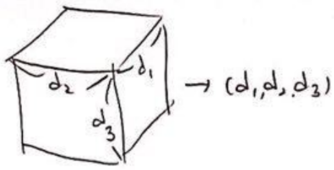
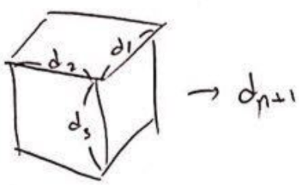
Shape

`torch.Tensor.size`, `torch.Tensor.stride`

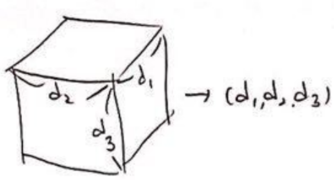
<table>
<tr><td colspan="2"><code>a.size() or a.stride()</code></td></tr>
<tr><td rowspan="4">
</td><td>Require</td></tr>
<tr><td>• $|\mathsf{a}| = (d_1, d_2, \ldots, d_k)$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>• $(d_1, d_2, \ldots, d_k)$를 튜플로 반환</td></tr>
</table>

$$\forall \texttt{ft} \in \{\texttt{size}, \texttt{stride}, \}, \qquad \frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash E.\texttt{ft}() \Rightarrow shapeToTuple(e), c}$$

<table>
<tr><td colspan="2"><code>a.size(n)</code></td></tr>
<tr><td rowspan="4">
</td><td>Require</td></tr>
<tr><td>• $|\mathsf{a}| = (d_1, d_2, \ldots, d_k)$
• $0 \leq n < k$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>• d_{n+1}을 숫자(int)로 반환
• n에 -1이 들어갈 수도 있음</td></tr>
</table>

$$\forall \texttt{ft} \in \{\texttt{size}, \texttt{stride}, \}, \qquad \frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ c' = \{(k \geq 1) \land (0 \leq n < k)\} \end{array}}{\sigma \vdash E.\texttt{ft}(n) \Rightarrow e[n+1], c \cup c'}$$

`torch.Tensor.shape`

<table>
<tr><td colspan="2"><code>a.shape</code></td></tr>
<tr><td rowspan="4">
</td><td>Require</td></tr>
<tr><td>• $|\mathsf{a}| = (d_1, d_2, \ldots, d_k)$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>• $(d_1, d_2, \ldots, d_k)$를 튜플로 반환</td></tr>
</table>

$$\frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash E.\mathsf{shape} \Rightarrow shapeToTuple(e), c}$$

Tensor Declarations

`torch.tensor`

<table>
<tr><td colspan="2"><code>torch.tensor(x)</code></td></tr>
<tr><td rowspan="2"></td><td>Comment</td></tr>
<tr><td><code>numpy</code>나 파이썬 리스트로 선언된 객체를 <code>torch</code>에서 호환가능한 형태로 바꾸는 함수</td></tr>
</table>

$$\frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash \texttt{tensor}(E) \Rightarrow e, c}$$

`torch.range`, `torch.arange`

<table>
<tr><td colspan="2"><code>torch.range(start=0, end, step=1, out=None, ...), torch.arange(...)</code></td></tr>
<tr><td rowspan="6">$(s, s+d, s+2d, \cdots, s+\lfloor \frac{e-s}{d} \rfloor d)$</td><td>Require</td></tr>
<tr><td>• $step \neq 0$
• $(end - start)/step > 0$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>• $|y| = (1 + \lfloor (end - start)/step \rfloor)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>• $(start, start + step, start + 2 \cdot step, \ldots)$를 반환
• out-텐서 인자가 있는 함수
• <code>torch.arange</code>도 똑같은 방식으로 작동함</td></tr>
</table>

$$\forall \texttt{ft} \in \{\texttt{range}, \texttt{arange}\}, \quad \frac{c = \{(d \neq 0) \wedge ((e - s)/d > 0)\}}{\sigma \vdash \texttt{ft}(s, e, d, out = None, ...) \Rightarrow (1 + \lfloor (e - s)/d \rfloor), c} \quad \text{Default: } s = 0, d = 1$$

`torch.linspace`

<table>
<tr><td colspan="2"><code>torch.linspace(start, end, steps=100, out=None, ...)</code></td></tr>
<tr><td rowspan="6">$\underbrace{(s, s+d, s+2d, \cdots, e)}_{\text{총 steps 개}}$</td><td>Require</td></tr>
<tr><td>• $steps \geq 0$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>• $|y| = (steps)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>• $(start, start + d, start + 2d, \ldots, end)$인데 원소가 $steps$개인 텐서 반환
• out-텐서 인자가 있는 함수</td></tr>
</table>

$$\frac{c = \{(steps \geq 0)\}}{\sigma \vdash \texttt{linspace}(s, e, steps = 100, out = None, ...) \Rightarrow (steps), c}$$

torch.zeros, torch.empty, torch.rand, torch.randn

<table>
<tr><td colspan="2"><code>torch.zeros(t1, t2, ..., tl, out=None, ...) or .empty, .rand, .randn</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td></td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = (t_1, t_2, \ldots, t_l)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>입력받은 형태대로 0, uninitialized, uniformly random, gaussian random 텐서를 반환
$(t_1, t_2, \ldots, t_l)$ 입력이 하나의 튜플로 들어오는 경우도 있음
out-텐서 인자가 있는 함수</td></tr>
</table>

$$\forall \texttt{ft} \in \{\texttt{zeros}, \texttt{empty}, \texttt{rand}, \texttt{randn}\}, \quad \frac{}{\sigma \vdash \texttt{ft}(t_1, t_2, \ldots, t_l, out = None) \Rightarrow (t_1, t_2, \ldots, t_l), \emptyset}$$

$$\forall \texttt{ft} \in \{\texttt{zeros}, \texttt{empty}, \texttt{rand}, \texttt{randn}\}, \quad \frac{}{\sigma \vdash \texttt{ft}((t_1, t_2, \ldots, t_l), out = None) \Rightarrow (t_1, t_2, \ldots, t_l), \emptyset}$$

torch.randint

<table>
<tr><td colspan="2"><code>torch.randint(low=0, high, shape, ..., out=None, ...)</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td>$low < high$
shape가 well-defined인 텐서 shape. (스칼라 타입은 X)</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = shape$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>low부터 $high - 1$까지 랜덤한 정수 할당
out-텐서가 있는 함수</td></tr>
</table>

$$\frac{}{\sigma \vdash \texttt{randint}(low = 0, high, (t_1, t_2, \ldots, t_l), ..., out = None, ...) \Rightarrow (t_1, t_2, \ldots, t_l), \{(low < high)\}}$$

torch.randperm

<table>
<tr><td colspan="2"><code>torch.randperm(n, out=None, ...)</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td>$n \geq 0$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = (n)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>out-텐서가 있는 함수</td></tr>
</table>

$$\frac{}{\sigma \vdash \texttt{randperm}(n, out = None, ...) \Rightarrow (n), \{(n \geq 0)\}}$$

torch.full

<table>
<tr><td colspan="2"><code>torch.full((t1, t2, ..., tl), fill_value, out=None, ...)</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td></td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = (t_1, t_2, \ldots, t_l)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>모든 원소가 $fill_value$로 채워진 텐서 반환
<code>empty</code>랑 비슷해보이는데, size인자를 항상 튜플로만 받음
out-텐서가 있는 함수</td></tr>
</table>

$$\frac{}{\sigma \vdash \texttt{full}((t_1, t_2, \ldots, t_l), fill\_value, out = None) \Rightarrow (t_1, t_2, \ldots, t_l), \emptyset}$$

torch.normal

<table>
<tr><td colspan="2"><code>torch.normal(mean, std, (t1, t2, ..., tl), out=None</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td>$std > 0$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = (t_1, t_2, \ldots, t_l)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td><code>empty</code>랑 비슷해보이는데, size인자를 항상 튜플로만 받음
out-텐서가 있는 함수</td></tr>
</table>

$$\frac{}{\sigma \vdash \texttt{normal}(mean, std, (t_1, t_2, \ldots, t_l), out = None) \Rightarrow (t_1, t_2, \ldots, t_l), \{(std > 0)\}}$$

torch.Tensor.new_empty

<table>
<tr><td colspan="2"><code>x.new_empty((t1, t2, ..., tl), ...)</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td></td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = (t_1, t_2, \ldots, t_l)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td><code>empty</code>랑 비슷해보이는데, <code>Tensor</code> 클래스에서만 사용 가능하고, out 인자도 없으며, size인자를 항상 튜플로만 받음
텐서 <code>x</code>의 shape도 똑같이 변함</td></tr>
</table>

$$\frac{}{\sigma \vdash x.\texttt{new\_empty}((t_1, t_2, \ldots, t_l), ...) \Rightarrow (t_1, t_2, \ldots, t_l), \emptyset}$$

x의 shape도 똑같이 $(t_1, t_2, \ldots, t_l)$로 변함

`torch.Tensor.new_full`

<table>
<tr><td colspan="2"><code>x.new_full((t1, t2, ..., tl), fill_value, ...)</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td></td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = (t_1, t_2, \ldots, t_l)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td><code>new_empty</code>랑 비슷해보이는데, <code>fill_value</code> 인자가 추가적으로 있음
텐서 <code>x</code>의 shape도 똑같이 변함</td></tr>
</table>

$$\frac{}{\sigma \vdash x.\texttt{new\_full}((t_1, t_2, \ldots, t_l), fill\_value...) \Rightarrow (t_1, t_2, \ldots, t_l), \emptyset}$$

x의 shape도 똑같이 $(t_1, t_2, \ldots, t_l)$로 변함

`torch.Tensor.clone`

<table>
<tr><td colspan="2"><code>x.clone(...)</code></td></tr>
<tr><td rowspan="4"></td><td>Require</td></tr>
<tr><td></td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = |x|$</td></tr>
</table>

$$\frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash E.\texttt{clone}(...) \Rightarrow e, c}$$

`torch.zeros_like`, `torch.empty_like`, `torch.rand_like`, `torch.randn_like`

<table>
<tr><td colspan="2"><code>torch.zeros_like(input, ...) or .empty_like, .rand_like, .randn_like</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td></td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = |input|$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>입력받은 텐서와 shape이 같은 0, uninitialized, uniformly random, gaussian random 텐서를 반환
out-텐서 인자가 없음!</td></tr>
</table>

$$\forall \texttt{ft} \in \{\texttt{zeros\_like}, \texttt{empty\_like}, \texttt{rand\_like}, \texttt{randn\_like}\}, \quad \frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash \texttt{ft}(E, ...) \Rightarrow e, c}$$

`torch.full_like`

`torch.full_like(input, fill_value, out=None, ...)`	
	Require
	Guarantees
	• $\lvert y \rvert = \lvert input \rvert$
	Comment
	• 입력받은 텐서와 shape이 같은 $fill_value$로 가득찬 텐서 반환 • 희한하게 이건 out-텐서 인자가 있음..

$$\frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash \texttt{full\_like}(E, fill\_value, out = None...) \Rightarrow e, c}$$

`torch.scalar_tensor`

`torch.scalar_tensor(scalar, ...)`	
5 → torch.tensor(5)	Require
	Guarantees
	• $\lvert y \rvert = ()$
	Comment
	• $scalar$ 값 하나만 가지는 `rank-0` 텐서 반환

$$\frac{}{\sigma \vdash \texttt{scalar\_tensor}(scalar, ...) \Rightarrow (), \emptyset}$$

`torch.eye`

`torch.eye(n, m=None, out=None, ...)`	
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	Require
	• $n \geq 0$ • $m = None$ or $m \geq 0$
	Guarantees
	• $\lvert y \rvert = \begin{cases} (n, n) & \text{if } m \text{ is } None \\ (n, m) & \text{otherwise} \end{cases}$
	Comment
	• 곱셈의 항등원 I_n를 리턴 • out 인자가 있는 함수

$$\frac{\begin{array}{l} e = \texttt{if } m = None \texttt{ then } (n, n) \texttt{ else } (n, m) \\ c = \{(n \geq 0) \land (m = None \lor m \geq 0)\} \end{array}}{\sigma \vdash \texttt{eye}(n, m, out = None, ...) \Rightarrow e, c}$$

Same Shape, Elementwise Operators

All these builtin functions `torch.*` are used with `torch.ft(input, out=None)` that output the same shapes of the `inputs`.

- round, floor, ceil
- exp, log, log10, log2, log1p, sigmoid
- sqrt, rsqrt
- cos, sin, tan, angle
- sign, neg, frac
- torch.Tensor.contiguous
  - 텐서 객체에서 사용 가능한 함수.
  - 똑같은 내용물이지만, 메모리 상에서 원소들이 연속하도록 배치해주므로 같은 shape을 반환
  - 즉, 얘는 `a.contiguous()` 이런 식으로 많이 쓰임

input 인자는 무조건 텐서 type이어야 합니다. (스칼라 X)

$$\frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash \texttt{ft}(E, out = None) \Rightarrow e, c}$$

`torch.clamp(input, min, max, out=None)` 함수는 *input* 텐서의 모든 원소가 $min \leq \cdot \leq max$가 성립하도록 만들어주는 것으로, 역시 텐서 shape이 보존됩니다.

$$\frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash \texttt{clamp}(E, min, max, out = None) \Rightarrow e, c}$$

torch.threshold, torch.nn.functional.threshold

torch.threshold(input, threshold, value, inplace=False)	
	Require
	Guarantees
	• $\lvert y\rvert = \lvert input\rvert$

$$\frac{\sigma \vdash E \Rightarrow e, c}{\sigma \vdash \texttt{threshold}(E, threshold, value, inplace = False) \Rightarrow e, c}$$

torch.softmax, torch.nn.functional.softmax, torch.log_softmax, torch.nn.functional.log_softmax

torch.softmax(input, dim=None, _stacklevel=3, dtype=None) or torch.log_softmax (same arguments)	
	Require
	• $-\texttt{rank}(\lvert input\rvert) \leq dim < \texttt{rank}(\lvert input\rvert)$
	Guarantees
	• $\lvert y\rvert = \lvert input\rvert$
	Comment
	• 원래 $dim = None$일 경우 last dimension이 기본값으로 들어갔지만, 이제는 $dim = None$으로 쓰는 방식이 deprecated 되었다고 나와있음.

$$\forall \texttt{ft} \in \{\texttt{softmax}, \texttt{log\_softmax}\}, \quad \frac{\sigma \vdash E \Rightarrow e, c \qquad c' = \{(-\texttt{rank}(e) \leq dim < \texttt{rank}(e))\}}{\sigma \vdash \texttt{ft}(E, dim = None, \_stacklevel = 3, dtype = None) \Rightarrow e, c \cup c'}$$

torch.inverse

<table>
<tr><td colspan="2"><code>torch.inverse(input, out=None)</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td>$|input| = (d_1, d_2, \ldots, d_k)$ $k \geq 2$ $d_{k-1} = d_k$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = |input|$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>정사각행렬의 곱의 역원
out-텐서 인자가 있는 함수</td></tr>
</table>

$$\frac{\begin{array}{l}\sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ c' = \{(k \geq 2) \wedge (e[k-1] = e[k])\}\end{array}}{\sigma \vdash \texttt{inverse}(E, out = None) \Rightarrow e, c \cup c'}$$

torch.flip

<table>
<tr><td colspan="2"><code>torch.flip(input, dims)</code></td></tr>
<tr><td rowspan="4"></td><td>Require</td></tr>
<tr><td>$|input| = (d_1, d_2, \ldots, d_k)$ $dims$ is a tuple or a list, and $\forall elt \in dims,\ -k \leq elt < k$.</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>$|y| = |input|$</td></tr>
</table>

$$\frac{\begin{array}{l}\sigma \vdash E \Rightarrow e, c \\ c' = \{(\forall elt \in dims,\ -\texttt{rank}(e) \leq dim < \texttt{rank}(e))\}\end{array}}{\sigma \vdash \texttt{flip}(E, dims) \Rightarrow e, c \cup c'}$$

tuple 형태로 반환

Broadcasted Shape, Binary Operators

All these builtin functions `torch.*` are used with `torch.ft(input, other, out=None)` that output the same shapes of the `inputs`.

- `eq`, `le`, `lt`, `ge`, `gt`
- `mul`, `div`, `fmod`, `atan2`

한 가지 독특한 성질은 *input*, *other* 인자가 스칼라로 들어오면 [ ]-shape 텐서로 변환된 후 계산됩니다.

- 즉, `torch.mul`$(1, 2)$`.shape`은 [ ] 입니다.

$$\frac{\begin{array}{c}\sigma \vdash E_1 \Rightarrow e_1, c_1 \\ \sigma \vdash E_2 \Rightarrow e_2, c_2\end{array}}{\sigma \vdash \texttt{ft}(E_1, E_2, out = None) \Rightarrow broadcast(e_1, e_2), c_1 \cup c_2 \cup broadcastable(e_1, e_2)}$$

The following builtin functions `torch.*` are slightly different. `torch.ft(input, other, out=None, alpha=1)` that output the same shapes of the `inputs`. (Additional *alpha* option!)

- `add`, `sub`
  - Calculates broadcasted $input \pm \alpha \cdot output$

마찬가지로 *input*, *other* 인자로 스칼라가 들어오면 [ ]-shape 텐서로 변환된 후 계산됩니다.

$$\frac{\sigma \vdash E_1 \Rightarrow e_1, c_1 \quad \sigma \vdash E_2 \Rightarrow e_2, c_2}{\sigma \vdash \texttt{ft}(E_1, E_2, out = None, alpha = 1) \Rightarrow broadcast(e_1, e_2), c_1 \cup c_2 \cup broadcastable(e_1, e_2)}$$

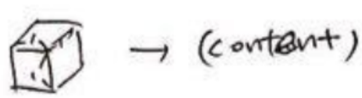
`torch.pow`

`torch.pow(input, exponent, out=None)`	
	Require
	• [*input* or *exponent* are scalar] or $broadcastable(\lvert input \rvert, \lvert exponent \rvert)$
	Guarantees
	• $broadcast(\lvert input \rvert, \lvert exponent \rvert)$
	Comment
	• elementwise binary operator라는 점에서 `mul`과 비슷하나, 인자 이름(exponent)이 달라서 따로 뺐습니다. (`pow(a, exponent=b)`같은 문제..) • 마찬가지로 인자에 스칼라가 들어오면 [ ]-shape 텐서로 변환된 후 계산됩니다. • *out* 인자가 있는 함수

$$\frac{\sigma \vdash E_1 \Rightarrow e_1, c_1 \quad \sigma \vdash E_2 \Rightarrow e_2, c_2}{\sigma \vdash \texttt{pow}(E_1, E_2, out = None) \Rightarrow broadcast(e_1, e_2), c_1 \cup c_2 \cup broadcastable(e_1, e_2)}$$

Indexing

`torch.Tensor.item`

`a.item()`	
	Require
	• $\lvert a \rvert = ()$ or $\lvert a \rvert = (1, 1, 1, \ldots, 1)$
	Guarantees
	• $\lvert y \rvert = e_n$
	Comment
	• Singular element tensor의 원소(스칼라 타입으로 반환)

$$\frac{\sigma \vdash E \Rightarrow e, c \quad k = \texttt{rank}(e) \quad c' = \{(\forall i = 1, 2, \ldots, k,\ e[i] = 1)\}}{\sigma \vdash E.\texttt{item}() \Rightarrow e_n, c \cup c'}$$

Shape Polymorphism

`torch.Tensor.view`

`a.view(n1, n2, ..., nl)`

Require

- $|\mathsf{a}| = (d_1, d_2, \ldots, d_k)$
- $reshapable((d_1, d_2, \ldots, d_k), (n_1, n_2, \ldots, n_l))$

Guarantees

- $(n_1, n_2, \ldots, n_l)$ as tensor

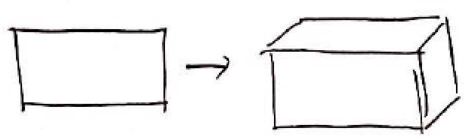
Comment

- 단, $n_1, n_2, \ldots, n_l$이 하나의 튜플로 입력이 들어올 수도 있음

$$\frac{\sigma \vdash \texttt{reshape}(E, n_1, n_2, \ldots, n_l) \Rightarrow e, c}{\sigma \vdash E.\texttt{view}(n_1, n_2, \ldots, n_l) \Rightarrow e, c}$$

`torch.Tensor.expand`

`a.expand(n1, n2, ..., nl)`



Require

- $|\mathsf{a}| = (d_1, d_2, \ldots, d_k)$
- $k \leq l$
- $\forall i = 1, 2, \ldots, l-k,\ (n_i > 0)$
- $\forall i = l-k+1, l-k+2, \ldots, l,\ [(n_i = -1) \text{ or } ((n_i > 0) \text{ and } ((d_{i-(l-k)} = 1) \text{ or } (d_{i-(l-k)} = n_i)))]$

Guarantees

- $(m_1, m_2, \ldots, m_l)$ as tensor where
  - $m_1 = n_1, m_2 = n_2, \cdots, m_{l-k} = n_{l-k}$
  - $m_{l-k+i} = \texttt{if } (n_{l-k+i} = -1) \texttt{ then } d_i \texttt{ else } n_{l-k+i}$ for the rests

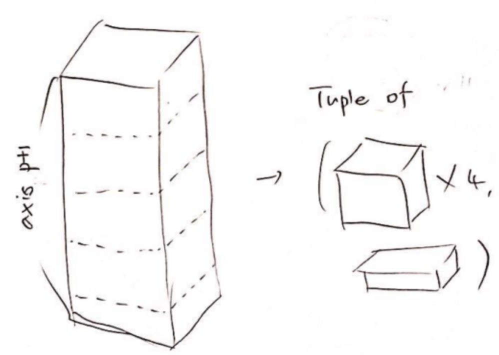
Comment

- 일방향 broadcast 함수; $(n_1, n_2, \ldots, n_l)$ shape이 목표
- $n_i = -1$인 경우, 본래 크기만큼 그대로 유지
- 단, $n_1, n_2, \ldots, n_l$이 하나의 튜플로 입력이 들어올 수도 있음

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ m_1 = n_1, m_2 = n_2, \cdots, m_{l-k} = n_{l-k} \\ \forall i \in \{1, 2, \ldots, k\},\ m_{l-k+i} = \texttt{if } (n_{l-k+i} = -1) \texttt{ then } d_i \texttt{ else } n_{l-k+i} \\ c_{base} = \{(k \leq l) \wedge (n_1 > 0) \wedge (n_2 > 0) \wedge \cdots \wedge (n_{l-k} > 0)\} \\ c_{expandable} = \{(\forall i \in \{1, 2, \ldots, k\},\ (n_{l-k+i} = -1) \vee [(n_{l-k+i} > 0) \wedge ((d_i = 1) \vee (d_i = n_{l-k+i}))])\} \end{array}}{\sigma \vdash E.\texttt{expand}(n_1, n_2, \ldots, n_l) \Rightarrow (m_1, m_2, \ldots, m_l), c \cup c_{base} \cup c_{expandable}}$$

`torch.split`

`torch.split(tensor, split_size_or_section, dim=0)`



Require

- $|tensor| = (d_1, d_2, \ldots, d_k)$
- $k \geq 1$
- $0 \leq dim < k$

Guarantees

- 아래 proof tree와 같이 $dim + 1$번째 axis가 최대 $split_size_or_section$개의 원소를 가지도록 $\lceil d_{dim+1}/\cdot \rceil$개 튜플 형태로 반환

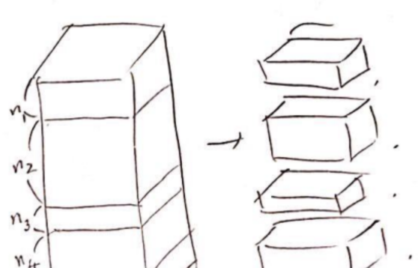
Comment

- Concat의 반대 역할 함수
- Divisible 여부 assert하지 않음
- 학습 및 테스트 데이터를 배치단위로 쪼개는 용도로 쓰일 것으로 추측

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ e_1 = e[1{:}p]@(n)@e[p+2{:}k] \\ e_2 = e[1{:}p]@(n)@e[p+2{:}k] \\ \quad \cdots \\ e_{l-1} = e[1{:}p]@(n)@e[p+2{:}k] \\ e_l = e[1{:}p]@(n')@e[p+2{:}k] \quad \text{where } e[p+1] = n(l-1) + n',\ 0 < n' \leq n \\ c' = \{(k \geq 1) \wedge (0 \leq p < k)\} \end{array}}{\sigma \vdash \texttt{split}(E, n, p = 0) \Rightarrow (e_1, e_2, \ldots, e_l), c \cup c'}$$

l-원소 tuple 형태로 반환

`torch.split(input, [n1, n2, ..., nl], dim=0)`



Require

- $|input| = (d_1, d_2, \ldots, d_k)$
- $k \geq 1$
- $0 \leq dim < k$
- $d_{dim+1} = n_1 + n_2 + \cdots + n_l$

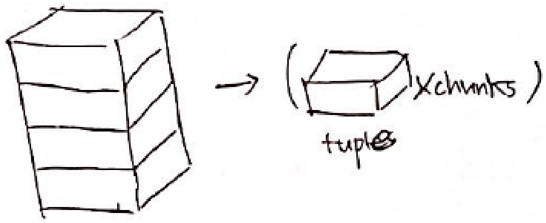
Guarantees

- 아래 proof tree와 같이 $dim + 1$번째 axis의 크기가 $n_1, n_2, \ldots, n_l$인 l개의 텐서 튜플을 반환
- n_i의 합과 d_{dim+1}이 같은지 assert

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ e_1 = e[1{:}p]@(n_1)@e[p+2{:}k] \\ e_2 = e[1{:}p]@(n_2)@e[p+2{:}k] \\ \quad \cdots \\ e_l = e[1{:}p]@(n_l)@e[p+2{:}k] \\ c' = \{(k \geq 1) \wedge (0 \leq x < k) \wedge (e[p+1] = n_1 + n_2 + \cdots + n_l)\} \end{array}}{\sigma \vdash \texttt{split}(E, [n_1, n_2, \ldots, n_l], p = 0) \Rightarrow (e_1, e_2, \ldots, e_l), c \cup c'}$$

l-원소 tuple 형태로 반환

torch.chunk

<table>
<tr><td colspan="2"><code>torch.chunk(input, chunks, dim=0)</code></td></tr>
<tr><td rowspan="6">
</td><td>Require</td></tr>
<tr><td>

- $|input| = (d_1, d_2, \ldots, d_k)$
- $k \geq 1$
- $chunks > 0$
- $0 \leq dim < k$

</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>

- Proof tree와 같이 최소 $chunks$개로 dim 축이 잘라진 텐서 튜플 반환

</td></tr>
<tr><td>Comment</td></tr>
<tr><td>

- `split`과 비슷하나 쪼개는 개수를 명시한다는 점이 다름

</td></tr>
</table>

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ d = \min\{e[p+1], chunks\} \\ n = \lceil e[p+1]/d \rceil \\ e_1 = e[1{:}p]@(n)@e[p+2{:}k] \\ e_2 = e[1{:}p]@(n)@e[p+2{:}k] \\ \quad \cdots \\ e_{l-1} = e[1{:}p]@(n)@e[p+2{:}k] \\ e_l = e[1{:}p]@(n')@e[p+2{:}k] \quad \text{where } e[p+1] = n(l-1)+n',\ 0 < n' \leq n \\ c' = \{(k \geq 1) \wedge (chunks > 0) \wedge (0 \leq p < k)\} \end{array}}{\sigma \vdash \texttt{chunk}(E, chunks, p=0) \Rightarrow (e_1, e_2, \ldots, e_l), c \cup c'} \qquad l\text{-원소 tuple 형태로 반환}$$

Dynamic Operations (Cannot Predict Statically)

torch.nonzero

<table>
<tr><td colspan="2"><code>torch.nonzero(input, out=None, as_tuple=False)</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td></td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>

- if as_tuple, then $|y| = (countNonzeros(input), \texttt{rank}(|input|))$
- otherwise, then $|y|$ is $\texttt{rank}(|input|)$-tuple of tensors shaped $(countNonzeros(input))$

</td></tr>
<tr><td>Comment</td></tr>
<tr><td>

- $countNonzeros$는 주어진 텐서에서 0이 아닌 항의 개수를 구하는 것
- Nonzero 인덱스 번호들을 반환함.
- out-텐서 인자가 있는 함수

</td></tr>
</table>

$$\frac{\begin{array}{c} \sigma \vdash E_1 \Rightarrow e_1, c_1 \\ \sigma \vdash E_2 \Rightarrow e_2, c_2 \end{array}}{\sigma \vdash \texttt{pow}(E_1, E_2, out = None) \Rightarrow broadcast(e_1, e_2), c_1 \cup c_2 \cup broadcastable(e_1, e_2)}$$

Example Codes:

```
print(torch.nonzero(torch.tensor([[0, 1, 0], [0, 1, 1]])))
  # output: tensor([[0, 1], [1, 1], [1, 2]])
```

```
print(torch.nonzero(torch.tensor([[0, 1, 0], [0, 1, 1]]), as_tuple=True))
  # output: (tensor([0, 1, 1]), tensor([1, 1, 2]))
```