Generalized shapes: Block sparsity, MaskedTensor, NestedTensor
Results below based on PyTorch nightly from Nov 1st 2022 on a single A100 with CUDA 11.7
<Github repo link with scripts to reproduce results>

**F.linear and block sparsity**
Increase performance with block sparse matrix multiplication

Access float32 BSR matrix multiplication routines backed by cuSPARSE

<table>
<thead>
<tr>
<th>CSR format</th>
<th>BSR format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 0</td>
<td>0 1 2 0</td>
</tr>
<tr>
<td>2 4 5 6</td>
<td>2 4 5 6</td>
</tr>
<tr>
<td>6 0 2 0</td>
<td>6 0 2 0</td>
</tr>
</tbody>
</table>

**MaskedTensor**
Compress unspecified elements

Distinguish between specified and unspecified values. Orthogonalize sparse storage and masked computation.

**Max with MaskedTensor**

```
import torch

k = torch.randn(64, 256, 1024, device='cuda')
b = torch.randn(1024, device='cuda')
def run_benchmark(blocksize, p):
    A = create_blocked_tensor(1024, 1024)
    A_sparse = A.to_sparse_bsr(blocksize=blocksize)
    dense_time = benchmark(F.linear, x, A, b)
    sparse_time = benchmark(F.linear, x, A_sparse, b)

    return dense_time, sparse_time
```

**MaskedTensor: Layout change**

```
x = torch.randn(64, 256, 1024, device='cuda')
pt = nn.MultiheadAttention(embed_dim=1024, num_heads=8, batch_first=True)
pt = pt.eval().half().cuda()
sq = [lens[randint(0, len(lens))] for _ in range(256)]
x = nested_tensor([torch.randn(s, embed_dim) for s in sq])
y = x.to_padded_tensor(0)
padding_ratio = x.numel() / y.numel()
padded_time = benchmark(pt, y, y, y, mask, need_weights=False)
nested_time = benchmark(pt, x, x, x, mask, need_weights=False)
```

**NestedTensor**
MultiheadAttention with xFormers and FlashAttention on enwik9

Access fused scaled dot product attention kernels on variably sized data. Comparison to idealized speedup of 1 / (padding ratio).

```
pt = nn.MultiheadAttention(embed_dim=1024, num_heads=8, batch_first=True)
pt = pt.eval().half().cuda()
sq = [lens[randint(0, len(lens))] for _ in range(256)]
x = nested_tensor([torch.randn(s, embed_dim) for s in sq])
y = x.to_padded_tensor(0)
padding_ratio = x.numel() / y.numel()
padded_time = benchmark(pt, y, y, y, mask, need_weights=False)
nested_time = benchmark(pt, x, x, x, mask, need_weights=False)
```