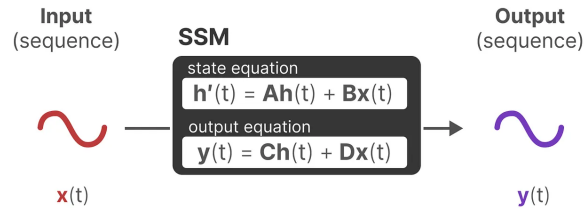
Assignment 2 S4D model report

Name : Van Nguyen , Preethal Reddy Yellareddygari

1. Background on $S4$ model.

State space models

- **State space models** (SSMs) assume that there exists an underlying hidden state that governs the sequence of observations we see. In essence, SSMs map input sequences to latent states with higher dimension , which are then used to predict future outputs.
- Traditionally, SSMs operate through two primary equations:



–

Figure 1: Source

  - **State Equation** : This describes how the current state evolves into the next state based on the input and internal dynamics.
  - **Output Equation**: This explains how the current state is translated into an observable output, ,and how the input directly influences the output.
- The **S4 model** integrates key concepts from state space models to effectively handle sequential data. This is achieved by utilizing a specialized approach to designing matrix A. Matrix A functions to carry information from the previous state to construct the new state, acting as the memory backbone of the model.
- To optimize the capability of matrix A, the **HiPPO** (High-order Polynomial Projection Operators) method is employed. This innovative approach compresses all observed input signals into a compact vector of coefficients, enabling matrix A to:
  - Effectively capture recent tokens or inputs with high fidelity.
  - Gradually decay the influence of older tokens, balancing the memory between old and new information.
- This structured approach leads to the development of the S4 model, which is composed of:
  - **State Space Models:** These provide the foundational framework for modeling dynamic systems.
  - **HiPPO:** Used to manage long-range dependencies within the model, ensuring robust memory and retention capabilities across extended sequences.
  - **Discretization:** This process adapts the continuous state space model into formats suitable for recurrent or convolutional processing, making it applicable to various sequence learning tasks.

How is S4D different from S4?

The S4 and S4D models differ primarily in their approach to handling state matrices, which impacts their computational complexity and application suitability. The S4 model employs a state matrix that includes both diagonal and low-rank components, allowing it to capture complex interdependencies between different state dimensions. This complexity requires sophisticated algorithms to manage, making the S4 model suitable for advanced applications that demand detailed modeling of intricate data interactions, such as sophisticated signal processing or complex time-series analysis. In contrast, the S4D model uses a state matrix comprised solely of diagonal components, which simplifies the computations by focusing on the independent evolution of each state dimension. This reduction in complexity results in easier implementation and faster processing, making S4D more suitable for less complex tasks where speed and computational efficiency are prioritized, such as basic sequence processing tasks. Thus, while S4 is geared towards complexity and detail for complex modeling tasks, S4D offers a streamlined, efficient approach ideal for simpler applications.

2. Model Architecture

While we initially considered Sashimi and S4 models, they have complexity with dependency files .Thus , through the training process in google collab , we encounter system crashes due to The model need to trace the file back and forth .Through exploration, we found that a d_state value of 10 have the optimal balance for our model's needs (it helps prevent output with low amplitude and vanishing gradient). For a deeper look of our model's architecture and specific implementation details, please refer to the provided code and visual block diagram.

```
# S4 model can be found here https://github.com/state-spaces/s4
class S4D_AE(nn.Module):
    def __init__(self, input_dim=None):
        super(S4D_AE, self).__init__()
        self.s4d_1 = S4D(1, d_state=10, dropout=0.0)
        self.d1 = nn.Conv1d(1, 2, 2, stride=2)
        self.s4d_2 = S4D(2, d_state=10, dropout=0.0)
        self.u2 = nn.ConvTranspose1d(2, 1, 2, stride=2)
        self.norm = nn.LayerNorm(90000)
        self.norm_1 = nn.LayerNorm(45000)
        self.gelu = nn.GELU()

    def forward(self, x): #(1,1,90000)
        x = self.norm(x)
        x = self.s4d_1(x)  # (B,H,L) -> (B,H,L)
        x = self.d1(x)  # 1,1,90000 -> 1,2,45000
        x = self.norm_1(x)
        x = self.s4d_2(x)  # (B,H,L) -> (B,H,L)
        x = self.gelu(x)
```

```
        x = self.norm_1(x)
        x = self.u2(x)  # 1,2,45000 -> 1,1,90000
        x = self.norm(x)
        x = self.s4d_1(x)  # (B,H,L) -> (B,H,L)
        x = self.gelu(x)
        return x
```

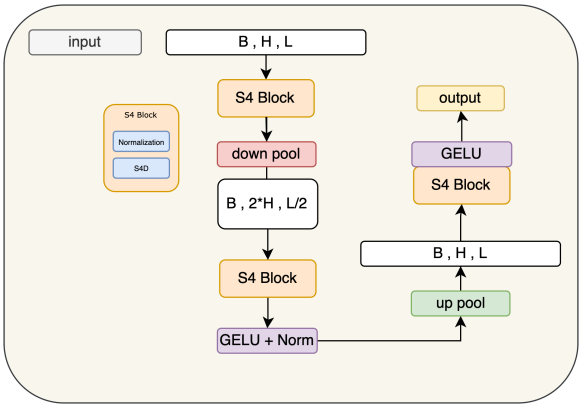


Figure 2: Architecture of model

3. Experimental Setup

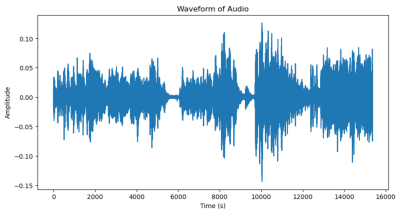
- The datasets and preprocessing we use same as assigment one .
- The loss function we using MSE loss since this is the continous data.
- Metric for comparasion is FID score
  - The Inception model used in our implementation is derived from the encoder of the VAE model developed in Assignment 1 [A1] .
    * The inception model is training on the classical genere , and sampling rate =3000
    * To ensure the fairness , We only compare upto 20 batch for train and test set , because the result will affect based on the size of data.
  - We will compare the result between the best VAE model in A1 and our implementation of S4 generative .

4. Results and Analysis

FID score comparasion	Train sample	Test sample
Best VAE from A1	20	96
S4D	5.12	20.76

- we can see that the S4D are :

  - have lower score than VAE (it can have X4 lower)
  - The test score is close to train score → model not overfitting.
- From observation
  - The audio can generate with more pattern , unlike the VAE they just repeat same patern with the litmit of latent space .
  - result of the generation waveform and its input sample . Eventhough there is still a lot of noise but we can find the pattern of the background



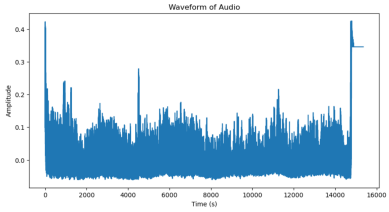
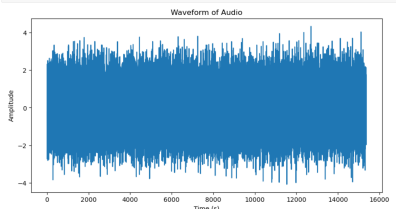


Figure 3: input sample **Figure 4:** generate output

- If we put a noise as input → The output can give us some reconizable audio pattern .
  - We can observe the wave from below



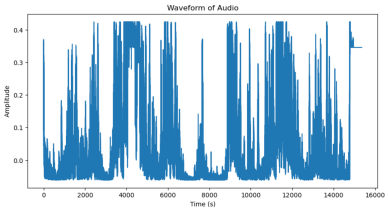


Figure 5: noise sample **Figure 6:** generate of noise

  -

5. Conclusion and Future Work

- Challenges :
  - We have try to experiment different S4 architecture (Sashimi , S4) , but since the training process issue on google collab we focus on building with S4D .
  - As we try working with S4D model , we found that while increase the dimension of state can help produce more complex for model to out put more feature map . But with this dataset if the the dimension is high (more than 20) we can easily have vanishing gradient (the loss are getting nan or very low result , some time they not changing) .
  - Even though S4D model is good but without normalization the result is very bad .
  - We also apply actication function between the layer to impove the output .
- Future Work
  - We have implement the VAE with S4 version , but since this model is sensitive with loss model . We keep getting vanishing and glitch the gradient (very high) . Eventhough we have change the weight of ELBO but it doesnt help much . In

future we will try to make it better .

- The SSM model are famous when dealing with long range input data , we will examinate with higher sampling rate on next implementation.

Citation:

1. Grootendorst, M. (2024, February 19). A Visual Guide to Mamba and State Space Models. Exploring Language Models. https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state#%C2%A7what-is-a-state-space
2. https://github.com/state-spaces/s4/tree/main/models
3. On the Parameterization and Initialization of Diagonal State Space Models (S4D)
4. Efficiently Modeling Long Sequences with Structured State Spaces by Albert Gu, Karan Goel, and Christopher R´e(2022,5 August)
5. It's Raw! Audio Generation with State-Space Models(Sashimi)