GCN - Kipf

File Edit View Insert Runtime Tools Help

Comment Share a

Files

+ Code + Text

RAM Disk Editing

```
..
sample_data
  README.md
  anscombe.json
  california_housing_test.csv
  california_housing_train.csv
  cora.cites
  cora.content
  mnist_test.csv
  mnist_train_small.csv
```

Disk 55.18 GB available

[74]
```
import numpy as np
import scipy.sparse as sp
import torch
```

[75]
```
def encode_onehot(labels):
    classes = set(labels)
    classes_dict = {c: np.identity(len(classes))[i, :] for i, c in
                    enumerate(classes)}
    labels_onehot = np.array(list(map(classes_dict.get, labels)),
                             dtype=np.int32)
    return labels_onehot
```

[76]
```
def load_data(path="/content/sample_data/cora", dataset="cora"):
    """Load citation network dataset (cora only for now)"""
    print('Loading {} dataset...'.format(dataset))

    idx_features_labels = np.genfromtxt("{}.content".format(path, dataset),
                                        dtype=np.dtype(str))
    features = sp.csr_matrix(idx_features_labels[:, 1:-1], dtype=np.float32)
    labels = encode_onehot(idx_features_labels[:, -1])

    # build graph
    idx = np.array(idx_features_labels[:, 0], dtype=np.int32)
    idx_map = {j: i for i, j in enumerate(idx)}
    edges_unordered = np.genfromtxt("{}.cites".format(path, dataset),
                                    dtype=np.int32)
    edges = np.array(list(map(idx_map.get, edges_unordered.flatten())),
                     dtype=np.int32).reshape(edges_unordered.shape)
    adj = sp.coo_matrix((np.ones(edges.shape[0]), (edges[:, 0], edges[:, 1])),
                        shape=(labels.shape[0], labels.shape[0]),
                        dtype=np.float32)

    # build symmetric adjacency matrix
    adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)

    features = normalize(features)
    adj = normalize(adj + sp.eye(adj.shape[0]))

    idx_train = range(140)
    idx_val = range(200, 500)
    idx_test = range(500, 1500)

    features = torch.FloatTensor(np.array(features.todense()))
    labels = torch.LongTensor(np.where(labels)[1])
    adj = sparse_mx_to_torch_sparse_tensor(adj)

    idx_train = torch.LongTensor(idx_train)
    idx_val = torch.LongTensor(idx_val)
    idx_test = torch.LongTensor(idx_test)

    return adj, features, labels, idx_train, idx_val, idx_test
```

[77]
```
def normalize(mx):
    """Row-normalize sparse matrix"""
    rowsum = np.array(mx.sum(1))
    r_inv = np.power(rowsum, -1).flatten()
    r_inv[np.isinf(r_inv)] = 0.
    r_mat_inv = sp.diags(r_inv)
    mx = r_mat_inv.dot(mx)
    return mx
```

[78]
```
def accuracy(output, labels):
    preds = output.max(1)[1].type_as(labels)
    correct = preds.eq(labels).double()
    correct = correct.sum()
    return correct / len(labels)
```

[79]
```
def sparse_mx_to_torch_sparse_tensor(sparse_mx):
    """Convert a scipy sparse matrix to a torch sparse tensor."""
    sparse_mx = sparse_mx.tocoo().astype(np.float32)
    indices = torch.from_numpy(
        np.vstack((sparse_mx.row, sparse_mx.col)).astype(np.int64))
    values = torch.from_numpy(sparse_mx.data)
    shape = torch.Size(sparse_mx.shape)
    return torch.sparse.FloatTensor(indices, values, shape)
```

[80]
```
import math
import torch

from torch.nn.parameter import Parameter
from torch.nn.modules.module import Module
```

[81]
```
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
```

```
            output = torch.spmm(adj, support)
            if self.bias is not None:
                return output + self.bias
            else:
                return output

        def __repr__(self):
            return self.__class__.__name__ + ' (' \
                   + str(self.in_features) + ' -> ' \
                   + str(self.out_features) + ')'
```

[82]
```
import torch.nn as nn
import torch.nn.functional as F
```

[83]
```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc2(x, adj)
        return F.log_softmax(x, dim=1)
```

[84]
```
from __future__ import division
from __future__ import print_function

import time
import argparse
import numpy as np

import torch
import torch.nn.functional as F
import torch.optim as optim
```

[85]
```
# Training settings
parser = argparse.ArgumentParser()
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='Disables CUDA training.')
parser.add_argument('--fastmode', action='store_true', default=False,
                    help='Validate during training pass.')
parser.add_argument('--seed', type=int, default=42, help='Random seed.')
parser.add_argument('--epochs', type=int, default=200,
                    help='Number of epochs to train.')
parser.add_argument('--lr', type=float, default=0.01,
                    help='Initial learning rate.')
parser.add_argument('--weight_decay', type=float, default=5e-4,
                    help='Weight decay (L2 loss on parameters).')
parser.add_argument('--hidden', type=int, default=16,
                    help='Number of hidden units.')
parser.add_argument('--dropout', type=float, default=0.5,
                    help='Dropout rate (1 - keep probability).')
```

```
_StoreAction(option_strings=['--dropout'], dest='dropout', nargs=None, const=None, default=0.5, type=<class 'float'>, choices=None, help='Dropout rate (1 - keep probability).', metavar=None)
```

```
args, unknown = parser.parse_known_args()
args.cuda = not args.no_cuda and torch.cuda.is_available()
```

[88]
```
np.random.seed()
torch.manual_seed(args.seed)
if args.cuda:
    torch.cuda.manual_seed(args.seed)
```

[89]
```
adj, features, labels, idx_train, idx_val, idx_test = load_data()
```

```
Loading cora dataset...
```

[90]
```
model = GCN(nfeat=features.shape[1],
            nhid=args.hidden,
            nclass=labels.max().item() + 1,
            dropout=args.dropout)
optimizer = optim.Adam(model.parameters(),
                       lr=args.lr, weight_decay=args.weight_decay)
```

[91]
```
if args.cuda:
    model.cuda()
    features = features.cuda()
    adj = adj.cuda()
    labels = labels.cuda()
    idx_train = idx_train.cuda()
    idx_val = idx_val.cuda()
    idx_test = idx_test.cuda()
```

[92]
```
def train(epoch):
    t = time.time()
    model.train()
    optimizer.zero_grad()
    output = model(features, adj)
    loss_train = F.nll_loss(output[idx_train], labels[idx_train])
    acc_train = accuracy(output[idx_train], labels[idx_train])
    loss_train.backward()
    optimizer.step()

    if not args.fastmode:
        # Evaluate validation set performance separately,
        # deactivates dropout during validation run.
        model.eval()
        output = model(features, adj)

    loss_val = F.nll_loss(output[idx_val], labels[idx_val])
    acc_val = accuracy(output[idx_val], labels[idx_val])
    print('Epoch: {:04d}'.format(epoch+1),
          'loss_train: {:.4f}'.format(loss_train.item()),
          'acc_train: {:.4f}'.format(acc_train.item()),
          'loss_val: {:.4f}'.format(loss_val.item()),
          'acc_val: {:.4f}'.format(acc_val.item()),
          'time: {:.4f}s'.format(time.time() - t))
```

```
[93] def test():
         model.eval()
         output = model(features, adj)
         loss_test = F.nll_loss(output[idx_test], labels[idx_test])
         acc_test = accuracy(output[idx_test], labels[idx_test])
         print("Test set results:",
               "loss= {:.4f}".format(loss_test.item()),
               "accuracy= {:.4f}".format(acc_test.item()))
```

```
[94] # Train model
     t_total = time.time()
     for epoch in range(args.epochs):
         train(epoch)
     print("Optimization Finished!")
     print("Total time elapsed: {:.4f}s".format(time.time() - t_total))
```

```
Epoch: 0145 loss_train: 0.5972 acc_train: 0.9000 loss_val: 0.7945 acc_val: 0.8033 time: 0.0051s
Epoch: 0146 loss_train: 0.5483 acc_train: 0.9286 loss_val: 0.7917 acc_val: 0.8067 time: 0.0051s
Epoch: 0147 loss_train: 0.5459 acc_train: 0.8857 loss_val: 0.7894 acc_val: 0.8133 time: 0.0052s
Epoch: 0148 loss_train: 0.5877 acc_train: 0.9071 loss_val: 0.7867 acc_val: 0.8200 time: 0.0051s
Epoch: 0149 loss_train: 0.5656 acc_train: 0.8929 loss_val: 0.7847 acc_val: 0.8200 time: 0.0050s
Epoch: 0150 loss_train: 0.5762 acc_train: 0.8643 loss_val: 0.7826 acc_val: 0.8200 time: 0.0052s
Epoch: 0151 loss_train: 0.5811 acc_train: 0.9429 loss_val: 0.7802 acc_val: 0.8200 time: 0.0052s
Epoch: 0152 loss_train: 0.5494 acc_train: 0.8929 loss_val: 0.7781 acc_val: 0.8200 time: 0.0051s
Epoch: 0153 loss_train: 0.5332 acc_train: 0.8929 loss_val: 0.7757 acc_val: 0.8167 time: 0.0051s
Epoch: 0154 loss_train: 0.5374 acc_train: 0.9214 loss_val: 0.7723 acc_val: 0.8167 time: 0.0051s
Epoch: 0155 loss_train: 0.5433 acc_train: 0.8786 loss_val: 0.7695 acc_val: 0.8067 time: 0.0051s
Epoch: 0156 loss_train: 0.5333 acc_train: 0.9214 loss_val: 0.7670 acc_val: 0.8033 time: 0.0052s
Epoch: 0157 loss_train: 0.5070 acc_train: 0.9143 loss_val: 0.7647 acc_val: 0.8033 time: 0.0071s
Epoch: 0158 loss_train: 0.5275 acc_train: 0.8929 loss_val: 0.7629 acc_val: 0.8000 time: 0.0080s
Epoch: 0159 loss_train: 0.5589 acc_train: 0.8929 loss_val: 0.7615 acc_val: 0.8000 time: 0.0050s
Epoch: 0160 loss_train: 0.5386 acc_train: 0.9071 loss_val: 0.7601 acc_val: 0.8000 time: 0.0051s
Epoch: 0161 loss_train: 0.5476 acc_train: 0.9000 loss_val: 0.7595 acc_val: 0.8000 time: 0.0051s
Epoch: 0162 loss_train: 0.5161 acc_train: 0.9071 loss_val: 0.7584 acc_val: 0.8000 time: 0.0050s
Epoch: 0163 loss_train: 0.5324 acc_train: 0.9429 loss_val: 0.7579 acc_val: 0.8033 time: 0.0055s
Epoch: 0164 loss_train: 0.5326 acc_train: 0.9000 loss_val: 0.7562 acc_val: 0.8067 time: 0.0051s
Epoch: 0165 loss_train: 0.5239 acc_train: 0.8857 loss_val: 0.7532 acc_val: 0.8067 time: 0.0051s
Epoch: 0166 loss_train: 0.4794 acc_train: 0.9143 loss_val: 0.7496 acc_val: 0.8067 time: 0.0051s
Epoch: 0167 loss_train: 0.4924 acc_train: 0.9286 loss_val: 0.7463 acc_val: 0.8100 time: 0.0049s
Epoch: 0168 loss_train: 0.5163 acc_train: 0.9000 loss_val: 0.7432 acc_val: 0.8100 time: 0.0050s
Epoch: 0169 loss_train: 0.4899 acc_train: 0.9071 loss_val: 0.7406 acc_val: 0.8100 time: 0.0060s
Epoch: 0170 loss_train: 0.5400 acc_train: 0.8571 loss_val: 0.7384 acc_val: 0.8100 time: 0.0049s
Epoch: 0171 loss_train: 0.4957 acc_train: 0.8929 loss_val: 0.7368 acc_val: 0.8067 time: 0.0052s
Epoch: 0172 loss_train: 0.5064 acc_train: 0.9357 loss_val: 0.7352 acc_val: 0.8067 time: 0.0050s
Epoch: 0173 loss_train: 0.5353 acc_train: 0.9071 loss_val: 0.7336 acc_val: 0.8067 time: 0.0054s
Epoch: 0174 loss_train: 0.5033 acc_train: 0.9000 loss_val: 0.7324 acc_val: 0.8067 time: 0.0051s
Epoch: 0175 loss_train: 0.4756 acc_train: 0.9143 loss_val: 0.7315 acc_val: 0.8067 time: 0.0069s
Epoch: 0176 loss_train: 0.5159 acc_train: 0.8929 loss_val: 0.7314 acc_val: 0.8033 time: 0.0053s
Epoch: 0177 loss_train: 0.4964 acc_train: 0.9214 loss_val: 0.7319 acc_val: 0.8067 time: 0.0054s
Epoch: 0178 loss_train: 0.4936 acc_train: 0.9429 loss_val: 0.7330 acc_val: 0.8067 time: 0.0052s
Epoch: 0179 loss_train: 0.5320 acc_train: 0.8929 loss_val: 0.7338 acc_val: 0.8067 time: 0.0054s
Epoch: 0180 loss_train: 0.4773 acc_train: 0.9286 loss_val: 0.7341 acc_val: 0.8100 time: 0.0052s
Epoch: 0181 loss_train: 0.5001 acc_train: 0.9214 loss_val: 0.7337 acc_val: 0.8033 time: 0.0053s
Epoch: 0182 loss_train: 0.4684 acc_train: 0.9214 loss_val: 0.7328 acc_val: 0.8033 time: 0.0051s
Epoch: 0183 loss_train: 0.4816 acc_train: 0.9143 loss_val: 0.7311 acc_val: 0.8033 time: 0.0063s
Epoch: 0184 loss_train: 0.4677 acc_train: 0.9429 loss_val: 0.7281 acc_val: 0.8033 time: 0.0097s
Epoch: 0185 loss_train: 0.4986 acc_train: 0.9143 loss_val: 0.7244 acc_val: 0.8033 time: 0.0054s
Epoch: 0186 loss_train: 0.4449 acc_train: 0.9286 loss_val: 0.7216 acc_val: 0.8000 time: 0.0052s
Epoch: 0187 loss_train: 0.5297 acc_train: 0.9143 loss_val: 0.7187 acc_val: 0.8000 time: 0.0054s
Epoch: 0188 loss_train: 0.4796 acc_train: 0.9143 loss_val: 0.7162 acc_val: 0.8000 time: 0.0052s
Epoch: 0189 loss_train: 0.4669 acc_train: 0.9286 loss_val: 0.7145 acc_val: 0.8067 time: 0.0053s
Epoch: 0190 loss_train: 0.4769 acc_train: 0.9143 loss_val: 0.7135 acc_val: 0.8033 time: 0.0053s
Epoch: 0191 loss_train: 0.4716 acc_train: 0.9143 loss_val: 0.7132 acc_val: 0.8067 time: 0.0055s
Epoch: 0192 loss_train: 0.4409 acc_train: 0.9357 loss_val: 0.7140 acc_val: 0.8067 time: 0.0068s
Epoch: 0193 loss_train: 0.4927 acc_train: 0.9214 loss_val: 0.7163 acc_val: 0.8100 time: 0.0052s
Epoch: 0194 loss_train: 0.4548 acc_train: 0.9286 loss_val: 0.7177 acc_val: 0.8100 time: 0.0050s
Epoch: 0195 loss_train: 0.4770 acc_train: 0.9071 loss_val: 0.7181 acc_val: 0.8100 time: 0.0054s
Epoch: 0196 loss_train: 0.4270 acc_train: 0.9571 loss_val: 0.7181 acc_val: 0.8100 time: 0.0053s
Epoch: 0197 loss_train: 0.4593 acc_train: 0.9500 loss_val: 0.7166 acc_val: 0.8067 time: 0.0052s
Epoch: 0198 loss_train: 0.4690 acc_train: 0.9286 loss_val: 0.7153 acc_val: 0.8067 time: 0.0050s
Epoch: 0199 loss_train: 0.4627 acc_train: 0.9286 loss_val: 0.7149 acc_val: 0.8067 time: 0.0051s
Epoch: 0200 loss_train: 0.4525 acc_train: 0.9286 loss_val: 0.7135 acc_val: 0.8067 time: 0.0051s
Optimization Finished!
Total time elapsed: 3.9699s
```

```
[95] # Testing
     test()
```

```
Test set results: loss= 0.7517 accuracy= 0.8360
```