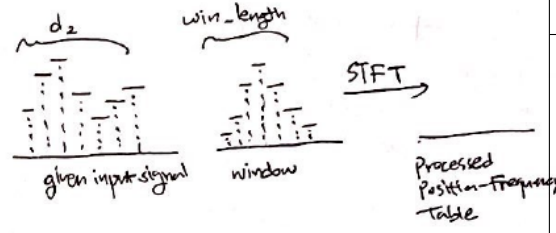


Builtin Fourier Transforms

torch.stft

torch.stft(input, n_fft, hop_length=None, win_length=None, window=None, center=True, pad_mode='reflect', normalized=False, onesided=True)	
	Require
	<ul style="list-style-type: none"> • $input = (d_1, d_2)$ or (d_2) (rank = 1 or 2) • $0 < n_fft < 2 \cdot d_2$ if $center == True$ • $0 < n_fft \leq d_2$ if $center == False$ • $hop_length > 0$ <ul style="list-style-type: none"> – if $hop_length == None$, then let $hop_length = \lfloor \frac{n_fft}{4} \rfloor$. • $0 < win_length \leq n_fft$ <ul style="list-style-type: none"> – if $win_length == None$, then let $win_length = n_fft$. • $window == None$ or $window = (win_length)$ (rank 1)
	Guarantees
	<ul style="list-style-type: none"> • $y = (d_1, a, b, 2)$ or $(a, b, 2)$... from the proof tree
	Comment
	<ul style="list-style-type: none"> • 음향분석에서 자주 쓰이는 short-time Fourier transform. • 마지막 텐서 부분은 real과 complex 부분을 가리키는 두 텐서. real 함수가 여기서 쓰임

$\sigma \vdash E \Rightarrow e, c$

$\sigma \vdash window \Rightarrow w, c_{win}$ if $window \neq None$, otherwise $c_{win} = \emptyset$

$hop_length = \lfloor \frac{n_fft}{4} \rfloor$ if $hop_length == None$

$win_length = n_fft$ if $win_length == None$

$n = e[\text{rank}(e)]$

$c_{dim} = \{(window = None \vee w = (win_length)) \wedge (0 < hop_length) \wedge (0 < win_length \leq n_fft)\}$

$c_{fft} = \{(\text{if } center \text{ then } 0 < n_fft < 2 \cdot n \text{ else } 0 < n_fft < n)\}$

$a = \text{if } onesided \text{ then } \lfloor \frac{n_fft}{2} \rfloor + 1 \text{ else } n_fft$

$b = \text{if } center \text{ then } \lfloor \frac{n}{hop_length} \rfloor + 1 \text{ else } \lfloor \frac{n-n_fft}{hop_length} \rfloor + 1$

$e' = \text{if } \text{rank}(e) == 2 \text{ then } (e[1], a, b, 2) \text{ else } (a, b, 2)$

$\sigma \vdash \text{stft}(E, n_fft, hop_length = None, win_length = None, ..., onesided = True) \Rightarrow e', c \cup c_{win} \cup c_{dim} \cup c_{fft}$

Example Codes:

```
a = torch.randn(5, 10000)
print(a.shape) # (5, 10000)
print(torch.stft(a, 4).shape) # (5, 3, 10001, 2)
print(torch.stft(a, 200, 100).shape) # (5, 101, 101, 2)
print(torch.stft(a, 200, 100, center=False).shape) # (5, 101, 99, 2)
print(torch.stft(a, 200, 100, center=False, onesided=False).shape) # (5, 200, 99, 2)
```

torch.rfft

torch.rfft(input, signal_ndim, normalized=False, onesided=True)	
<p>input as polynomial f, $j \triangleq \sqrt{-1}$ Compute $f(e^{\frac{2\pi j k}{N}})$ for each k. Onesided options is because $\overline{e^{kj}} = e^{-kj}$</p>	Require
	<ul style="list-style-type: none"> • $input = (d_1, d_2, \dots, d_k)$ • $signal_ndim \in \{1, 2, 3\}$ • $signal_ndim \leq k$
	Guarantees
	<ul style="list-style-type: none"> • $y = (d_1, d_2, \dots, d_{k-1}, d'_k, 2)$ where – $d'_k = \text{if onesided then } \lfloor \frac{d_k}{2} \rfloor + 1 \text{ else } d_k$
	Comment
	<ul style="list-style-type: none"> • Real-to-complex Discrete Fourier Transform • 마지막 텐서 부분은 real과 complex 부분을 가리키는 두 텐서. real 함수가 여기서 쓰임

$$\sigma \vdash E \Rightarrow e, c$$

$$k = \text{rank}(e)$$

$$c_{dim} = \{(signal_ndim \in \{1, 2, 3\}) \wedge (signal_ndim \leq k)\}$$

$$f = \text{if onesided then } \lfloor \frac{e[k]}{2} \rfloor + 1 \text{ else } e[k]$$

$$e' = (e[1], e[2], \dots, e[k-1], f, 2)$$

$$\frac{}{\sigma \vdash \text{rfft}(E, signal_ndim, normalized = False, onesided = True) \Rightarrow e', c \cup c_{dim}}$$

torch.fft

torch.fft(input, signal_ndim, normalized=False)	
	Require
	<ul style="list-style-type: none"> • $input = (d_1, d_2, \dots, d_k)$ • $d_k = 2$ (real and complex values) • $signal_ndim \in \{1, 2, 3\}$ • $signal_ndim \leq k - 1$
	Guarantees
	<ul style="list-style-type: none"> • $y = (d_1, d_2, \dots, d_{k-1}, d_k) = input$ (same shape)
	Comment
	<ul style="list-style-type: none"> • Complex-to-complex Discrete Fourier Transform • 마지막 텐서 부분은 real과 complex 부분을 가리키는 두 텐서. real 함수가 여기서 쓰임

$$\sigma \vdash E \Rightarrow e, c$$

$$k = \text{rank}(e)$$

$$c_{last} = \{(e[k] = 2)\}$$

$$c_{dim} = \{(signal_ndim \in \{1, 2, 3\}) \wedge (signal_ndim \leq k - 1)\}$$

$$\frac{}{\sigma \vdash \text{fft}(E, signal_ndim, normalized = False) \Rightarrow e, c \cup c_{last} \cup c_{dim}}$$

Window Declarations

torch.hann_window

torch.hann_window(window_length, periodic=True, dtype=None, layout=torch.strided, device=None, requires_grad=False)	
	Require
	<ul style="list-style-type: none">$window_length \geq 0$
	Guarantees
	<ul style="list-style-type: none">$y = (window_length)$
	Comment
	<ul style="list-style-type: none">\sin^2 가중치의 주기성 윈도우 선언

$$\overline{\sigma \vdash \text{hann_window}(window_length, \dots) \Rightarrow (window_length), \{(window_length \geq 0)\}}$$

torch.bartlett_window

torch.bartlett_window(window_length, periodic=True, dtype=None, layout=torch.strided, device=None, requires_grad=False)	
	Require
	<ul style="list-style-type: none">$window_length \geq 0$
	Guarantees
	<ul style="list-style-type: none">$y = (window_length)$
	Comment
	<ul style="list-style-type: none">절대값함수 가중치 윈도우 선언

$$\overline{\sigma \vdash \text{bartlett_window}(window_length, \dots) \Rightarrow (window_length), \{(window_length \geq 0)\}}$$

torch.hamming_window

torch.hamming_window(window_length, periodic=True, alpha=0.54, beta=0.46, dtype=None, layout=torch.strided, device=None, requires_grad=False)	
	Require
	<ul style="list-style-type: none">$window_length \geq 0$
	Guarantees
	<ul style="list-style-type: none">$y = (window_length)$
	Comment
	<ul style="list-style-type: none">삼각함수 가중치의 주기성 윈도우 선언

$$\overline{\sigma \vdash \text{hamming_window}(window_length, \dots) \Rightarrow (window_length), \{(window_length \geq 0)\}}$$

Linear Algebra

torch.svd

torch.svd(input, some=True, compute_uv=True, out=None)	
<p>Hand-drawn diagram illustrating the SVD decomposition: $\text{input} = U \times \text{diag}(S) \times V^T$. Below it, an alternative representation for <code>some=False</code> is shown: $\text{input} = U \times V^T$.</p>	Require
	<ul style="list-style-type: none"> $input = (*, m, n)$ (rank ≥ 2)
	Guarantees
	<ul style="list-style-type: none"> $U = (*, m, m), S = (*, m), V = \dots$ <ul style="list-style-type: none"> if $(\text{compute_uv}) \wedge (\text{some})$ then $V = (*, n, m)$ (as default) otherwise, $V = (*, n, n)$ $y = (U , S , V)$
	Comment
	<ul style="list-style-type: none"> (U, S, V) 형태의 $(Tensor, Tensor, Tensor)$ 출력 반환 $input = U \times \text{diag}(S) \times V^T$ <code>out</code>-텐서 인자가 있는 함수

$\sigma \vdash E \Rightarrow e, c$

$k = \text{rank}(e)$

$u = e[1:k-2]@(e[k-1], e[k-1])$

$s = e[1:k-2]@(e[k-1])$

$v_{tail} = \text{if } (\text{compute_uv}) \wedge (\text{some}) \text{ then } (n, m) \text{ else } (n, n)$

$v = e[1:k-2]@v_{tail}$

$c_{dim} = \{(k \geq 2)\}$

$\sigma \vdash \text{svd}(E, \text{some} = \text{True}, \text{compute_uv} = \text{True}, \text{out} = \text{None}) \Rightarrow (u, s, v), c \cup c_{dim}$

3-tensor tuple로 반환

torch.diag

torch.diag(input, diagonal=0, out=None)	
<p>Hand-drawn diagram illustrating the <code>diag</code> function. For 1D input, it creates a diagonal matrix. For 2D input, it creates a diagonal matrix with the input on the diagonal.</p>	Require
	<ul style="list-style-type: none"> $input = (d_1)$ or (d_1, d_2). (rank 1 or 2) if rank = 1, then there is no requirement. if rank = 2, then $-d_1 \leq \text{diagonal} \leq d_2$
	Guarantees
	<ul style="list-style-type: none"> if rank = 1, then $y = (d_1 + \text{diagonal} , d_1 + \text{diagonal})$ if rank = 2, then $y = (\min\{d_1, d_2, d_1 + \text{diagonal}, d_2 - \text{diagonal}\})$
	Comment
	<ul style="list-style-type: none"> 1차원, 2차원 텐서 입력에 따라 역할이 달라짐 <code>out</code>-텐서 인자가 있는 함수

$\sigma \vdash E \Rightarrow e, c$

$k = \text{rank}(e)$

$c_{dim} = \{(k \in \{1, 2\}) \wedge (k = 1 \vee (-e[1] \leq \text{diagonal} \leq e[2]))\}$

$e' = \text{if } k = 1 \text{ then } (e[1] + |\text{diagonal}|, e[1] + |\text{diagonal}|) \text{ else } (\min\{e[1], e[2], e[1] + \text{diagonal}, e[2] - \text{diagonal}\})$

$\sigma \vdash \text{diag}(E, \text{diagonal} = 0, \text{out} = \text{None}) \Rightarrow e', c \cup c_{dim}$

torch.bilinear, torch.nn.functional.bilinear

torch.nn.functional.bilinear(input1, input2, weight, bias=None)	
	Require
	<ul style="list-style-type: none"> • $input_1 = (d_1, d_2, \dots, d_m, h_1)$ • $input_2 = (d_1, d_2, \dots, d_m, h_2)$ <ul style="list-style-type: none"> – Same ranks with $\mathbf{rank} \geq 1$ and – for all $i \in \{1, 2, \dots, m\}$, $input_1 [i] = input_2 [i]$ • $weight = (s, h_1, h_2)$ • $bias$ is <i>None</i> or $bias = (s)$
	Guarantees
	<ul style="list-style-type: none"> • $y = (d_1, d_2, \dots, d_m, s)$
	Comment
	<ul style="list-style-type: none"> • 특이사항으로 torch.nn.functional.bilinear에서는 bias의 기본값이 <i>None</i>으로 설정되어있지만, torch.bilinear에서는 그렇지 않아서 무조건 값을 넣어줘야합니다.

$$\sigma \vdash E_1 \Rightarrow e_1, c_1$$

$$\sigma \vdash E_2 \Rightarrow e_2, c_2$$

$$\sigma \vdash W \Rightarrow w, c_w$$

$$\sigma \vdash B \Rightarrow b, c_b \quad \text{if } B \text{ is not } None, \text{ otherwise } c_b = \emptyset$$

$$c_{dim} = \{(\mathbf{rank}(e_1) = \mathbf{rank}(e_2)) \wedge (\mathbf{rank}(e_1) \geq 1) \wedge (\mathbf{rank}(w) = 3)\}$$

$$k = \mathbf{rank}(e_1)$$

$$c_{prefix} = \{(\forall i \in \{1, 2, \dots, k-1\}, e_1[i] = e_2[i])\}$$

$$c_{suffix} = \{(e_1[k] = w[2]) \wedge (e_2[k] = w[3]) \wedge (B = None \vee b = (w[1]))\}$$

$$e' = e_1[1:k-1] @ (w[1])$$

$$\sigma \vdash \mathbf{bilinear}(E_1, E_2, W, B = None) \Rightarrow e', c \cup c_{dim} \cup c_{prefix} \cup c_{suffix}$$