Embedding

CLASS torch.nn.Embedding(*num_embeddings*, *embedding_dim*, *padding_idx=None*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*, *sparse=False*, *_weight=None*, *_freeze=False*, *device=None*, *dtype=None*) [SOURCE]

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int, optional*) – If specified, the entries at `padding_idx` do not contribute to the gradient; therefore, the embedding vector at `padding_idx` is not updated during training, i.e. it remains as a fixed "pad". For a newly constructed Embedding, the embedding vector at `padding_idx` will default to all zeros, but can be updated to another value to be used as the padding vector.
- **max_norm** (*float, optional*) – If given, each embedding vector with norm larger than `max_norm` is renormalized to have norm `max_norm`.
- **norm_type** (*float, optional*) – The p of the p-norm to compute for the `max_norm` option. Default `2`.
- **scale_grad_by_freq** (*bool, optional*) – If given, this will scale gradients by the inverse of frequency of the words in the mini-batch. Default `False`.
- **sparse** (*bool, optional*) – If `True`, gradient w.r.t. `weight` matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

Variables

weight (*Tensor*) – the learnable weights of the module of shape (num_embeddings, embedding_dim) initialized from $\mathcal{N}(0, 1)$

Shape:

- Input: $(*)$, IntTensor or LongTensor of arbitrary shape containing the indices to extract
- Output: $(*, H)$, where $*$ is the input shape and $H = \text{embedding\_dim}$

• NOTE

Keep in mind that only a limited number of optimizers support sparse gradients: currently it's `optim.SGD` (*CUDA* and *CPU*), `optim.SparseAdam` (*CUDA* and *CPU*) and `optim.Adagrad` (*CPU*)

• NOTE

When `max_norm` is not `None`, `Embedding`'s forward method will modify the `weight` tensor in-place. Since tensors needed for gradient computations cannot be modified in-place, performing a differentiable operation on `Embedding.weight` before calling `Embedding`'s forward method requires cloning `Embedding.weight` when `max_norm` is not `None`. For example:

```
n, d, m = 3, 5, 7
embedding = nn.Embedding(n, d, max_norm=1.0)
W = torch.randn((m, d), requires_grad=True)
idx = torch.tensor([1, 2])
a = embedding.weight.clone() @ W.t()  # weight must be cloned for this to be differentiable
b = embedding(idx) @ W.t()  # modifies weight in-place
out = (a.unsqueeze(0) + b.unsqueeze(1))
loss = out.sigmoid().prod()
loss.backward()
```

Examples:

```
>>> # an Embedding module containing 10 tensors of size 3
>>> embedding = nn.Embedding(10, 3)
>>> # a batch of 2 samples of 4 indices each
>>> input = torch.LongTensor([[1, 2, 4, 5], [4, 3, 2, 9]])
>>> embedding(input)
tensor([[[-0.0251, -1.6902,  0.7172],
         [-0.6431,  0.0748,  0.6969],
         [ 1.4970,  1.3448, -0.9685],
         [-0.3677, -2.7265, -0.1685]],

        [[ 1.4970,  1.3448, -0.9685],
         [ 0.4362, -0.4004,  0.9400],
         [-0.6431,  0.0748,  0.6969],
         [ 0.9124, -2.3616,  1.1151]]])


>>> # example with padding_idx
>>> embedding = nn.Embedding(10, 3, padding_idx=0)
>>> input = torch.LongTensor([[0, 2, 0, 5]])
>>> embedding(input)
tensor([[[ 0.0000,  0.0000,  0.0000],
         [ 0.1535, -2.0309,  0.9315],
         [ 0.0000,  0.0000,  0.0000],
         [-0.1655,  0.9897,  0.0635]]])

>>> # example of changing `pad` vector
>>> padding_idx = 0
>>> embedding = nn.Embedding(3, 3, padding_idx=padding_idx)
>>> embedding.weight
Parameter containing:
tensor([[ 0.0000,  0.0000,  0.0000],
        [-0.7895, -0.7089, -0.0364],
        [ 0.6778,  0.5803,  0.2678]], requires_grad=True)
>>> with torch.no_grad():
...     embedding.weight[padding_idx] = torch.ones(3)
>>> embedding.weight
Parameter containing:
tensor([[ 1.0000,  1.0000,  1.0000],
        [-0.7895, -0.7089, -0.0364],
        [ 0.6778,  0.5803,  0.2678]], requires_grad=True)
```

CLASSMETHOD from_pretrained(*embeddings*, *freeze=True*, *padding_idx=None*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*, *sparse=False*) [SOURCE]

Create Embedding instance from given 2-dimensional FloatTensor.

Parameters

- **embeddings** (*Tensor*) – FloatTensor containing weights for the Embedding. First dimension is being passed to Embedding as `num_embeddings`, second as `embedding_dim`.
- **freeze** (*bool*, *optional*) – If `True`, the tensor does not get updated in the learning process. Equivalent to `embedding.weight.requires_grad = False`. Default: `True`
- **padding_idx** (*int*, *optional*) – If specified, the entries at `padding_idx` do not contribute to the gradient; therefore, the embedding vector at `padding_idx` is not updated during training, i.e. it remains as a fixed "pad".
- **max_norm** (*float*, *optional*) – See module initialization documentation.
- **norm_type** (*float*, *optional*) – See module initialization documentation. Default `2`.
- **scale_grad_by_freq** (*bool*, *optional*) – See module initialization documentation. Default `False`.
- **sparse** (*bool*, *optional*) – See module initialization documentation.

Examples:

```
>>> # FloatTensor containing pretrained weights
>>> weight = torch.FloatTensor([[1, 2.3, 3], [4, 5.1, 6.3]])
>>> embedding = nn.Embedding.from_pretrained(weight)
>>> # Get embeddings for index 1
>>> input = torch.LongTensor([1])
>>> embedding(input)
tensor([[ 4.0000,  5.1000,  6.3000]])
```

‹ Previous Next ›

Resources

Find development resources and get your questions answered

View Resources