CHEAT SHEET: PYTORCH AND TENSORFLOW

Jacques Bourg

October 4, 2024

Keywords Deep learning frameworks, Pytorch, Tensorflow

1 Imports

```
import torch
```

```
import tensorflow as tf
```

2 GPU availability

```
torch.cuda.is_available()
```

```
logical_gpus =
    tf.config.experimental.list_logical_devices('GPU')
if len(logical_gpus) > 0:
   print("GPU is available and can be used by
       TensorFlow")

memory_growth =
    tf.config.experimental.get_memory_growth()
for device, growth in
    zip(tf.config.list_physical_devices('GPU'),
    memory_growth):
   print(f"{device.name} memory growth:
       {growth}")
```

3 Variables

```
d0 = torch.ones(1)
d1 = torch.ones(2)
d2 = torch.ones(2, 2)
d3 = torch.ones(2, 2, 2)
```

```
d0 = tf.ones((1,))
d1 = tf.ones((2,))
d2 = tf.ones((2, 2))
d3 = tf.ones((2, 2, 2))
```

4 Conversion

```
d0_np = torch.ones(2).numpy()
d0_torch = torch.from_numpy(numpy_array)
```

```
d0_np = tf.ones((2,)).numpy()
d0_tf = tf.convert_to_tensor(numpy_array)
```

5 Basic operations

```
result = tensor1 + tensor2
result = tensor1 * tensor2
result = torch.matmul(tensor1, tensor2)
x_t = x.t()
```

```
result = tf.add(tensor1, tensor2)
result = tf.multiply(tensor1, tensor2)
result = tf.matmul(tensor1, tensor2)
x_t = tf.transpose(x)
```

6 Dimensionality

```
x.size()
x = torch.cat([tensor1, tensor2], dim=0)
y = x.view(a,b,...)
y = x.view(-1,a)
y = x.transpose(a,b)
y = x.permute(*dims)
y = x.unsqueeze(dim)
y = x.unsqueeze(dim=2)
y = x.squeeze()
y = x.squeeze(dim=1)
```

```
tf.shape(x)
x = tf.concat([tensor1, tensor2], axis=0)
y = tf.reshape(x, new_shape)
y = tf.reshape(x, (-1, a))
y = tf.transpose(x, perm)
y = tf.transpose(x, perm)
y = tf.expand_dims(x, axis)
y = tf.expand_dims(x, axis=2)
y = tf.squeeze(x)
y = tf.squeeze(x, axis=1)
```

7 Automatic differentiation

```
x = torch.randn(2, 2, requires_grad=True)
y = x.pow(2).sum()
y.backward()
print(x.grad)
```

```
with tf.GradientTape() as tape:
  y = x**2

grads = tape.gradient(y, x)
```

8 Neural network layers and activation functions

```
import torch.nn as nn
linear_layer = nn.Linear(input_size,
    output_size)
conv_layer = nn.Conv2d(in_channels,
    out_channels, kernel_size)
nn.ReLU()
nn.Sigmoid()
nn.Tanh()
nn.MaxPoolXd(s)
nn.BatchNormXd

nn.RNN/LSTM/GRU
nn.Dropout(p=0.5, inplace=False)
nn.Dropout2d(p=0.5, inplace=False)
nn.Embedding(num_embeddings, embedding_dim)
```

```
import tensorflow.keras.layers as layers
linear_layer = layers.Dense(units=output_size)
conv_layer =
    layers.Conv2D(filters=out_channels,
    kernel_size=kernel_size)
tf.keras.layers.ReLU()
tf.keras.layers.Sigmoid()
tf.keras.layers.Tanh()
tf.keras.layers.MaxPoolXd(pool_size=s,
    padding='valid')
tf.keras.layers.BatchNormalization()
tf.keras.layers.SimpleRNN/LSTM/GRU
tf.keras.layers.Dropout(rate=0.5)
tf.keras.layers.SpatialDropout2D(rate=0.5)
tf.keras.layers.Embedding(input_dim=num_embeddings,
    output_dim=embedding_dim)
```

9 Networks

```
class Net(nn.Module):
    def __init__(self, input_size, output_size,
         hidden_sizes):
        super(Net, self).__init__()
        layers = [nn.Linear(input_size,
            hidden_sizes[0])]
        for i in range(1, len(hidden_sizes)):
            layers.append(nn.ReLU())
            layers.append(nn.Linear(hidden_sizes[i-1],
                hidden_sizes[i]))
        layers.append(nn.ReLU())
        layers.append(nn.Linear(hidden_sizes[-1],
            output_size))
        self.sequential = nn.Sequential(*layers)

    def forward(self, x):
        return self.sequential(x)

net = Net(input_size=10, output_size=2,
    hidden_sizes=[5, 10])
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

net = Sequential()
net.add(Dense(units=5, input_dim=10,
    activation='relu'))
net.add(Dense(units=10, activation='relu'))
net.add(Dense(units=2))

net.compile(optimizer='adam', loss='mse')
```

10 Model and optimizer

```
import torch.optim as optim
model = nn.Sequential(
    nn.Linear(input_size, hidden_size),
    nn.ReLU(),
    nn.Linear(hidden_size, output_size))
optimizer = optim.Adam(model.parameters(),
    lr=0.001, betas=(0.9, 0.999), eps=1e-8,
    weight_decay=1e-5)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(hidden_size,
        activation='relu',
        input_shape=(input_size,)),
    tf.keras.layers.Dense(output_size)])
model.compile(optimizer='adam', loss='mse')
```

11 Train a network

```
for i, (inputs, labels) in
    enumerate(train_loader):
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if i % 100 == 0:
            print(f"Epoch {epoch+1}, Batch
                {i+1}, Loss: {loss.item():.4f}"))
```

```
for epoch in range(num_epochs):
    for inputs, labels in train_dataset:
        loss = model.train_on_batch(inputs,
            labels)
        if i % 100 == 0:
            print(f"Epoch {epoch+1}, Batch
                {i+1}, Loss: {loss:.4f}")
```