] and also used by Adamcic et al. in the previous year's master project [Ada+24].

Backward Message For the backward message $m_{f\to a}(a)$, we marginalize out b and z:

$$m_{f\to a}(a) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \delta(z - ab)\, m_{b\to f}(b)\, m_{z\to f}(z)\, \mathrm{d}b\, \mathrm{d}z \tag{5.13}$$

where $\delta(\cdot)$ is the delta function (see Appendix A.2). The product factor $\delta(z - ab)$ enforces the constraint $a = z/b$. The resulting message $m_{f\to a}(a)$ is the distribution of a conditioned on b and z. Again, this is not Gaussian due to the non-linear factor, and we need to come up with an approximation. We can use expectation propagation to iteratively refine the posterior approximation or variational inference to optimize a lower bound of the marginal likelihood iteratively. Of course, sampling is also possible but very expensive. Therefore, we are more interested in finding a closed-form solution that yields a good approximation, for example, via moment matching.

A good first approach is to perform moment matching on the marginal of a, which we could then use to calculate an approximate message $m_{f\to a}(a)$. However, Adamcic et al. show that this message can not be integrable in general [Ada+24]. They explored different ways of numerically approximating the marginal but did not arrive at a satisfactory approximation that avoids an iterative approach like expectation maximization.

On the other hand, Stern et al. [SHG09] propose an approximation for the reverse message $m_{f\to a}(a)$ that achieves satisfactory results in practice. They state that it was derived by minimizing the reverse KL divergence. Because the paper does not include a derivation, we provide an attempt at a derivation (see Subsection A.6) that arrives at the same result shown in Equation 5.14 and is visualized exemplarily in Figure 5.3:

Product Factor - Backward Message (Gaussian Approximation)

$$m_{f\to a}(a) = \mathcal{N}\left(a; \frac{\mathbb{E}[m_{z\to f}]\,\mathbb{E}[b]}{\mathbb{E}[b^2]}, \frac{\mathbb{V}[m_{z\to f}]}{\mathbb{E}[b^2]}\right). \tag{5.14}$$

Product Factor - Backward Message (MoG component-wise approximation)

$$m_{f\to a}(a) = \sum_{i,j} w_i w'_j \mathcal{N}(a_{ij}; \frac{\mathbb{E}[z_i]\,\mathbb{E}[b_j]}{\mathbb{E}[b_j^2]}, \frac{\mathbb{V}[z_i]}{\mathbb{E}[b_j^2]}) \tag{5.15}$$

Because products follow the commutative law, the message to the other input variable b is analogous to the message to a. As discussed, this is not a robust approximation because it minimizes the reverse KL on the message, not the marginal. A correct approximation is still open for future work. We include our partial progress in Appendix A.6 as a starting point.

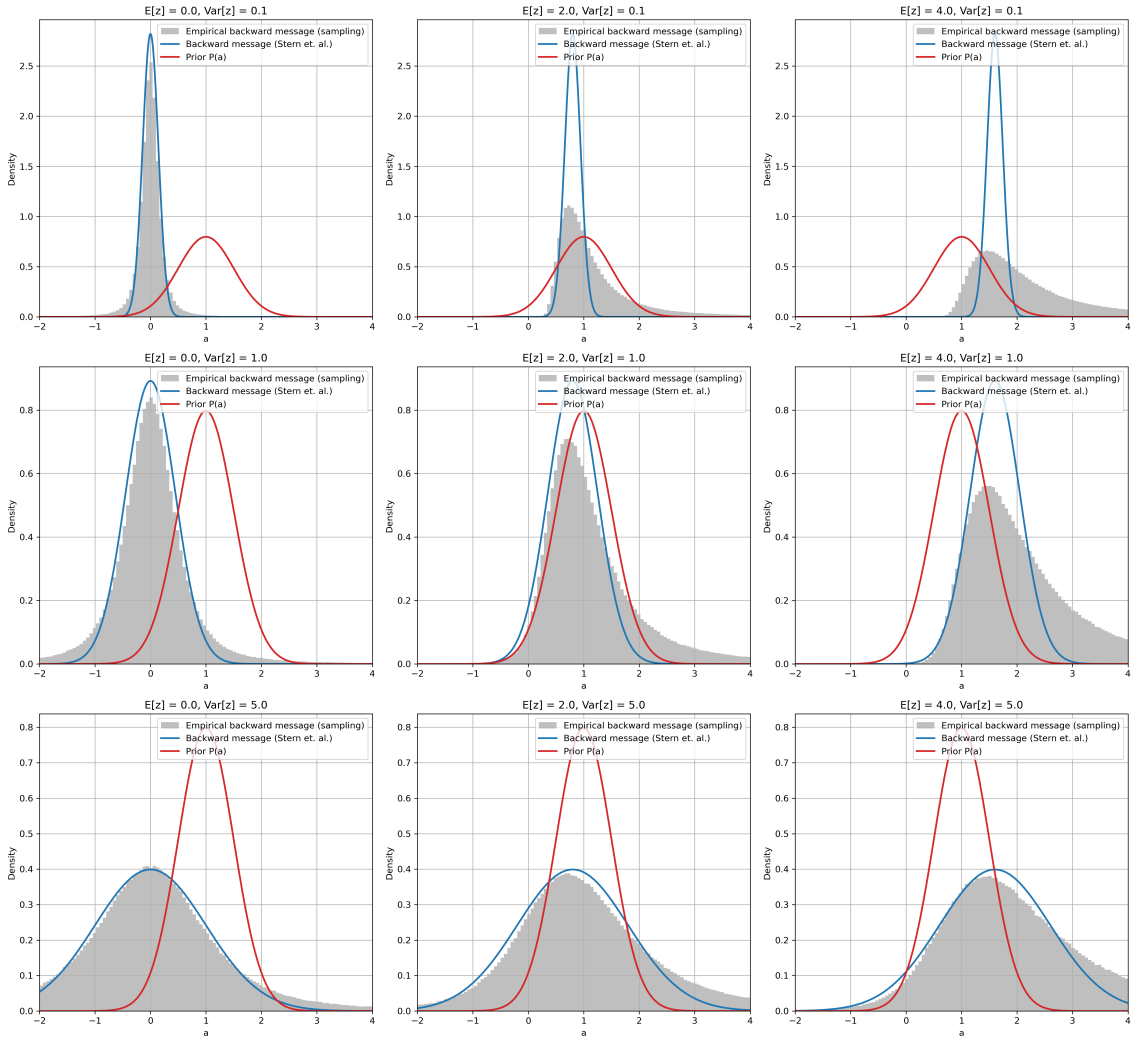


Figure 5.3: Approximations of the product factor backward message $m_{f\to a}(a)$ using the formula derived by Stern et al. [SHG09] for different means and variances of z. The prior over a in the example is fixed to the parameters $\mu_a = 1$ and $\sigma_a^2 = \frac{1}{2}$.

5.2.2 ReLu and Leaky ReLU Factor

In the context of factor graphs, the Leaky ReLU (Rectified Linear Unit) serves as a valuable non-linear activation function that introduces a small, non-zero gradient for negative inputs, mitigating the potential issue of dead neurons typical for standard ReLU activations. Unlike the standard ReLU, which outputs zero for all negative inputs, the Leaky ReLU allows for a small, controlled

flow of information even when $z < 0$. Leaky ReLU is defined as:

$$\text{Leaky ReLU}_\alpha : \mathbb{R} \to \mathbb{R}, z \mapsto \begin{cases} z & \text{for } z \geq 0 \\ \alpha z & \text{for } z < 0 \end{cases}$$

where α is the leak. We model the application of nonlinearity as the factor $f = \delta(a - \text{LeakyReLU}_\alpha(z))$ where z is the input into the factor and a is the output (see Figure 5.4.

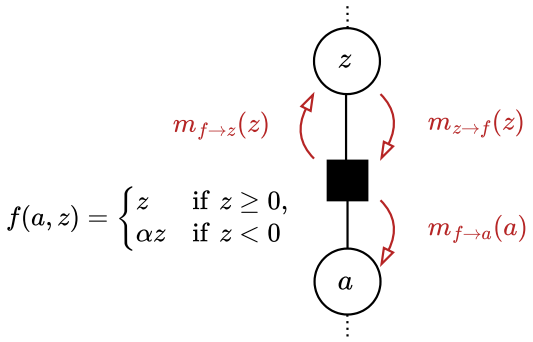


Figure 5.4: Factor for the Leaky ReLU activation function $a = \text{Leaky ReLU}_\alpha(z)$

Forward Message The forward message is defined as follows:

$$m_{f\to a}(a) = \int_{z\in\mathbb{R}} \delta(a - \text{Leaky ReLU}_\alpha(z)) m_{z\to f}(z)\, dz = \begin{cases} m_{z\to f}(a) & \text{if } a \geq 0, \\ m_{z\to f}(a/\alpha) & \text{if } a < 0 \end{cases}$$

To approximate this message, we split each component $w_i \mathcal{N}(y \mid \mu_i, \sigma_i^2)$ of the message $m_{z\to f}$ at 0 into two Gaussians: $\mathcal{N}_i^-$ and $\mathcal{N}_i^+$, approximating the truncated parts of $\mathcal{N}_i$ below and above 0 using moment matching. For $\mathcal{N}_i^-$ and $\mathcal{N}_i^+$, we compute the probability masses P_i^- and P_i^+ as follows:

$$P_i^- = \Phi_{N_i}(0) = \Phi\left(\frac{-\mu_i}{\sigma_i}\right), \quad P_i^+ = 1 - P_i^- = \Phi\left(\frac{\mu_i}{\sigma_i}\right),$$

where $\Phi_{N_i}(\cdot)$ is the CDF of the normal distribution with mean μ_i and standard deviation σ_i, and $\Phi(\cdot)$ is the CDF of the standard normal distribution.

The truncated Gaussians $\mathcal{N}_i^+$ and $\mathcal{N}_i^-$ are defined as:

$$\mathcal{N}_i^+(y) = \mathcal{N}(y \mid \mu_i^+, \sigma_i^{+2}), \quad \mathcal{N}_i^-(y) = \mathcal{N}(y \mid \mu_i^-, \sigma_i^{-2}),$$

where the means and variances are given by:

$$\mu_i^+ = \mu_i + \sigma_i v\left(\frac{\mu_i}{\sigma_i}\right), \quad (\sigma_i^+)^2 = \sigma_i^2\left(1 - w\left(\frac{\mu_i}{\sigma_i}\right)\right),$$

$$\mu_i^- = \mu_i - \sigma_i v\left(\frac{\mu_i}{\sigma_i}\right), \quad (\sigma_i^-)^2 = \sigma_i^2\left(1 - w\left(-\frac{\mu_i}{\sigma_i}\right)\right).$$

Here, $w(t)$ and $v(t)$ are defined as:

$$v(t) = \frac{\phi(t)}{\Phi(t)}, \quad w(t) = v(t) \cdot (v(t) + t),$$

where $\phi(t)$ is the PDF of the standard normal distribution, and $\Phi(t)$ is its CDF. In the resulting message, the positive components remain unchanged due to the definition of Leaky ReLU. The components smaller than 0 get multiplied by α to achieve $m_{f\to a}(a) = m_{z\to f}(a/\alpha)$. The weights ω are scaled by the proportion of the Gaussian lying below or above 0 before the transformation:

Leaky ReLU - Forward Message

$$m_{f\to a}(a) = \sum_{k=1}^{K} \left(\omega_k P_k^+ \mathcal{N}_k^+(y \mid \mu_k^+, (\sigma_k^+)^2) + \omega_k P_k^- \mathcal{N}_k^-(y \mid \alpha\mu_k^-, (\alpha\sigma_k^-)^2)\right) \tag{5.16}$$

The output message $m_{f\to a}(a)$ is a new Gaussian mixture model with twice as many components as the incoming message.

Backward Message By the sifting property of the Dirac delta, the backward message is equal to:

$$m_{f\to z}(z) = \int_{a\in\mathbb{R}} \delta(a - \text{LeakyReLU}_\alpha(z)) m_{a\to f}(a)\, da = m_{a\to f}(\text{LeakyReLU}_\alpha(z)) = \begin{cases} m_{a\to f}(z) & \text{for } z \geq 0 \\ m_{a\to f}(\alpha z) & \text{for } z < 0 \end{cases}$$

The backward message can be calculated analogously to the forward message. The difference is that instead of multiplying by alpha, we divide by it:

Leaky ReLU - Backward Message

$$m_{f\to z}(z) = \sum_{k=1}^{K} \left(\omega_k P_k^+ \mathcal{N}_k^+(z \mid \mu_k^+, (\sigma_k^+)^2) + \omega_k P_k^- \mathcal{N}_k^-(z \mid \mu_k^-/\alpha, (\sigma_k^-/\alpha)^2)\right) \tag{5.17}$$

After splitting the components into positive and negative ones, the μ and σ remain the same for the components with $z >= 0$. For the components $z < 0$, we multiply μ and σ by $1/\alpha$ to invert the operations of the forward pass.

Figure 5.5 and Figure 5.6 show an example of how an incoming message is transformed after applying a Leaky ReLU activation.

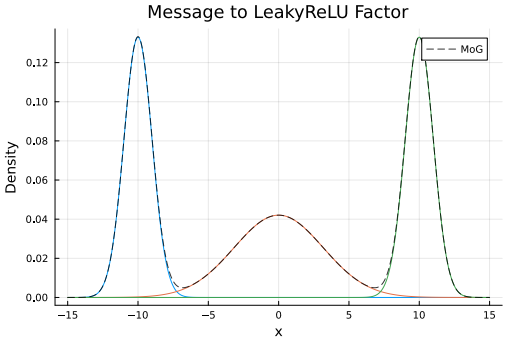


Figure 5.5: Message to Leaky ReLU factor with 0.5 Leak showing the individual components

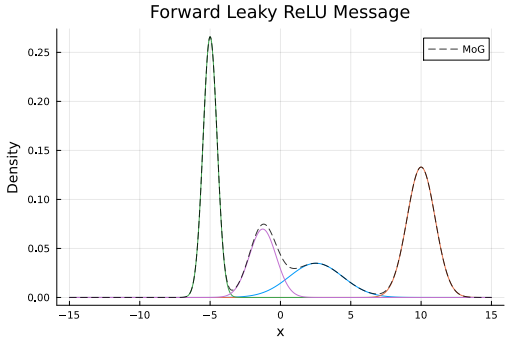


Figure 5.6: Forward Leaky ReLU message with 0.5 Leak showing the message and the components

5.2.3 Regression Factor

The regression factor is comparatively simple. It is connected to three variables in the context of Bayesian neural networks: an output neuron $\hat{y}$, β, and y (see Figure 5.7). y is the assumed ground truth, and β models our uncertainty regarding the ground truth. During training, y is known, and the message for the backward pass is the distribution of y with a noise constant β^2 as the added variance. For the forward pass during predictions, y is unknown. The prediction is the distribution of $\hat{y}$ with added uncertainty β^2 similar to the Gaussian mean factor in TrueSkill as in Subsection 2.4.2.

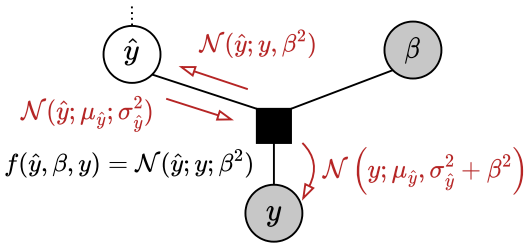


Figure 5.7: Regression factor with Gaussian messages

Regression - Forward Message for Predictions

$$m_{f \to y}(y) = \sum_k w_k \mathcal{N}(y_i; \mu_k, \sigma_k^2 + \beta^2) \tag{5.18}$$

Regression - Backward Message

$$m_{f \to \hat{y}}(\hat{y}) = \mathcal{N}(\hat{y}; y, \beta^2) \tag{5.19}$$

The backward message can also be a MoG.

5.2.4 Building a simple MLP

Now we have all the factors and can put them together. For a detailed explanation of how to build Bayesian neural networks with factor graphs, please refer to [Som24] and [SHH25]. As described in [SHH25] for regression networks, each training example gets modeled with its own factors and variables, for example, for inputs and intermediate results. Weights and biases are shared between training examples. The main difference when building an MLP with MoGs is the form of the passed messages and the calculation of outgoing messages from variables. The passed messages are all represented as MoGs. We do not divide the marginals by incoming messages to get outgoing messages, but instead multiply all incoming messages as explained in Section 4.2.

The following factor graph in Figure 5.8 displays the architecture for a simple neuron with two input values and a subsequently applied Leaky ReLU and regression factor.

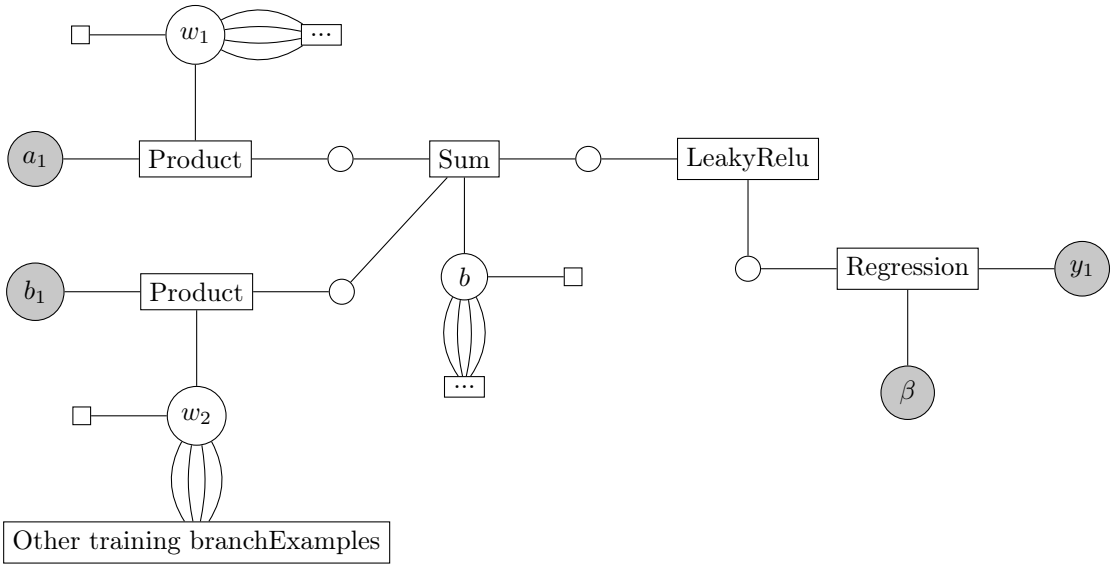


Figure 5.8: Illustration Bayesian neural network consisting of a single neuron with two inputs with subsequent Leaky ReLU and regression Factor. The weights and bias are connected to other training examples in the overall factor graph.

For future work, other factors for training, such as the argmax factor [SHH25], could be adapted for Bayesian neural networks with MoGs.

Chapter 6

Experiments, Results and Discussion

Based on the theoretical and conceptual ideas presented in the previous chapters, this section illustrates the application on different datasets. In Section 6.1, we describe the objectives, rationale, and methodology of our approach. In addition, we outline the different datasets and the reasons for including them in these experiments. In Section 6.2, we provide more detailed information about the different experimental scenarios and showcase the results. Taking all the experiments into account, we discuss the findings in Section 6.3 and interpret them in context.

6.1 Experiment Design

6.1.1 Approach and Objectives

Our project aims to evaluate the effect of different representation techniques in TrueSkill and their performance. Specifically, we explore and compare discrete distributions (see Subsection 6.2.1) and Gaussian mixtures (see Subsection 6.2.2) with the original Gaussian TrueSkill representation. The key goal is to determine how these different representations affect the accuracy of skill estimates based on match outcomes. For all comparison experiments, the TrueSkill implementation is based on the initial paper by Herbrich et al. [HMG07]. All distributions are initialized with a normal prior $\mathcal{N}(25, 25/3)$, and we use a beta value of $25/6$. This follows the original implementation. The code for all described experiments is written in Julia version 1.11.1 and publicly available in our GitHub.

TrueSkill with Discrete Distributions We first compare the different initialization techniques for the discrete distributions to set the bucket thresholds. Here, we observe and compare the KL divergence (KLD) of the three approaches as described in Section 3.1. The objective is to answer which of the quantile threshold initialization techniques (equal width, equal density, derivation-based) yields the highest approximation quality of a randomly chosen Gaussian.

Next, the goal is to identify if the discrete distribution representation leads to higher accuracy than the Gaussian variant on real-world data. Therefore, we use a single-player dataset of tennis matches. We run through 5,000 matches. The metric used for comparison is accuracy over time. In short, we refer to it as accuracy, computed with a sliding window of size 10,000. To inspect the metric, we create a plot and visually inspect how the different approaches behave over time. We always include the unimodal Gaussian performance as a baseline in the plots (see Section 1.2). Due to the unsatisfactory results of the discrete representations, we then focus on evaluating and comparing the performance of MoGs.

TrueSkill with Mixture of Gaussians For the MoG approach, we apply the same experiment multiple times on different datasets. On the one hand, we want to model how well single-player game outcomes can be predicted. On the other hand, we run the experiment on known team-based match outcomes. Lastly, we want to investigate the effect of a bimodal prior to player skills with a synthetically generated dataset. A different dataset with the required attributes is used for each tested scenario. These are outlined in Subsection 6.1.2. In order to estimate the performance of TrueSkill in another real-world setting, we run it on the LMArena dataset and compare the results against the commonly used Elo ranking algorithm (see Section 1.2). We research their ranking order, ranking curve, and accuracy over time. Additionally, we take a look at the win-rate matrices of each algorithm in order to gauge their confidence and calibration.

Addressing Model Accuracy with EM and Calibration Next, we conduct two experiments to improve and investigate the accuracy metric. First, we apply the expectation-maximization (EM) algorithm to refine the model's message updates by calculating the weighted sum and truncation messages. The goal is to investigate whether this technique improves model accuracy and ensures more reliable predictions, regardless of the number of input components.

After the EM experiment, we focus on calibration, reassessing the accuracy metric. By using calibration plots, we gain deeper insights into the reliability of the model's probabilistic predictions. We apply this method to evaluate the calibration of the Gaussian- and MoG-based TrueSkill models using a single-player tennis dataset. The objective is to compare how well each model's predicted probabilities align with observed outcomes and assess the impact of calibration on predictive reliability.

Neural Nets with Factor Graphs In addition to these experiments focused on the effect of distributional representation in TrueSkill, we want to explore the applicability of using factor graphs beyond TrueSkill based on mixtures of Gaussians. Therefore, we first perform a first experiment to check if the product factor, which is essential for neural net setups, behaves as expected. We test the product factor's ability to correctly backpropagate messages by providing two MoGs as input and visually comparing the results with the ground truth. The next experiment tests linear and cubic regressions to see if product and sum factors can backpropagate messages correctly in a factor graph. The objective is to verify message propagation using the mean of each distribution as the parameter value. Lastly, we build a factor-graph-based MLP and train it on two simple datasets: a step function and a parabola. We investigate the results visually and derive the first conclusions and next steps.

6.1.2 Datasets

The experiments are carried out both on synthetic and real-life datasets:

Tennis Dataset 1 vs. 1 The tennis dataset 1 vs. 1 is a database of 447,000 registered tennis matches scraped from the website of the Association of Tennis Professionals (ATP) [ATP25; Lan25]. ATP scraped datasets have been used before to explore the capabilities of TrueSkill (e.g., [LWH24]), which makes this dataset an ideal basis for comparison of the newly developed approaches. The dataset used is a collection of matches between August 16, 1915, and December 03, 2020. We only consider the winner (w1_id) and loser (l1_id) of the games to make the predictions.

Tennis Dataset 2 vs. 2 This dataset includes matches where two players on each team competed against each other. It was collected by filtering for doubles matches (doubles = t) from another ATP tennis dataset containing matches played between January 21, 1950, and August 22, 2018 [Hal20]. We then split the two players into the player_id and opponent_id columns by underscores, obtaining one entry per player. player_1 and player_2 are the winning players, while opponent_1 and opponent_2 represent the losing players. As a result, we get a subset of 800,000 matches

from the original dataset. We include this subset to have a simple, real-life, data-based example of team-based match outcomes.

League of Legends League of Legends (LoL) is a free multiplayer online battle arena (MOBA) game with millions of players. We use a dataset of over 100,000 ranked games with match outcomes [Shi20]. Here, ranked signifies that five players per team were matched against each other using LoL's internal matchmaking system, which implies that they likely have similar skill levels. We did not distinguish between different player roles. The LoL dataset is included as an example of an online multiplayer game.

LMArena LMArena is a 1 vs. 1 dataset in which the players are large language models (LLM) competing against each other. Specifically, the data is derived from the online platform Chatbot Arena[Chi+], where users can chat with two models simultaneously and rank their answers according to perceived quality [Chi+; Chi+24]. Importantly, the identity of the models is not known to the users, ensuring a fair grading. Our data sample contains all rankings until August 14, 2024, including recent models such as ChatGPT-4-o and Gemini-1.5-Pro. It includes a solid 1,6 million entries. There are many matches labeled either "tie" or "tie (bothbad)" indicating that both answers were bad. Even though TrueSkill can handle tie situations [Mos10], we drop these rows for simplicity reasons. This leaves us with 1 million 1 vs. 1 ranking entries. This application of ranking systems is one of the most recent emerging trends in computer science, making our investigation of TrueSkill's performance on this dataset a compelling research direction.

Synthetic Dataset To benchmark our MoG approach in a scenario where multimodality is the underlying assumption, we create a synthetic dataset of 1 vs. 1 match outcomes. Specifically, we generate random bimodal MoG skill distributions for 100 independent and hypothetical players. In addition, we simulate $15,000$ different 1 vs. 1 game outcomes by sampling a random performance for each player based on their underlying skill distribution, such that the player with the higher skill wins the match. In the following, we describe in detail how the dataset is generated and what properties it possesses.

In the first step, the skill is modeled. Therefore, a MoG with $M = 2$ components is constructed by assigning random weights to each component. These weights, w_i, are sampled from a uniform distribution and normalized to sum to one:

$$w_i = \frac{w_i}{\sum_{j=1}^{M} w_j}, \quad i = 1, ..., M. \tag{6.1}$$

Each component is parameterized by a mean μ_i and a standard deviation σ_i. The mean values are drawn from a normal distribution with zero mean and standard deviation σ_μ, which we set to $\sigma_\mu = 10$:

$$\mu_i \sim \mathcal{N}(0, \sigma_\mu) \tag{6.2}$$

The standard deviation σ_i is obtained by sampling from a normal distribution scaled by σ_v, taking the absolute value to ensure non-negativity, and adding a minimum variance term $\sigma_{\min}$, where we use $\sigma_v = 5$ and $\sigma_{\min} = 0.1$:

$$\sigma_i = \max(\sigma_{\min}, |\sigma_v \cdot \mathcal{N}(0, 1)|) \tag{6.3}$$

The final Gaussian mixture model is given by:

$$p(x) = \sum_{i=1}^{M} w_i \mathcal{N}(x \mid \mu_i, \sigma_i) \tag{6.4}$$

where $\mathcal{N}(x \mid \mu_i, \sigma_i)$ represents a Gaussian distribution with mean μ_i and standard deviation σ_i.

As seen in Figure 6.1, 99.7% of the means of the components lie within the range $[-30, 30]$ because they are drawn from a normal distribution with a standard deviation of 10. This aligns with the expected behavior due to the three-sigma rule. By this rule, most values fall within $3 \times 10 = 30$. Hence, the means are expected to lie within the interval $[-30, 30]$. The variances generated from a half-normal distribution mostly fall within the range $[0.1, 15.0]$. The lower bound of 0.1 ensures non-negative variances and the three-sigma rule determines the 99.7%-upper bound of 15.0.

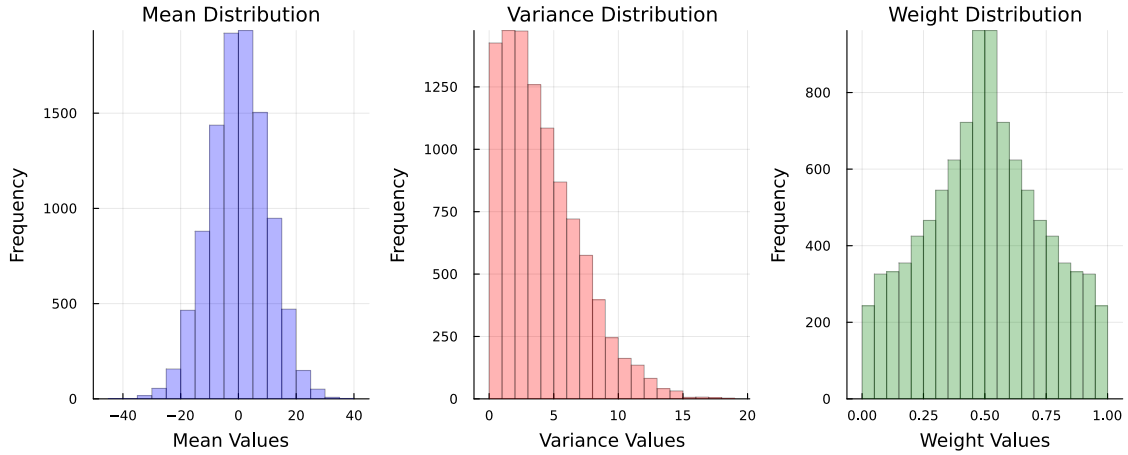


Figure 6.1: Component-wise statistics for mean, variance, and weight distribution using our hyperparameters

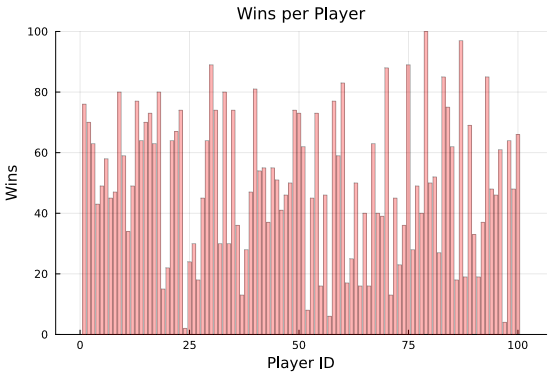
Given a MoG with M components, the process of drawing a random sample, for example, a performance from the skill distribution, follows a two-step procedure. First, a component index k is selected, according to the categorical distribution defined by the component weights w_i, where $i = 1, \ldots, M$. This selection process follows:

$$P(k = i) = w_i, \quad \text{for } i = 1, \ldots, M. \tag{6.5}$$

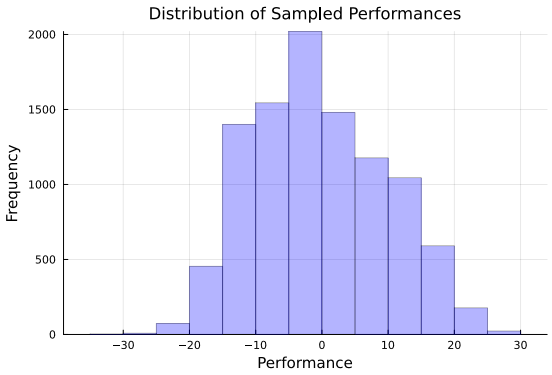
Once a component k is chosen, a sample x is drawn from the corresponding Gaussian distribution $\mathcal{N}(\mu_k, \sigma_k)$, where μ_k and σ_k are the mean and standard deviation of the selected component:

$$x \sim \mathcal{N}(\mu_k, \sigma_k) \tag{6.6}$$

Once the performance of two players has been sampled, the player with the higher performance wins the game. This results in an illustrative win distribution, as seen in Figure 6.2a, which is reasonably random. Additionally, it can be observed in Figure 6.2b that the performance samples of all players over many matches are roughly normally distributed.



(a) Example win distribution of 100 players after 5,000 random match-ups



(b) Example performance samples from all players over 5,000 match-ups

Figure 6.2: MoG match sampling

Regression and MLP Dataset For the linear regression, we sample x values from a standard normal distribution, multiply them with a slope, and add an intercept to get the y values. We do the same for cubic regression but with a cubic function, which adds additional uncertainty to the y values. The MLP experiments use a parabola dataset with 20 points sampled in the same way. Based on the theoretical and conceptual ideas presented in the previous chapters, this section illustrates their application across different datasets. In our experiments, the primary objective of our experiments is to systematically evaluate how different skill representation techniques in TrueSkill affect accuracy. To achieve this, we conduct multiple tests that compare discrete distributions, mixtures of Gaussians (MoGs), and the original Gaussian TrueSkill model.

6.2 Implementation and Results

In this chapter, we compare TrueSkill implemented with Gaussians, bucket and quantile distributions, and mixtures of Gaussians on different datasets. We further dive deeper into a comparison of mixtures of Gaussians with Elo and TrueSkill. Subsequently, we show the results we were able to achieve using regression and MLPs with MoGs.

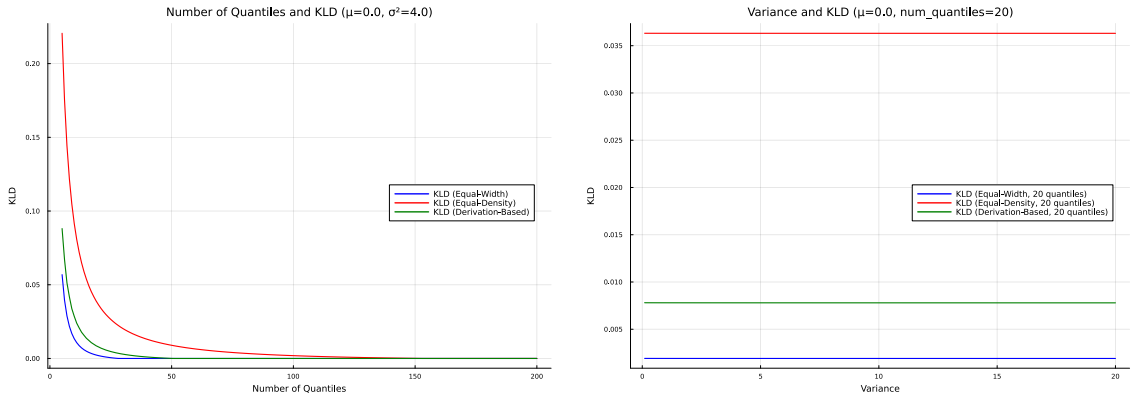
6.2.1 TrueSkill with Discrete Distributions

Initialization Our three presented initialization approaches have different characteristics in terms of how well they approximate a randomly chosen true Gaussian distribution. We refer to this aspect as the approximation quality of the approach. As introduced in Section 3.1, we differentiate between three approaches. The equal-width approach defines the bucket thresholds based on an equal distance on the x-axis. The equal-density approach shows an equal distance on the y-axis. The last approach derives the thresholds based on the derivation, leading to a higher resolution in areas with a high slope and a lower resolution for low-slope areas. To compare them, we analyze the effect of the number of quantiles, i.e., the number of buckets, and the variance on the KL divergence (KLD). A lower KLD indicates that the approximation is closer to the true distribution, signifying better approximation quality.

Using a Gaussian with $\mu = 0$ and $\sigma = 2$ as a comparison ground, we first use varying numbers of quantiles while keeping the mean and variance fixed (see Figure 6.3a). It can be observed that the equal-density approach performs worst (see red line), especially when the number of quantiles is low. This is likely due to the fact that most of its resolution is allocated to the center of the mass, while the tails of the normal distribution are neglected. The equal-width and derivation-based approaches show similar KLD values, even though the equal-width technique is slightly better. While the reasons for this require further research, it can be said that the equal-width approach

allocates uniform resolution to the defined range of the x-axis, reducing the risk of large deviations from the ground truth. As the KLD penalizes higher deviations more than lower ones, we observe higher KLD values for the equal-density and derivation-based approaches.

Next, we chose a number of quantiles $n = 20$ where the three approaches still show differences in approximation quality to explore the effect of changes in variance. With a fixed number of quantiles and $\mu = 0$, Figure 6.3b shows that all three approaches seem robust against changes in distribution variance.



(a) KLD with varying number of quantiles

(b) KLD with varying variance

Figure 6.3: Quality comparison of initialization approaches based on KL divergence

Tennis 1 vs. 1 For the MoU models, we treat the overlap handling, slope, and shifted mean calculations as different multiplication approaches (see Subsection 2.5.2). Considering possible combinations of the three multiplication variants with the three initialization strategies, we reach a total of nine different approaches (see). We execute each MoU approach ten times to mitigate randomness in the sampling process and averaged the results. Due to runtime constraints, we restrict TrueSkill to the first 5,000 matches when using MoU approaches.

As shown in Figure 6.4, the Gaussian baseline outperforms all MoU variants over the first 5,000 matches. By the end of the experiment, the Gaussian achieves nearly 70 % accuracy, whereas the MoUs stabilize around 65 %.

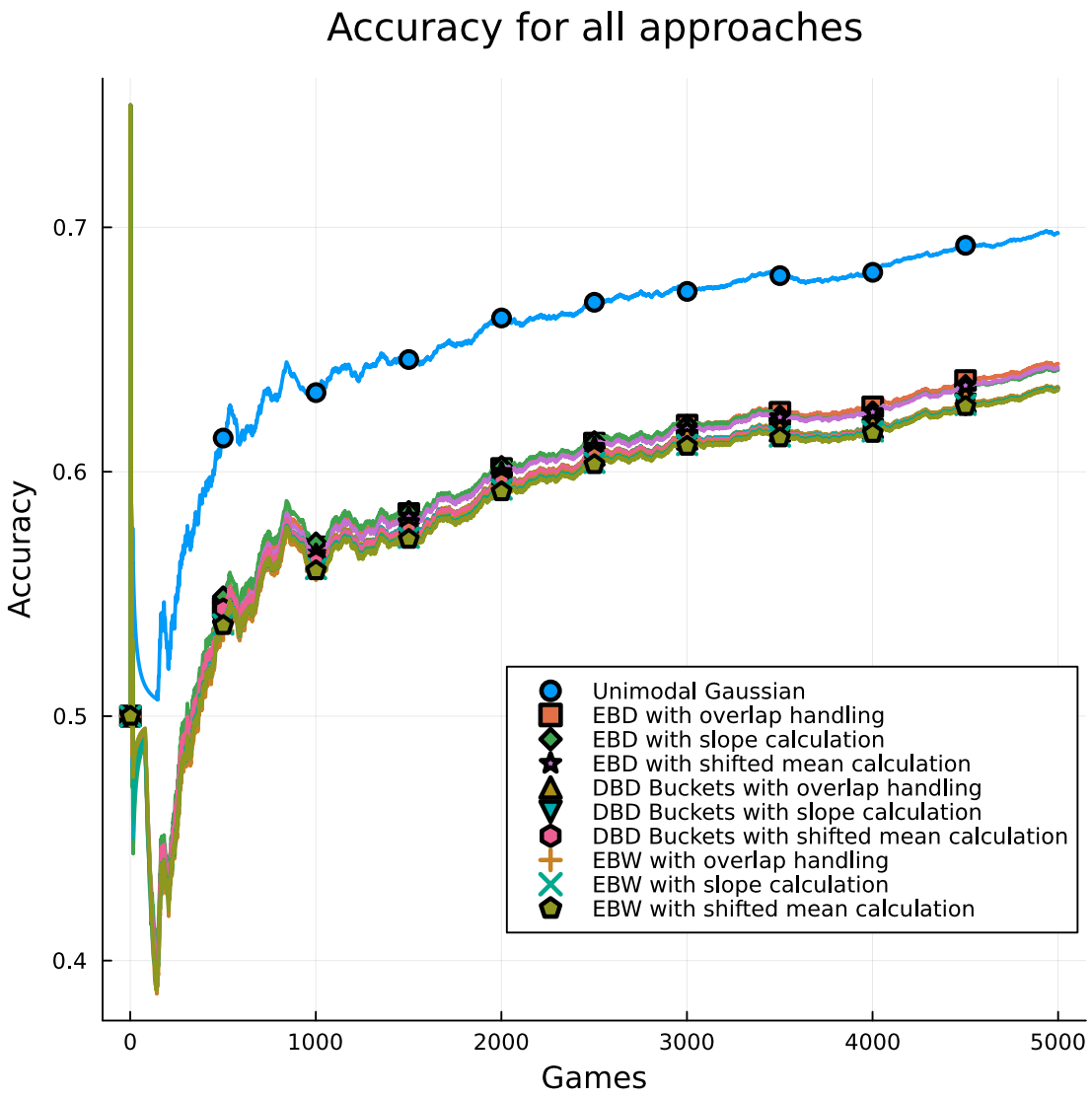
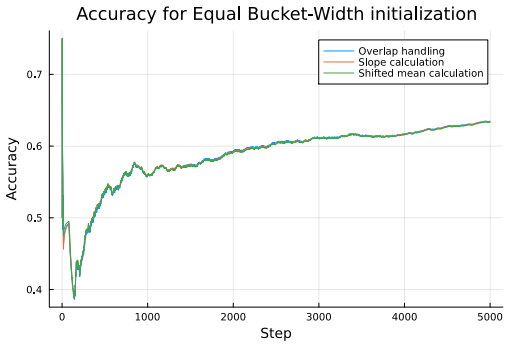
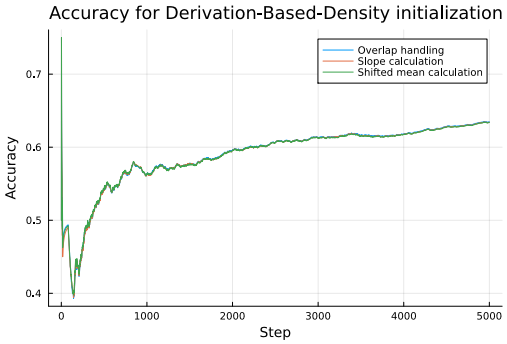


Figure 6.4: Performance comparison of TrueSkill on the 1 vs. 1 tennis dataset. The plot compares different density estimation approaches: Equal-bucket density (EBD), derivation-based density (DBD), and equal-bucket width (EBW) (see Section 3.1). Each method is further evaluated with variations, including overlap handling, slope calculation, and shifted mean calculation (see Section 3.2).

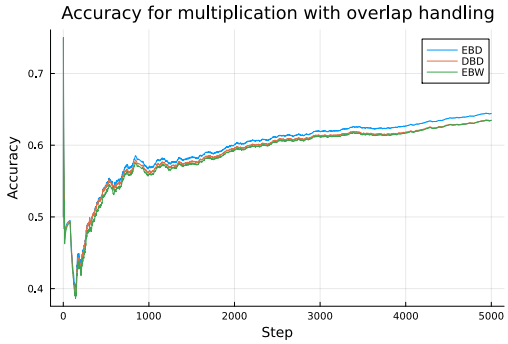
Figure 6.5 further compares the different initialization strategies and multiplication methods. While the choice of multiplication technique had no significant impact on accuracy, the equal-bucket width initialization consistently performed slightly better throughout the experiment.



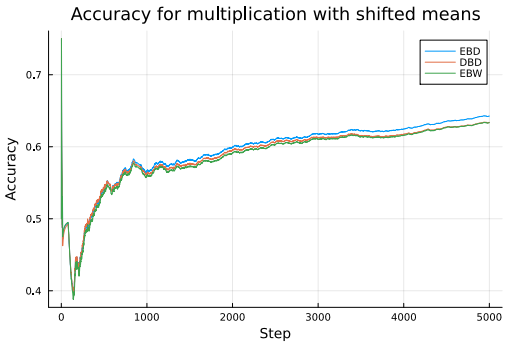
(a) Equal Bucket-Width



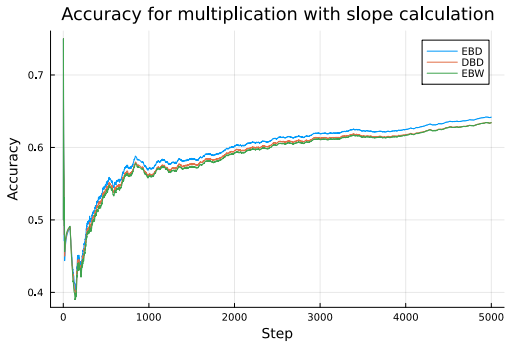
(b) Derivation-Based Density



(c) Multiplication with Overlap Handling



(d) Multiplication with Shifted Means



(e) Multiplication with Slope Calculation

Figure 6.5: Comparison of different initialization strategies and multiplication methods: Equal-bucket density (EBD), derivation-based density (DBD), and equal-bucket width (EBW) (see Section 3.1)

The quantile-based approaches did not lead to an improvement over the initial Gaussian baseline. On the other hand, the approach led to a high overload of parameters, which increased drastically according to the number of quantiles. Therefore, we do not perform further experiments using the MoU approach.

6.2.2 TrueSkill with Mixture of Gaussians

Tennis 1 vs. 1 As seen in Figure 6.6, the reference TrueSkill implementation with Gaussians outperforms our approach with MoGs. Theoretically, the MoG with unimodal prior and the reference implementation should have virtually the same performance. When all MoGs in the TrueSkill graph are initialized with an unimodal prior, all MoGs remain unimodal Gaussians, and the calculation should be identical to the traditional Gaussian approach. The lines in the graph of

the MoG with unimodal prior and Gaussian are almost perfectly parallel, except for the slightly lower start of the MoG. We suspect that this is due to rounding differences when calculating the win probabilities at the beginning when all players have identical skills.

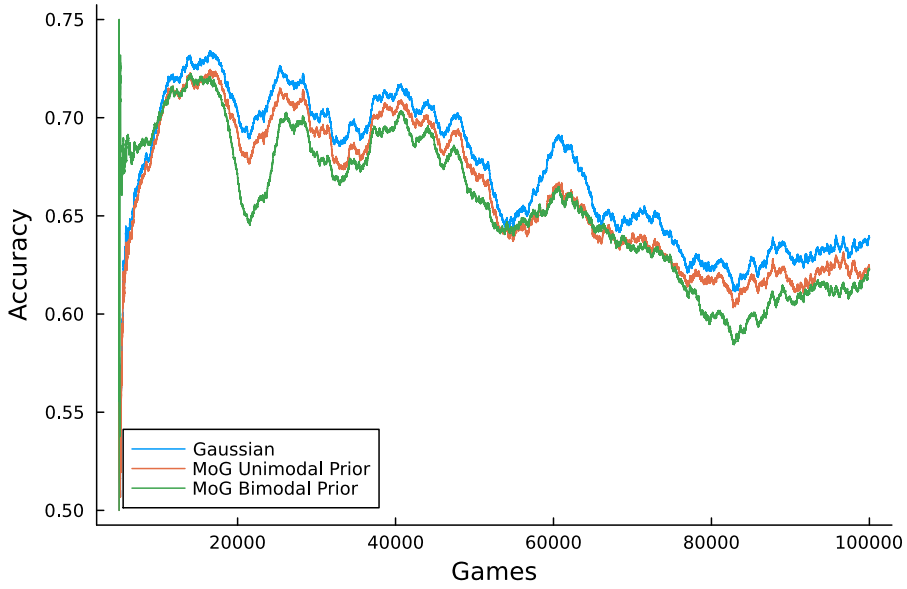


Figure 6.6: Performance comparison of TrueSkill on 1 vs. 1 tennis dataset. The maximum number of components for MoG is four. Accuracy is calculated with a sliding window of size 5,000.

Tennis 2 vs. 2 On the 2 vs. 2 tennis data, the MoG with a bimodal prior appears to learn faster, outperforming the traditional Gaussians by 3 percentage points at the start but returning to the same performance over the long run. The unimodal prior is again underperforming the Gaussian reference implementation. This can be seen in Figure 6.7.

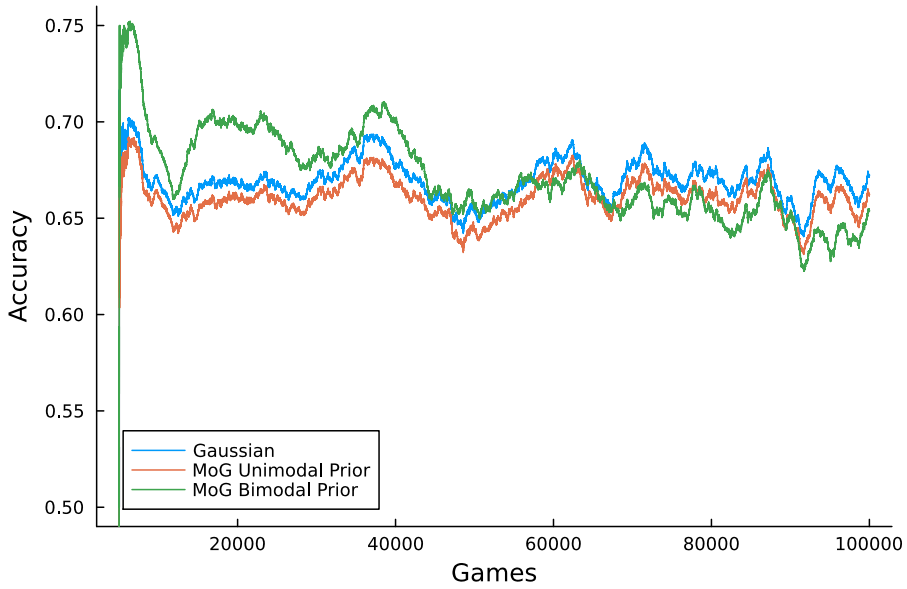


Figure 6.7: Performance comparison of TrueSkill on the 2 vs. 2 tennis dataset. The maximum number of components per mixture of Gaussians is four, and the accuracy is calculated using a sliding window of size 5,000.

League of Legends In the League of Legends games, the MoG with bimodal prior performs slightly worse. The results are shown in Figure 6.8. However, all three approaches cannot predict the game much better than by chance. This is likely because the data comes from ranked games that are supposed to be close and where the players' performances are likely already very similar. In the dataset, over the 100,000 games, each player plays on average 4 times, which is insufficient to accurately model the player's skills.

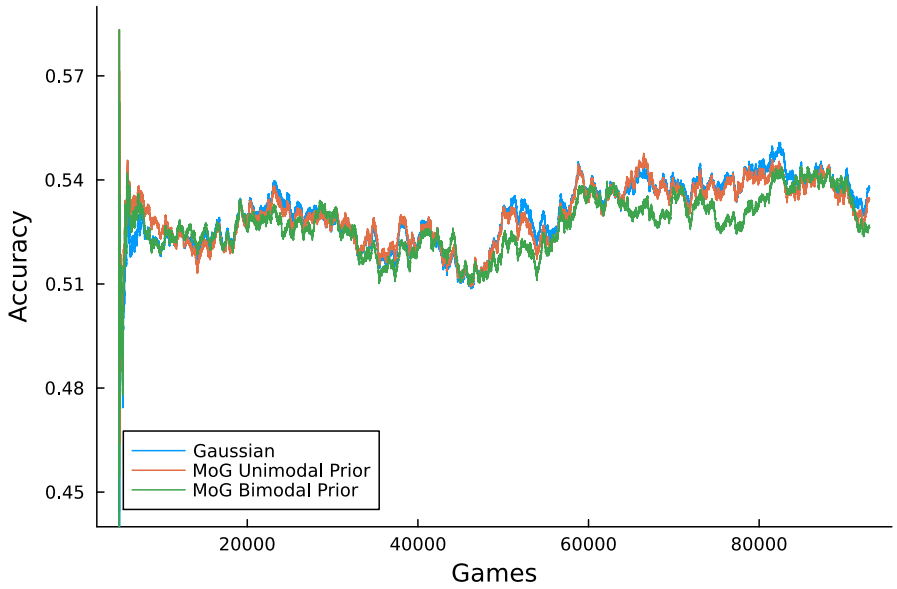


Figure 6.8: Performance comparison of TrueSkill variations on the 5 vs. 5 League of Legends dataset. The maximum number of components per mixture of Gaussians is four, and the accuracy is calculated using a sliding window of size 5,000.

Synthetic Data As shown in Figure 6.9, the MoG approach outperforms the Gaussian TrueSkill on 10,000 synthetically generated match outcomes between 100 players. However, the performance gap diminishes slowly over time. This makes sense since the skill priors are bimodal. Therefore, the MoG is able to better fit this data than the single Gaussian after the first iterations. This confirms that MoGs can be beneficial in situations with high uncertainty and less data, while Gaussians catch up once more data becomes available.

LMArena The results for the LMArena dataset, shown in Figure 6.10, indicate that Elo (see Section 1.2) drastically underperforms. The traditional Gaussian model and the two variants of MoGs perform almost identically.

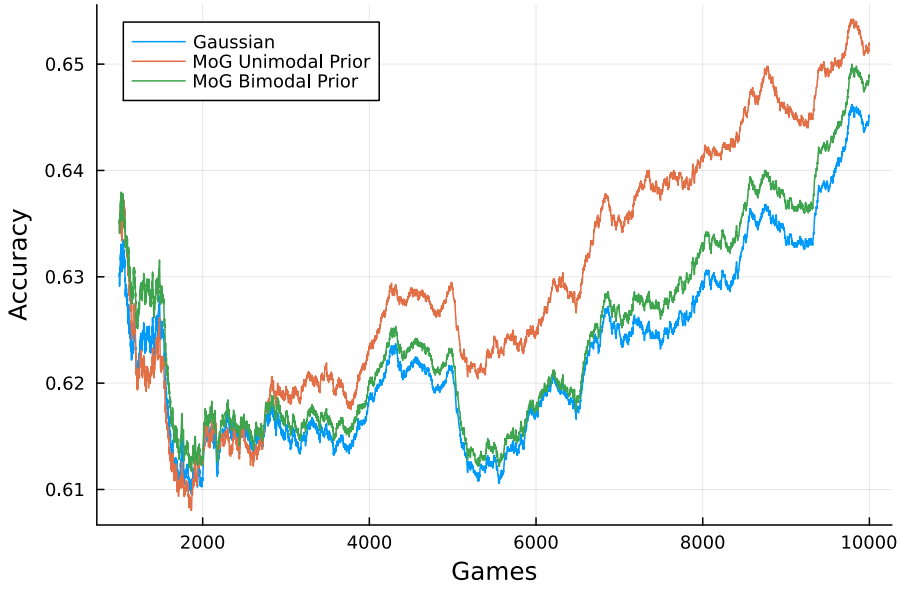


Figure 6.9: Performance comparison of TrueSkill on the 1 vs. 1 synthetic dataset. The maximum number of components per mixture of Gaussians is 8, and the accuracy is calculated using a sliding window of size 5,000.

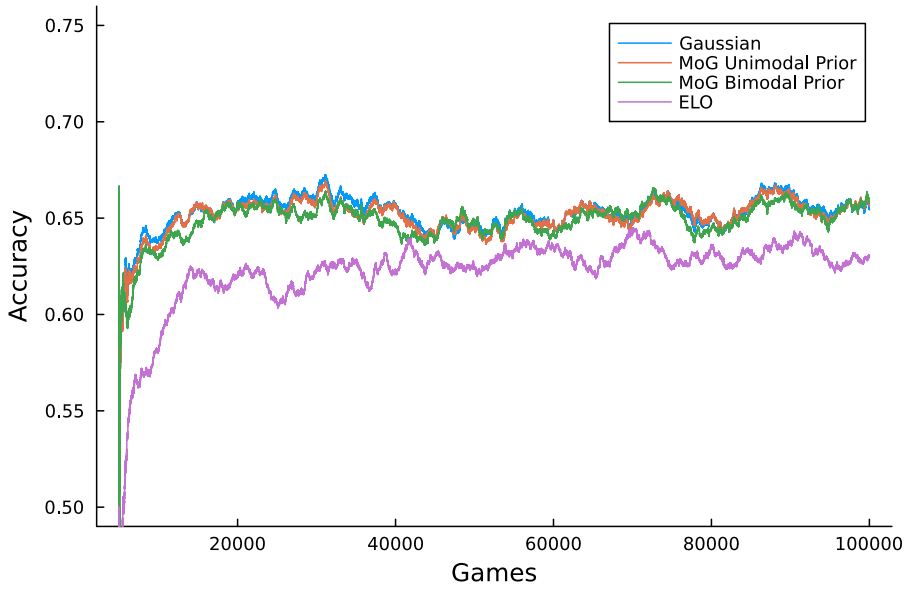


Figure 6.10: Performance comparison of TrueSkill and Elo on the LMArena dataset. The maximum number of components per mixture of Gaussians is four, and the accuracy is calculated using a sliding window of size 5,000.

Additionally, we can see the win rates predicted by both TrueSkill with MoGs and Elo in Figure 6.11. Win rates are a commonly used measure in ranking and preference learning systems [ZR25]. As observed, TrueSkill tends to assign very clear win rates close to 1.0 or 0.0 to the model. This may result from diminishing variance after training on over one million matches. Conversely, Elo shows relatively small differences in win rates between models. Interestingly, the ranking itself remains quite similar between both approaches. This can be seen in both the top 30 models from Figure 6.11 and the overall ranking curve in Figure 6.13. In the latter, we observe that the overall distribution of scores follows a similar curve for TrueSkill and Elo. However, Elo assigns larger confidence intervals (95%) generated via bootstrapping 100 times. Out of the 129 tested models,

both approaches classify ChatGPT-4o as the highest-performing one. Nonetheless, there are slight differences in ranking, such as TrueSkill placing GPT-4-Turbo higher than Athene-70b.

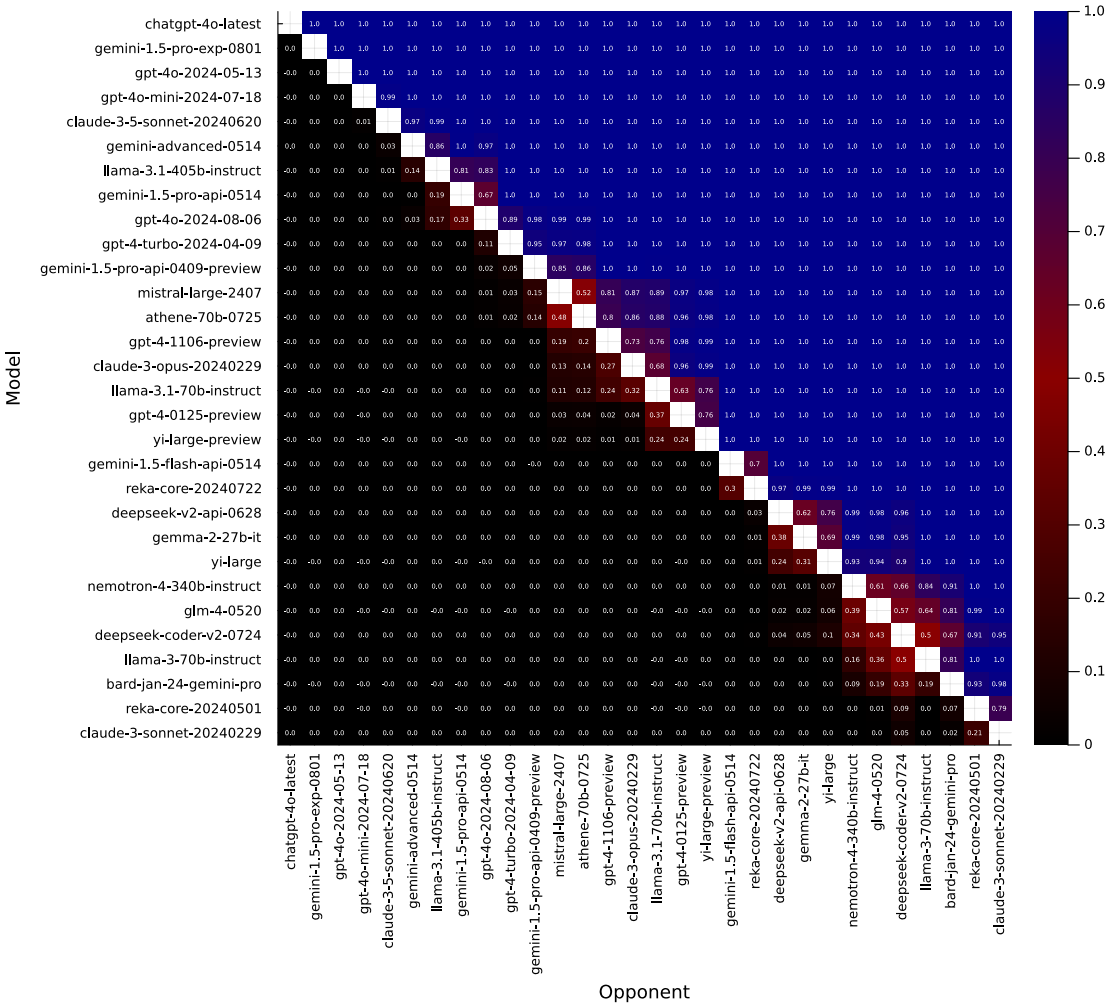


Figure 6.11: Win-rate prediction of the top 30 models using the final TrueSkill distributions with MoGs on the LMArena dataset. The MoG has very high win-rate predictions due to the small uncertainty in ratings after training on the large data corpus.

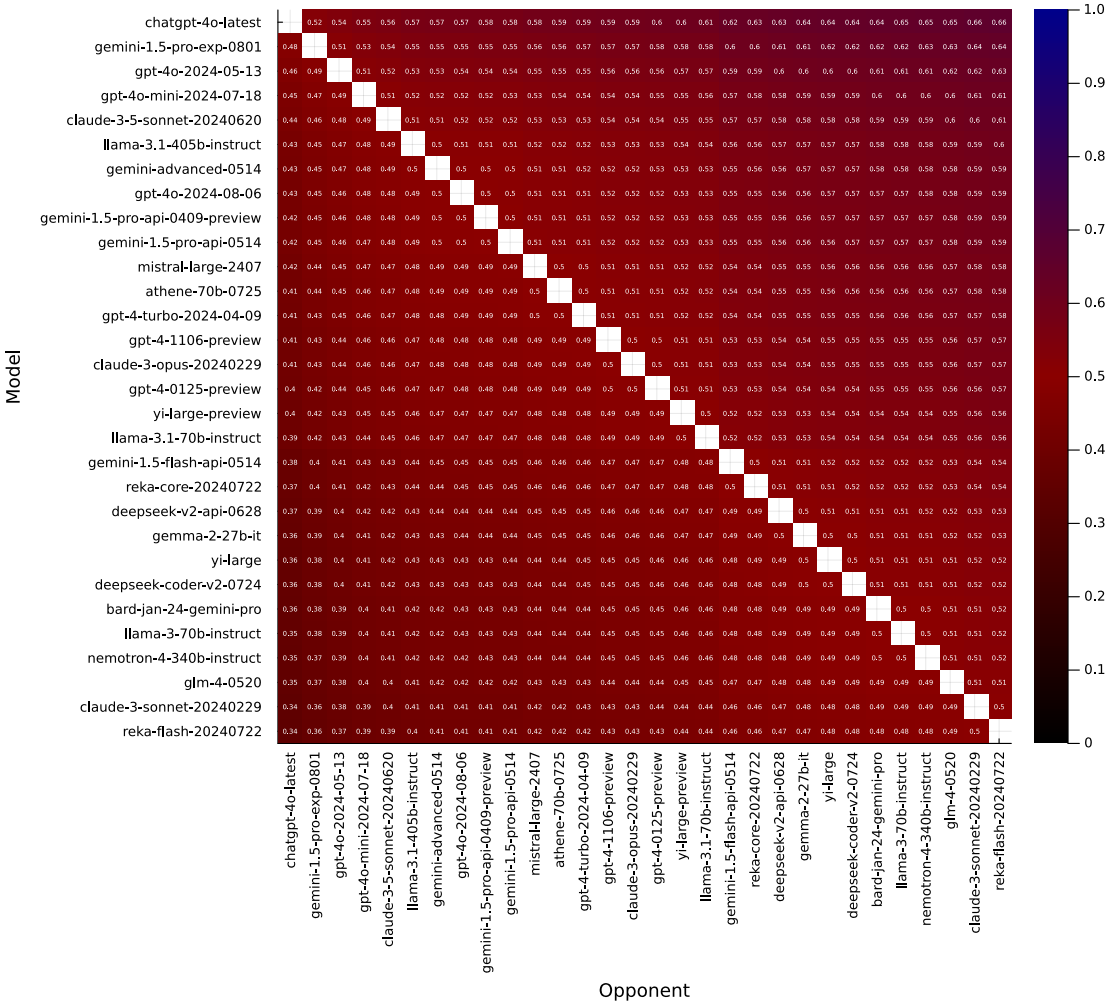


Figure 6.12: Win-rate prediction of the top 30 models using Elo predictions on the LMArena dataset. Elo maintains a higher level of uncertainty compared to TrueSkill.

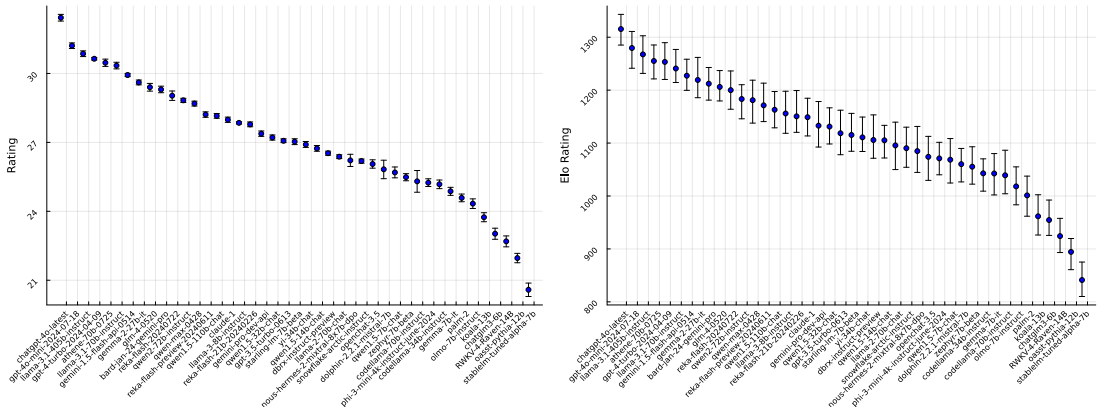


Figure 6.13: Ranking comparison of TrueSkill with bimodal MoG (left) and Elo (right) on the LMArena dataset. Both approaches rank models similarly, while TrueSkill has smaller confidence intervals. To maintain the curve's shape, the 40 depicted models have been sampled with even intervals from all models sorted by TrueSkill ranking. The Elo curve (right) then uses the same models as the one sampled in the TrueSkill curve (left).

6.2.3 Addressing Model Accuracy with EM and Calibration

Performance Comparison using EM The prior experiments show that the MoG approach cannot outperform the traditional one with unimodal priors reliably. This makes sense, as all our approximations are on a component level. When a factor receives messages with MoG consisting of a single component, it always produces messages with only one component. We thus explore the performance of the EM approach to calculate weighted sum and truncation messages, which can produce more accurate messages regardless of the number of input components. As seen in Figure 6.14, the use of EM did not lead to a high gain in accuracy, still performing worse than the Gaussian baseline and similar to the MoG without EM. However, the EM approach exhibits a much steeper learning curve in the initial stages.

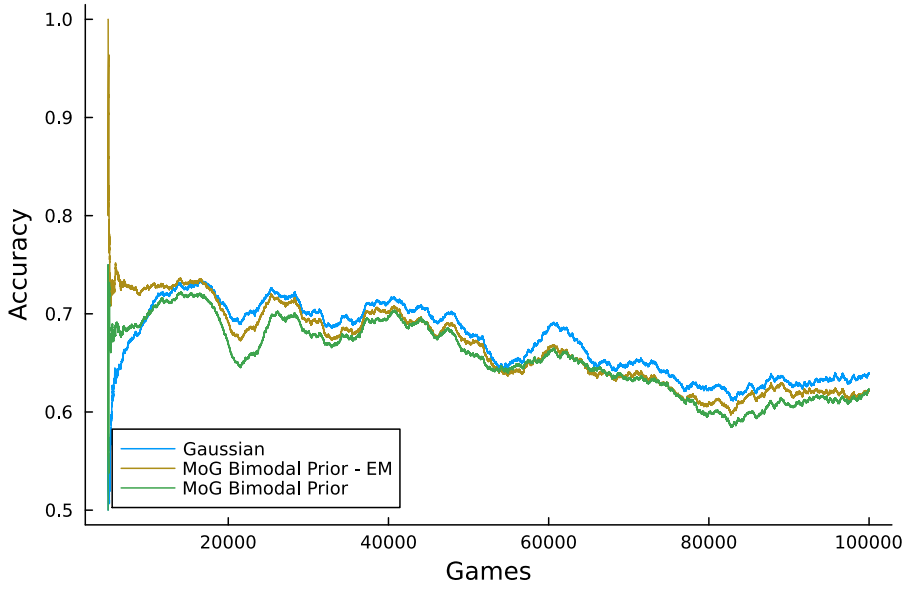


Figure 6.14: Performance comparison of TrueSkill on the 1 vs. 1 tennis dataset when approximating messages with expectation maximization (EM). A sliding window of size 5,000 is used.

Calibration Testing Unlike typically reported performance metrics like accuracy that focus solely on discrimination ability (correctly predicting or ordering outcomes), a calibration plot shows whether the predicted probabilities align with the actual outcomes. Therefore, we look at the calibration plot to evaluate the reliability of probabilistic predictions generated by the model. Ideally, the data points would lie on the 45-degree line (the line of "perfect calibration"), indicating that empirical outcomes align with the model's output confidence. For example, if the model predicts that player 1 wins with 45% probability, the plot shows how many percent player 1 actually won out of all times that the model predicted it would win with 45 %. To draw more informed interpretations of the plot, we also visualize the number of times the model predicted a certain outcome probability (aka confidence) via the size of bubbles. Figure Figure 6.15 shows the calibration of the original 1D Gaussian true skill model trained on the entire tennis dataset introduced in Subsection 6.1.2 compared to the calibration of the TrueSkill model with Gaussian mixtures. According to the plot, both approaches are overconfident where the actual probability is higher than 0.5 and underconfident elsewhere. In Figure 6.16, we compare the quantile distribution to the 1D TrueSkill implementation. Interestingly, the quantile distribution is underconfident, where the actual probability is over 0.5, and overconfident elsewhere. The horizontal symmetry around 0.5 predicted probability can be explained because 1 vs. 1 matches are a zero-sum game, and if player one wins with probability p, the probability of the other player winning is simply $1 - p$.

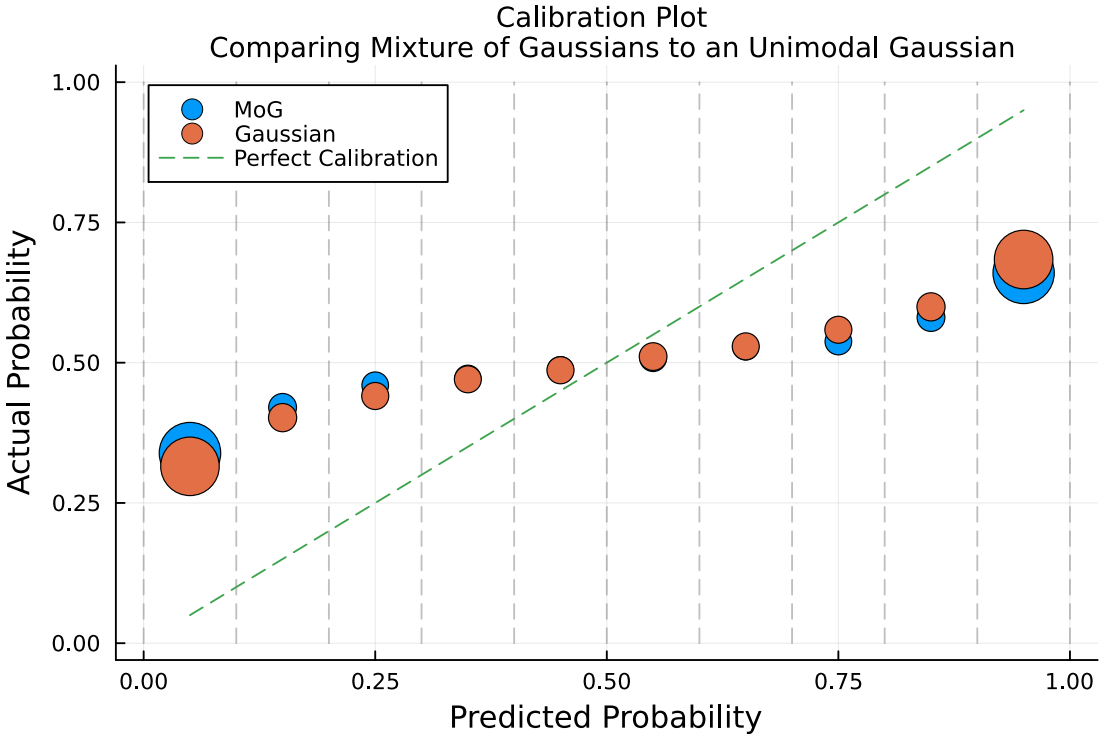


Figure 6.15: Comparison calibration 1D TrueSkill and MoG TrueSkill on the 1 vs. 1 tennis dataset. The predicted win probability is calculated in 10 bins highlighted in the plot.

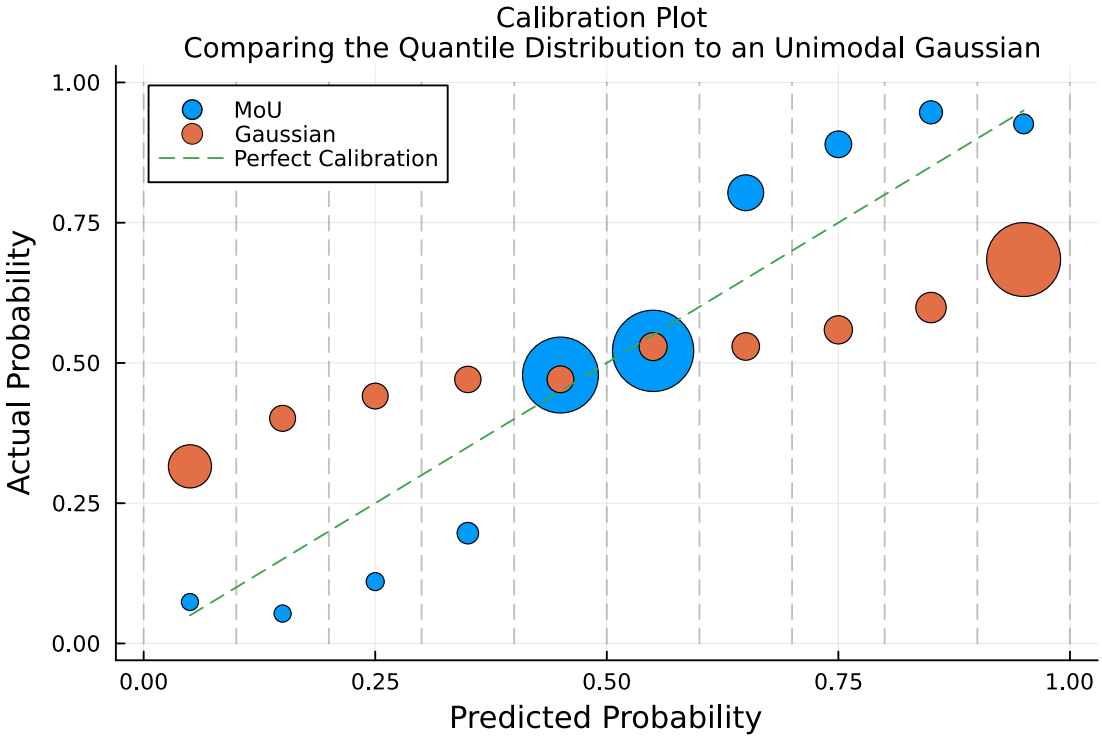


Figure 6.16: Comparison calibration 1D TrueSkill and quantile distribution on the 1 vs. 1 tennis dataset. The predicted win probability is calculated in 10 bins highlighted in the plot.

6.2.4 Neural Nets with Factor Graphs

Product Factor First, we test the implementation of the product factor with a straightforward experiment to see if the factor can correctly backpropagate messages. We take two MoGs as input messages to the product factor and ground truth, which is the backward message the product factor receives. The results of a simple experiment can be seen in Figure 6.17 and Figure 6.18. They indicate that the product factor can find the right modes.

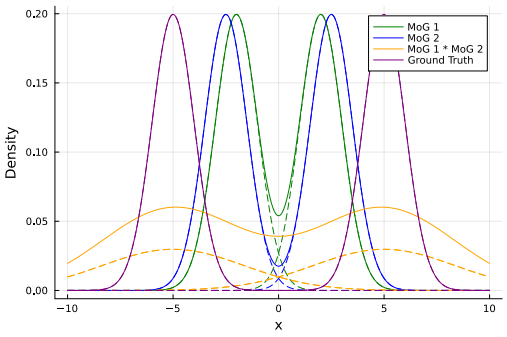


Figure 6.17: Example of two symmetrical MoGs, the product of their random variables and the desired ground truth

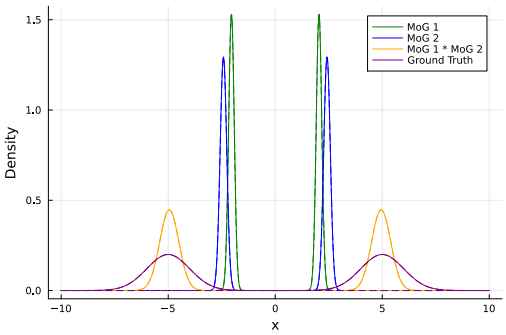


Figure 6.18: Result (in orange) after backpropagating the ground truth through the factor graph as a message to the product factor

However, we also find undesirable behaviors. For instance, if the priors are symmetric, but the ground truth is not, the factor cannot break the symmetry as shown in Figure 6.19 and Figure 6.20.

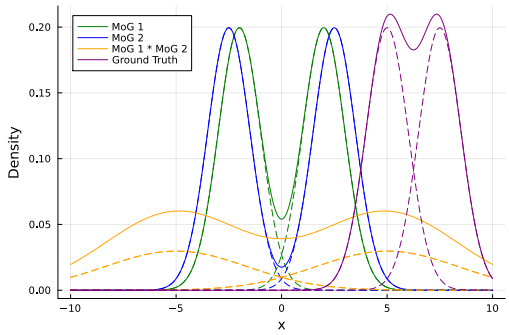


Figure 6.19: Example of two symmetrical MoGs, the product of their random variables and a positive ground truth

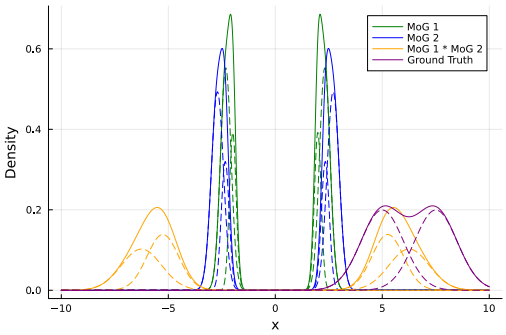


Figure 6.20: Result (in orange) after back-propagating the ground truth through the factor graph as a message to the product factor

With asymmetrical priors, we can fit modes with bimodal distributions, but aligning the input factors to a single mode remains challenging. When fitting a positive mode, both factors should be positive or negative, but if the distribution models both the negative and positive modes of the input factors, a negative mode of the output is created as a byproduct. This is shown in Figure 6.22.

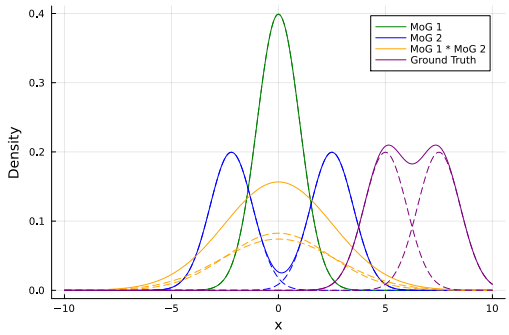


Figure 6.21: Example of two symmetrical MoGs, the product of their random variables and a positive ground truth

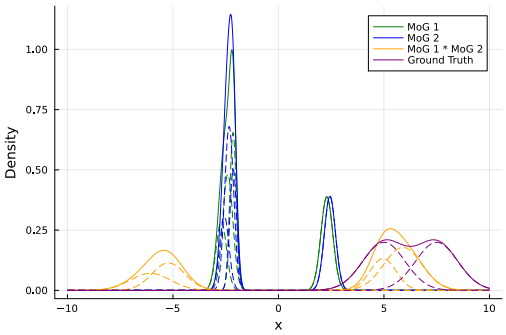


Figure 6.22: Result (in orange) after back-propagating the ground truth through the factor graph as a message to the product factor

The mode fitting issue could be resolved by not allowing messages to have positive and negative modes at the same time.

Regression We also implement linear and cubic regressions of the form $y = ax + b$ and $y = ax^3 + bx^2 + cx + d$, respectively. We use a symmetric bimodal prior with modes around 1 and -1 for all model parameters. These experiments indicate that individual product and sum factors can correctly back-propagate message updates for a small number of nodes in the factor graph. The lines are plotted by taking each distribution's mean as the parameter's value.

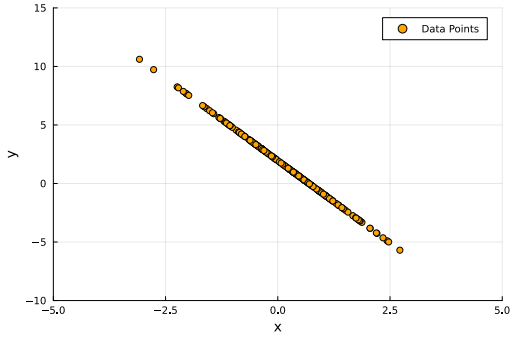


Figure 6.23: Example for a linear regression using MoGs

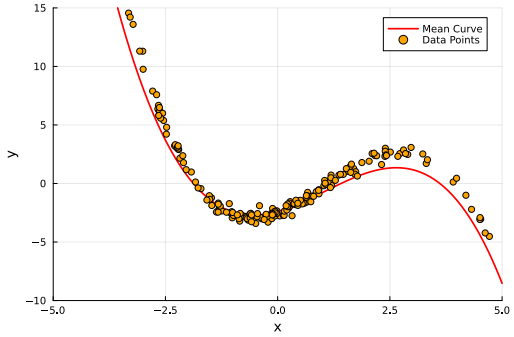


Figure 6.24: Example for a cubic regression using MoGs

However, these models do not need multimodal messages, as they can be perfectly fitted with unimodal messages. And this is indeed what happens with all parameters collapsing on a single mode.

Therefore, we designed a dataset consisting of two parallel lines. The model produced a bimodal probability distribution over the bias parameter with the modes we used to generate the data in some cases. However, this depended heavily on the choice of prior.

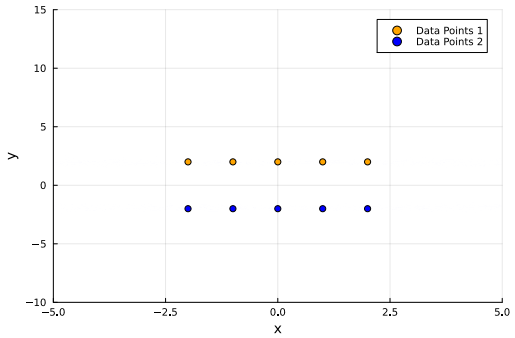


Figure 6.25: Example for a linear regression with two parallel lines with slope 0

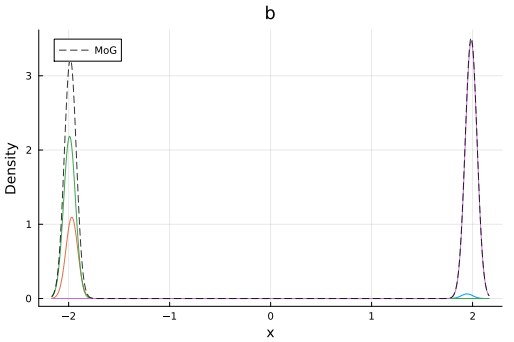


Figure 6.26: Probability distribution over the bias

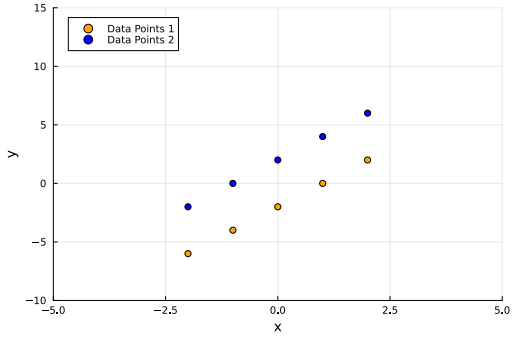


Figure 6.27: Example for a linear regression with two parallel lines with positive slope

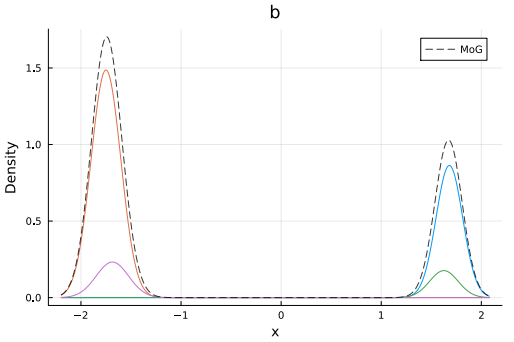


Figure 6.28: Probability distribution over the bias

Bayesian Multi-Layer Perceptrons We built an MLP and tried to train it on two simple datasets: a step function and a parabola, as seen in Figure 6.29-Figure 6.32.

We used simple Gaussians and MoGs with up to four components. The network architecture was

three layers of six neurons each, followed by LeakyReLU activation functions. As a result, the model can fit the rough shape of the data distribution.

However, with an increased number of layers, more data points, or more training iterations, the code runs into errors. We did not find the cause for the errors. Presumably, the messages deteriorate or run into still undiscovered edge cases. It is also possible that the unresolved issues with the product factor lead to this behavior. Overall, the training is not yet good enough to meaningfully compare the performance of the MLPs with Gaussians with the MoGs.

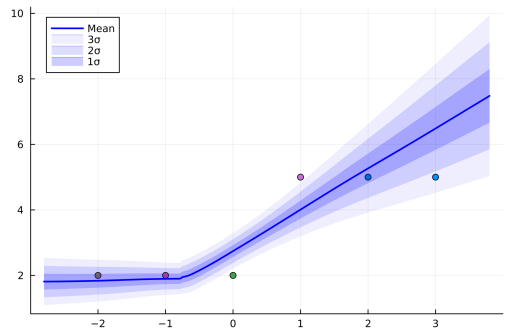


Figure 6.29: Output of MLP with Gaussians fitted on small step dataset

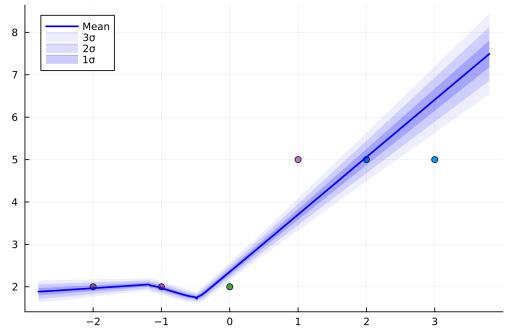


Figure 6.30: Output of MLP with MoGs with up to four components fitted on small step dataset

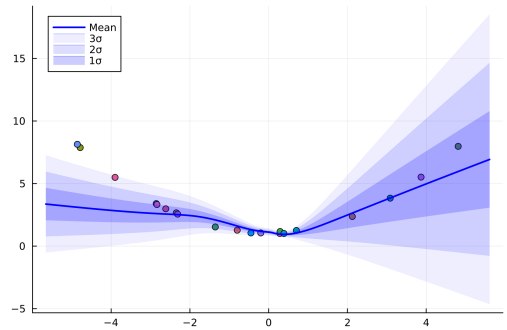


Figure 6.31: Output of MLP with Gaussians fitted on parabola

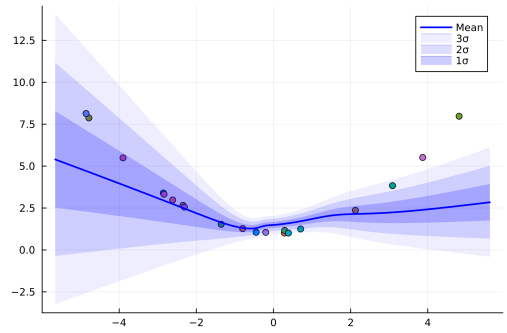


Figure 6.32: Output of MLP with MoGs with up to four components fitted on parabola

6.3 Discussion

TrueSkill with Discrete Distributions Our experiments with different discrete approaches (see Figure 6.3a) have shown that out of the investigated approaches, the equal-width approach results in the lowest KLD, and therefore has the best approximation quality. However, achieving an optimal KLD requires increasing the number of quantiles, i.e. $n \to \infty$. This leads to a high parameter overhead, resulting in inefficient computations with $\mathcal{O}(n)$ space complexity. In comparison, the initial Gaussian representation is fully defined by only two parameters, μ and σ, making it more efficient with constant space complexity $\mathcal{O}(1)$. The different multiplication methods (overlap handling, slope calculation, shifted mean) did not drastically improve the accuracy. As explained in Subsection 3.2.3, the multiplication methods become less accurate the less overlap the buckets have. Due to these observations, we can conclude that the Gaussian representation used in the original TrueSkill implementation leads to higher computational efficiency and accuracy than MoUs. However, this conclusion only accounts for our experiments on data with Gaussian priors.

TrueSkill with Mixtures of Gaussians All of the MoG experiments show that the MoG approach cannot reliably outperform the traditional one with unimodal priors. This makes sense, as all our approximations are on a component level. When a factor receives messages with MoG consisting of a single component, it always produces messages with only one component. The more data we had on a player, the better the accuracy evolved. The results indicate that the MoG model has advantages when data is multimodal. For example, for the synthetic data with bimodal priors, MoGs provided better initial performance by capturing the complexity of the underlying distributions. While MoGs outperformed the traditional Gaussian approach early on, the performance gap slowly diminished as more data became available. This leads to the hypothesis that MoGs are most useful in the early stages of learning when data is multimodal. In contrast, when the data distribution is presumably simple, the traditional Gaussian model performed equally well, or even better, from the beginning on. In these cases, the added complexity of MoGs did not provide any benefit.

Considering the LoL dataset, we did not distinguish between champions. Separating skills for different champions around a player's mean skill (similarly to the Gaussian mean factor, see Subsection 2.4.2) might be a good direction to obtain even better results since a player usually prefers to play certain roles and is not as proficient in the other roles. Therefore, the played role could be considered when predicting the match's outcome to increase accuracy.

Looking at the LMArena experiment, the high agreement between TrueSkill and Elo regarding ranking and accuracy indicates the soundness of both ranking systems. However, the confidence of the TrueSkill model might be too high to justify, considering that the answers of high-ranking LLMs might be very close to each other in quality, even though TrueSkill mostly assigns binary win rates of 1.0 or 0.0. This indicates a miscalibration of the model, aligning with the calibration findings in Figure 6.15 and Figure 6.16 regarding the 1 vs. 1 tennis matches, where the highest number of predictions was also located in the probability area 0.0 or 1.0. In general, the repeated pattern of miscalibration in multiple experiments highlights the need for more research.

In summary, MoGs may better capture more complex skill distributions. However, its advantages diminish as the size of the dataset increases and the model refines its estimates. Our results indicate that using Gaussian distributions in TrueSkill over MoGs is the more robust and efficient choice given the approximation methods for TrueSkill factors that we investigated in this work. An encouraging result is that expectation maximization is a feasible approach. It can work with as few as 1000 samples and 10 iterations, which reduces the runtime from hours (when using $> 100k$ samples and > 100 iterations) to seconds or minutes, making it only slightly slower compared to other less accurate approximations.

Neural Nets with Factor Graphs In our experiments, the product factor successfully recovered the correct modes under ideal conditions but struggled to break the symmetry with symmetric priors and asymmetric backward messages, sometimes inadvertently generating additional modes. Regression tasks revealed that while multimodal messages can capture distinct features, parameters often collapse to a single mode when an unimodal representation suffices, highlighting the sensitivity to the prior choice. Finally, Bayesian MLPs leveraging these factor graph components could approximate simple functions. However, scaling network complexity led to errors, suggesting unresolved challenges in message propagation and factor interactions that must be addressed for deeper architectures with more parameters.

Chapter 7

Conclusion

7.1 Summary

Initially, we developed discrete probability distribution representations. Their main issue is storage complexity, which requires thousands or tens of thousands of parameters to represent a simple Gaussian accurately in practice. Additionally, they are not closed under multiplication, making them virtually unusable for factor graphs. For small factor graphs, such as TrueSkill, and using moment matching, we showed that they cannot outperform the traditional approach.

Afterward, we developed a framework for multimodal message representation in factor graphs using a mixture of Gaussians. The framework implements all operations that are required for TrueSkill and MLPs, such as multiplication, addition, and scaling. Most of these operations are, however, approximations. The framework is complemented by an EM approach to approximate different messages using MoGs. Using this framework, we are able to better approximate messages in factor graphs and increase their expressiveness compared to unimodal Gaussians. On a synthetic dataset, we have shown that if the player skills are bimodal (i.e., there is a group of good and a group of not-so-good players), our model is able to approximate the real player skills better. However, we could not outperform the original TrueSkill implementation on real-world competition datasets.

Lastly, we implemented a framework for building MLPs with MoG message representations. We derived a Leaky ReLU factor and discussed various approximations for a product factor. Additionally, to the best of our knowledge, we have presented a novel application of segment trees to factor graphs. With this approach, we are able to compute outgoing messages without division in $O(log(N))$ instead of $O(N)$, where N is the number of incoming edges of that variable. We also investigated using a prefix-postfix-product with an even faster product computation in $O(1)$ time. However, the segment tree is likely the better data structure in practice due to its $O(log(N))$ updates compared to $O(N)$ updates for the prefix-postfix-product.

We have shown that linear and cubic regression work using this framework and that the model is able to accurately represent uncertainties when fitting on toy example datasets, such as parallel lines. However, the MLPs are not yet fully functional and unable to fit larger non-linear datasets. We believe that the biggest limiting factor currently is the accurate representation of backward messages for the product factor. Due to the multimodal nature of messages, we have shown that, when calculating the backward message for the product factor, the model cannot fit on a single positive output mode without simultaneously fitting on its negative twin. This is attributed to the fundamental property that when multiplying two signed factors to get a positive product, both or none of the factors can be negative. This results in two possible solutions. It remains an open problem how to solve this issue to enable accurate backpropagation through multiple layers in an MLP with MoG message representations.

7.2 Future Work

In our experiments, we could not perform better with MoUs or MoGs in TrueSkill. However, our approach could be tested on other factor graphs in different application areas where more accurate message representations might offer significant benefits.

In order to provide a more in-depth analysis of the introduced TrueSkill MoU approaches, some ideas could be further investigated. Here, the objective could be to define where the quantile distribution intermediate results differ from the unimodal Gaussian representation. Also, the quantile distribution approach could be explored using non-Gaussian priors. This benefit might make using the quantile distributions instead of Gaussians favorable for specific use cases.

A key component of our MoG framework is the component reduction algorithm. Yet, we have not thoroughly compared the different approaches. Future research could explore various reduction algorithms in the context of factor graphs and assess whether alternatives with lower time complexity than Runalls' algorithm can achieve comparable performance. For instance, West's algorithm runs in $O(n^2)$ and only keeps the k components with the highest weight in $O(n \cdot min(log(n), k)$. Other approximations might also be worth investigating, particularly factor-specific ones. For example, input messages are currently multiplied in a binary tree structure with local approximations at each node in the weighted sum factor. Exploring a single global approximation such as EM instead of multiple local ones might yield more accurate results.

Furthermore, we found that TrueSkill tends to have bad calibration, as demonstrated on the 1v1 tennis and LMArena datasets. We therefore think that more investigation is needed to learn about the mechanisms behind this phenomenon and how to improve upon it.

Finding the best approximations for the product factor remains an open challenge. This is especially true for multimodal distributions, where the mirroring issue remains unsolved. If this problem is solved, an interesting next step would be to test whether deeper MLP architectures can be trained reliably. Therefore, further relevant factors for MLPs need to be developed.

Ultimately, we aimed to enable more expressive message representations in factor graphs to improve marginal approximations. However, other methods may achieve similar or better results. Our framework using MoGs represents just one attempt to address this challenge.

Appendix

A Mathematical Background

A.1 Alpha-Divergences

When approximating a complicated true distribution $p(x)$ by a simpler distribution family $q(x;\theta)$ (e.g., a Gaussian or Gaussian Mixture with parameters θ), a closeness metric between two probability density functions has to be specified and minimized to find the optimal approximation. For this purpose, we can use a generalization of the typically used Kullback-Leibler (KL) divergence [KL51], which is called the family of α-divergences. It was introduced by [Ama07] and allows for penalizing $\{x : p(x) \neq q(x;\theta)\}$ based on certain characteristics.

Definition .1 (Alpha Divergence)**.** Let $p, q \in \mathcal{P}$ be two probability density functions (with $p(x) \geq 0$ and $\int_\Omega p(x)\,\mathrm{d}x = 1$) over a sample space Ω. Then, the α-divergence $D_\alpha(p,q) : (\mathcal{P} \times \mathcal{P} \to \mathbb{R}_{\geq 0})$ with $\alpha \in \mathbb{R}\setminus\{0,1\}$ is defined as

$$D_\alpha(p,q) = \frac{1}{\alpha(1-\alpha)}\left(1 - \int_x p(x)^\alpha q(x;\theta)^{1-\alpha}\,\mathrm{d}x\right)$$

Its key properties are:

- There is no symmetric relationship between the input argument order and the computed $D_\alpha(p,q)$ score. Thus, it is a divergence and not a distance metric.
- The α hyperparameter controls where the penalty focuses: $\alpha > 1$ weights regions higher where $p(x) > q(x;\theta)$, and $\alpha < 1$ places more weight on regions where $q(x;\theta) > p(x)$.
- $D_\alpha(p,q) = \begin{cases} 0, & \text{if } p = q \\ > 0, & \text{otherwise} \end{cases}$

A useful divergence that can be derived from the α-divergence is the Kullback Leibler divergence $\mathrm{KL} : (\mathcal{P} \times \mathcal{P} \to \mathbb{R}_{\geq 0})$. The forward $\mathrm{KL}(p \parallel q)$ and reverse $\mathrm{KL}(p \parallel q)$ are special cases of D_α for $\lim_{\alpha\to 1}$ and $\lim_{\alpha\to 0}$, respectively. We show their different characteristics (fitting to one mode vs. covering all modes) in Figure 1. Here, we derive the forward $\mathrm{KL}(p \parallel q)$, which we often use in this report.

Forward KL Divergence:

$$\lim_{\alpha\to 1} D_\alpha(p,q) = \lim_{\alpha\to 1} \frac{1 - \int_x p(x)^\alpha q(x;\theta)^{1-\alpha}\,\mathrm{d}x}{\alpha(1-\alpha)} \tag{1}$$

Applying L'Hôpital's rule [Tay52] $\lim_{\alpha\to x} \frac{f(\alpha)}{g(\alpha)} = \lim_{\alpha\to x} \frac{f'(\alpha)}{g'(\alpha)}$ for the indeterminate $0/0$ form:

$$
\begin{aligned}
\lim_{\alpha\to 1} D_\alpha(p,q) &= \lim_{\alpha\to 1} \frac{\frac{d}{d\alpha}\left(1 - \int_x p(x)^\alpha q(x;\theta)^{1-\alpha}\,\mathrm{d}x\right)}{\frac{d}{d\alpha}\left(\alpha(1-\alpha)\right)} \\
&= \lim_{\alpha\to 1} \frac{-\int_x \frac{d}{d\alpha}\left(p(x)^\alpha q(x;\theta)^{1-\alpha}\right)\,\mathrm{d}x}{\frac{d}{d\alpha}\alpha - \alpha^2} \\
&= \lim_{\alpha\to 1} \frac{-\int_x \left(\frac{d}{d\alpha}p(x)^\alpha \cdot q(x;\theta)^{1-\alpha} + p(x)^\alpha \cdot \frac{d}{d\alpha}q(x;\theta)^{1-\alpha}\right)\,\mathrm{d}x}{\frac{d}{d\alpha}\alpha - \alpha^2} \\
&= \lim_{\alpha\to 1} \frac{-\int_x \left(p(x)^\alpha \ln p(x)\cdot q(x;\theta)^{1-\alpha} - p(x)^\alpha \cdot q(x;\theta)^{1-\alpha}\ln q(x;\theta)\right)\,\mathrm{d}x}{\frac{d}{d\alpha}\alpha - \alpha^2} \\
&= \lim_{\alpha\to 1} \frac{-\int_x \left(p(x)^\alpha q(x;\theta)^{1-\alpha}\left(\ln p(x) - \ln q(x;\theta)\right)\right)\,\mathrm{d}x}{1-2\alpha} \\
&= \lim_{\alpha\to 1} \frac{-\int_x \left(p(x)^\alpha q(x;\theta)^{1-\alpha}\ln\frac{p(x)}{q(x;\theta)}\right)\,\mathrm{d}x}{1-2\alpha}
\end{aligned}
\tag{2}
$$

Evaluating at $\alpha = 1$:

$$
\begin{aligned}
\lim_{\alpha\to 1} D_\alpha(p,q) &= \frac{-\int_x p(x)\ln\frac{p(x)}{q(x;\theta)}\,\mathrm{d}x}{1-2(1)} \\
&= \int_x p(x)\ln\frac{p(x)}{q(x;\theta)}\,\mathrm{d}x \\
&= \mathbb{E}\left[\ln\frac{p(x)}{q(x;\theta)}\right] \\
&= \mathrm{KL}(p \parallel q).
\end{aligned}
\tag{3}
$$

The chosen order of the input arguments of the KL divergence minimization criterion impacts the characteristics of the resulting approximation. The forward $\mathrm{KL}(p \parallel q)$ criterion penalizes heavily when $p(x) \gg q(x;\theta)$, i.e., in high-density regions that $q(x;\theta)$ does not cover. As a result, the forward KL favors approximations that cover all peaks, leading to approximations with extremely high variance. This is especially problematic when the approximator has fewer modes than the target. Previous works [Ada+24; SHG09; Som24] have solved this issue by minimizing the reverse $\mathrm{KL}(q \parallel p)$, which has high penalties when $q(x;\theta) \gg p(x)$. This resolves the high-variance problem but typically causes the function to fit a subset of the target density modes. This is symmetry breaking because it practically selects a subset of possible 'solutions' (high-density regions).

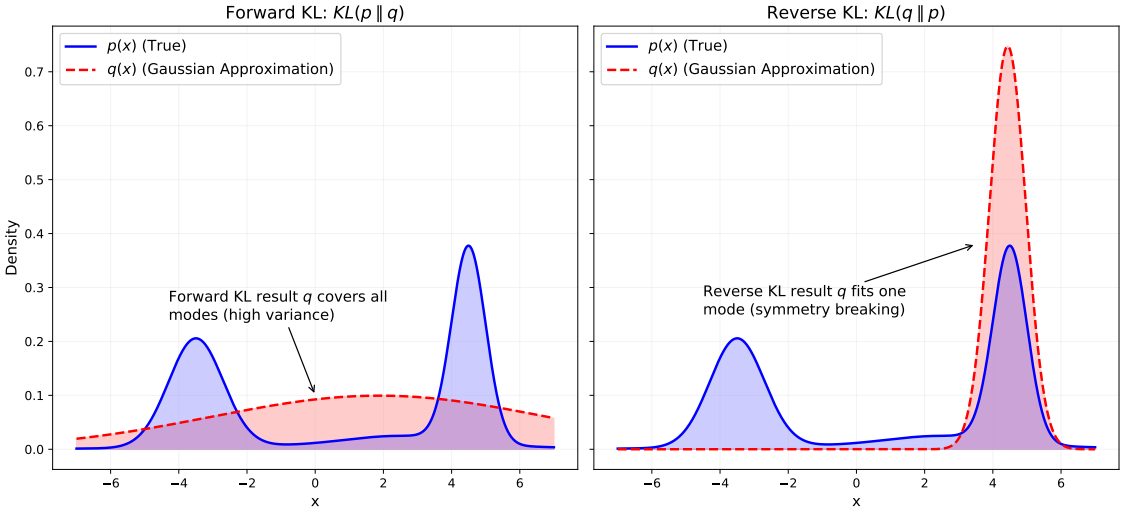


Figure 1: Divergences comparison for different settings of α. The left subplot shows an approximation that minimizes the forward KL divergence ($\alpha = 1$) and the right subplot shows an approximation that minimizes the reverse KL divergence ($\alpha = 0$).

A.2 Dirac Delta Function

The Dirac delta function ($\delta(x)$) is a distribution that is defined by the following conditions:

1. Zero everywhere except at zero:

$$\delta(x) = \begin{cases} \infty, & x = 0, \\ 0, & x \neq 0. \end{cases}$$

2. Normalization condition:

$$\int_{-\infty}^{\infty} \delta(x)\, dx = 1.$$

3. Scaling property: For any non-zero constant a:

$$\delta(ax) = \frac{1}{|a|}\delta(x). \tag{4}$$

The *sifting property* of the Dirac delta function is its most important feature, allowing it to "pick out" the value of a function at a specific point. For any continuous function $f(x)$ and $x_0 \in \mathbb{R}$:

$$\int_{-\infty}^{\infty} f(x)\delta(x - x_0)\, dx = f(x_0). \tag{5}$$

This property effectively reduces the integral to the value of $f(x)$ at the location where the delta function is centered, $x = x_0$.

In factor graphs and message-passing algorithms, the Dirac delta function often appears to encode equality constraints. For example, consider an integral where the Dirac delta function imposes a constraint on a random variable z:

$$m_{f\to z}(z) = \int_{a\in\mathbb{R}} \delta(a - g(z))m_{a\to f}(a)\, da.$$

Using the sifting property, this integral simplifies to:

$$m_{f\to z}(z) = m_{a\to f}(g(z)),$$

where $g(z)$ is the transformation imposed by the delta function.

A.3 Moment Matching

An important property we will use in this work is that for the exponential family, which includes the Normal distribution, $\mathrm{KL}(p \parallel q)$ is minimized by moment matching [Her05], also known as the method of moments. This is also closely related to assumed-density filtering [Min13], which uses moment matching for approximate Bayesian inference. In this section, we will prove that moment matching minimizes the KL-divergence for the univariate Gaussian case, which is sufficient for this work. For the multivariate case, we refer the interested reader to Herbrich's proof [Her05].

Derivation for a univariate Gaussian. The forward KL divergence between the true distribution $p(x)$ and the approximating Gaussian distribution $q(x;\theta) = \mathcal{N}(x;\mu,\sigma^2)$ is given by:

$$\mathrm{KL}(p \parallel q) = \int_x p(x) \ln \frac{p(x)}{\mathcal{N}(x;\mu,\sigma^2)}\, \mathrm{d}x \tag{6}$$

Expanding $\ln(\frac{a}{b}) = \ln(a) - \ln(b)$ and $\mathcal{N}(x; \mu, \sigma^2)$, we get:

$$\begin{aligned}
\mathrm{KL}(p \parallel q) &= \int_x p(x) \left(\ln p(x) - \ln \mathcal{N}(x; \mu, \sigma^2) \right) \mathrm{d}x \\
&= \int_x p(x) \ln p(x) \, \mathrm{d}x - \int_x p(x) \ln \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \right) \mathrm{d}x
\end{aligned} \tag{7}$$

We are minimizing concerning μ and σ^2, so the first term is constant and can be dropped:

$$\begin{aligned}
\arg\min_{\mu,\sigma^2} \mathrm{KL}(p \parallel q) &= \arg\min_{\mu,\sigma^2} \left[\cancel{\int_x p(x) \ln p(x) \, \mathrm{d}x} - \int_x p(x) \ln \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \right) \mathrm{d}x \right] \\
&= \arg\min_{\mu,\sigma^2} \left[- \int_x p(x) \ln \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \right) \mathrm{d}x \right] \\
&= \arg\min_{\mu,\sigma^2} \left[\int_x p(x) \left(\ln \left(\sqrt{2\pi\sigma^2} \right) + \frac{(x-\mu)^2}{2\sigma^2} \right) \mathrm{d}x \right] \\
&= \arg\min_{\mu,\sigma^2} \left[\frac{1}{2} \ln(2\pi\sigma^2) \underbrace{\int_x p(x) \, \mathrm{d}x}_{=1} + \int_x p(x) \frac{(x-\mu)^2}{2\sigma^2} \mathrm{d}x \right] \\
&= \arg\min_{\mu,\sigma^2} \left[\frac{1}{2} \ln(2\pi\sigma^2) + \frac{1}{2\sigma^2} \int_x p(x)(x-\mu)^2 \, \mathrm{d}x \right]
\end{aligned} \tag{8}$$

Differentiating with respect to μ:

$$\begin{aligned}
\frac{\partial}{\partial \mu} \left[\cancel{\frac{1}{2} \ln(2\pi\sigma^2)} + \frac{1}{2\sigma^2} \int_x p(x)(x-\mu)^2 \, \mathrm{d}x \right] &= \frac{1}{2\sigma^2} \int_x p(x) \frac{\partial}{\partial \mu} \left[(x-\mu)^2 \right] \mathrm{d}x \\
&= -\frac{1}{\sigma^2} \int_x p(x)(x-\mu) \, \mathrm{d}x
\end{aligned} \tag{9}$$

Setting the derivative to 0:

$$\begin{aligned}
0 &= -\frac{1}{\sigma^2} \int_x p(x)(x-\mu) \, \mathrm{d}x \\
&= -\frac{1}{\sigma^2} \left(\underbrace{\int_x p(x) x \, \mathrm{d}x}_{=\mathbb{E}[x]} - \mu \underbrace{\int_x p(x) \, \mathrm{d}x}_{=1} \right) \\
&= -\frac{1}{\sigma^2} \left(\mathbb{E}[x] - \mu \right) \\
&= \mathbb{E}[x] - \mu
\end{aligned} \tag{10}$$

Solving for μ, we retrieve the optimal mean as $\mu^* = \mathbb{E}[x]$, which is the first moment. Now, we will derive the optimal variance.

Differentiating with respect to σ^2:

$$\begin{aligned}
\frac{\partial}{\partial \sigma^2} \left[\frac{1}{2} \ln(2\pi\sigma^2) + \frac{1}{2\sigma^2} \int_x p(x)(x-\mu)^2 \, \mathrm{d}x \right] &= \frac{1}{2} \left(\cancel{\frac{\partial}{\partial \sigma^2} [\ln(2\pi)]} + \frac{\partial}{\partial \sigma^2} \ln\left(\sigma^2\right) \right) + \frac{\partial}{\partial \sigma^2} \left[\frac{1}{2\sigma^2} \underbrace{\int_x p(x)(x-\mu)^2 \, \mathrm{d}x}_{\mathbb{V}[x]} \right] \\
&= \frac{1}{2\sigma^2} - \frac{1}{2\sigma^4} \mathbb{V}[x]
\end{aligned} \tag{11}$$

Setting the derivative to 0:

$$\begin{aligned} 0 &= \frac{1}{2\sigma^2} - \frac{1}{2\sigma^4}\,\mathbb{V}[x] \\ \frac{1}{\sigma^2} &= \frac{\mathbb{V}[x]}{\sigma^4} \\ (\sigma^2)^* &= \mathbb{V}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \end{aligned} \tag{12}$$

Thus, we have shown that moment matching yields the optimal approximation that minimizes the KL-divergence for the case where the approximation is a univariate Gaussian.

Approximate Message Passing. One important caveat is that we can perform moment matching only on marginal distributions $p(x)$. When dealing with factors that operate on these distributions (e.g., Uniform, Gaussian, or Mixture Models) over the input random variables $x \in \text{Ne}(f)\backslash\{x\}$, if the operation isn't closed for such distributions, an approximation for the messages has to be computed using the marginal distribution approximations:

Approximate Message Passing via Moment Matching for Factor-to-Variable Messages

The approximated message $\hat{m}_{f\to x}(x)$ can be computed using the approximated marginal $q(x;\theta) \approx p(x)$:

$$\hat{m}_{f\to x}(x) = \frac{q(x;\theta)}{m_{x\to f(x)}}$$

The messages must be computed using an approximated marginal $q(x;\theta)$ because only marginals can be approximated using distributions. This is because messages can represent arbitrary functions that do not necessarily need to integrate to 1. For example, truncating a Gaussian distribution results in a step function message with an infinite area under the curve; see subsection 2.4.4.

A.4 Gaussian Mean Factor Message to Variable: Convolution of two Gaussians

As stated in Section 2.4.2 for the Gaussian mean factor, when the incoming message from the skill s_i to the factor is Gaussian, we get the following factor-to-variable message by applying the message-passing algorithm 2.5:

$$m_{f\to p_i}(p_i) = \int_{-\infty}^{\infty} \underbrace{\mathcal{N}(p_i; s_i, \beta^2)}_{f(s_i,p_i)} \cdot \underbrace{\mathcal{N}(s_i; \mu_{s_i}, \sigma_{s_i}^2)}_{m_{s_i\to f(s_i)}} \,\mathrm{d}s_i = \mathcal{N}(p_i; \mu_{s_i}, \sigma_{s_i}^2 + \beta^2) \tag{13}$$

In this section, we validate that the convolution of two Gaussians yields another Gaussian. Moreover, we derive the parameters of the $m_{f\to p_i}(p_i)$. To begin, we can write out the product of the two Gaussians inside the integral:

$$\begin{aligned} \mathcal{N}(p_i; s_i, \beta^2)\mathcal{N}(s_i; \mu_{s_i}, \sigma_{s_i}^2) &= \left[\frac{1}{\sqrt{2\pi}\beta}\exp\left(-\frac{(p_i - s_i)^2}{2\beta^2}\right)\right] \cdot \left[\frac{1}{\sqrt{2\pi}\sigma_{s_i}}\exp\left(-\frac{(s_i - \mu_{s_i})^2}{2\sigma_{s_i}^2}\right)\right] \\ &= \frac{1}{2\pi\beta\sigma_{s_i}}\exp\left(-\frac{1}{2}\left(\frac{(p_i - s_i)^2}{\beta^2} + \frac{(s_i - \mu_{s_i})^2}{\sigma_{s_i}^2}\right)\right) \\ &= \frac{1}{2\pi\beta\sigma_{s_i}}\exp\left(-\frac{1}{2}\left(\frac{p_i^2}{\beta^2} - 2\frac{p_i s_i}{\beta^2} + \frac{s_i^2}{\beta^2} + \frac{s_i^2}{\sigma_{s_i}^2} - 2\frac{\mu_{s_i} s_i}{\sigma_{s_i}^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^2}\right)\right) \\ &= \frac{1}{2\pi\beta\sigma_{s_i}}\exp\left(-\frac{1}{2}\left(\underbrace{\left(\frac{1}{\beta^2} + \frac{1}{\sigma_{s_i}^2}\right)}_{a} s_i^2 - 2\underbrace{\left(\frac{p_i}{\beta^2} + \frac{\mu_{s_i}}{\sigma_{s_i^2}}\right)}_{b} s_i + \underbrace{\frac{p_i^2}{\beta^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^2}}_{c}\right)\right) \end{aligned}$$

We can simplify this quadratic equation by completing the square, i.e., making it a perfect square equation by adding a constant at the end:

$$\begin{aligned} as_i^2 - 2bs_i + c &= a\left(s_i^2 - 2\frac{b}{a}s_i\right) + c \\ &= a\left(s_i - \frac{b}{a}\right)^2 + c - \frac{b^2}{a} \end{aligned}$$

Hence, we have

$$\begin{aligned} \int_{-\infty}^{\infty} \mathcal{N}(p_i; s_i, \beta^2)\mathcal{N}(s_i; \mu_{s_i}, \sigma_{s_i}^2)\,\mathrm{d}s_i &= \int_{-\infty}^{\infty} \frac{1}{2\pi\beta\sigma_{s_i}} \exp\left(-\frac{1}{2}a\left(s_i - \frac{b}{a}\right)^2 - \frac{1}{2}\left(c - \frac{b^2}{a}\right)\right)\mathrm{d}s_i \\ &= \int_{-\infty}^{\infty} \frac{1}{2\pi\beta\sigma_{s_i}} \exp\left(-\frac{1}{2}a\left(s_i - \frac{b}{a}\right)^2\right)\exp\left(-\frac{1}{2}\left(c - \frac{b^2}{a}\right)\right)\mathrm{d}s_i \\ &= \frac{1}{2\pi\beta\sigma_{s_i}} \exp\left(-\frac{1}{2}\left(c - \frac{b^2}{a}\right)\right)\int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}a\left(s_i - \frac{b}{a}\right)^2\right)\mathrm{d}s_i \end{aligned}$$

The only part dependent on s_i is $\exp\left(-\frac{1}{2}a\left(s_i - \frac{b}{a}\right)^2\right)$. By rewriting it as $\exp\left(-\frac{1}{2}\frac{\left(s_i - \frac{b}{a}\right)^2}{\frac{1}{a}}\right)$, we can see that it is an unnormalized Gaussian with mean $\mu_{m_{f\to p_i}} = \frac{b}{a}$ and variance $\sigma^2_{m_{f\to p_i}} = \frac{1}{a}$. For a Gaussian, we know that the normalization constant is:

$$\int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}a\left(s_i - \frac{b}{a}\right)^2\right)\mathrm{d}s_i = \sqrt{\frac{2\pi}{a}}$$

Thus, the message becomes:

$$m_{f\to p_i}(p_i) = \frac{1}{2\pi\beta\sigma_{s_i}}\sqrt{\frac{2\pi}{a}}\exp\left(-\frac{1}{2}\left(c - \frac{b^2}{a}\right)\right)$$

Substituting in a, b and c, we have:

$$m_{f\to p_i}(p_i) = \frac{1}{2\pi\beta\sigma_{s_i}}\sqrt{\frac{2\pi}{a}}\exp\left(-\frac{1}{2}\left(\frac{p_i^2}{\beta^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^2} - \frac{\left(\frac{p_i}{\beta^2} + \frac{\mu_{s_i}}{\sigma_{s_i}^2}\right)^2}{\frac{1}{\beta^2} + \frac{1}{\sigma_{s_i}^2}}\right)\right)$$

To make further progress, we multiply with the reciprocal of the denominator and then expand the terms:

$$\begin{aligned} m_{f\to p_i}(p_i) &= \frac{1}{2\pi\beta\sigma_{s_i}}\sqrt{\frac{2\pi}{a}}\exp\left(-\frac{1}{2}\left(\frac{p_i^2}{\beta^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^2} - \frac{\frac{p_i^2}{\beta^4} + 2\frac{p_i\mu_{s_i}}{\beta^2\sigma_{s_i}^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^4}}{\frac{\sigma_{s_i}^2+\beta^2}{\beta^2\sigma_{s_i}^2}}\right)\right) \\ &= \frac{1}{2\pi\beta\sigma_{s_i}}\sqrt{\frac{2\pi}{a}}\exp\left(-\frac{1}{2}\left(\frac{p_i^2}{\beta^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^2} - \left(\frac{p_i^2}{\beta^4} + 2\frac{p_i\mu_{s_i}}{\beta^2\sigma_{s_i}^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^4}\right)\frac{\beta^2\sigma_{s_i}^2}{\sigma_{s_i}^2+\beta^2}\right)\right) \\ &= \frac{1}{2\pi\beta\sigma_{s_i}}\sqrt{\frac{2\pi}{a}}\exp\left(-\frac{1}{2}\left(\frac{p_i^2}{\beta^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^2} - \frac{p_i\sigma_{s_i}^2}{\beta^2(\sigma_{s_i}^2+\beta^2)} - \frac{2p_i\mu_{s_i}}{\sigma_{s_i}^2+\beta^2} - \frac{\mu_{s_i}^2\beta^2}{\sigma_{s_i}^2(\sigma_{s_i}^2+\beta^2)}\right)\right) \end{aligned}$$

Simplifying the terms in the exponent by grouping two of the terms with p_i^2, we get:

$$\frac{p_i^2}{\beta^2} - \frac{p_i^2\sigma_{s_i}^2}{\beta^2(\sigma_{s_i}^2+\beta^2)} = \frac{p_i^2}{\beta^2}\left(1 - \frac{\sigma_{s_i}^2}{\sigma_{s_i}^2+\beta^2}\right) = \frac{p_i^2}{\beta^2}\left(\frac{(\sigma_{s_i}^2+\beta^2)-\sigma_{s_i}^2}{\sigma_{s_i}^2+\beta^2}\right) = \frac{p_i^2}{\beta^2}\left(\frac{\beta^2}{\sigma_{s_i}^2+\beta^2}\right) = \frac{p_i^2}{\sigma_{s_i}^2+\beta^2}$$

and similarly we can combine two of the $\mu_{s_i}^2$ terms:

$$\frac{\mu_{s_i}^2}{\sigma_{s_i}^2} - \frac{\mu_{s_i}^2\beta^2}{\sigma_{s_i}^2(\sigma_{s_i}^2+\beta^2)} = \frac{\mu_{s_i}^2}{\sigma_{s_i}^2}\left(1 - \frac{\beta^2}{\sigma_{s_i}^2+\beta^2}\right) = \frac{\mu_{s_i}^2}{\sigma_{s_i}^2}\left(\frac{(\sigma_{s_i}^2+\beta^2)-\beta^2}{\sigma_{s_i}^2+\beta^2}\right) = \frac{\mu_{s_i}^2}{\sigma_{s_i}^2}\left(\frac{\sigma_{s_i}^2}{\sigma_{s_i}^2+\beta^2}\right) = \frac{\mu_{s_i}^2}{\sigma_{s_i}^2+\beta^2}$$

Finally, we can insert both terms into the equation and simplify the exponent:

$$\begin{aligned} m_{f\to p_i}(p_i) &= \frac{1}{2\pi\beta\sigma_{s_i}}\sqrt{\frac{2\pi}{a}}\exp\left(-\frac{1}{2}\left(\frac{p_i^2}{\sigma_{s_i}^2+\beta^2} + \frac{\mu_{s_i}^2}{\sigma_{s_i}^2+\beta^2} - \frac{2p_i\mu_{s_i}}{\sigma_{s_i}^2+\beta^2}\right)\right) \\ &= \frac{1}{2\pi\beta\sigma_{s_i}}\sqrt{\frac{2\pi}{a}}\exp\left(-\frac{1}{2}\left(\frac{(p_i-\mu_{s_i})^2}{\sigma_{s_i}^2+\beta^2}\right)\right) \end{aligned}$$

This is an unnormalized Gaussian. Thus, we can retrieve the mean and variance of the factor-to-variable message $m_{f\to p_i}(p_i)$:

$$\begin{aligned} \mathbb{E}[m_{f\to p_i}] &= \mu_{s_i} \\ \mathbb{V}[m_{f\to p_i}] &= \sigma_{s_i}^2 + \beta^2 \end{aligned}$$

Thus, the factor-to-variable message of the Gaussian mean factor is:

$$m_{f\to p_i}(p_i) = \mathcal{N}(p_i; \mu_{s_i}, \sigma_{s_i}^2 + \beta^2) \tag{14}$$

A.5 Symmetric Gaussian Messages

Here we briefly show that $\mathcal{N}(x; y, \beta^2) = \mathcal{N}(y; x, \beta^2)$. Given the two normal distributions, we can write out their PDFs:

$$\begin{aligned} \mathcal{N}(x; y, \beta^2) &= \frac{1}{\sqrt{2\pi\beta^2}}\exp\left(-\frac{(x-y)^2}{2\beta^2}\right) \\ \mathcal{N}(y; x, \beta^2) &= \frac{1}{\sqrt{2\pi\beta^2}}\exp\left(-\frac{(y-x)^2}{2\beta^2}\right) \end{aligned}$$

We can see that the squared term $(x-y)^2$ in the exponent is the same as $(y-x)^2$, because squaring the difference makes it symmetric. If both PDFs also have the same variance, the distributions are identical. Thus, we have derived the following equality using this symmetry:

$$\mathcal{N}(x; y, \beta^2) = \mathcal{N}(y; x, \beta^2) \tag{15}$$

A.6 Product Factor Backward Message Derivation

Partial progress towards a marginal approximation. In this section, we make partial progress towards a Gaussian approximation that optimizes the reverse KL-divergence on the marginal of a for the backward message of the product factor $f(a, b, z) = \delta(z - ab)$. The resulting marginal approximation would be useful for deriving a factor-to-variable message approximation of $m_{f\to a}(a)$ with better guarantees compared to the approximation provided by Stern et al. [SHG09].

We start from the reverse KL-divergence between the approximation q and the true marginal p:

$$\begin{aligned}
\mathrm{KL}(q \parallel p) &= \int_{-\infty}^{\infty} q(a) \ln\left(\frac{q(a)}{p(a)}\right) \mathrm{d}a \\
&= \int_{-\infty}^{\infty} q(a) \ln q(a) \,\mathrm{d}a - \int_{-\infty}^{\infty} q(a) \ln p(a) \,\mathrm{d}a \\
&= \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2) \ln \mathcal{N}(a; \mu, \sigma^2) \,\mathrm{d}a - \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2) \ln p(a) \,\mathrm{d}a \\
&= \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2) \left[-\frac{1}{2}\ln(2\pi\sigma^2) - \frac{(a-\mu)^2}{2\sigma^2}\right] \mathrm{d}a - \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2) \ln p(a) \,\mathrm{d}a \\
&= -\frac{1}{2}\ln(2\pi\sigma^2) \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2) \,\mathrm{d}a - \frac{1}{2\sigma^2} \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2)(a-\mu)^2 \,\mathrm{d}a \\
&\quad - \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2) \ln p(a) \,\mathrm{d}a \\
&= -\frac{1}{2}\ln(2\pi\sigma^2) \cdot 1 - \frac{1}{2\sigma^2} \cdot \sigma^2 - \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2) \ln p(a) \,\mathrm{d}a \\
&= -\frac{1}{2}\ln(2\pi\sigma^2) - \frac{1}{2} - \int_{-\infty}^{\infty} \mathcal{N}(a; \mu, \sigma^2) \ln p(a) \,\mathrm{d}a
\end{aligned}$$

At this point we did not continue the derivation due to time constraints. Further steps could be to apply the arg min with respect to the parameters μ, σ^2 of the Gaussian approximation and then solve for μ and σ^2 individually after setting the first derivative of the term to zero.

A practical derivation. With the not yet completed marginal approximation in mind, we will derive a sufficiently good Gaussian approximation of the product factor's backward message $m_{f\to a}(a)$ for practical purposes. Even though we do not try to explicitly minimize a divergence criterion, our result matches the approximation from Stern et al. [SHG09] that minimizes the reverse KL divergence $\mathrm{KL}\left(\mathcal{N}(a; \mu_a^*, \sigma_a^{2*}) \parallel m_{f\to a}(a)\right)$. First, we marginalize out b and z using the factor-to-variable message definition from message passing 2.5:

$$m_{f\to a}(a) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(z - ab)\, m_{b\to f}(b)\, m_{z\to f}(z) \,\mathrm{d}b \,\mathrm{d}z \tag{16}$$

Next, we can use the sifting property (see Equation 5) of the delta function $\delta(z - ab)$, which allows us to pick out the point where $z = ab$ from the integral over z, leaving us with only the integral over b:

$$m_{f\to a}(a) = \int_{-\infty}^{\infty} m_{b\to f}(b)\, m_{z\to f}(ab) \,\mathrm{d}b \tag{17}$$

Writing the approximation in its natural form with canonical parameters $\frac{\mu_z}{\sigma_z^2}, \frac{1}{\sigma_z^2}$ and sufficient statistics $z = ab, -\frac{z^2}{2} = -\frac{a^2b^2}{2}$, we get:

$$
\begin{aligned}
m_{f\to a}(a) &= \int_{-\infty}^{\infty} m_{b\to f}(b)\, \exp\left(\frac{\mu_z}{\sigma_z^2} z - \frac{1}{2\sigma_z^2} z^2\right) \mathrm{d}b \\
&= \int_{-\infty}^{\infty} m_{b\to f}(b)\, \exp\left(\frac{\mu_z}{\sigma_z^2} ab - \frac{1}{2\sigma_z^2} a^2 b^2\right) \mathrm{d}b
\end{aligned}
\tag{18}
$$

Now, we introduce an approximation and replace the functions of b by their expected values. The approximated message is a constant function that uses the first two moments of b:

$$
\begin{aligned}
m_{f\to a}(a) &= \int_{-\infty}^{\infty} m_{b\to f}(b)\, \exp\left(\frac{\mu_z}{\sigma_z^2} ab - \frac{1}{2\sigma_z^2} a^2 b^2\right) \mathrm{d}b \\
&= \mathbb{E}_{b\sim m_{b\to f}(\cdot)}\left[\exp\left(\frac{\mu_z}{\sigma_z^2} ab - \frac{1}{2\sigma_z^2} a^2 b^2\right)\right] \\
&\approx \exp\left(\frac{\mu_z}{\sigma_z^2} a\, \mathbb{E}[b] - \frac{1}{2\sigma_z^2} a^2\, \mathbb{E}[b^2]\right) \\
&= \exp\left(\frac{\mu_z\, \mathbb{E}[b]}{\sigma_z^2} a - \frac{\mathbb{E}[b^2]}{2\sigma_z^2} a^2\right)
\end{aligned}
\tag{19}
$$

The idea behind this approximation is similar to moment-matching (we use the first two moments to approximate the interval over b), however the two are not equivalent. Our approximation will result in the same form as in Stern et al. [SHG09] and they say that their message minimizes the reverse KL divergence. Therefore this can not be equivalent to moment-matching, which minimizes the forward KL divergence. We note that the soundness of our approximation is arguable, but it works well in practice, as shown in Figure 5.3.

The above form is an unnormalized Gaussian approximation $\hat{m}_{f\to a}(a) \propto \exp\left(\lambda a - \frac{\lambda'}{2} a^2\right)$ of the message $m_{f\to a}(a)$. Thus, we can simply retrieve its natural parameters:

$$
\begin{aligned}
\lambda' &= \frac{1}{\sigma_a^{2*}} = \frac{\mathbb{E}[b^2]}{\sigma_z^2} \implies \sigma_a^{2*} = \frac{\sigma_z^2}{\mathbb{E}[b^2]} \\
\lambda &= \frac{\mu_a^*}{\sigma_a^{2*}} = \frac{\mu_z\, \mathbb{E}[b]}{\sigma_z^2} \implies \mu_a^* = \frac{\mu_z\, \mathbb{E}[b]\, \sigma_a^{2*}}{\sigma_z^2} = \frac{\mu_z\, \mathbb{E}[b]}{\mathbb{E}[b^2]}
\end{aligned}
\tag{20}
$$

Putting the parameters together and using the backpropagated message $m_{z\to f}(z) = \mathcal{N}(z; \mu_z, \sigma_z^2)$, we arrive at the Gaussian backward message approximation presented in Stern et al. [SHG09]:

$$
m_{f\to a}(a) = \mathcal{N}\left(a; \frac{\mathbb{E}[m_{z\to f}]\, \mathbb{E}[b]}{\mathbb{E}[b^2]}, \frac{\mathbb{V}[m_{z\to f}]}{\mathbb{E}[b^2]}\right). \tag{21}
$$

A.7 Factor to Variable Messages

Starting from the variable-to-factor message equation (as described in the message-passing algorithm in Equation 2.6):

$$
m_{x\to f_i}(x) = \prod_{f\in \mathrm{Ne}(x)\setminus\{f_i\}} m_{f\to x}(x) \tag{22}
$$

Also recall that the marginal distribution for the variable x is given by the product of all incoming messages (see Equation 2.4):

$$
p(x) = \prod_{f\in \mathrm{Ne}(x)} m_{f\to x}(x) \tag{23}
$$

Since the neighborhood $\mathrm{Ne}(x)$ can be partitioned as $\{f_i\} \cup (\mathrm{Ne}(x) \setminus \{f_i\})$, we can rewrite the

product as:

$$p(x) = m_{f_i \to x}(x) \cdot \underbrace{\prod_{f \in \mathrm{Ne}(x) \setminus \{f_i\}} m_{f \to x}(x)}_{m_{x \to f_i}(x), \text{ see Equation 22}} = m_{f_i \to x}(x) \cdot m_{x \to f_i}(x) \tag{24}$$

Solving for $m_{f_i \to x}(x)$, we arrive at the equation for factor-to-variable messages:

$$m_{f_i \to x}(x) = \frac{p(x)}{m_{x \to f_i}(x)} \tag{25}$$

Bibliography

[Jen06] J. L. W. V. Jensen. "Sur les fonctions convexes et les inégalités entre les valeurs moyennes". In: *Acta Mathematica* 30 (1906), pp. 175–193. URL: https://api.semanticscholar.org/CorpusID:120669169.

[KL51] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[Tay52] A. E. Taylor. "L'Hospital's Rule". In: *The American Mathematical Monthly* 59.1 (1952), pp. 20–24. DOI: 10.2307/2307183. URL: https://doi.org/10.2307/2307183.

[Ros62] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Spartan Books, 1962. URL: https://books.google.de/books?id=7FhRAAAAMAAJ.

[Har67] Kenneth Harkness. *Official Chess Handbook*. New edition; with contributions from the United States Chess Federation. D. McKay Company, 1967.

[ES78] Arpad E Elo and Sam Sloan. *The rating of chessplayers: Past and present*. Batsford, 1978. ISBN: 0668047216.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536. URL: https://api.semanticscholar.org/CorpusID:205001834.

[Dav88] H. A. David. *The Method of Paired Comparisons*. 2nd, revised. London and New York: C. Griffin and Oxford University Press, 1988. ISBN: 978-0852642900.

[Wes93] M. West. "Approximate posterior distributions by mixture". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 55.2 (1993), pp. 409–422.

[Moo96] T.K. Moon. "The expectation-maximization algorithm". In: *IEEE Signal Processing Magazine* 13.6 (1996), pp. 47–60. DOI: 10.1109/79.543975.

[Fre+97] Brendan J Frey et al. "Factor graphs and algorithms". In: *Proceedings of the Annual Allerton Conference on Communication Control and Computing*. Vol. 35. Citeseer. 1997, pp. 666–680.

[Lec+98] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[Gli99] Mark E Glickman. "Parameter estimation in large dynamic paired comparison experiments". In: *Journal of the Royal Statistical Society Series C: Applied Statistics* 48.3 (1999), pp. 377–394.

[RQD00] Douglas A Reynolds, Thomas F Quatieri, and Robert B Dunn. "Speaker verification using adapted Gaussian mixture models". In: *Digital signal processing* 10.1-3 (2000), pp. 19–41.

[KFL01] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. "Factor graphs and the sum-product algorithm". In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 498–519. DOI: 10.1109/18.910572.

[Min01] Thomas P. Minka. "A Family of Algorithms for Approximate Bayesian Inference". PhD Thesis. Cambridge, MA: Massachusetts Institute of Technology, Jan. 2001.

[Hes02] Tom Heskes. "Stable fixed points of loopy belief propagation are local minima of the bethe free energy". In: *Advances in neural information processing systems* 15 (2002).

[SZ04] Katie Salen and Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. MIT Press, 2004.

[Her05] Ralf Herbrich. "Minimising the Kullback-Leibler Divergence". In: 2005. URL: https://api.semanticscholar.org/CorpusID:124661640.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[Ama07] Shun-ichi Amari. "Integration of stochastic models by minimizing alpha-divergence". In: *Neural Computation* 19.10 (2007), pp. 2780–2796. DOI: 10.1162/neco.2007.19.10.2780.

[Dan+07] Pierre Dangauthier et al. "TrueSkill Through Time: Revisiting the History of Chess". In: *Advances in Neural Information Processing Systems 20*. The MIT Press, 2007, pp. 931–938.

[HMG07] Ralf Herbrich, Tom Minka, and Thore Graepel. "TrueSkill(TM): A Bayesian Skill Rating System". In: *Advances in Neural Information Processing Systems 20*. MIT Press, Jan. 2007, pp. 569–576. URL: https://www.microsoft.com/en-us/research/publication/trueskilltm-a-bayesian-skill-rating-system/.

[Loe+07] Hans-Andrea Loeliger et al. "The factor graph approach to model-based signal processing". In: *Proceedings of the IEEE* 95.6 (2007), pp. 1295–1322.

[Run07] A. R. Runnalls. "Kullback-Leibler approach to Gaussian mixture reduction". In: *IEEE Transactions on Aerospace and Electronic Systems* 43.3 (2007), pp. 989–999.

[SH09] Dennis Schieferdecker and Marco F. Huber. "Gaussian mixture reduction via clustering". In: *2009 12th International Conference on Information Fusion*. 2009, pp. 1536–1543.

[SCI09] Huiying Shen, James Coughlan, and Volodymyr Ivanchenko. "Figure-ground segmentation using factor graphs". In: *Image and Vision Computing* 27.7 (2009), pp. 854–863.

[SHG09] David H. Stern, Ralf Herbrich, and Thore Graepel. "Matchbox: large scale online bayesian recommendations". In: *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. Madrid, Spain: Association for Computing Machinery, 2009, pp. 111–120. ISBN: 9781605584874. DOI: 10.1145/1526709.1526725. URL: https://doi.org/10.1145/1526709.1526725.

[ZJ09] Lei Zhang and Qiang Ji. "Image segmentation with a unified graphical model". In: *IEEE transactions on pattern analysis and machine intelligence* 32.8 (2009), pp. 1406–1425.

[Mos10] Jeff Moser. "The Math Behind TrueSkill". In: (2010). Last accessed on 25.01.2025. Copyright 2010, Jeff Moser. URL: https://www.moserware.com/assets/computing-your-skill/The%20Math%20Behind%20TrueSkill.pdf.

[Cro+11] David F. Crouse et al. "A look at Gaussian mixture reduction algorithms". In: *14th International Conference on Information Fusion*. 2011, pp. 1–8. URL: https://ieeexplore.ieee.org/document/5977695.

[JJD11] Jorge Jiménez-Rodrıguez, Guillermo Jiménez-Dıaz, and Belén Dıaz-Agudo. "Matchmaking and case-based recommendations". In: *19th international conference on case based reasoning*. 2011.

[Sfe11] Nicolae Sfetcu. *The game of chess*. Morrisville, NC: Lulu.com, Sept. 2011.

[EMK12] Gal Elidan, Ian McGraw, and Daphne Koller. "Residual belief propagation: Informed scheduling for asynchronous message passing". In: *arXiv preprint arXiv:1206.6837* (2012).

[GVP13] Dan Geiger, Tom S. Verma, and Judea Pearl. *d-Separation: From Theorems to Algorithms*. 2013. arXiv: 1304.1505 [cs.AI]. URL: https://arxiv.org/abs/1304.1505.

[Min13] Thomas P. Minka. *Expectation Propagation for approximate Bayesian inference*. 2013. arXiv: 1301.2294 [cs.AI]. URL: https://arxiv.org/abs/1301.2294.

[HJM16] Ella Horton, Daniel Johnson, and Jo Mitchell. "Finding and building connections: moving beyond skill- based matchmaking in videogames". In: *Proceedings of the 28th Australian Conference on Computer-Human Interaction - OzCHI '16*. OzCHI '16. ACM Press, 2016, pp. 656–658. DOI: 10.1145/3010915.3011857. URL: http://dx.doi.org/10.1145/3010915.3011857.

[BKM17] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. "Variational Inference: A Review for Statisticians". In: *Journal of the American Statistical Association* 112.518 (Apr. 2017), pp. 859–877. ISSN: 1537-274X. DOI: 10.1080/01621459.2017.1285773. URL: http://dx.doi.org/10.1080/01621459.2017.1285773.

[DK+17] Frank Dellaert, Michael Kaess, et al. "Factor graphs for robot perception". In: *Foundations and Trends® in Robotics* 6.1-2 (2017), pp. 1–139.

[Guo+17] Chuan Guo et al. "On calibration of modern neural networks". In: *International conference on machine learning*. PMLR. 2017, pp. 1321–1330.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[Rud17] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG]. URL: https://arxiv.org/abs/1609.04747.

[VJS17] Aleksi Visti, Tapani N. Joelsson, and Jouni Smed. "Beyond skill-based rating systems: analyzing and evaluating player performance". In: *Proceedings of the 21st International Academic Mindtrek Conference*. AcademicMindtrek'17. ACM, Sept. 2017. DOI: 10.1145/3131085.3131096. URL: http://dx.doi.org/10.1145/3131085.3131096.

[Gar+18] Timur Garipov et al. *Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs*. 2018. arXiv: 1802.10026 [stat.ML]. URL: https://arxiv.org/abs/1802.10026.

[He+18] Yong He et al. "Object recognition in images via a factor graph model". In: *Ninth International Conference on Graphic and Image Processing (ICGIP 2017)*. Vol. 10615. SPIE. 2018, pp. 326–335.

[MCZ18] Tom Minka, Ryan Cleven, and Yordan Zaykov. "Trueskill 2: An improved bayesian skill rating system". In: *Technical Report* (2018).

[GAB19] Jianxiong Gao, Zongwen An, and Xuezong Bai. "A new representation method for probability distributions of multimodal and irregular data based on uniform mixture model". In: *Annals of Operations Research* 311.1 (Apr. 2019), pp. 81–97. ISSN: 1572-9338. DOI: 10.1007/s10479-019-03236-9. URL: http://dx.doi.org/10.1007/s10479-019-03236-9.

[LH19] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG]. URL: https://arxiv.org/abs/1711.05101.

[MZ19] Tianle Ma and Aidong Zhang. "Incorporating biological knowledge with factor graph neural network for interpretable deep learning". In: *arXiv preprint arXiv:1906.00537* (2019).

[Erl20] Lukas Erlenbach. "Active Learning for Bayesian Neural Networks with Gaussian Processes". Advisor: Prof. Dr. Jochen Garcke; Second Advisor: Prof. Dr. Martin Rumpf. Born 01.08.1994 in Nastätten. Master's Thesis in Mathematics. Universität Bonn, Institut für Numerische Simulation, Oct. 2020. URL: https://ins.uni-bonn.de/media/public/publication-media/Masterthesis_Lukas_Erlenbach.pdf (visited on 02/09/2025).

[Hal20] Evan Hallmark. *A Large Tennis Dataset for ATP and ITF Betting*. Accessed: 2025-03-01. 2020. URL: https://www.kaggle.com/datasets/ehallmar/a-large-tennis-dataset-for-atp-and-itf-betting.

[MBW20] Wesley J. Maddox, Gregory Benton, and Andrew Gordon Wilson. *Rethinking Parameter Counting in Deep Models: Effective Dimensionality Revisited*. 2020. arXiv: 2003.02139 [cs.LG]. URL: https://arxiv.org/abs/2003.02139.

[Qu+20] Jiahui Qu et al. "Anomaly detection in hyperspectral imagery based on Gaussian mixture model". In: *IEEE Transactions on Geoscience and Remote Sensing* 59.11 (2020), pp. 9504–9517.

[Shi20] Minyong Shin. *League of Legends(LOL) - Ranked Games 2020*. 2020. URL: https://www.kaggle.com/datasets/gyejr95/league-of-legendslol-ranked-games-2020-ver1/data.

[ZWL20] Zhen Zhang, Fan Wu, and Wee Sun Lee. "Factor graph neural networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 8577–8587.

[For+21] Pierre Foret et al. *Sharpness-Aware Minimization for Efficiently Improving Generalization*. 2021. arXiv: 2010.01412 [cs.LG]. URL: https://arxiv.org/abs/2010.01412.

[Min+21] Matthias Minderer et al. "Revisiting the Calibration of Modern Neural Networks". In: *CoRR* abs/2106.07998 (2021). arXiv: 2106.07998. URL: https://arxiv.org/abs/2106.07998.

[Gli22] Mark E. Glickman. *Example of the Glicko-2 system*. Accessed: 2025-02-24. Mar. 2022. URL: https://www.glicko.net/glicko/glicko2.pdf.

[Mur23] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. URL: http://probml.github.io/book2.

[Vas+23] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[Ada+24] Janina Adamcic et al. *Bayesian Neural Networks*. Technical Report. Hasso Plattner Institute, Chair for Artificial Intelligence and Sustainability, July 2024.

[Chi+24] Wei-Lin Chiang et al. *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*. 2024. arXiv: 2403.04132 [cs.AI].

[CW24] Yoni Choukroun and Lior Wolf. "Factor Graph Optimization of Error-Correcting Codes for Belief Propagation Decoding". In: *arXiv preprint arXiv:2406.12900* (2024).

[Her24] Ralf Herbrich. *Introduction to Probabilistic Machine Learning - Unit 5: Bayesian Ranking*. 2024.

[Hu+24] Zhengyu Hu et al. *Explaining Length Bias in LLM-Based Preference Evaluations*. 2024. arXiv: 2407.01085 [cs.LG]. URL: https://arxiv.org/abs/2407.01085.

[LWH24] Jianan Li, Yichen Wu, and Kunhua Huang. "A Study of Tennis Match Momentum Based on Machine Learning and the Trueskill-H Algorithm". In: *2024 2nd International Conference on Mechatronics, IoT and Industrial Informatics (ICMIII)*. 2024, pp. 292–297. DOI: 10.1109/ICMIII62623.2024.00060.

[She+24] Yuesong Shen et al. *Variational Learning is Effective for Large Deep Networks*. 2024. arXiv: 2402.17641 [cs.LG]. URL: https://arxiv.org/abs/2402.17641.

[Som24] Romeo Sommerfeld. "Scalable Approximate Message Passing for Bayesian Neural Networks". Skalierbares Approximatives Message Passing für Bayes'sche Neuronale Netze. Submitted on October 20, 2024. Master Thesis. Universität Potsdam, Oct. 2024.

[Yuk24] Cem Yuksel. "Skill-Based Matchmaking for Competitive Two-Player Games". In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7.1 (May 2024), pp. 1–19. ISSN: 2577-6193. DOI: 10.1145/3651303. URL: http://dx.doi.org/10.1145/3651303.

[ATP25] ATP Tour. *ATP Tour - Tennis Players, Rankings, and Tournaments*. Accessed: 2025-03-03. 2025. URL: https://www.atptour.com/en.

[Lan25] Gustavo Landfried. *TrueSkillThroughTime.jl - Examples*. Accessed: 2025-03-03. 2025. URL: https://glandfried.github.io/TrueSkillThroughTime.jl/man/examples/.

[SHH25] Romeo Sommerfeld, Christian Helms, and Ralf Herbrich. *Approximate Message Passing for Bayesian Neural Networks*. 2025. arXiv: 2501.15573 [cs.LG]. URL: https://arxiv.org/abs/2501.15573.

[Ten25] Association of Tennis Professionals. *ATP World Tour Tennis Data*. https://github.com/datasets/atp-world-tour-tennis-data. Accessed: 2025-02-28. 2025.

[ZR25] Lily H. Zhang and Rajesh Ranganath. *Preference learning made easy: Everything should be understood through win rate*. 2025. arXiv: 2502.10505 [cs.LG]. URL: https://arxiv.org/abs/2502.10505.

[Chi+] Wei-Lin Chiang et al. *LLMArena Platform*. https://lmarena.ai/. Accessed: 2025-03-01.

[SFH] Oliver C. Schrempf, Olga Feiermann, and Uwe D. Hanebeck. *Optimal Mixture Approximation of the Product of Mixtures*. URL: https://isas.iar.kit.edu/pdf/Fusion05_Schrempf-MixtureReduction.pdf.

[US nd] US Chess. *US Chess Rule Book: Online Only Edition*. Accessed: 2025-02-24. n.d. URL: https://new.uschess.org/sites/default/files/media/documents/us-chess-rule-book-online-only-edition-chapters-1-2-10-11-9-1-20.pdf.