### **PyTorch functions with their sub-functions and descriptions** :would be extremely extensive because PyTorch is a massive library with hundreds of functions covering tensors, neural networks, optimizers, and various utilities. Instead, here's a structured overview of the primary PyTorch modules, including essential functions and sub-functions, along with brief descriptions.

**1. torch Module (Core Tensor Operations)**

**Tensor Creation Functions**

* **torch.tensor(data):** Creates a tensor from data (list, NumPy array, etc.).
* **torch.zeros(size):** Returns a tensor filled with zeros.
* **torch.ones(size):** Returns a tensor filled with ones.
* **torch.arange(start, end, step):** Creates a tensor with values in the given range.
* **torch.linspace(start, end, steps):** Creates a tensor with linearly spaced values.
* **torch.rand(size):** Returns a tensor with random values uniformly distributed in [0, 1).
* **torch.randn(size):** Returns a tensor with random values sampled from a normal distribution.

**Tensor Manipulation Functions**

* **torch.reshape(tensor, shape):** Reshapes the tensor without changing its data.
* **torch.transpose(tensor, dim0, dim1):** Swaps two dimensions.
* **torch.squeeze(tensor):** Removes dimensions of size 1.
* **torch.unsqueeze(tensor, dim): Adds** a dimension of size 1.
* **torch.cat(tensors, dim):** Concatenates multiple tensors along a specified dimension.
* **torch.stack(tensors, dim):** Stacks multiple tensors along a specified dimension.

**Mathematical Operations**

* **torch.add(x, y):** Element-wise addition.
* **torch.sub(x, y):** Element-wise subtraction.
* **torch.mul(x, y):** Element-wise multiplication.
* **torch.div(x, y):** Element-wise division.
* **torch.matmul(x, y):** Matrix multiplication.
* **torch.pow(tensor, exponent):** Raises each element to the given power.
* **torch.sqrt(tensor):** Computes the square root of each element.

**Reduction Operations**

* **torch.sum(tensor, dim):** Sums tensor elements across specified dimension(s).
* **torch.mean(tensor, dim):** Computes mean along specified dimension.
* **torch.max(tensor, dim):** Returns maximum values along a specified dimension.
* **torch.min(tensor, dim):** Returns minimum values along a specified dimension.
* **torch.argmax(tensor, dim):** Returns indices of the maximum values along a specified dimension.
* **torch.argmin(tensor, dim):** Returns indices of the minimum values along a specified dimension.

**Logical Operations**

* **torch.eq(x, y):** Element-wise equality check.
* **torch.ne(x, y):** Element-wise inequality check.
* **torch.gt(x, y):** Element-wise greater-than check.
* **torch.lt(x, y):** Element-wise less-than check.

**2. torch.nn Module (Neural Network Layers and Functions)**

**Basic Layers**

* **torch.nn.Linear(in\_features, out\_features):** Fully connected linear layer.
* **torch.nn.Conv2d(in\_channels, out\_channels, kernel\_size):** 2D convolutional layer.
* **torch.nn.Conv1d and torch.nn.Conv3d:** 1D and 3D convolutional layers, respectively.
* **torch.nn.MaxPool2d(kernel\_size):** 2D max pooling layer.
* **torch.nn.AvgPool2d(kernel\_size):** 2D average pooling layer.
* **torch.nn.Dropout(p):** Applies dropout for regularization.

**Activation Functions**

* **torch.nn.ReLU():** ReLU activation function.
* **torch.nn.Sigmoid():** Sigmoid activation function.
* **torch.nn.Tanh():** Tanh activation function.
* **torch.nn.Softmax(dim):** Softmax function for classification.
* **torch.nn.LeakyReLU(negative\_slope):** Leaky ReLU activation.

**Loss Functions**

* **torch.nn.MSELoss():** Mean Squared Error loss.
* **torch.nn.CrossEntropyLoss():** Cross-entropy loss for classification.
* **torch.nn.BCELoss():** Binary Cross-Entropy loss.
* **torch.nn.NLLLoss(): Negative** Log-Likelihood loss.

**Normalization Layers**

* **torch.nn.BatchNorm2d(num\_features):** Batch normalization for 2D inputs.
* **torch.nn.LayerNorm(normalized\_shape):** Layer normalization.
* **torch.nn.GroupNorm(num\_groups, num\_channels):** Group normalization.

**Recurrent Layers**

* **torch.nn.RNN(input\_size, hidden\_size):** Basic RNN layer.
* **torch.nn.LSTM(input\_size, hidden\_size):** Long Short-Term Memory layer.
* **torch.nn.GRU(input\_size, hidden\_size):** Gated Recurrent Unit layer.

**3. torch.optim Module (Optimization Algorithms)**

**Optimization Algorithms**

* **torch.optim.SGD(params, lr):** Stochastic Gradient Descent optimizer.
* **torch.optim.Adam(params, lr):** Adam optimizer.
* **torch.optim.Adagrad(params, lr):** Adagrad optimizer.
* **torch.optim.RMSprop(params, lr):** RMSprop optimizer.
* **torch.optim.AdamW(params, lr):** Adam optimizer with weight decay.

**Learning Rate Scheduling**

* **torch.optim.lr\_scheduler.StepLR(optimizer, step\_size, gamma):** Decreases learning rate by a factor of gamma every specified step.
* **torch.optim.lr\_scheduler.ReduceLROnPlateau(optimizer, mode, factor):** Reduces learning rate when a metric has stopped improving.
* **torch.optim.lr\_scheduler.CosineAnnealingLR(optimizer, T\_max):** Cosine annealing learning rate schedule.

**4. torch.autograd Module (Automatic Differentiation)**

* **torch.autograd.grad(outputs, inputs):** Computes gradients of outputs with respect to inputs.
* **torch.autograd.backward(tensor):** Computes the gradient of the tensor.
* **torch.autograd.no\_grad():** Context manager that disables gradient computation.
* **torch.autograd.Function:** Extends PyTorch by defining custom autograd operations.

**5. torch.utils.data Module (Data Handling)**

* **torch.utils.data.Dataset:** Abstract class for creating custom datasets.
* **torch.utils.data.DataLoader:** Provides batching, shuffling, and parallel loading of data.
* **torch.utils.data.TensorDataset:** Dataset wrapping tensors.
* **torch.utils.data.ConcatDataset:** Concatenates multiple datasets.

**6. torchvision Module (Computer Vision)**

* **torchvision.datasets:** Collection of popular datasets such as MNIST, CIFAR-10, etc.
* **torchvision.transforms:** Collection of data transformation functions, e.g., Resize, RandomCrop, ToTensor.
* **torchvision.models:** Pretrained computer vision models (ResNet, VGG, etc.).
* **torchvision.io:** For reading and writing image/video files.

**7. torchtext Module (Natural Language Processing)**

* **torchtext.datasets:** Popular NLP datasets (AG News, IMDb, etc.).
* **torchtext.vocab:** Vocabulary and embedding management (e.g., GloVe, FastText).
* **torchtext.data.Field:** Defines how data should be processed.
* **torchtext.data.BucketIterator:** Iterator with efficient batching for variable-length sequences.

**8. torchaudio Module (Audio Processing)**

* **torchaudio.transforms:** Common audio transformations (e.g., Spectrogram, MelSpectrogram).
* **torchaudio.datasets:** Datasets for audio research (e.g., CommonVoice, YESNO).
* **torchaudio.io:** Audio I/O functions for loading and saving audio data.
* **torchaudio.models:** Pre-trained models and architectures for audio processing.

**9. torch.jit Module (Scripting and Tracing for Optimization)**

* **torch.jit.script:** Compiles PyTorch code for improved performance.
* **torch.jit.trace:** Traces a model and creates a static graph.
* **torch.jit.save:** Saves a scripted/traced model.
* **torch.jit.load:** Loads a scripted/traced model.

**10. torch.distributed Module (Distributed Training)**

* **torch.distributed.init\_process\_group:** Initializes the distributed environment.
* **torch.distributed.all\_reduce:** Aggregates tensors across processes.
* **torch.distributed.scatter:** Distributes elements from one tensor to all processes.
* **torch.distributed.gather:** Gathers tensors from all processes into one.

**11. torch.cuda Module (CUDA Operations for GPUs)**

* **torch.cuda.is\_available():** Checks if CUDA-capable devices are available.
* **torch.cuda.manual\_seed(seed):** Sets the seed for CUDA.
* **torch.cuda.empty\_cache():** Releases cached memory on the GPU.
* **torch.cuda.memory\_allocated():** Checks the current GPU memory usage.

**Other Utility Functions**

* torch.save(): Saves an object to