Introduction to PyTorch

Georgia Tech CS 4650

(Part of tutorials and slides are made by Nihal Singh, Jingfeng Yang)

Outline

- **Pytorch**
  - **Introduction**
  - **Basics**
  - **Examples**

Introduction to PyTorch

What is PyTorch?

- Open source machine learning library
- Developed by Facebook's AI Research lab
- It leverages the power of GPUs
- Automatic computation of gradients
- Makes it easier to test and develop new ideas.

Other libraries?

Microsoft CNTK

Caffe

Caffe2

PYTORCH

Chainer

Keras

TensorFlow

theano

dy/net

mxnet

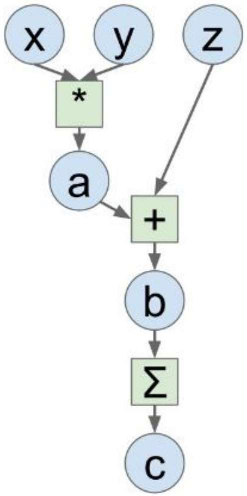
GLUON

Why PyTorch?

- It is pythonic - concise, close to Python conventions
- Strong GPU support
- Autograd - automatic differentiation
- Many algorithms and components are already implemented
- Similar to NumPy

Why PyTorch?

Computation Graph



Numpy

```python
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```

Tensorflow

```python
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

PyTorch

```python
import torch

N, D = 3, 4

x = torch.rand((N, D),requires_grad=True)
y = torch.rand((N, D),requires_grad=True)
z = torch.rand((N, D),requires_grad=True)

a =x * y
b =a + z
c=torch.sum(b)

c.backward()
```

Getting Started with PyTorch

Installation

Via Anaconda/Miniconda:

```
conda install pytorch
```

Via pip:

```
pip3 install torch
```

PyTorch Basics

iPython Notebook Tutorial

bit.ly/pytorchbasics

Tensors

Tensors are similar to NumPy’s ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

Common operations for creation and manipulation of these Tensors are similar to those for ndarrays in NumPy. (rand, ones, zeros, indexing, slicing, reshape, transpose, cross product, matrix product, element wise multiplication)

Tensors

Attributes of a tensor 't':

- `t= torch.randn(1)`

`requires_grad`- making a trainable parameter

- By default False
- Turn on:
    - `t.requires_grad_()` or
    - `t = torch.randn(1, requires_grad=True)`
  - Accessing tensor value:
    - t.data
  - Accessingtensor gradient
    - t.grad

`grad_fn`- history of operations for autograd

- t.grad_fn

```python
import torch

N, D = 3, 4

x = torch.rand((N, D),requires_grad=True)
y = torch.rand((N, D),requires_grad=True)
z = torch.rand((N, D),requires_grad=True)

a = x * y
b = a + z
c=torch.sum(b)

c.backward()

print(c.grad_fn)
print(x.data)
print(x.grad)
```

```
<SumBackward0 object at 0x7fd0cb970cc0>
tensor([[0.4118, 0.2576, 0.3470, 0.0240],
        [0.7797, 0.1519, 0.7513, 0.7269],
        [0.8572, 0.1165, 0.8596, 0.2636]])
tensor([[0.6855, 0.9696, 0.4295, 0.4961],
        [0.3849, 0.0825, 0.7400, 0.0036],
        [0.8104, 0.8741, 0.9729, 0.3821]])
```

Loading Data, Devices and CUDA

Numpy arrays to PyTorch tensors

- `torch.from_numpy(x_train)`
- `Returns a cpu tensor!`

PyTorch tensor to numpy

- `t.numpy()`

Using GPU acceleration

- `t.to()`
- Sends to whatever device (cuda or cpu)

Fallback to cpu if gpu is unavailable:

- `torch.cuda.is_available()`

Check cpu/gpu tensor OR numpy array ?

- `type(t)` or `t.type()` returns
  - numpy.ndarray
  - torch.Tensor
    - CPU - torch.cpu.FloatTensor
    - GPU - torch.cuda.FloatTensor

Autograd

- Automatic Differentiation Package
- Don't need to worry about partial differentiation, chain rule etc.
  - `backward()` does that
- Gradients are accumulated for each step by default:
  - Need to zero out gradients after each update
  - `tensor.grad_zero()`

```python
# Create tensors.
x = torch.tensor(1., requires_grad=True)
w = torch.tensor(2., requires_grad=True)
b = torch.tensor(3., requires_grad=True)

# Build a computational graph.
y = w * x + b    # y = 2 * x + 3

# Compute gradients.
y.backward()

# Print out the gradients.
print(x.grad)    # x.grad = 2
print(w.grad)    # w.grad = 1
print(b.grad)    # b.grad = 1
```

Optimizer and Loss

Optimizer

- Adam, SGD etc.
- An optimizer takes the parameters we want to update, the learning rate we want to use along with other hyper-parameters and performs the updates

Loss

- Various predefined loss functions to choose from
- L1, MSE, Cross Entropy

```python
a = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
b = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)


# Defines a SGD optimizer to update the parameters
optimizer = optim.SGD([a, b], lr=lr)

for epoch in range(n_epochs):
    yhat = a + b * x_train_tensor
    error = y_train_tensor - yhat
    loss = (error ** 2).mean()

    loss.backward()

    optimizer.step()

    optimizer.zero_grad()

print(a, b)
```

Model

In PyTorch, a model is represented by a regular Python class that inherits from the Module class.

- Two components
    - `__init__(self)` : it defines the parts that make up the model- in our case, two parameters, a and b
    - `forward(self, x)` : it performs the actual computation, that is, it outputs a prediction, given the inputx

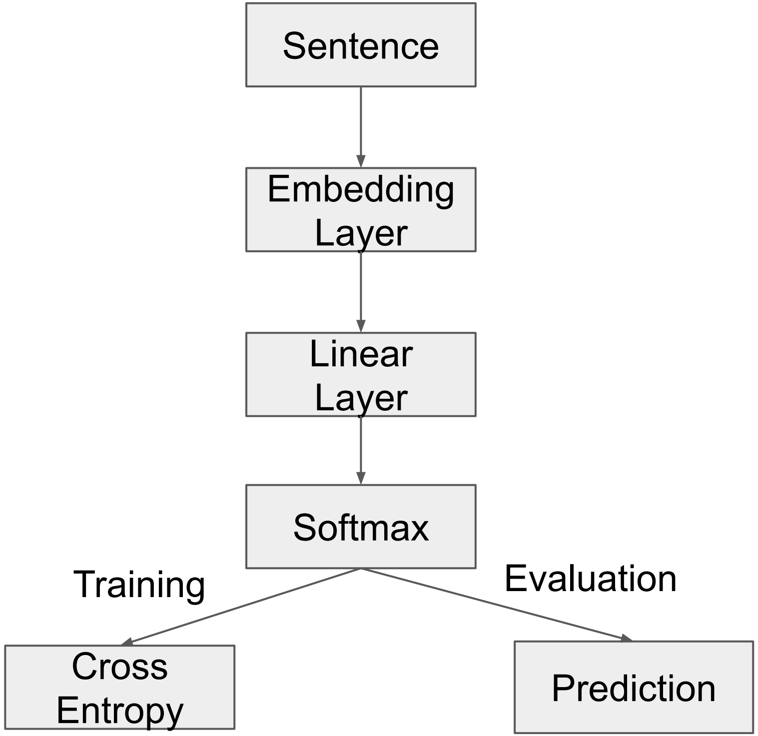
```
class ManualLinearRegression(nn.Module):
    def __init__(self):
        super().__init__()
        # To make "a" and "b" real parameters of the model, we need to wrap them with nn.Parameter
        self.a = nn.Parameter(torch.randn(1, requires_grad=True, dtype=torch.float))
        self.b = nn.Parameter(torch.randn(1, requires_grad=True, dtype=torch.float))

    def forward(self, x):
        # Computes the outputs / predictions
        return self.a + self.b * x
```

PyTorch Example
(neural bag-of-words (ngrams) text classification)

bit.ly/pytorchexample

Overview


Sentence
Embedding Layer
Linear Layer
Softmax
Training
Evaluation
Cross Entropy
Prediction

Design Model

- Initilaize modules.
- Use linear layer here.
- Can change it to RNN, CNN, Transformer etc.

- Randomly initilaize parameters

- Foward pass

```
import torch.nn as nn
import torch.nn.functional as F
class TextSentiment(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super().__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=True)
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc(embedded)
```

Preprocess

- Build and preprocess dataset

```
import torch
import torchtext
from torchtext.datasets import text_classification
NGRAMS = 2
import os
if not os.path.isdir('./.data'):
    os.mkdir('./.data')
train_dataset, test_dataset = text_classification.DATASETS['AG_NEWS'](
    root='./.data', ngrams=NGRAMS, vocab=None)
BATCH_SIZE = 16
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

- Build vocabulary

```
VOCAB_SIZE = len(train_dataset.get_vocab())
EMBED_DIM = 32
NUN_CLASS = len(train_dataset.get_labels())
model = TextSentiment(VOCAB_SIZE, EMBED_DIM, NUN_CLASS).to(device)
```

Preprocess

- One example of dataset:

```
print(train_dataset[0])
```

```
(2, tensor([    572,     564,       2,    2326,   49106,     150,      88,       3,
           1143,      14,      32,      15,      32,      16,  443749,       4,
            572,     499,      17,      10,  741769,       7,  468770,       4,
             52,    7019,    1050,     442,       2,   14341,     673,  141447,
         326092,   55044,    7887,     411,    9870,  628642,      43,      44,
            144,     145,  299709,  443750,   51274,     703,   14312,      23,
        1111134,  741770,  411508,  468771,    3779,   86384,  135944,  371666,
           4052]))
```

- Create batch (Used in SGD)
- Choose pad or not (Using [PAD])

```
def generate_batch(batch):
    label = torch.tensor([entry[0] for entry in batch])
    text = [entry[1] for entry in batch]
    offsets = [0] + [len(entry) for entry in text]
    # torch.Tensor.cumsum returns the cumulative sum
    # of elements in the dimension dim.
    # torch.Tensor([1.0, 2.0, 3.0]).cumsum(dim=0)

    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
    text = torch.cat(text)
    return text, offsets, label
```

Training each epoch

Iterable batches

Before each optimization, make previous gradients zeros

Forward pass to compute loss

Backforward propagation to compute gradients and update parameters

After each epoch, do learning rate decay (optional)

```
from torch.utils.data import DataLoader

def train_func(sub_train_):

    # Train the model
    train_loss = 0
    train_acc = 0
    data = DataLoader(sub_train_, batch_size=BATCH_SIZE, shuffle=True,
                      collate_fn=generate_batch)
    for i, (text, offsets, cls) in enumerate(data):
        optimizer.zero_grad()
        text, offsets, cls = text.to(device), offsets.to(device), cls.to(device)
        output = model(text, offsets)
        loss = criterion(output, cls)
        train_loss += loss.item()
        loss.backward()
        optimizer.step()
        train_acc += (output.argmax(1) == cls).sum().item()

    # Adjust the learning rate
    scheduler.step()

    return train_loss / len(sub_train_), train_acc / len(sub_train_)
```

Test process

Do not need back propagation or parameter update !

```
def test(data_):
    loss = 0
    acc = 0
    data = DataLoader(data_, batch_size=BATCH_SIZE, collate_fn=generate_batch)
    for text, offsets, cls in data:
        text, offsets, cls = text.to(device), offsets.to(device), cls.to(device)
        with torch.no_grad():
            output = model(text, offsets)
            loss = criterion(output, cls)
            loss += loss.item()
            acc += (output.argmax(1) == cls).sum().item()

    return loss / len(data_), acc / len(data_)
```

The whole training process

- Use CrossEntropyLoss() as the criterion. The input is the output of the model. First do logsoftmax, then compute cross-entropy loss.
- Use SGD as optimizer.
- Use exponential decay to decrease learning rate

Print information to monitor the training process

```
import time
from torch.utils.data.dataset import random_split
N_EPOCHS = 5
min_valid_loss = float('inf')

criterion = torch.nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=4.0)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1, gamma=0.9)

train_len = int(len(train_dataset) * 0.95)
sub_train_, sub_valid_ = \
    random_split(train_dataset, [train_len, len(train_dataset) - train_len])

for epoch in range(N_EPOCHS):

    start_time = time.time()
    train_loss, train_acc = train_func(sub_train_)
    valid_loss, valid_acc = test(sub_valid_)

    secs = int(time.time() - start_time)
    mins = secs / 60
    secs = secs % 60

    print('Epoch: %d' %(epoch + 1), " | time in %d minutes, %d seconds" %(mins, secs))
    print(f'\tLoss: {train_loss:.4f}(train)\t|\tAcc: {train_acc * 100:.1f}%(train)')
    print(f'\tLoss: {valid_loss:.4f}(valid)\t|\tAcc: {valid_acc * 100:.1f}%(valid)')
```

Evaluation with test dataset or random news

```
print('Checking the results of test dataset...')
test_loss, test_acc = test(test_dataset)
print(f'\tLoss: {test_loss:.4f}(test)\t|\tAcc: {test_acc * 100:.1f}%(test)')
```

```
import re
from torchtext.data.utils import ngrams_iterator
from torchtext.data.utils import get_tokenizer

ag_news_label = {1 : "World",
                 2 : "Sports",
                 3 : "Business",
                 4 : "Sci/Tec"}

def predict(text, model, vocab, ngrams):
    tokenizer = get_tokenizer("basic_english")
    with torch.no_grad():
        text = torch.tensor([vocab[token]
                            for token in ngrams_iterator(tokenizer(text), ngrams)])
        output = model(text, torch.tensor([0]))
        return output.argmax(1).item() + 1
```

```
ex_text_str = "MEMPHIS, Tenn. – Four days ago, Jon Rahm was \
    enduring the season’s worst weather conditions on Sunday at The \
    Open on his way to a closing 75 at Royal Portrush, which \
    considering the wind and the rain was a respectable showing. \
    Thursday’s first round at the WGC-FedEx St. Jude Invitational \
    was another story. With temperatures in the mid-80s and hardly any \
    wind, the Spaniard was 13 strokes better in a flawless round. \
    Thanks to his best putting performance on the PGA Tour, Rahm \
    finished with an 8-under 62 for a three-stroke lead, which \
    was even more impressive considering he’d never played the \
    front nine at TPC Southwind."

vocab = train_dataset.get_vocab()
model = model.to("cpu")

print("This is a %s news" %ag_news_label[predict(ex_text_str, model, vocab, 2)])
```