Deep Learning Lab4

312554054 林辰翰

1. Derivate conditional VAE formula

In a standard VAE, the goal is to maximize the following objective, which represents the variational lower bound:

$$\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$

Here, P(X) is the likelihood of the data, Q(z | X) is the encoder that approximates the true posterior P(z | X), and P(X | z) is the decoder that generates data given latent variables.

The VAE model can be limited because the encoder and decoder do not account for variations within the data that could be explained by additional variables, like labels or types.

Conditional Encoder and Decoder

To make the VAE capable of generating specific types of data based on conditional information c (like class labels), we condition both the encoder and the decoder on c. The encoder then models the latent variable z based on both the input data X and the condition c, represented as Q(z | X,c). Similarly, the decoder models the output X based on z and c, denoted as P(X | z,c).

Revised Objective

With these modifications, the variational lower bound is reformulated to incorporate the conditioning variable c:

$$\log P(X|c) - D_{KL}[Q(z|X,c)\|P(z|X,c)] = E[\log P(X|z,c)] - D_{KL}[Q(z|X,c)\|P(z|c)]$$

This new formulation suggests that:

(1) The latent variable z is now drawn from a conditional distribution P(z | c), which adapts to the given condition c.

(2) The reconstruction of X from z takes c into account, improving the specificity and relevance of the generated data.

Implementation Notes:

(1) Encoder: It not only encodes X into a latent representation z but does so by also considering the additional condition c. This can be implemented by feeding both X and c into the neural network.

(2) Decoder: It reconstructs X from z while being influenced by c, ensuring that the output is appropriate for the given condition.
By conditioning on c, the CVAE allows for more control over the generated outputs, making it suitable for tasks that require generation of specific types of data based on additional attributes or classes.

2. Introduction

In this assignment, we need to implement conditional video prediction in a VAE-based model. It revolves around designing a model capable of generating future video frames conditioned on both observed video frames and corresponding label information. The context consists of modules essential for this purpose, including a Generator for generating frames, RGB and Label Encoders for feature extraction, a Gaussian Predictor for latent space prediction, and a Decoder Fusion module for combining features and generating future frames.

In the training part, it includes routines for both training and validation stages, along with mechanisms for adjusting parameters such as the learning rate and teacher forcing ratio. The training loop involves calculating reconstruction loss and KL divergence loss to optimize the VAE-based model.

Meanwhile, the modules.py script encapsulates the architecture of various components of our model, such as the Generator, RGB Encoder, Label Encoder, Gaussian Predictor, and Decoder Fusion. These modules predict latent variables and generate future frames based on learned representations.

By integrating these components and training the model using provided datasets, we aim to achieve robust and accurate conditional video prediction capabilities, as outlined in the assignment requirements.

3. Implementation details

Dataloader.py:

In the dataloader.py, it initializes the basic components in the implementation of conditional video prediction model.

The get_key function is utilized to ensure proper sorting of file paths, crucial for maintaining the sequential order of image frames within video sequences. Additionally, the __getitem__ method of the Dataset_Dance class enables the retrieval of individual samples (video sequences) from the dataset. This method handles the loading of image and label frames, applies any specified transformations, and organizes the data into tensors, thereby facilitating seamless access to training and validation data for the conditional video prediction model.

To address an unexpected data path error encountered in the original sample code provided by the TA, I made use of the os library to ensure the accuracy of the file paths. This modification ensures that the file paths for both image and label frames are constructed correctly, thereby resolving the error. By leveraging the os.path.join function to concatenate directory paths and filenames, I ensured the proper formation of file paths, which is essential for successful data loading. This adjustment enhances the reliability and robustness of the data loading process, contributing to the seamless operation of the conditional video prediction model implemented in the dataloader.py module.

Trainer.py:

The function kl_criterion is to compute the KL divergence which measures the difference between two probability distributions.

The class of kl_annealing is responsible for managing the annealing

process for the beta parameter used in the Variational Autoencoder (VAE) loss function, particularly the KL divergence term.
The constructor initializes the class with the provided arguments and the current epoch count. It sets beta to 0.0 and defines two strategies "**Cyclical**" and "**Monotonic**" in annealing respectively. If the annealing type is 'Cyclical', it updates the beta value by cycling through the precomputed beta values based on the current epoch. If the annealing type is 'Monotonic', it linearly increases the beta value from 0.0 to 1.0 over the course of training.
In the function "**frange_cycle_linear**", it was designed to generate a list of beta values that cyclically change linearly from start to stop.

The training_one_step method is design to execute a single step of the training process. It begins by zeroing out the gradients to prepare for parameter updates. Then, it preprocesses the input data, extracting the first frame of each video sequence and transforming it using the frame and label encoders. Next, it feeds the encoded data through the Gaussian predictor and decoder fusion modules to generate reconstructed images from the generator. The method computes the reconstruction loss using the Mean Squared Error (MSE) loss between the reconstructed images and the input images. Additionally, it calculates the KL divergence loss using the kl_criterion function, which measures the difference between two probability distributions. The total loss is then computed as the sum of the reconstruction loss and the KL divergence loss multiplied by the annealed beta. Finally, the method performs backpropagation to compute gradients and updates the model parameters based on the computed loss.
The val_one_step method performs a single step of validation. It also begins by zeroing out the gradients and preprocesses the input data and then disable gradient computation in order to save memory and computation time. The method then computes the reconstruction loss using the same MSE loss as in the training step. The computed loss is returned without performing backpropagation, as validation is solely focused on evaluating the model's performance and not updating its

parameters.

modules.py:

The reparametrize function in "Gaussian_Prediction" class implements the reparameterization trick for sampling latent variables in a Variational Autoencoder (VAE). The method first computes the standard deviation (std) from the logarithm of the variance (logvar). Then, it generates random noise (eps) from a standard normal distribution with the same shape as std. Finally, it combines the mean (mu) with the product of std and eps to produce the sampled latent variable (z). This technique enables the VAE to learn continuous and meaningful representations in the latent space while allowing for efficient gradient-based optimization during training.

The role of noise is to ensure the differentiability of the sampling process for latent variables. In Variational Autoencoders (VAEs), we aim to sample from a known probability distribution to generate latent variables, which are then used to generate data. However, directly sampling from the standard deviation (std) and mean (mu) may lead to non-differentiable operations, posing challenges for backpropagation.

Tester.py:

In val_one_step function, I first initialize the previous_frame with first image frame(img[0]) from the input sequence, then loop through all the remaing frames in the input sequence(label).

Firstly, the current pose information is extracted from the label sequence. This involves obtaining the pose information for the current frame, referred to as current_pose.

Following this, encoded features are generated. The current pose is transformed into an encoded representation, known as encoded_pose, utilizing the label_transformation module. Simultaneously, the previously generated frame (previous_frame) is encoded using the frame_transformation module, resulting in the acquisition of encoded_previous_frame.

Next, random noise, denoted as noise, is generated from a standard

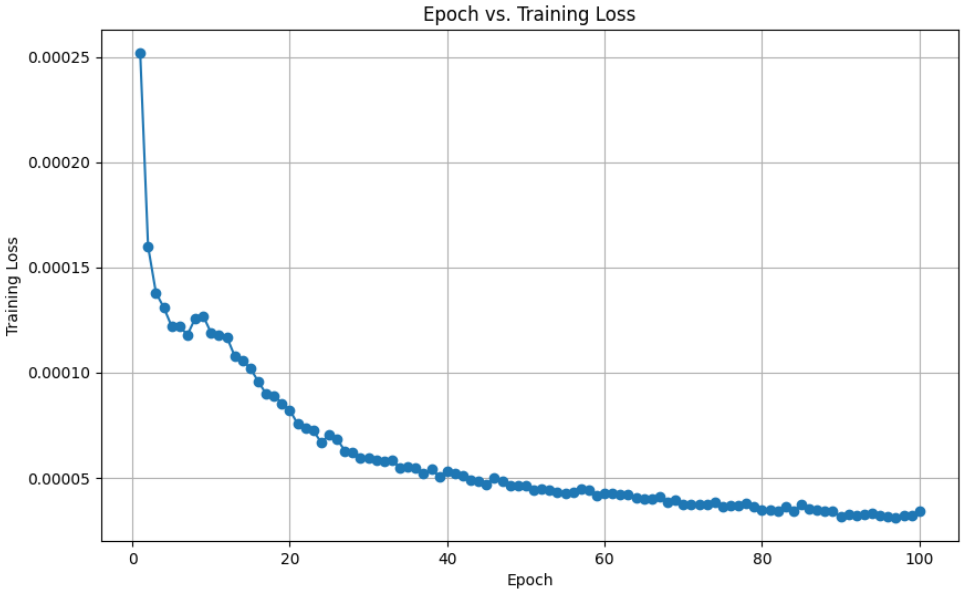
normal distribution. This noise, possessing the same shape as the frame (args.N_dim, args.frame_H, args.frame_W), serves to introduce stochasticity during the frame generation process.

Utilizing the Decoder_Fusion module, the encoded pose, encoded previous frame, and noise are combined to produce the subsequent frame in the sequence, referred to as generated_frame.

Upon generation of the frame, the previous_frame variable is updated with the newly generated frame, preparing it for use in the next iteration of the loop.

Throughout this process, the generated frame (generated_frame) and the current pose (current_pose) are appended to their respective lists (decoded_frame_list and label_list). These lists serve as repositories for the generated frames and pose information, facilitating visualization and evaluation tasks.

4. Analysis & Discussion



In the "Monotonic" strategy, the gradual adjustment of the KL annealing parameter allows for a controlled transition in the model's focus from prioritizing reconstruction to emphasizing the KL divergence term. Initially, when the KL annealing parameter is low, the model tends to prioritize

reconstruction loss, resulting in a higher reconstruction loss at the beginning of training. As training progresses and the KL annealing parameter increases, the model gradually shifts its focus towards the KL divergence term, leading to a decrease in reconstruction loss and an increase in KL divergence loss. Eventually, the total loss converges to a balanced level, indicating that the model has successfully learned to generate accurate reconstructions while also regularizing the latent space.

In contrast, the "Cyclical" strategy involves oscillations in the KL annealing parameter over the course of training. This approach alternates between phases of prioritizing reconstruction and emphasizing the KL divergence term, leading to fluctuations in both reconstruction loss and KL divergence loss throughout training. While this strategy may introduce additional complexity, it can potentially help the model explore different regions of the latent space more effectively.

On the other hand, training without KL annealing entirely removes the regularization effect of the KL divergence term from the loss function. In this scenario, the model solely focuses on minimizing the reconstruction loss, which may result in overfitting to the training data and a less structured latent space representation. Consequently, the total loss may not converge to a balanced level, and the model may exhibit issues such as posterior collapse or instability during training.

Overall, the choice of KL annealing strategy impacts the training dynamics and the learned latent space representation. The "Monotonic" strategy provides a smooth transition between reconstruction and regularization, the "Cyclical" strategy introduces oscillations to explore different latent space regions, while training without KL annealing removes the regularization entirely, each with its respective implications for model performance and stability.

5. Reference

(1) https://agustinus.kristia.de/techblog/2016/12/17/conditional-vae/

(2) https://blog.csdn.net/wwyy2018/article/details/101599862