SSY316: Take-Home Exam

Benjamin Pettersson
Carl Odqvist
-
Group 23

January 10, 2026

P1. Skill Estimation in Competitive Games with TrueSkill

1.1 Model Formulation

In a match between two players 1 and 2. Each team i has a random variable $s_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ representing the skill of a team, there are 20 teams. The outcome of a match between two players also has a Gaussian random variable $t \sim \mathcal{N}(s_1 - s_2, \sigma_t^2)$. The binary variable y for the result of the match is $y = 1$ if $t > 0$ (player 1 wins). Consequently, $y = -1$ if $t \leq 0$ (player 2 wins).

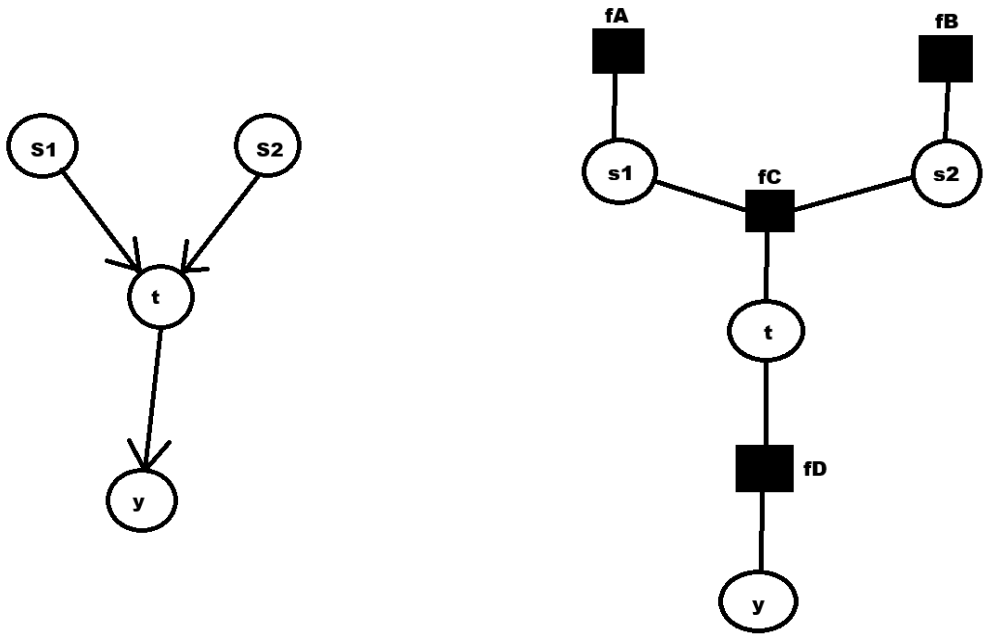
$$p(y|t) = \begin{cases} 1 & \text{if } y = 1 \text{ and } t > 0 \\ 1 & \text{if } y = -1 \text{ and } t \leq 0 \\ 0 & \text{otherwise.} \end{cases} = \begin{cases} 1 & \text{if } y > 0 \text{ and } t > 0 \\ 1 & \text{if } y \leq 0 \text{ and } t \leq 0 \\ 0 & \text{otherwise.} \end{cases} = \mathbb{I}\{y \cdot t > 0\} \text{ , meaning 100}$$

The joint probability is: $p(s_1, s_2, t, y) = p(s_1) \cdot p(s_2) \cdot p(t \mid s_1, s_2) \cdot p(y \mid t) =$

$= \mathcal{N}(s_1, \sigma_1^2) \cdot \mathcal{N}(s_2, \sigma_2^2) \cdot \mathcal{N}(s_1 - s_2, \sigma_t^2) \cdot \mathbb{I}\{y \cdot t > 0\}$

y is just the outcome, not a hyperparameter. The five hyper paramters are μ_1, μ_2, σ_1^2, σ_2^2 and σ_t^2.

1.2 Bayesian Network and Factor Graph



(a) Bayesian network (b) Factor graph

Figure 1: Bayesian network and factor graph.

The first graph is the Bayesian network for $p(s_1, s_2, t, y) = p(s_1) \cdot p(s_2) \cdot p(t \mid s_1, s_2) \cdot p(y \mid t)$

The second graph is the factor graph where: $f_A = p(s_1)$, $f_B = p(s_2)$, $f_C = p(t \mid s_1, s_2)$, $f_D = p(y \mid t)$

1.3 Conditional Probabilities and Posterior

- $p(s_1, s_2 \mid t, y) = p(s_1, s_2 \mid t) \propto p(t \mid s_1, s_2)P(s_1)P(s_2)$, these terms are known.
- $p(t \mid y, s_1, s_2) = \frac{p(y|t,s_1,s_2)p(t|s_1,s_2)}{p(y|s_1,s_2)} \propto p(y \mid t, s_1, s_2)p(t \mid s_1, s_2) = p(y \mid t)p(t \mid s_1, s_2)$, these terms are also known. Here $p(y \mid t)$ will truncate the distribution of $p(t \mid s_1, s_2)$ to not give any probability of t not matching y.
- The probability of player 1 winning ($p(y = 1)$) can be calcualted by the cumulative probability of t being above zero. Since $s_1 - s_2 \sim \mathcal{N}(\mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2)$. Thus, $t \sim \mathcal{N}(s_1 - s_2, \sigma_t^2) = \mathcal{N}(\mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2 + \sigma_t^2)$ since the variance of σ_t^2 is just added.

Regarding the truncation mentioned in the second point, to make the new distribution a probability distribution it has to be scaled. This is done by dividing the distribution by the cumulative distribution of t matching the y. Otherwise the distribution would just be a cut of gaussian distribution where the integral of the entire distribution would not be equal to one.

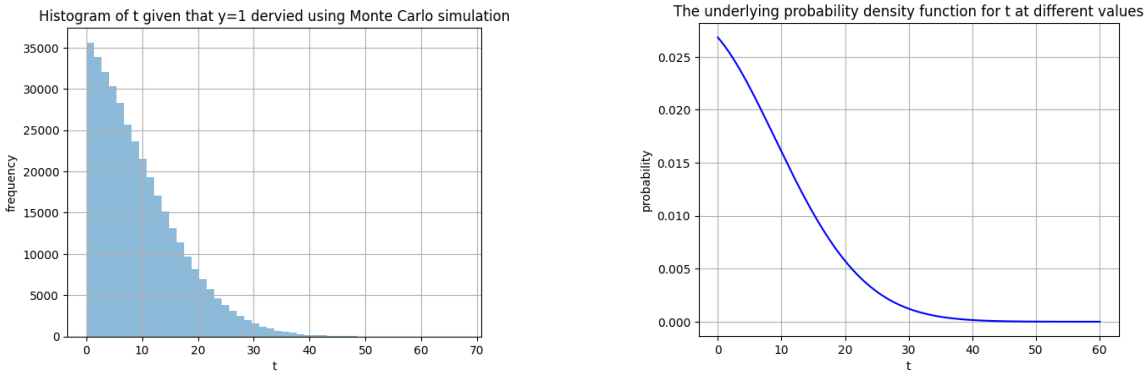
$$p(t \mid y, s_1, s_2) = \begin{cases} \frac{\mathcal{N}(t; s_1 - s_2, \sigma_t^2)}{p(t>0|s_1,s_2)} & \text{if } y = 1 \text{ and } t > 0 \\ \frac{\mathcal{N}(t; s_1 - s_2, \sigma_t^2)}{p(t\leq 0|s_1,s_2)} & \text{if } y = -1 \text{ and } t \leq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The code gives us results that validate the formulas:

Monte Carlo win rate of player 1:
36.0371%

Win rate of player 1 from derived
formula: 35.9958%

Since we only compare the histograms of the Monte Carlo simulation with the underlying distribution, it is sufficient to plot a distribution proportional to the underlying one. Thus avoids the need to normalize the distribution.



(a) Histogram of values for t.

(b) The distribution derived with the formula.

Figure 2: Comparing the distribution of t with the histogram.

These figures seem to validate the formulas for the posterior distributions.

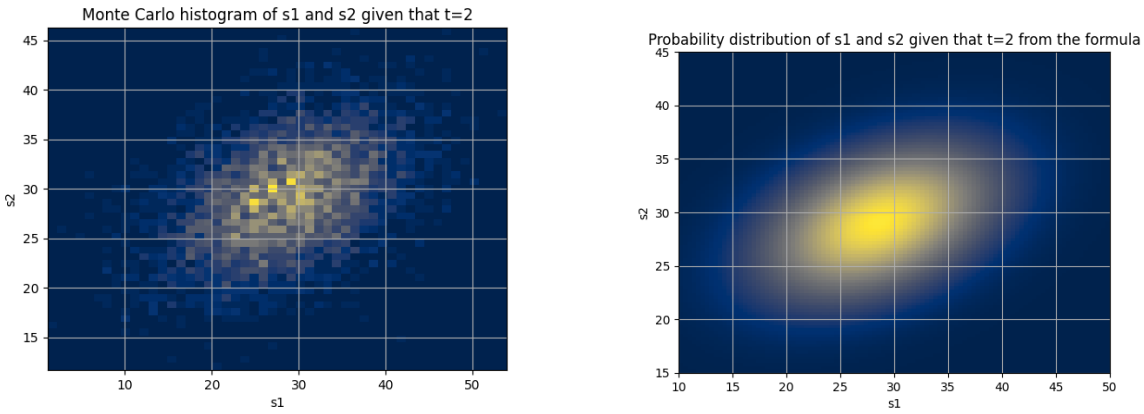


Figure 3: Comparing the distribution of t with the histogram.

1.4 Gibbs Sampling

Sampling s_1 According to the formula for the normal distribution, it is proportional to function on the following form. $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \propto \exp\left(-\frac{1}{2}\left(\frac{x^2-2x\mu+\mu^2}{\sigma^2}\right)\right) = \exp\left(-\frac{1}{2}\left(\frac{x^2}{\sigma^2} - \frac{2x\mu}{\sigma^2} + \frac{\mu^2}{\sigma^2}\right)\right) = \exp\left(-\frac{1}{2}\left(x^2\frac{1}{\sigma^2} - 2x\frac{\mu}{\sigma^2} + \frac{\mu^2}{\sigma^2}\right)\right) = (*)$

$$p(s_1|t,s_2) \propto p(t|s_1,s_2)p(s_1) \propto \exp\left(\frac{-(t-(s_1-s_2))^2}{2\sigma_t^2}\right)\cdot\exp\left(\frac{-(s_1-\mu_1)^2}{2\sigma_1^2}\right) = \exp\left(-\frac{1}{2}\left(\frac{t^2-2t(s_1-s_2)+(s_1-s_2)^2}{\sigma_t^2} + \frac{s_1^2-2s_1\mu_1+\mu_1^2}{\sigma_1^2}\right)\right)$$
$$= \exp\left(-\frac{1}{2}\left(\frac{t^2-2ts_1+2ts_2+s_1^2-2s_1s_2+s_2^2}{\sigma_t^2} + \frac{s_1^2-2s_1\mu_1+\mu_1^2}{\sigma_1^2}\right)\right) = \exp\left(-\frac{1}{2}\left(\frac{-2ts_1+s_1^2-2s_1s_2}{\sigma_t^2} + \frac{s_1^2-2s_1\mu_1}{\sigma_1^2} + \text{constants}\right)\right)$$
$$= \exp\left(-\frac{1}{2}\left(s_1^2\left(\frac{1}{\sigma_t^2}+\frac{1}{\sigma_1^2}\right) - 2s_1\left(\frac{t+s_2}{\sigma_t^2}+\frac{\mu_1}{\sigma_1^2}\right) + \text{constants}\right)\right)$$

Constants are variables that does not depend on s_1. Now the expression for the joint probability has the same form as (*), thus the posterior distribution can be derived.

$$s_1^2\frac{1}{\sigma_{\text{new}}^2} = s_1^2\left(\frac{1}{\sigma_t^2}+\frac{1}{\sigma_1^2}\right) \implies \frac{1}{\sigma_{\text{new}}^2} = \frac{1}{\sigma_t^2}+\frac{1}{\sigma_1^2} \implies \sigma_{\text{new}}^2 = \left(\frac{1}{\sigma_t^2}+\frac{1}{\sigma_1^2}\right)^{-1}$$

$$-2s_2\frac{\mu_{\text{new}}}{\sigma_{\text{new}}^2} = -2s_2\left(\frac{t+s_2}{\sigma_t^2}+\frac{\mu_1}{\sigma_1^2}\right) \implies \frac{\mu_{\text{new}}}{\sigma_{\text{new}}^2} = \frac{t+s_2}{\sigma_t^2}+\frac{\mu_1}{\sigma_1^2} \implies \mu_{\text{new}} = \sigma_{\text{new}}^2\left(\frac{t+s_2}{\sigma_t^2}+\frac{\mu_1}{\sigma_1^2}\right)$$

Sampling s_2**.** For sampling s_2 we need the corresponding distribution

$$p(s_2|t,s_1) \propto p(t|s_2,s_1)p(s_2) \propto \exp\left(\frac{-(t-(s_1-s_2))^2}{2\sigma_t^2}\right)\cdot\exp\left(\frac{-(s_2-\mu_2)^2}{2\sigma_2^2}\right) = \exp\left(-\frac{1}{2}\left(\frac{t^2-2t(s_1-s_2)+(s_1-s_2)^2}{\sigma_t^2} + \frac{s_2^2-2s_2\mu_2+\mu_2^2}{\sigma_2^2}\right)\right)$$
$$= \exp\left(-\frac{1}{2}\left(\frac{t^2-2ts_1+2ts_2+s_1^2-2s_1s_2+s_2^2}{\sigma_t^2} + \frac{s_2^2-2s_2\mu_2+\mu_2^2}{\sigma_2^2}\right)\right) = \exp\left(-\frac{1}{2}\left(\frac{2ts_2-2s_1s_2+s_2^2}{\sigma_t^2} + \frac{s_2^2-2s_2\mu_2}{\sigma_2^2} + \text{constants}\right)\right)$$
$$= \exp\left(-\frac{1}{2}\left(s_2^2\left(\frac{1}{\sigma_t^2}+\frac{1}{\sigma_2^2}\right) - 2s_2\left(\frac{s_1-t}{\sigma_t^2}+\frac{\mu_2}{\sigma_2^2}\right) + \text{constants}\right)\right)$$

Constants are variables that does not depend on s_2. As before, the expression for the joint probability has the same form as (*), thus the posterior distribution can be derived.

$$s_2^2\frac{1}{\sigma_{\text{new}}^2} = s_2^2\left(\frac{1}{\sigma_t^2}+\frac{1}{\sigma_2^2}\right) \implies \frac{1}{\sigma_{\text{new}}^2} = \frac{1}{\sigma_t^2}+\frac{1}{\sigma_2^2} \implies \sigma_{\text{new}}^2 = \left(\frac{1}{\sigma_t^2}+\frac{1}{\sigma_2^2}\right)^{-1}$$

$$-2s_2\frac{\mu_{\text{new}}}{\sigma_{\text{new}}^2} = -2s_2\left(\frac{s_1-t}{\sigma_t^2}+\frac{\mu_2}{\sigma_2^2}\right) \implies \frac{\mu_{\text{new}}}{\sigma_{\text{new}}^2} = \frac{s_1-t}{\sigma_t^2}+\frac{\mu_2}{\sigma_2^2} \implies \mu_{\text{new}} = \sigma_{\text{new}}^2\left(\frac{s_1-t}{\sigma_t^2}+\frac{\mu_2}{\sigma_2^2}\right)$$

Sampling t

From 1.3, it is known that given s_1, s_2 and y: $t \sim \mathcal{N}(s_1 - s_2, \sigma_t^2)$ But truncated to only positive or only negative values of t, depending on the provided y.

Priors

The mean values are initially set to 25 since it is in the middle of the range from zero to 50. The standard deviation is set to 25/3 to ensure that it will take three standard deviation to get outside the range. These setting is taken from the Microsoft webpage about TrueSkill.

Result from the implementation

For evaluating the distribution of the posterior $\mu_1 = 20$, $\mu_2 = 30$, $\sigma_1 = \sigma_2 = \sigma_t = \frac{25}{3}$ and $y = 1$.

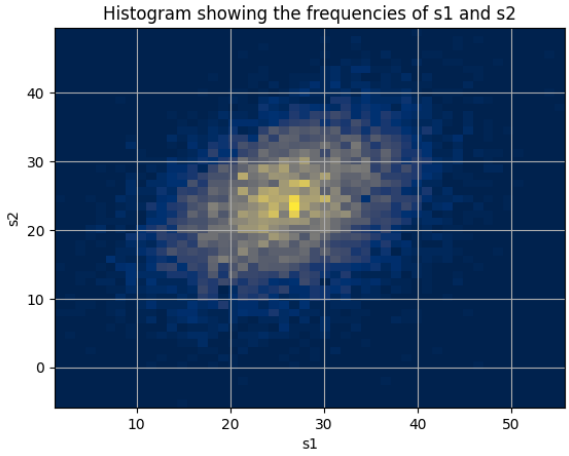


Figure 4: Histogram of samples from the Gibbs sampler

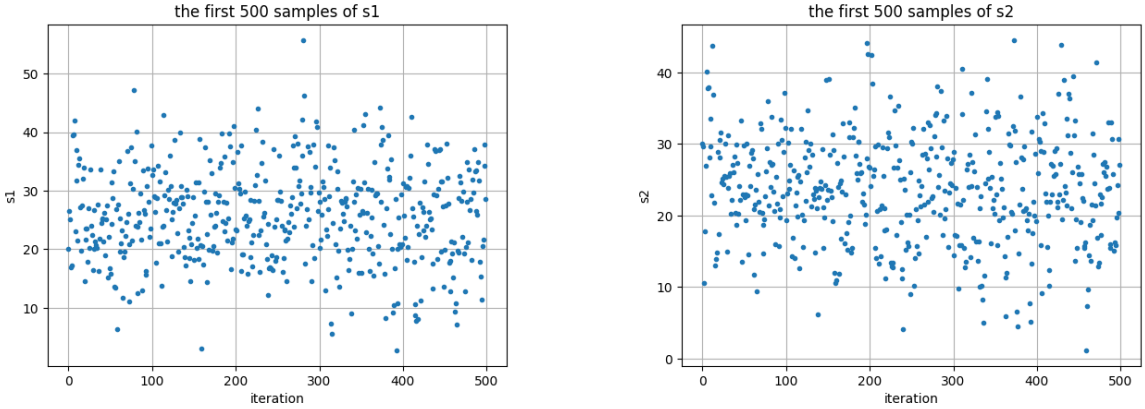


Figure 5: The first 500 samples of s_1 and s_2.

There is no obvious burn-in period. However, the burn-in period is set to 200 in the following plots, just in case there is some convergence at the beginning. The lack of a burn-in period could be due to the posterior mean being close enough to the priors. Thus, the samples converge very quickly in the first few samples, which is why it can be seen clearly in the plots.

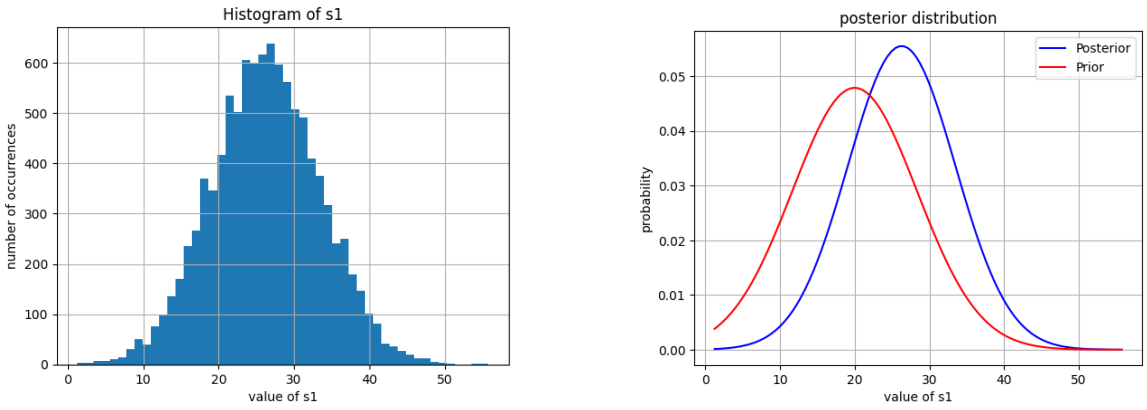


Figure 6: The histogram of the posterior samples and the distributions of s_1.

The samples in the histogram give:

$\mu_1 = 26.2850$, $\mu_2 = 23.8542$, $\sigma_1 = 7.1853$, $\sigma_2 = 7.2623$

showing that after one match where $y = 1$, it now determines player 1 to have a higher mean than player 2. The priors were the opposite.

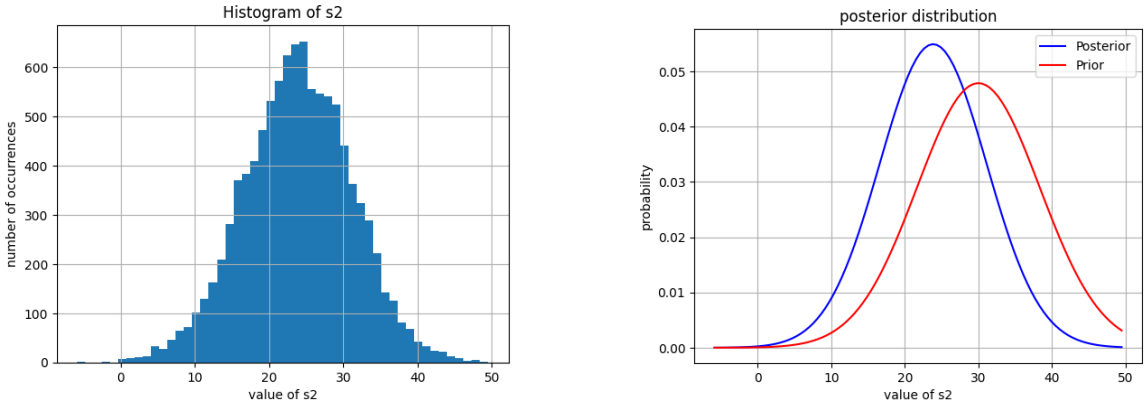


Figure 7: The histogram of the posterior samples and the distributions of s_2.

Trade-offs

The mean and standard deviation are computed for 10 000 samples and used in a normal distribution. This distribution is used to calculate the probability of samples. Then the log likelihood is calculated for different sample sizes which can be seen in Figure 8. But instead of plotting the log likelihood directly, it is divided by the number of samples to produce the average log likelihood. The likelihood was chosen as a measurement of accuracy since it signals how well samples follow a distribution. Setting the samples size to 4 000 seems to be a good trade-off since the samples do not seem to follow the distribution any better for more samples. This provides maximum performance in as little time as possible.

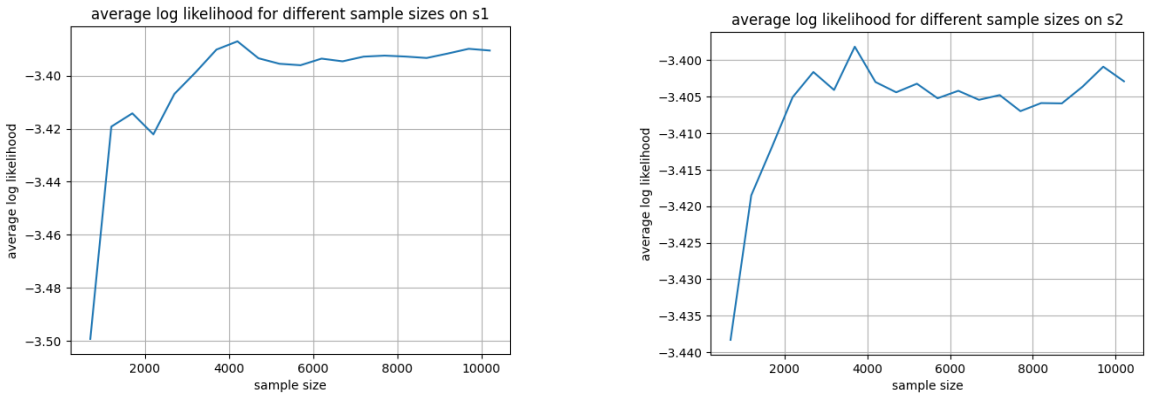
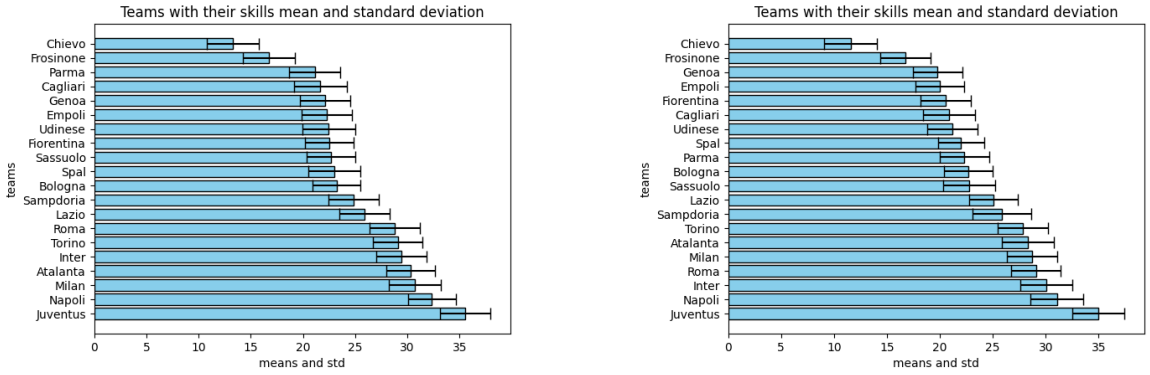


Figure 8: The average log likelihood for the sampled s_1 and s_2.

1.5



(a) No shuffling of the data.

(b) With shuffling of the data.

Figure 9: The ranking of the teams. The bar represents the μ and the interval is the mean plus and minus one standard deviation.

Juventus seems clearly be the best team and Chievo is the worst team, which corresponds the the actual results of Serie A that year. The variance of the skill distribution can be interpreted as how consistent a team's performance is.

The ranking is different depending on whether the data is shuffled or not. This is probably due to the algorithm being sequential and where each match affects the sampling in the next match. For example, a team that over- or underperforms in the first match will have a big influence on the hyperparameters which will then be used throughout the ADF learning. But on the other hand, if the same match is played last, it will have very little influence.

1.6 One-Step-Ahead Predictions

The model starts off with similar performance to random guessing. But after 150 matches, it converges to a prediction rate of 70-75%.

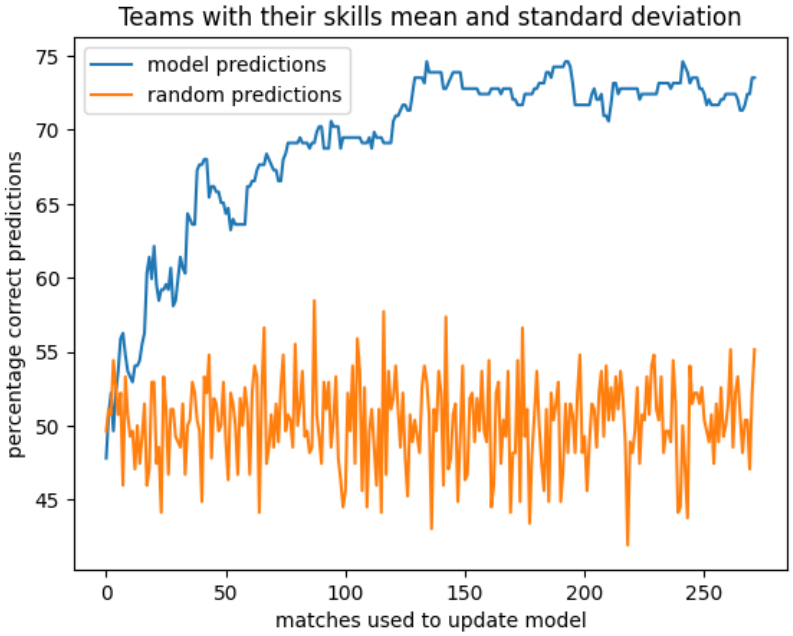


Figure 10: Prediction rate of the model and random guessing

1.8 Custom Dataset

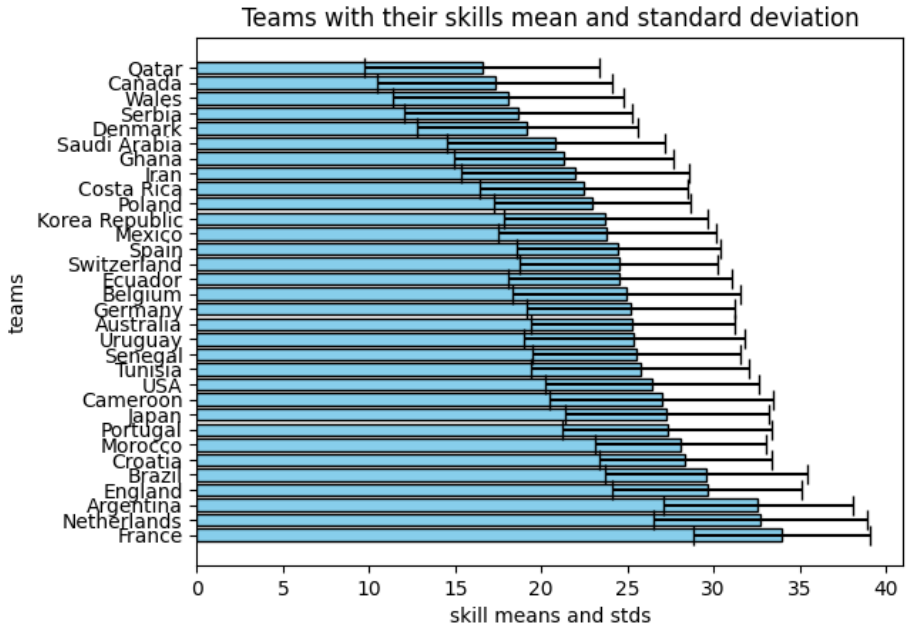


Figure 11: The ranking from the World Cup.

The dataset is all the matches from the football World Cup in 2022. It was downloaded from https://fixturedownload.com/ which provides data about sports competitions. The only preprocessing was to modify the dataloader from exercise 1.5 to convert the dataset into the same format as before with team1, team2 and y as the columns. This makes it possible to reuse the functions from before.

In the real competition, Argentina won over France and this model predicts both of these teams to be in the top 3. Compared to the previous dataset, the World Cup has fewer matches and more teams, so the variance is significantly higher. Also, the further a team progresses in the competition, the more matches they get to play. This could partly explain why the best teams have less variance than the worst teams. But this could also be due to the better teams being more consistent.

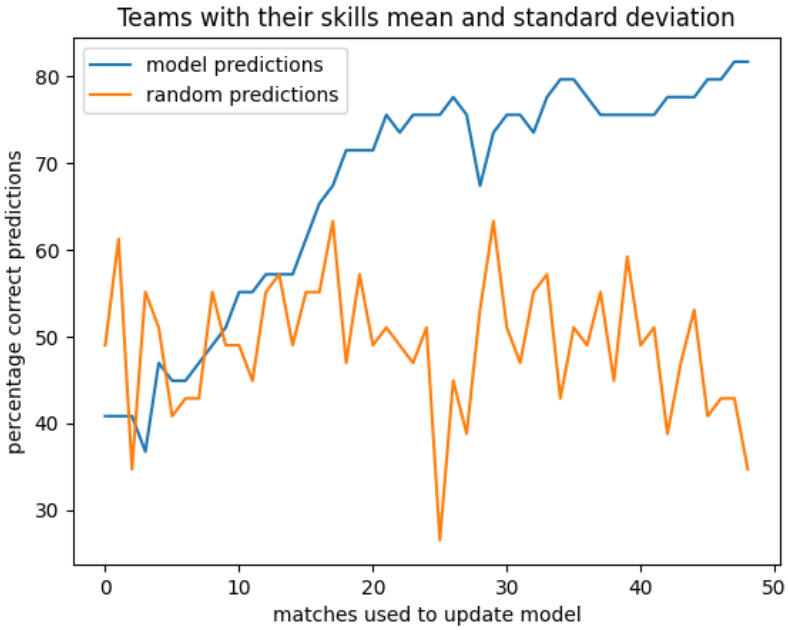


Figure 12: The prediction rate in the World Cup.

We can also see that the model reached a prediction rate of 80% which is significantly better than in previous exercises, even though there are more teams and fewer matches. But one worrying observation is that the prediction rate continuously increases as more data is provided. This means that the model has not converged which was the case on the Serie A dataset.

1.9 Model Extensions

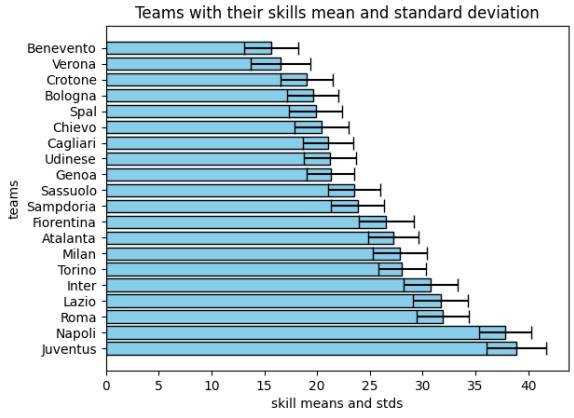
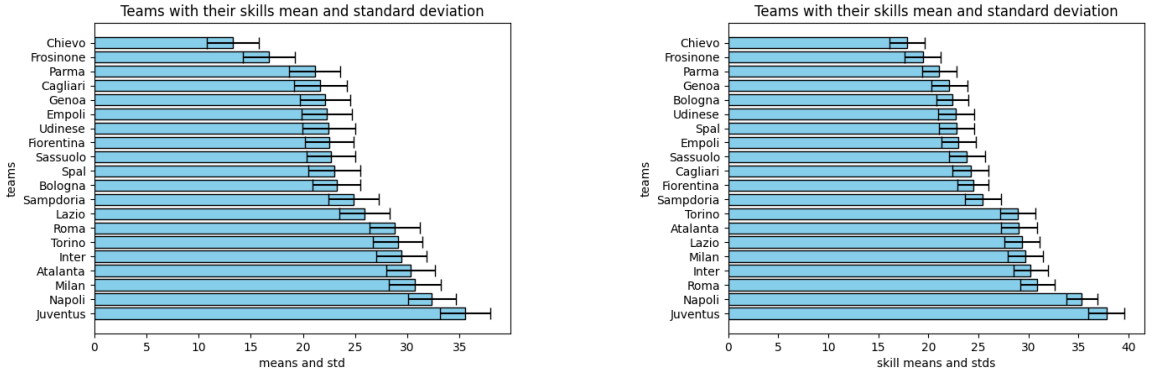


Figure 13: The Ranking from the 2017/2018 season. These are the means and standard deviations used to better predict the next season.

One problem with the original model is that it takes many matches before the predictions are decent. But it is not the first time these teams have played, so it is unreasonable to assume that all teams are equally good at the start of the season. This extension uses data of the previous season (2017/2018) to calculate the priors used for the next season, these values can be seen in Figure 13. But each year the three worst teams in Serie A are replaced by the three best teams from Serie B. Thus there are three new teams in the 2018/2019 season that we do not have prior knowledge of. So for these three teams the priors are initialized as before with $\mu = 25$ and $\sigma = \frac{25}{3}$. This data extension aims to achieve better predictions, especially early in a season. More data should give it better skills estimation. But the computational complexity has not improved since the original model still has to be used as a part of the extension.



(a) Ranking by the original model from exercise 1.5

(b) Ranking with priors derived from the previous season (extension).

Figure 14: The ranking with and without prior information of the 2018/2019 season.

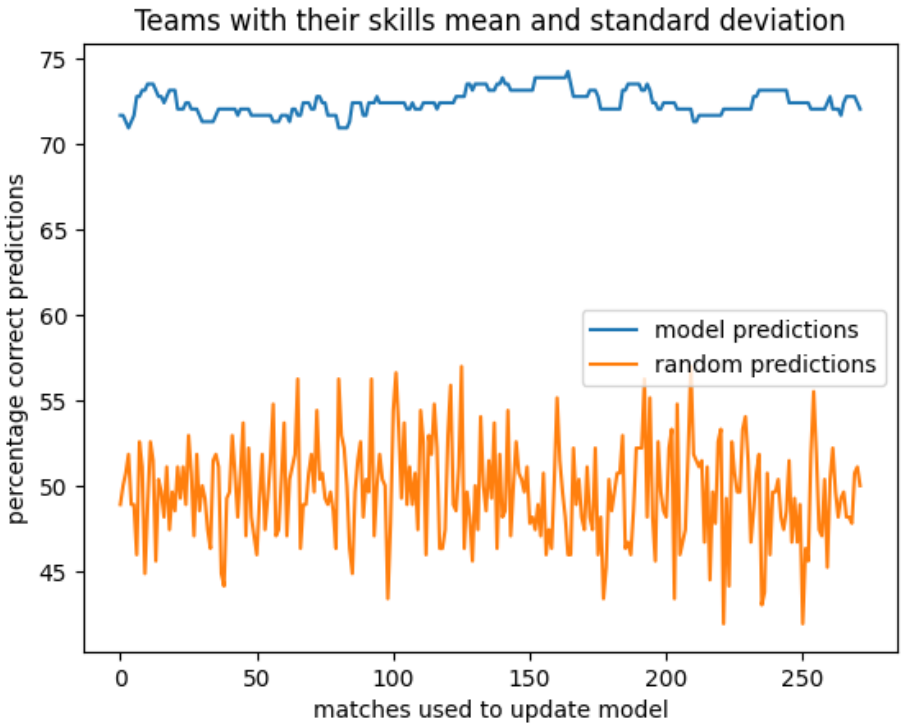


Figure 15: The prediction rate for the 2018/2019 season with the extension.

The extension does improve prediction early in the season dramatically as can be seen in Figure 15, where the prediction rate in the beginning of the season is just as good as at the end of the season. This can be attributed to the teams playing at a similar level as in the previous season. But the prediction rate is just as good as the original model before the extension, see Figure 10. This is a clear disappointment that might indicate that the model has reached its limitations and won't improve with more data or that the outcome of a match cannot be predicted better solely based on which two teams are competing.

However, the two rankings in Figure 14 show that the standard deviation has decreased with the extension. This can be interpreted as the skill estimation has been improved. But since this does not show up as better results in the prediction rate, it can be doubted how well the skill estimation actually is improved.

In summary, the extension uses data from a previous season to derive better priors. It results in good performance immediately in a new season, which is advantageous. But at the end of the season, it does not have any advantage over the original model (which lacks the extension). Also, the extended model has less variance in its skill estimation.

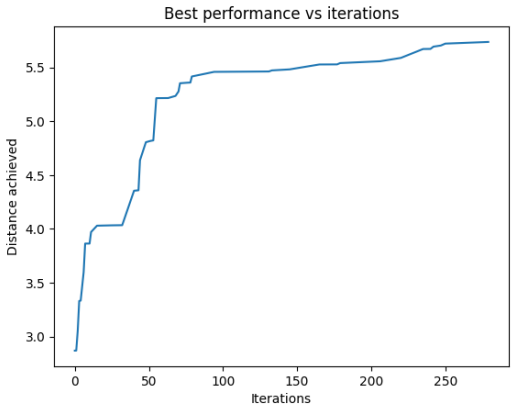
P2. QWOP

Two models have been implemented and evaluated. The first one is a genetic algorithm, which is an evolutionary algorithm. The plans are interpreted as genetics. A gene can mutate (random changes occur) and there can be crossover between two genes, similar to breeding. There is also a selection of the fittest. In the implementation each of the 40 instructions in the plan has a 3% probability of mutating to a random value between -1 and 1. There is a 10% probability that a gene will crossover with another gene. During crossover a random number n is drawn in the interval 1-40, and the first n instructions of the gene is swapped with another gene. There is also a 5% probability that a gene will be replaced by the best gene, which represents the selection. There are in total 50 competing genes, which are randomly assigned in the initialization.

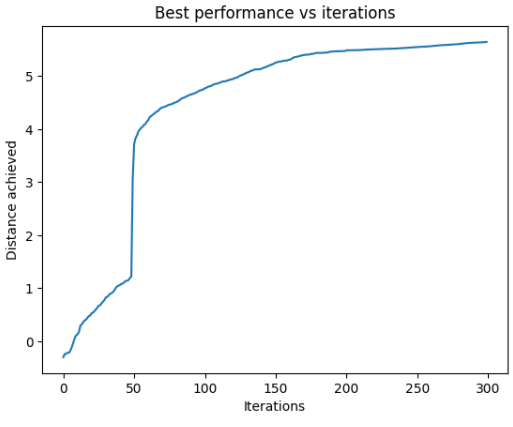
The second model is a custom version of a gradient-based algorithm. It starts with a randomly initialized plan. During one iteration, it goes through every instruction in the plan. For each instruction, it tries out whether the performance improves if the step size value is added to or subtracted from the instruction value. The plan will be updated with the best instruction value. The instructions in the plan is updated one at a time.

Typically, gradient methods differentiate a loss function to find in which direction the parameters should be updated. But since this simulation is a black box that cannot be differentiated, the second model takes one step in each direction to find where to go. Most gradient-based methods first calculate the best direction and then update all parameters at the same time. But the choice to update on parameters (instruction value) at a time is simply due to the other approach did not result in stable training.

After 300 iterations, the evolutionary algorithm (Darwin's Plan) achieved a total distance of 5.73, and the gradient-based method (Gradient Plan) achieved a total distance of 5.64, as shown in Figure 16.



(a) The performance of the evolutionary model.



(b) The performance of the gradient-based model.

Figure 16: The performance of the gradient-based method and evolutionary model.

The plans, which consist of the instruction values, are shown in Table 1. Both methods produced similar results but the evolutionary algorithms is slightly better. The gradient-based method learns more smoothly than the genetic algorithm.

If a gradient can be computed by differentiation of a loss function, then they are much more efficient than genetic algorithms, which just do random changes, hoping some will be an improvement.

Even though both methods managed to produce decent plans, the performance of the gradient-based algorithm varied much more on the initialization. Sometimes it produced very bad plans and other times it was just as good as the genetic algorithm. This is probably due to the model getting stuck in a plan that just throws itself forward, head first, which is a good tactic in the short term, but its a dead end. The genetic algortihm on the other hand has 50 different plans all changing in random. Having multiple plans in the same model means that not all will have a bad initialization. Also, the random changes and combinations of different plans mean that it is better at getting out of local optimums.

One advantage of the gradient-based method is that it has fewer hyperparameters than the genetic algorithm. This makes it easier to use and saves time and computational resources for hyperparameter-tuning.

One interesting observation is that the Gradient Plan has the same value for the last eleven instructions is probably due to it never reaching that part, see Table 1. This also means that during training, when it reaches further than it has previously done before, there will not even be a tactic there to try out.

Table 1: Plans

Index	**Darwin's Plan (x_i)**	**Gradient Plan (x_i)**
0	-0.704145	0.558841
1	0.918923	0.586214
2	0.601930	0.490212
3	-0.854984	0.531860
4	-0.732656	-0.377920
5	-0.954028	0.486438
6	0.183273	0.910000
7	0.329492	1.000000
8	0.881201	1.000000
9	0.996588	-0.108419
10	-0.085373	0.894129
11	-0.680869	0.580721
12	-0.480875	-0.408750
13	-0.179579	0.497230
14	-0.299167	1.000000
15	-0.809113	0.860000
16	0.592342	0.950000
17	0.743240	0.513938
18	-0.040156	0.970000
19	0.863114	0.103145
20	0.524515	0.402973
21	0.236824	-0.648052
22	0.864455	1.000000
23	0.988886	1.000000
24	-0.838303	-0.245296
25	-0.843065	1.000000
26	0.016072	0.970000
27	-0.751476	0.480000
28	0.424378	1.000000
29	0.768947	1.000000
30	-0.678933	1.000000
31	-0.686788	1.000000
32	-0.095773	1.000000
33	-0.759903	1.000000
34	-0.392481	1.000000
35	-0.888618	1.000000
36	0.259479	1.000000
37	-0.649286	1.000000
38	0.031201	1.000000
39	-0.470888	1.000000

P3-1. Generative Models

P3-1.1 Variational Autoencoder (VAE)

Model

See that appended Python code for the full implementation. An overview of the architecture is given in Figure 17. The likelihood chosen to implement is that of a Bernoulli decoder, which uses a sigmoid at the output, as seen as the last "layer" in the aforementioned figure.

As seen in the overview of the model architecture, the encoder consists of four linear layers in total. The first linear layer takes the input and feeds it out with "hidden dimensions". Initially, only one layer was used in the sequential part (2-1); however, it was found that this was not sufficient (high loss values). Hence, the encoder now consists of two initial linear layers mixed with activation functions in order to allow for non-linearity. A standard choice of ReLU was utilized. The sequential layer in the encoder afterwards feeds into two different linear layers: the mean-value layer and the log_var layer. It's from these two layers that we learn the mean and variance, which later outputs the latent features (of which $d_z = 20$). Initially, a $d_z = 10$ was utilized, however, as with the number of layers, it was found that 10 latent dimensions were not sufficient since the loss was high. From our understanding of working with the implementation of neural networks, choosing the number of layers and dimensions is a bit of trial and error for the problem at hand.

The architecture of the decoder is not much different, besides the fact that it aims to restore ("up-sample") the data initially fed into the VAE. Hence, in the decoder, we find three layers, also found by testing. The first two layers pass the output to ReLU activation functions as used in the encoder, and the final output of the layer is into the sigmoid activation function. This is due to the Bernoulli likelihood considered, and also why BCELoss was used. If a sigmoid was not used at the end of the decoder, then simply a BCEWithLogitLoss could have been used as a loss function.

Layer (type:depth-idx)	Output Shape	Param #
VAE	[1, 1, 784]	--
├─Encoder: 1-1	[1, 1, 20]	--
│ └─Sequential: 2-1	[1, 1, 256]	--
│ │ └─Linear: 3-1	[1, 1, 256]	200,960
│ │ └─ReLU: 3-2	[1, 1, 256]	--
│ │ └─Linear: 3-3	[1, 1, 256]	65,792
│ │ └─ReLU: 3-4	[1, 1, 256]	--
│ └─Linear: 2-2	[1, 1, 20]	5,140
│ └─Linear: 2-3	[1, 1, 20]	5,140
├─Decoder: 1-2	[1, 1, 784]	--
│ └─Sequential: 2-4	[1, 1, 784]	--
│ │ └─Linear: 3-5	[1, 1, 256]	5,376
│ │ └─ReLU: 3-6	[1, 1, 256]	--
│ │ └─Linear: 3-7	[1, 1, 256]	65,792
│ │ └─ReLU: 3-8	[1, 1, 256]	--
│ │ └─Linear: 3-9	[1, 1, 784]	201,488
│ │ └─Sigmoid: 3-10	[1, 1, 784]	--

Figure 17: Overview of the Variational Autoencoder architecture.

ELBO Derivation

The roles of all components:

- the latent variable z: models the hidden (latent) representation of an input x (image in this problem). In other words, an input gets mapped to this space, such as a mathematical function that maps its input from some domain to a target range that can have a completely different dimension. The purpose of z is to map the latent underlying factors of the input.
- the encoder $q_\phi(z|x)$: the approximate posterior that is used to learn parameters to be able to map from $\mathbf{x}$ to $\mathbf{z}$. In other words, from the observed data $\mathbf{x}$, we learn parameters to map from the data space that $\mathbf{x}$ is in, to the latent dimension. So $q_\phi(z|x)$ is a learnable (the parameters ϕ) approximate posterior.
- the decoder $p_\theta(x|z)$: the decoder will act as a complement to the encoder, it learns parameters (weights θ) that can be used to map data points in the $\mathbf{z}$ dimension back to the originating dimension from where $\mathbf{x}$ came from. In other words, from latent variables back to the original format of the data (back to the image in this problem).
- the prior $p(z)$: specifies the type of distribution we are working with, so that the VAE knows what to learn between the encoder and decoder.

To begin with, we have the log-likelihood of the data distribution as

$$\log p_\theta(x).$$

If our encoder works as intended, then the expected value of our encoder for our log-likelihood should also be the same as the log-likelihood, ie

$$\log p_\theta(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x)]$$

We note that $p_\theta(x) = \frac{p_\theta(z,x)}{p_\theta(z|x)}$ and use this to rewrite the expected value as

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(z,x)}{p_\theta(z|x)}\right].$$

Hereafter, we rewrite the expression once again, replacing the joint probability $p_\theta(z,x) = p_\theta(z)\, p_\theta(x|z)$ as

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(z)\, p_\theta(x|z)}{p_\theta(z|x)}\right].$$

Thereafter, break the expected value into two separate parts (thanks to log-rules)

$$= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(z)}{p_\theta(z|x)}\right]$$

Now, extend the expression on the right-hand log expression by $\frac{q_\phi(z|x)}{q_\phi(z|x)}$

$$= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)\, p_\theta(z)}{p_\theta(z|x)\, q_\phi(z|x)}\right]$$

$$= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(z)}{q_\phi(z|x)}\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]$$

Now, we find that our original expression equals

$$\log p_\theta(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(z)}{q_\phi(z|x)}\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right].$$

We rewrite the middle term using simple algebra and log rules (and also the rightmost term)

$$\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(z)}{q_\phi(z|x)}\right] = -\,\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right] = -\mathrm{KL}(q_\phi(z|x)\,\|\,p_\theta(z))\,,$$

so that we obtain

$$\log p_\theta(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)\,\|\,p_\theta(z)) - \underbrace{\mathrm{KL}(p_\theta(z|x)\,\|\,q_\phi(z|x))}_{\geq 0} \implies$$

$$\implies \log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)\,\|\,p_\theta(z))\,.$$

Which is what was supposed to be proved. The two terms have different objectives:

- $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$: this is a measurement of the reconstruction between how the actual image looked and what the trained VAE manages to reconstruct. Hence, we expect to see this decrease during training.
- $\mathrm{KL}(q_\phi(z|x)\,\|\,p_\theta(z))$: in this problem, we assume a multivariate normal distribution as a prior p(z). The KL term aims to measure how close the learned distribution $q_\phi(z|x)$ is to the prior, i.e., during training, we are provided with a measurement of how close the distributions are to each other.

The negative elbo term that we aim to minimize is hence

$$\mathcal{L}_{negative} = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathrm{KL}(q_\phi(z|x)\,\|\,p_\theta(z))\,.$$

However, this is not the explicit form used to minimize the negative ELBO in practice. To get that, one needs to insert a Bernoulli distribution in the log expression of the expected value in the expression above. Thereafter, the normal multivariate distribution is inserted into the KL divergence term. The derivation/approach to this was not successful, and in the instructions, it says to ""Write the final objective you minimize in practice (negative ELBO)", hence instead, we include the explicit form of the negative ELBO that is used in the original paper of VAEs by Kingma and Welling (2014) as:

$$\mathcal{L}_{\text{negative}} = \underbrace{\sum_i \Big[- x_i \log \hat{x}_i - (1 - x_i)\log(1 - \hat{x}_i)\Big]}_{\text{reconstruction}} + \underbrace{\left(-\frac{1}{2}\sum_i \left(1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2\right)\right)}_{\text{KL term}}.$$

Training the VAE

The following hyperparameters were utilized:

- Learning Rate = 10^{-3} (but also used a scheduler that reduced the learning rate by 1% for every epoch).
- Batch Size = 64
- Number of epochs = 150
- Latent dimensions $d_z = 20$

Figure 18 illustrates the negative ELBO (loss metric) over the training epochs.

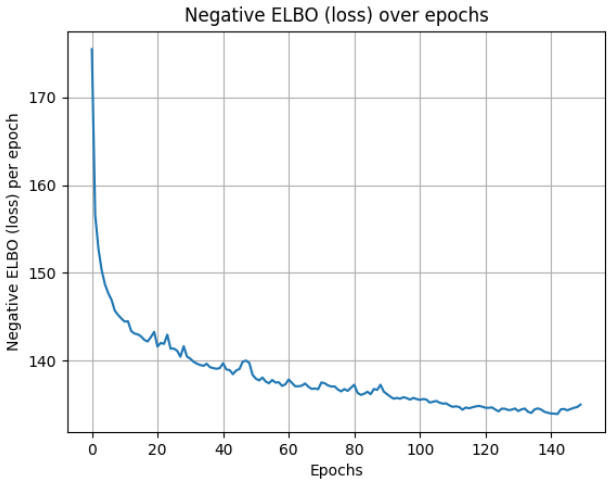


Figure 18: Negative ELBO loss during training.

In Figure 19, the Kullback-Leibler divergence is measured against the reconstruction. A linear relation between the relations is illustrated. In other words, we see that as reconstruction loss increases, i.e., worse reconstructions, the less of a KL divergence. And vice versa, when we have the highest divergence measures, we have the lowest reconstruction loss.

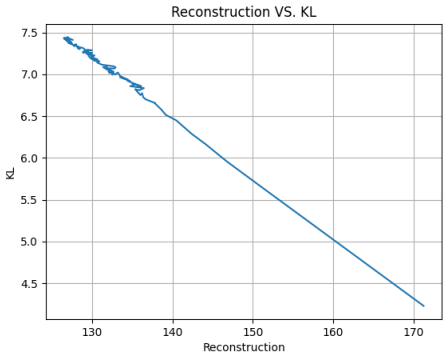


Figure 19: KL divergence vs. reconstruction.

Generating with the VAE

Figure 20 shows 32 samples drawn from a multivariate normal prior $N(\mathbf{0}, \mathbf{I})$ and used to generate an image with the decoder component. The quality of most generated samples is high, comparable to numbers found in the mnist dataset. However, some generated samples lack in quality, such as the second sample on the first row. Mostly, it's just a blur, but one could assume that it is either an 8 or a 9. When it comes to poorly generated images, it's usually due to blur and pixels "fusing". In the sixth sample on the last row, most of the pixels are tightly clustered together, making it difficult to determine which number the generated sample is supposed to represent.

Generally, most numbers can still be identified. The poor image quality is often due to an excessive number of pixels in some areas, causing them to "fuse" with one another. Conversely, there are samples with too few pixels, which results in missing parts of some numbers.

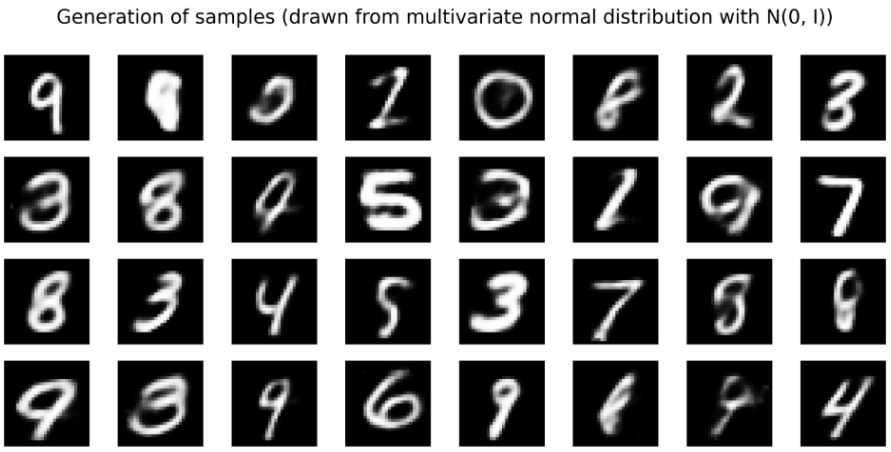


Figure 20: Generated images from the decoder.

P3.2 Score-Based Diffusion Model on a Spiral Distribution

Forward SDE and discretization

We aim to derive $p_k(\boldsymbol{x})$, i.e., the distribution of $\boldsymbol{x}_k$ at step k. We start by noting what we already know and what is given:

$$q(x_k \mid x_0) = \mathcal{N}\big(x_k;\ \sqrt{\bar{\alpha}_k}\, x_0,\ (1-\bar{\alpha}_k)\, I\big)$$

$$p_0(x_0) = \frac{1}{M}\sum_{m=1}^{M} \mathcal{N}\big(x_0;\ \mu(\tau_m),\ \sigma_0^2 I\big)$$

Now, we know that x_0 is a continuous random variable, and what we aim to do is to marginalize out this variable. Due to it being continuous, use the integral of x_0 to "remove it"/account for it and only consider x_k

$$p_k(x_k) = \int p(x_k, x_0)\, dx_0 = \{p(x_k, x_0) = p(x_k|x_0)p_0(x_0) = q(x_k|x_0)p_0(x_0)\} = \int q(x_k \mid x_0)\, p_0(x_0)\, dx_0$$

We now insert the formulas (the normal distributions that we wrote down earlier) into the expression and find that

$$p_k(x_k) = \frac{1}{M}\sum_{m=1}^{M} \int \mathcal{N}\big(x_k;\ \sqrt{\bar{\alpha}_k}\, x_0,\ (1-\bar{\alpha}_k)I\big)\, \mathcal{N}\big(x_0;\ \mu(\tau_m),\ \sigma_0^2 I\big)\ dx_0$$

And now, for simplicity, only take a look at one of the m at the time. Initially, multiplying the multivariate Gaussian was tested, but it did not result in anything useful but a messy formula. Hence, instead, a Gaussian transformation was considered. For instance, if $x_0 \sim \mathcal{N}(\mu, \Sigma)$, assume that $x_k = Ax_0 + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \Sigma_\varepsilon)$. Then, one can write xk as:

$$\mathcal{N}\big(x_k; A\mu,\ A\Sigma A^\top + \Sigma_\varepsilon\big)$$

And we know the A, $\sum$, and $\sum_\epsilon$ from the other formulas where $A = \sqrt{\bar{\alpha}_k}\, I$, $\Sigma = \sigma_0^2 I$, $\Sigma_\varepsilon = (1-\bar{\alpha}_k)I$. Thus, setting the pieces together:

$$\mathcal{N}\big(x_k; \sqrt{\bar{\alpha}_k}\, \mu(\tau_m),\ \big(\bar{\alpha}_k\sigma_0^2 + 1 - \bar{\alpha}_k\big)\, I\big)$$

Then, if we finally consider all m instead of one at a time, we find that

$$p_k(x_k) = \frac{1}{M}\sum_{m=1}^{M} \mathcal{N}\big(x_k;\ \sqrt{\bar{\alpha}_k}\, \mu(\tau_m),\ \big(\bar{\alpha}_k\sigma_0^2 + 1 - \bar{\alpha}_k\big)\, I\big)$$

Running the forward process is illustrated for different time steps in Figure 21. At K=300, the last step, then $\boldsymbol{x}_{300}$ looks like a gaussian. The reason for this has to do with the formula for $\bar{a}_k$

$$\bar{a}_k = \prod_{j=1}^{k} a_j$$

since when $k \to \infty$ then $\bar{a}_k \to 0$ and thus, the derived expression becomes

$$p_\infty(x_\infty) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{N}(x_\infty;\, \mathbf{0},\, \boldsymbol{I})$$

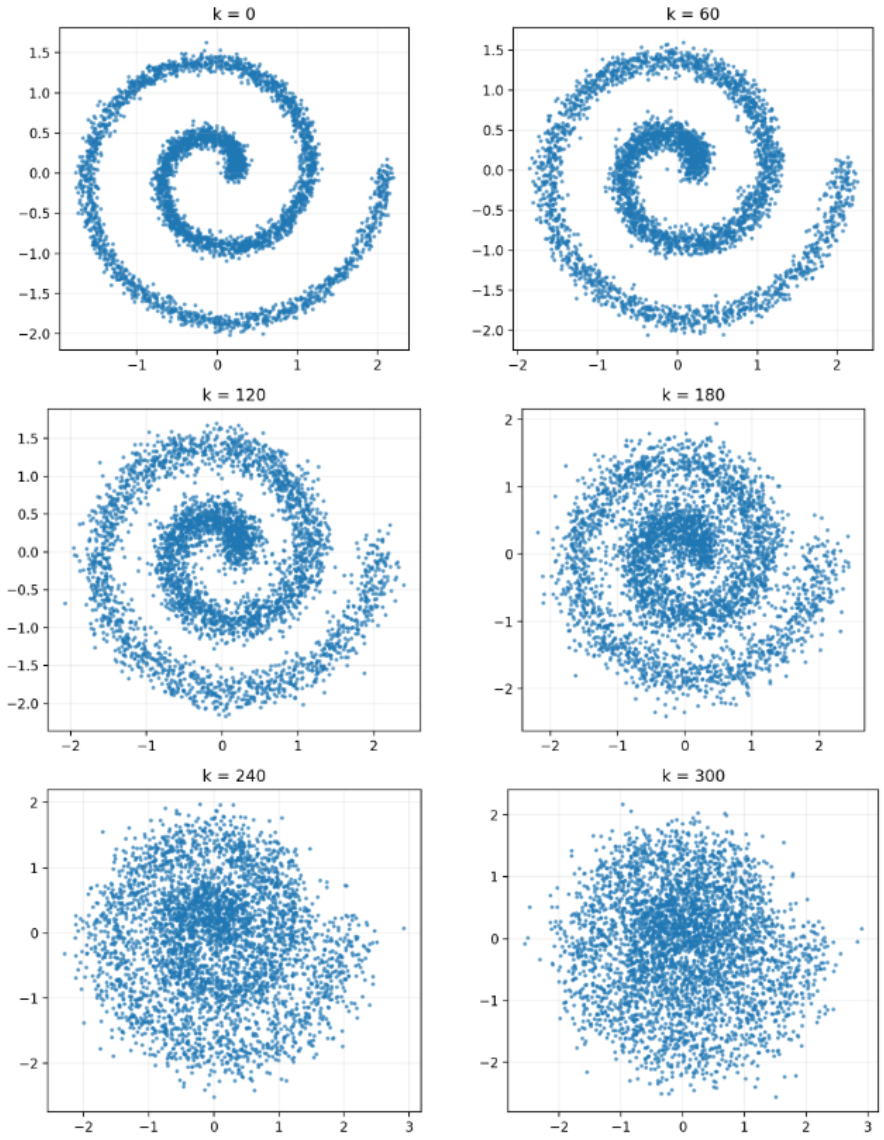


Figure 21: The forward process of adding noise, including the start and the end.

Reverse process using the true score

As derived before, we know that we have the following distribution to work with:

$$p_k(x) = \frac{1}{M}\sum_{m=1}^{M}\mathcal{N}(x; \sqrt{\bar{\alpha}_k}\mu(\tau_m), (\bar{\alpha}_k\sigma_0^2 + 1 - \bar{\alpha}_k)I)$$

To reduce the mess during derivation, let's redefine some variables for shorthand notation

$$\mu_m = \sqrt{\bar{\alpha}_k}\mu(\tau_m) \quad \text{and} \quad \Sigma = (\bar{\alpha}_k\sigma_0^2 + 1 - \bar{\alpha}_k)I$$

We know that the score function for this problem at hand is defined as

$$S_k(x) := \nabla_x \log(p_k(x)) = \nabla_x \log\left(\frac{1}{M}\sum_{m=1}^{M}\mathcal{N}(x; \mu_m, \Sigma)\right)$$

$\log(\frac{1}{M})$ can be canceled by log rules, and we will later differentiate with respect to x. So applying the chain rule $\nabla \log(v) = \frac{1}{v}\nabla v$:

$$S_k(x) = \frac{\sum_{m=1}^{M}\nabla_x\mathcal{N}(x; \mu_m, \Sigma)}{\sum_{m=1}^{M}\mathcal{N}(x; \mu_m, \Sigma)}$$

Instead of differentiating the whole sum at once in the numerator, only consider one component of m and derive that one explicitly (then repeat this for all other m').

$$\begin{aligned}
\nabla_x\mathcal{N}(x; \mu_m, \Sigma) &= \nabla_x\left(\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}e^{-\frac{1}{2}(x-\mu_m)^\top\Sigma^{-1}(x-\mu_m)}\right) \\
&= \mathcal{N}(x; \mu_m, \Sigma)\cdot\nabla_x\left(-\frac{1}{2}(x-\mu_m)^\top\Sigma^{-1}(x-\mu_m)\right) \\
&= \mathcal{N}(x; \mu_m, \Sigma)\cdot\left(-\Sigma^{-1}(x-\mu_m)\right) \\
&= \mathcal{N}(x; \mu_m, \Sigma)\cdot\Sigma^{-1}(\mu_m - x)
\end{aligned}$$

So now, we have an expression for the derivative of the multivariate normal distribution we are working with with respect to x. We can then reinsert that into the score function, such as:

$$S_k(x) = \frac{\sum_{m=1}^{M}\mathcal{N}(x; \mu_m, \Sigma)\cdot\Sigma^{-1}(\mu_m - x)}{\sum_{m=1}^{M}\mathcal{N}(x; \mu_m, \Sigma)}$$

Having found a differentiated expression that consists of two sums, we can then reinsert the variables that were substituted out in the beginning for shorthand notation and we find the following expression

$$S_k(x) = \frac{\sum_{m=1}^{M}\mathcal{N}(x; \sqrt{\bar{\alpha}_k}\mu(\tau_m), \Sigma)\left[\left((\bar{\alpha}_k\sigma_0^2 + 1 - \bar{\alpha}_k)I\right)^{-1}(\sqrt{\bar{\alpha}_k}\mu(\tau_m) - x)\right]}{\sum_{m=1}^{M}\mathcal{N}(x; \sqrt{\bar{\alpha}_k}\mu(\tau_m), \Sigma)}$$

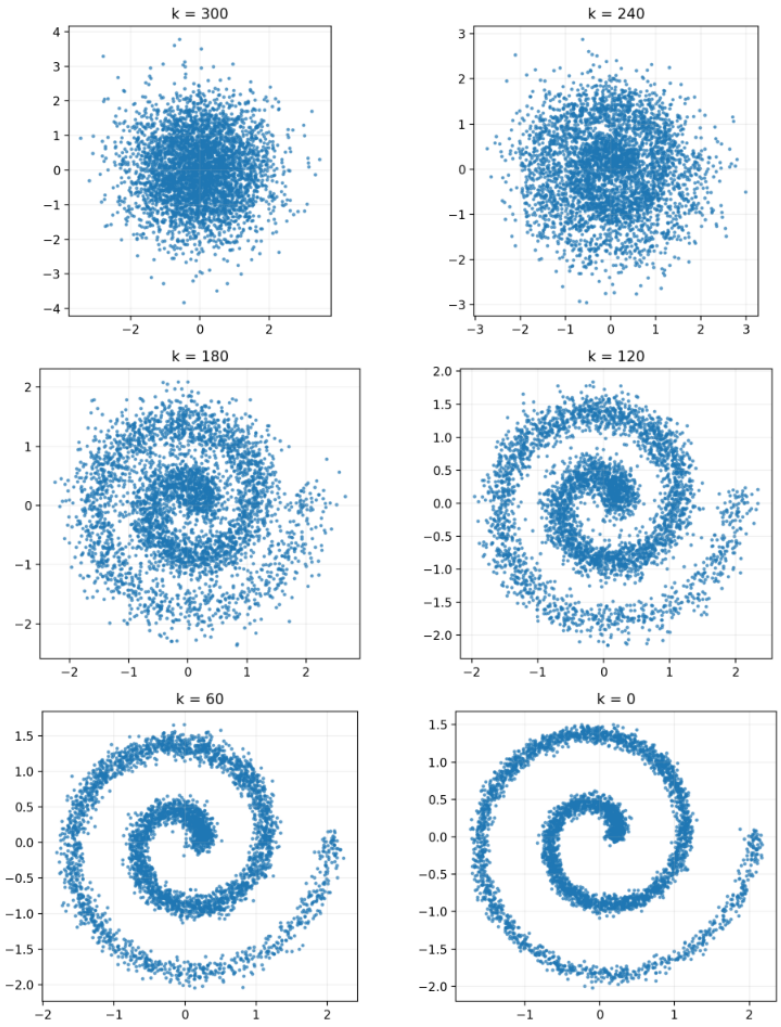


Figure 22: The backward process, using $\boldsymbol{x}_k \sim \mathcal{N}(0, \boldsymbol{I})$, retrieving the original distribution.

The reverse-time discretizations using Euler have been implemented in the appended code. The process of the distributions along the reverse trajectory is plotted in Figure 22. Note that the initial distribution was drawn with samples $\boldsymbol{x}_k \sim \mathcal{N}(0, \boldsymbol{I})$.

Due to interest, the final distribution from the noising in the forward step was also run backward. I.e, the first plot of noising in Figure 21 was run backward in Figure 23.

The final generated samples when reaching k=0 are compared to the original clean spiral samples in Figure 24.

Explain how this reverse process can be used for generation.

What we have seen is that random noise can be turned into structured data in small iterative steps. By defining a score-function, we can then, in every iterative step, nudge the random noise into some distribution of our choice. Hence, by the last time step, the random noise has been pushed and pulled into regions given by the score-function. As for the spiral distribution toy example, it began by moving data points towards a center, and at every later stage, it defined a smaller and clearer move closer to the spiral distribution.

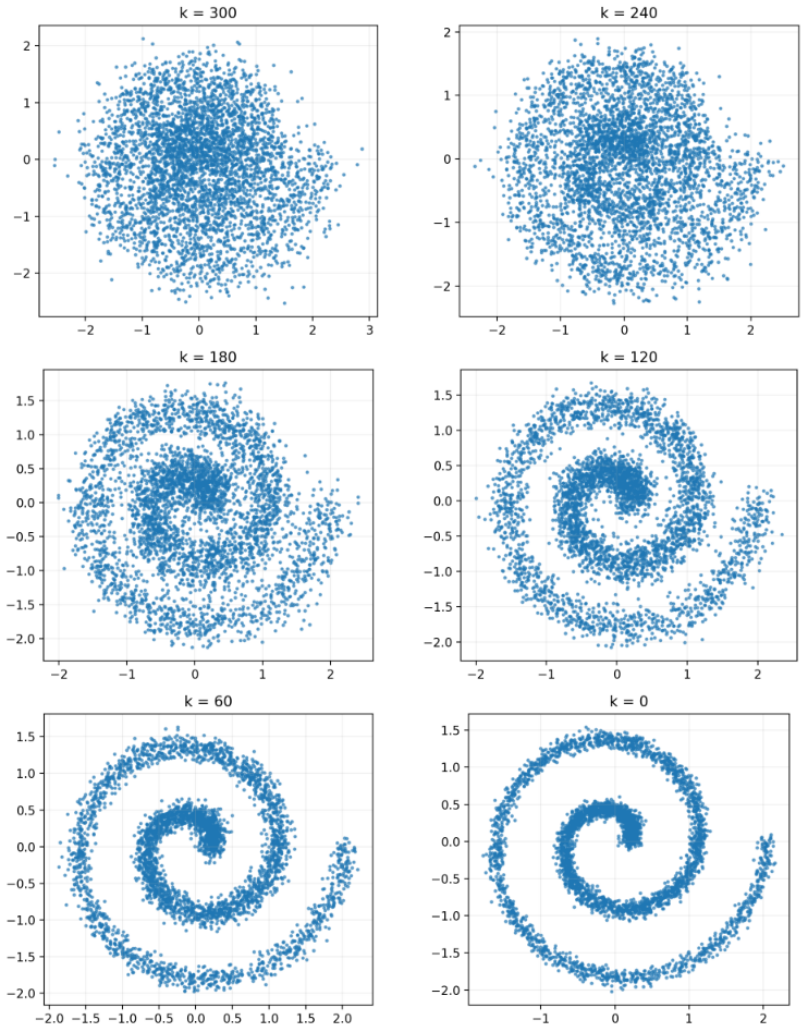


Figure 23: The backward process, using the final distribution after k forwarding steps, retrieving the original distribution.

Explain why the score term is essential (what happens if you remove it?).

The score term is essential for knowing which direction to move the data points. Without the score function, data points would only move around randomly at all time. As we will see, the score term itself is essentially a vector field at different time-steps that tells the data points where to move.

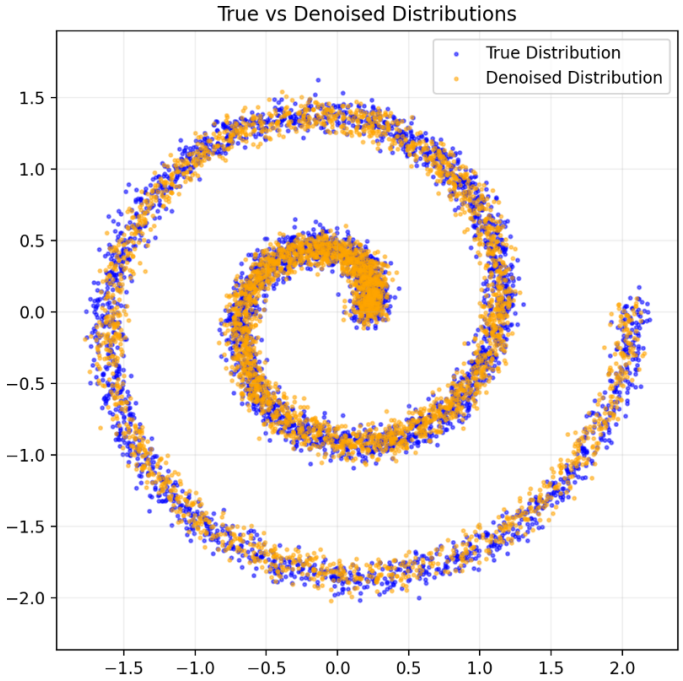


Figure 24: Original generated spiral vs. the spiral retrieved using the exact score from the distribution.

Learning the score

Design your score network: Describe the pain of trying to use RELU, but it does not work... SILU was a must

The network used to learn the score function consists of three inputs: the coordinates (x_o, x_1) for each data point and the time step k, turned into continuous time t_k. The neural network outputs a score for the position at each time step, which essentially is a 2d vector telling the original input data point where to move. I.e., for all points in space at each time step, we learn the direction of where to move. So if we are on the spiral distribution in a late time-step during the reversal process, one should assume that the score vector should be 0 (i.e., no necessity to move).

The network is designed using two hidden layers with a hidden size. Between the input layer and the hidden layers, SiLU activation functions were utilized. Note: initially, ReLU was tested since it is usually a standard first-choice for activation functions; however, it turns out that the model was not able to learn the scores when using ReLU activation functions. Every reverse process would end up random. Instead, SiLU was tested and worked as expected. Assumably, this has to do with ReLU not allowing negative gradients, which are necessary for the score-function, and SiLU solved this problem by being differentiable at every point.

Explain how you include time t (e.g. concatenate t to x).

The continuous time step t_k is included by concatenating the coordinates and x. Hence, the first input layer to the neural network takes two coordinates and the time step (i.e., nn.Linear(3, hidden)). This allows the network to learn the score for different time-steps by simply concatenating the time onto the coordinates.

The training process is illustrated in Figure 25.

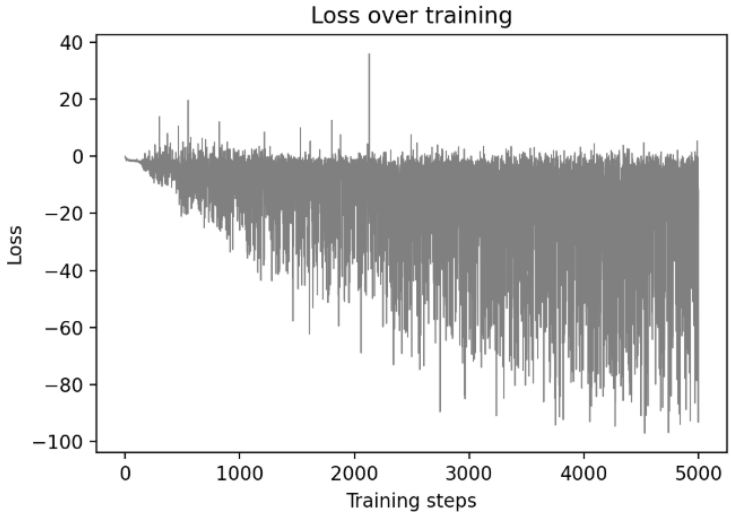


Figure 25: Implemented loss over the training steps.

The vector field for $s_\theta(x, t)$ is visualized in a 2D grid for various noise levels t in Figure 26.

Generation with the learned score

A comparison between the clean spiral data, the samples generated from the true score in P3.2.2, and the samples generated using the neural network approximation is illustrated in Figure 28

Briefly discuss: what is the main gap between the learned-score generation and the true-score generation?

In Figure 28, it is shown that the denoised distribution using the true-score generation is most similar to the true distribution. The learned score distribution struggles to keep the data points on the spiral within the spiral. In other words, at the outer side of the spiral, points generally stay on or close to the spiral, but when looking at the inner parts of the spiral, where the "mess"/noise is from the learned score generation, then the data points are drawn to different parts of the spiral. Therefore, we observe that the learned score has difficulty generating convincing vector fields, as the vectors are likely offset by different sections of the spiral arm. The true-score generation does not have this problem, since it's based on the differentiation of a true probability function. Hence, the main difference between the true distribution and the generated true-score distribution should be tiny random noise. But in the case of the learned-score function, a convincing vector field has not yet been fully learned, and hence, the mess around the inner part of the spiral distribution appears.

So, in summary, the true score creates the perfect path and is only affected by the random noise. The learned score is affected by both the noise and approximation errors, particularly in some regions where there might be a lack of training examples.

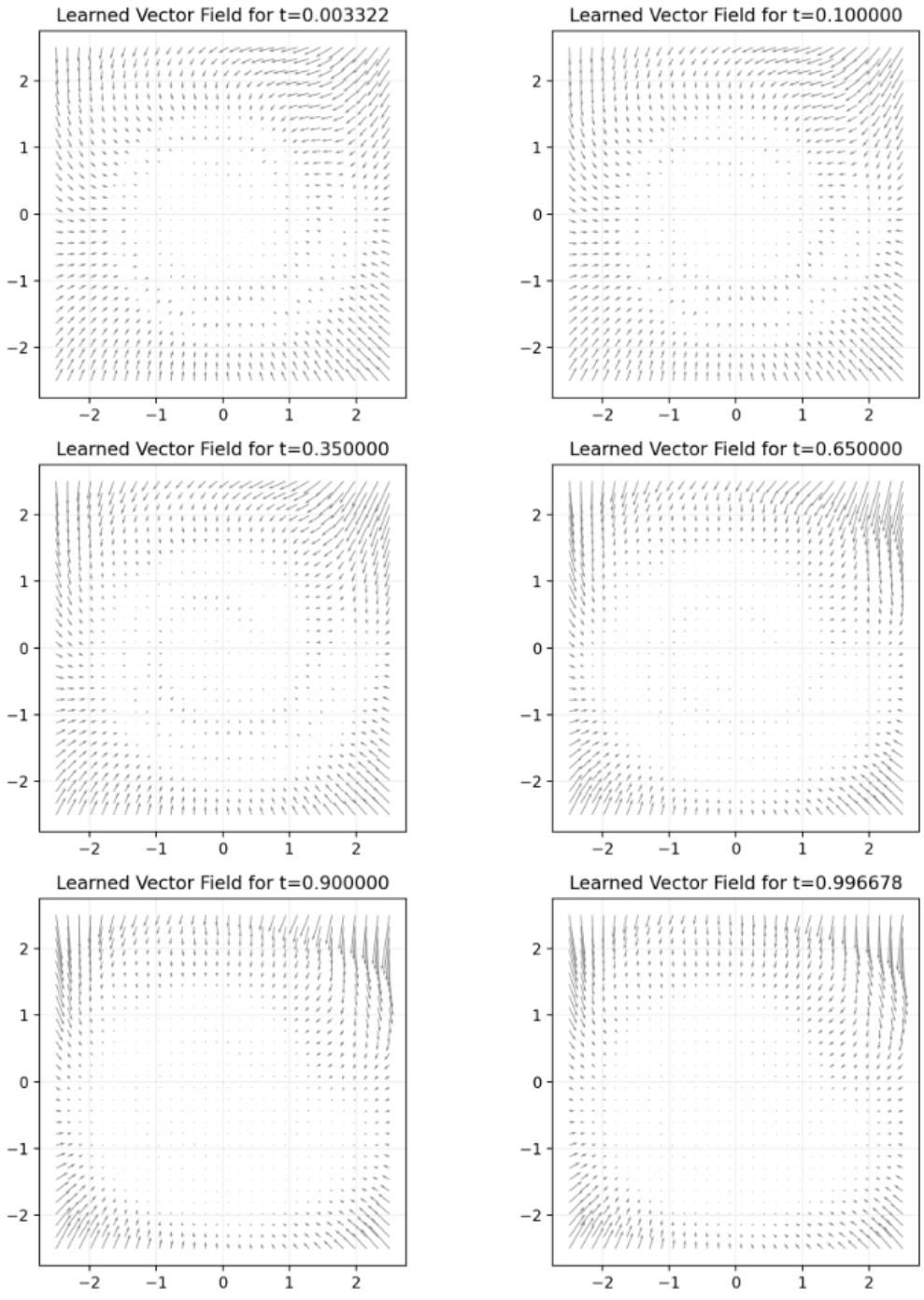


Figure 26: Visualization of the learned vector field for some noise levels

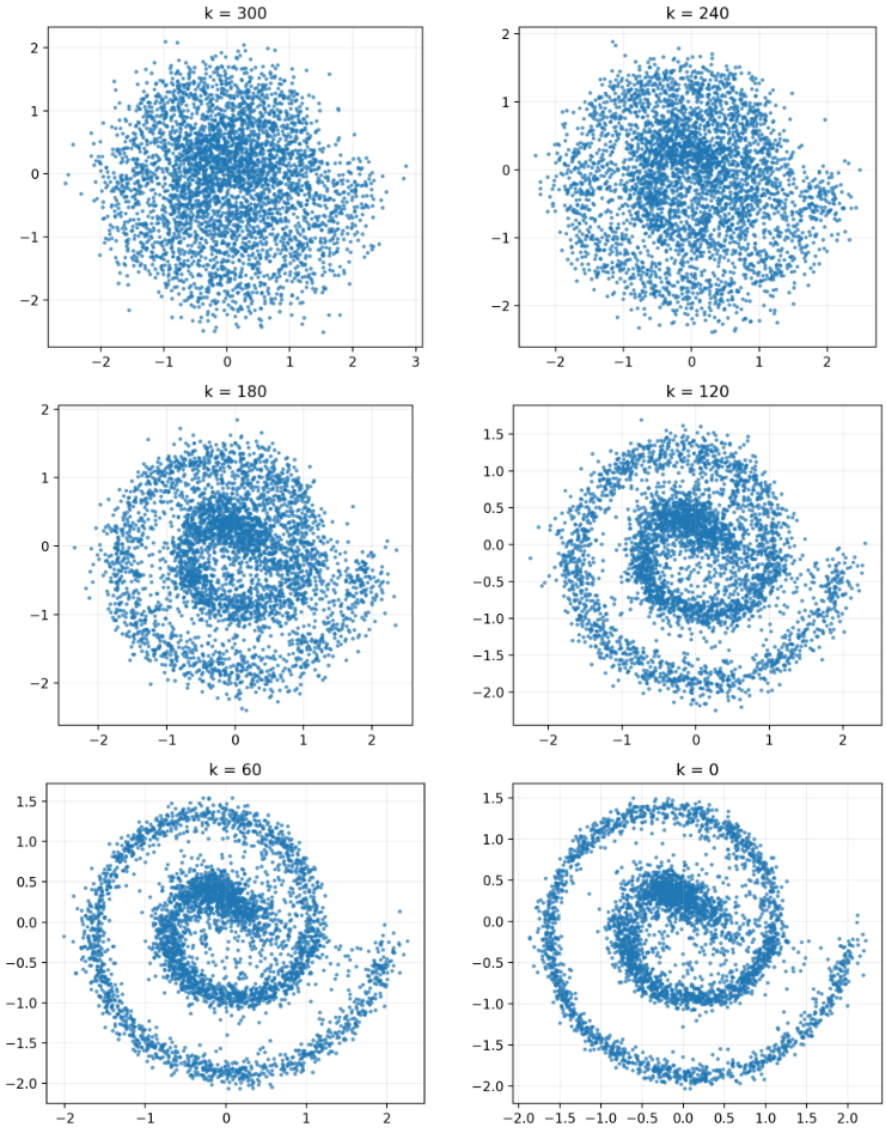


Figure 27: The backward process using the learned weights with a neural network, retrieving the original distribution

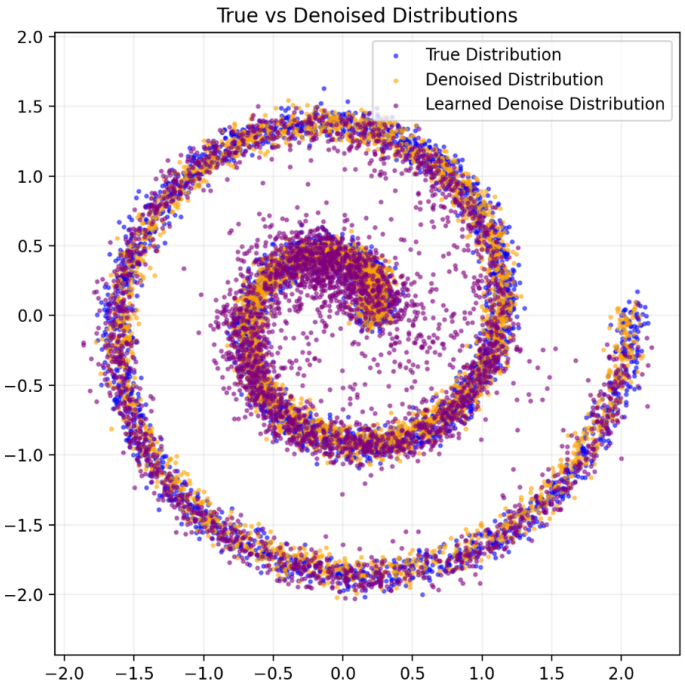


Figure 28: Original generated spiral vs. the spiral retrieved using the exact score from the distribution vs. the spiral retrieved by learning the score.

P3.3 Latent Diffusion Models

Why diffusion on pixels is hard

A sampling procedure for image pixels is challenging due to the high-dimensional pixel space. In this example, we will be working with images that have dimensions of 28 by 28 pixels, which means a total of 784 pixels. Normal images are far larger than the ones in the MNIST dataset, and also consist of three bands of red, green, and blue. Hence, the computational expense becomes vast when working with diffusion on pixels. For instance, consider only an image of the size 400 by 1200 pixels, then all of a sudden, approximately 1.5 million pixels must be processed at each part of the reversal process. Furthermore, not all of these pixels would have meaning, where most of them would just be used to model a certain color or a part of an image. In the MNIST scenario, the meaning of the pixels is dense; remove some, and the digit might lose some of its meaning. Hence, diffusion of pixels will be challenging since pixel space is vast and not a lot of the pixels might have meaning.

Instead, as we will see in the sub-task, it makes more sense to encode images into a lower-dimensional latent space and perform the diffusion on the latent space instead of the true pixel space.

Map MNIST to a low-dimensional latent space

The MNIST images $\boldsymbol{x}$ have been mapped using the encoder mean into a latent space with latent codes $\{\boldsymbol{z}_i\}$

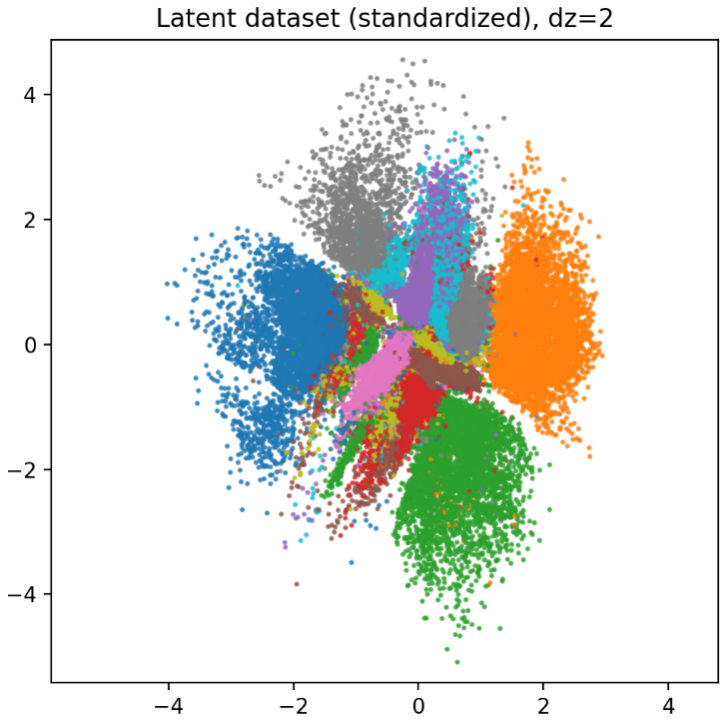


Figure 29: Scatter plot of latent dataset (images passed through encoder)

Learn a score model in latent space

The forward process of adding noise in the latent space is implemented in the Python script called P3_3_latent_diffusion_template.py. The score network $s_\theta(z_k, t_k)$ has been trained using the divergence-based (Hyvärinen) score, and the training curve is plotted in Figure 30. It shows that the loss is minimized during the training of the model; however, due to the stochasticity of choosing training examples, the loss looks to have high variance. However, the general trend of the training is downward, i.e., a decrease in the loss.

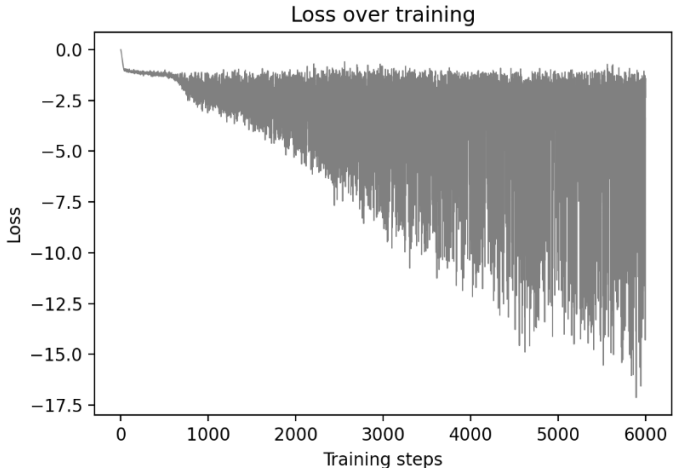


Figure 30: Hyvärinen loss over training.

Decode generated latents back to images

Latent samples $\tilde{\boldsymbol{z}}$ have been generated by running the reversal process in the latent space and have been passed through the decoder. The generation of samples from the latent space is illustrated in Figure 31.

Compare Part 1 vs Part 3

Both methods seem to be comparable when comparing the results in Figure 31 of the latent diffusion generation and Figure 32 of generated samples from the VAE. Both methods produce sharp and blurry images. However, it seems that the VAE might produce slightly more blur in the images. Both of the methods seem to be able to produce borderline nonsense, but once again, it might be that the latent-diffusion model is slightly better at producing higher-quality images.

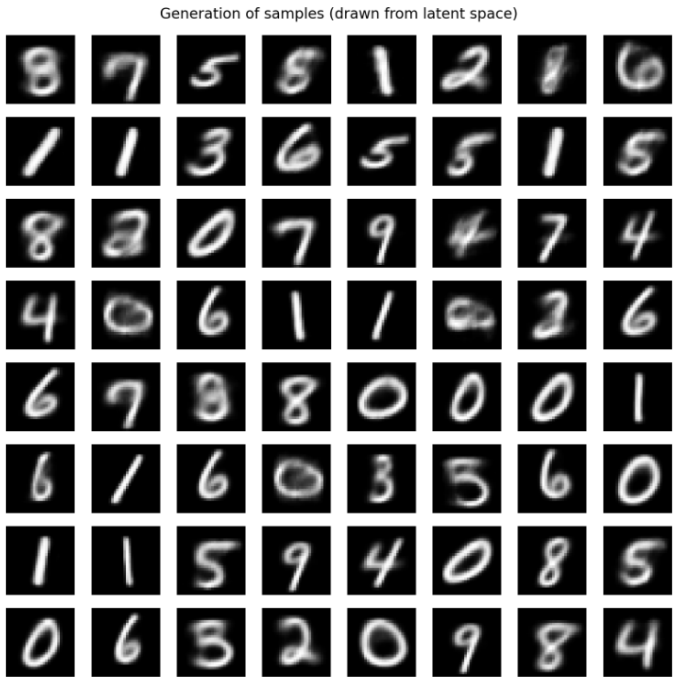


Figure 31: Latent diffusion generation of samples.

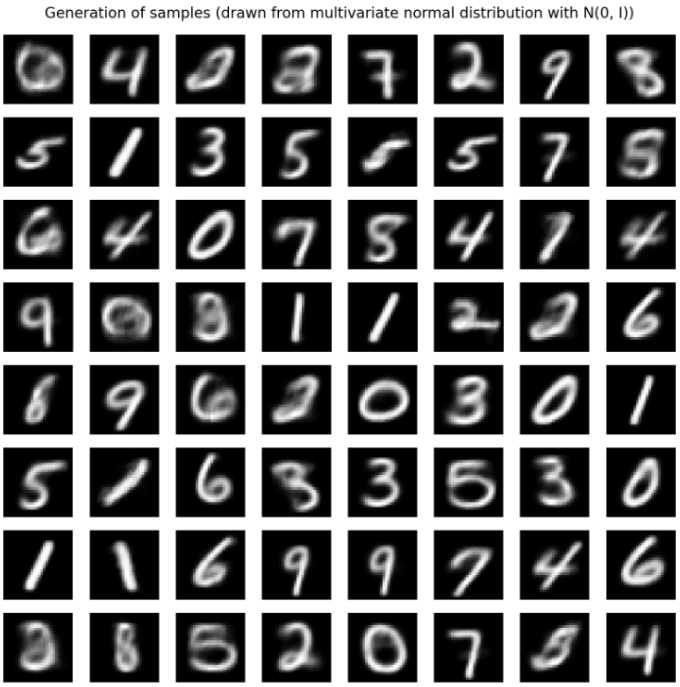


Figure 32: VAE generation of samples drawn from $\mathcal{N}(0, \boldsymbol{I})$

P3.4 Conditional Generation

A conditional model score model $s_\theta(z, k, y)$ where z denotes the latent representation of an image, k denotes the time-step, and y denotes the target label was developed and trained. Most of the code is similar to the code found in part 3.3 (Latent Diffusion Models), where a score network and the Hyvärinen loss were used. The main difference in the code for the conditional generation of images is the inclusion of the target label and the inclusion of the discrete time step. In the aforementioned task, the time-step was $t_k \in [0, 1]$, but in this task, it was interpreted to use a discrete time-step k (since $s_\theta(z, k, y)$ is given in the task).

Before adding the time k and the target y into the network feed, both of the inputs were embedded. The target label is embedded using an nn.embedding layer that maps the integer to a vector with the shape of the specified hidden dimension. This was done to prevent non-embedded targets from being interpreted as magnitudes, i.e., where 8 would be considered a higher magnitude than 2. In order to add the time aspect k to the neural network, the time dimension was also embedded.

The latent variables z, time, and the target embedding were then concatenated and fed into the main part of the neural network for training. This consisted of some linear layers and SiLU activation functions. The rest of the training was conducted in the same manner as the previous code given in the template. Hence, only a few adjustments had to be made. The code can be found in the Python script called P3_4_generate.py

The trained score-based network based on latent feature representation was thereafter trained, and generated digits are illustrated in Figure 33 and Figure 34. It is shown that the conditional method used is not perfect, since generated samples sometimes resulted in generating incorrect numbers. However, the method itself might be good as a conditional generative model, but this implementation could have been improved. For instance, the encoder used to represent the latent variables only consisted of $d_z = 2$, or in other words, the same model as used in previous tasks was used. And we saw in Figure 29, the scatter plot of the latent dataset in 2D, that the encoder would struggle to separate classes at times. Hence, to improve the model, one might consider training a better encoder that is used for more than visually representing the latent space.

However, Figure 33 and Figure 34 illustrate that latent-diffusion-based methodologies can be successful at generating conditional images, in this case, the numbers 0-9.

Figure 33: Generation of digit 7: success varies.

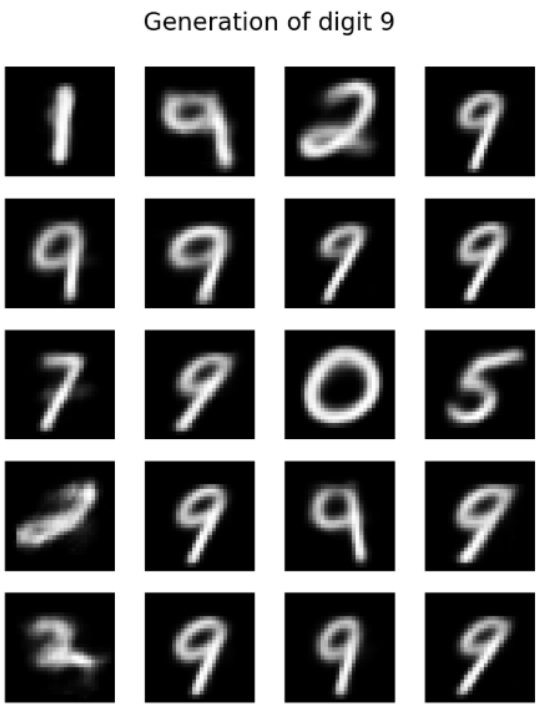


Figure 34: Generation of digit 9: success varies.