Applied Deep Learning with Tensorflow and Pytorch Chapter 2

Unless stated otherwise, for each item, the first command is in PyTorch and the second one is in Tensorflow.

Data Manipulation

The Tensor classes both in Tensorflow and PyTorch are similar to NumPy's ndarray with support on GPU and automatic differentiation.

CPU-GPU Support

- **PyTorch:** The device should be defined. It is GPU per default.
device = torch.device("cpu")
x = torch.arange(4.0, device=device)
- **Tensorflow:** To use CPU, tensorflow library should be installed.
To use GPU, tensorflow-gpu library should be installed.

Getting Started

- **Row vector with values from** 0 **to** *n***:** torch.arange(n), tf.range(n)
- **Tensor's shape:** x.shape, x.shape
- **Number of elements:** x.numel(), tf.size(x).numpy()
- **Reshaping to shape** *n***,** *m***:** x.reshape(n, m), tf.reshape(x, (n, m))
- **Tensor with only** 0**s:** torch.zeros((n,m,k)), tf.zeros((n,m,k))
- **Tensor with only** 1**s:** torch.ones((n,m,k)), tf.ones((n,m,k))
- **Reshaping Gaussian distribution:**
torch.randn(n, m), tf.random.normal(shape=[n, m])
- **Specifying all values manually:**
torch.tensor([[l,m,n],[o,p,q]]), tf.constant([[l,m,n],[o,p,q]])

Basic Operations

- **Basic Operations (for both):** +,-,*,/,**
- **Exponentiation:** torch.exp(x), tf.exp(x)
- **Concatenating tensors X, Y over dimension** *n***:**
torch.cat((X, Y), dim=n), tf.concat([X, Y], axis=n)
- **Summing all elements:** X.sum(), tf.reduce_sum(X)

Indexing and Slicing

- **Indexing and Slicing in PyTorch:**
X[m, n] = k, X[m:n, :] = k
- **Indexing and Slicing in Tensorflow:**
X = tf.Variable(X), X[m, n].assign(k)
X[m:n, :].assign(tf.ones(X[m:n,:].shape,dtype = tf.float32) * 12)

Conversion to other Python Objects

- **Converting to Numpy:** X.numpy(), X.numpy()
- **Converting from Numpy:** torch.tensor(X), tf.constant(X)
- **Converting to a scalar from Pytorch or Tensorflow single value Tensor** *a***:** a.item(), float(a), int(a)

Linear Algebra

Scalars, Vectors, Matrices and Tensors

- **Scalar:** torch.tensor([n]), tf.constant([n])
- **Number of elements of vector x:** len(x), len(x)
- **Transpose of matrix X:** X.T, tf.transpose(X)
- **Summation of tensor X and scalar** *a***:** a + X, a + X
- **Multiplication of tensor X and scalar** *a***:** a * X, a * X

Reduction

- **Sum of tensor X over an axis** *n***:**
X.sum(axis=n), tf.reduce_sum(X,axis=n)
- **Sum of tensor X over an axes** *n*,*m***:**
X.sum(axis=[n,m]), tf.reduce_sum(X,axis=[n,m])
- **Mean of tensor X:** X.mean(), tf.reduce_mean(X)
- **Mean of tensor X over an axis** *n***:**
X.mean(axis=n), tf.reduce_mean(X, axis=n)

Non-reduction Sum

- **Mean of tensor X over an axis** *n* **without dimension reduction:**
X.sum(axis=n, keepdims=True),
tf.reduce_sum(X,axis=n, keepdims=True)
- **Cumulative sum of tensor X over an axis** *n***:**
X.cumsum(axis=n), tf.cumsum(X, axis=n)

Product

- **Dot product of two vectors x, y:**
torch.dot(x, y), tf.tensordot(x, y, axes=1)
- **Matrix A - Vector x Product:**
torch.mv(A, x), tf.linalg.matvec(A, x)
- **Matrix A - Matrix B Product:** torch.mm(A, B) , tf.matmul(A, B)

Norm

- L_1 **norm of vector x:** torch.abs(x).sum(), tf.reduce_sum(tf.abs(x))
- L_2 **norm of vector x:** torch.norm(x), tf.norm(x)
- **Frobenius norm of matrix X:** torch.norm(X), tf.norm(X)

Automatic Differentiation

- $\partial y/\partial x$ **for function y = 2x^2 in PyTorch:**
x.requires_grad_(True)
y = 2 * torch.dot(x, x)
y.backward()
x.grad
- $\partial y/\partial x$ **for function y = 2x^2 in Tensorflow:**
x = tf.Variable(x)
with tf.GradientTape() as t:
  y = 2 * tf.tensordot(x, x, axes=1)
x_grad = t.gradient(y, x)
x_grad
- In order to calculate a new gradient, the gradient should be reset to 0, since the gradients are accumulated in PyTorch.
x.grad.zero_()
- In the gradient calculation of z w.r.t x, if we want to treat y (a function of x) as a constant, we can detach y as a new variable u.
u = y.detach(), u = tf.stop_gradient(y)
- One benefit of automatic differentiation is that it also works if the computational graph includes different control flows.

Probability

- PyTorch has a module distributions, TensorFlow has random
- Extensive probabilistic add-on packages:
for PyTorch: Pyro (https://pyro.ai/), for TensorFlow: TensorFlow Probability (https://www.tensorflow.org/probability)
- **Importing Libraries:**
from torch.distributions import multinomial
import tensorflow_probability as tfp