Sinusoidal Time Embedding Explanation and Implementation

1 Mathematical Formulation

Given a scalar timestep $t \in \mathbb{R}$ and an embedding dimension $d \in \mathbb{N}$ (where d is even), the sinusoidal time embedding is computed using alternating sine and cosine functions applied at exponentially scaled frequencies. The output is a vector $\mathbf{e}_t \in \mathbb{R}^d$ defined as:

$$e_t[2i] = \sin\left(\frac{t}{10000^{2i/d}}\right)$$

$$e_t[2i+1] = \cos\left(\frac{t}{10000^{2i/d}}\right)$$

for $i = 0, 1, \ldots, \frac{d}{2} - 1$.
This allows to encode continuous, smooth representations of time.

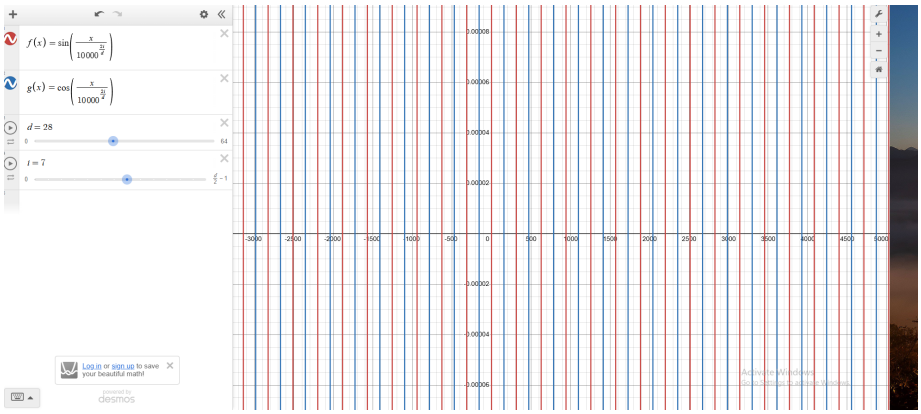

Figure 1: Cool desmos graph of the above functions.

2 Explanation

- The embedding uses a fixed, non-learned set of sinusoidal functions (compared to other methods where we use machine learning and fully-connected layers to learn the representation of time embeddings).
- Even indices (0, 2, 4, ...) use the sine function.

- Odd indices (1, 3, 5, ...) use the cosine function.

3 PyTorch Implementation

```python
# My version
def get_time_embeddings(timesteps: torch.Tensor, embedding_dim: int) -> torch.Tensor:
    assert embedding_dim % 2 == 0, "embedding_dim must be even"

    # Create the frequency spectrum
    half_dim = embedding_dim // 2
    exponent = -math.log(10000.0) / (half_dim - 1)
    freq = torch.exp(torch.arange(half_dim, dtype=torch.float32) * exponent)

    # Expand timesteps for broadcasting
    timesteps = timesteps.float().unsqueeze(1)  # (N, 1)
    args = timesteps * freq.unsqueeze(0)        # (N, half_dim)

    # Concatenate sin and cos
    embedding = torch.cat([torch.sin(args), torch.cos(args)], dim=1)  # (N, embedding_dim)

    return embedding
```