Deep Learning

Graph Convolutional Network - Lab

U Kang
Seoul National University

In This Lecture

- Implement graph convolutional network for node classification problem
- Data from Cora citation dataset

Outline

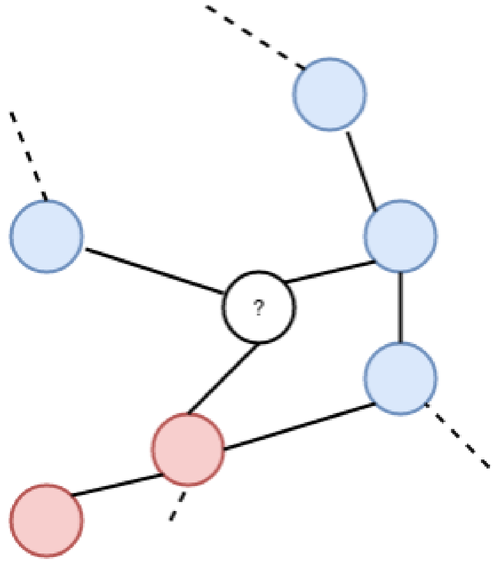
- **Introduction**
- Data
- Preprocessing Codes
- Answers

Motivation

- Graph is frequently used data structure
- Node classification is a classification problem given graph structure
- If we can utilize not only node features but also graph structure, e.g. edge information, we will get better performance

Goals

- Classify each node's label in graph structure



Problem Definition

- **Given:** Graph structure and node features from cora citation network
- **Predict:** classify each node to one of seven classes

Outline

- [x] Introduction
- [ ] **Data**
- [ ] Preprocessing Codes
- [ ] Answers

Training Dataset

- Graph from cora dataset
- Cora dataset is a citation network
  - 2708 scientific publications (node)
  - 5429 links (edge)
  - Each node has 0/1-valued vector indicating absence/presence of corresponding word from dictionary (1433 unique words are in dictionary)

Providing data

- Text files
- cora/cora.cites
  - Each line describes a link between papers
  - <ID of cited paper> < ID of citing paper>
  - i.e., A B means B -> A
- cora/cora.content
  - Each line describes a paper
  - <paper id> <word attributes>+ <class label>

Outline

- ☑ Introduction
- ☑ Data
- ☐ **Preprocessing Codes**
- ☐ Answers

Import libraries

- We will use libraries below, so install these using 'pip install'
  - tensorflow
  - numpy
  - sklearn
  - pandas
  - tdqm

Loading the Dataset

- Read csv file using pandas
  - csv file can easily handle complex data types, e.g. number and string simultaneously

```
cora_content = pd.read_csv('./cora/cora.content', sep='\t', header=None)
cora_content.head()
```

	0	1	2	3	4	5	6	7	8	9	...	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434
0	31336	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	Neural_Networks
1	1061127	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	Rule_Learning
2	1106406	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Reinforcement_Learning
3	13195	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Reinforcement_Learning
4	37879	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Probabilistic_Methods

Preprocess (1)

- Extract <paper id>, <node features>, <labels> from dataframe

```
ids = cora_content[0].values # paper(node) ids
vecs = cora_content[cora_content.columns[1:1434]].values # node features
labels = cora_content[1434].values # node label

print(np.unique(labels))
```

```
['Case_Based' 'Genetic_Algorithms' 'Neural_Networks'
 'Probabilistic_Methods' 'Reinforcement_Learning' 'Rule_Learning' 'Theory']
```

Preprocess (2)

- Apply one hot encoding on label using sklearn

```python
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
```

```python
# node label one hot encoding
labels_onehot = LabelEncoder().fit_transform(labels)
labels_onehot = np.expand_dims(labels_onehot, axis=1)
labels_onehot = OneHotEncoder().fit_transform(labels_onehot).toarray()
```

```python
inds = np.arange(ids.shape[0]) # use index at identifying each node
x = vecs
y = labels_onehot
print(ids.shape, x.shape, y.shape)
```

```
(2708,) (2708, 1433) (2708, 7)
```

Preprocess (3)

- Split train, valid and test set
  - You may need index of each node because GCN is semi-supervised transductive learning algorithm. In graph, transductive learning algorithm means at train time you will use whole graph structure including nodes in valid set and test set.
    - At train time, edge information including valid/test nodes are used
    - At valid/test time, each node's label information is also used

Preprocess (4)

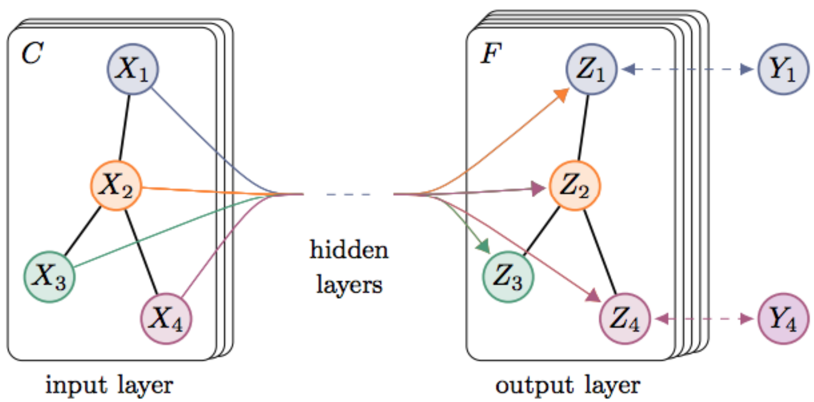
- Split train, valid and test set

```
num_classes = 7
num_per_train = 10
num_per_test = 100
x_train, x_test, y_train, y_test, idx_train, idx_test = train_test_split(x, y, inds, stratify=y,
                                                    train_size=num_classes*num_per_train,
                                                    test_size=num_classes*num_per_test,
                                                    random_state=42)

x_train, x_valid, y_train, y_valid, idx_train, idx_valid = train_test_split(x_train, y_train, idx_train,
                                                    stratify=y_train,
                                                    train_size=int(num_classes*num_per_train*0.8),
                                                    test_size=int(num_classes*num_per_train*0.2),
                                                    random_state=42)

print(idx_train.shape, x_train.shape, y_train.shape) # 10 examples per class
print(idx_valid.shape, x_valid.shape, y_valid.shape) # 10 examples per class
print(idx_test.shape, x_test.shape, y_test.shape) # 100 examples per class
```

```
(56,) (56, 1433) (56, 7)
(14,) (14, 1433) (14, 7)
(700,) (700, 1433) (700, 7)
```

Architecture – GCN (1)

- As CNN shares kernel between regions in image, GCN shares weights on all location of graph



Architecture – GCN (2)

- Layer wise propagation rules are as below

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

- $\hat{A} = A + I_N$, self loop added adjacent matrix
- $\widehat{D} = \sum_j \hat{A}_{ij}$, degree matrix
- $H^{(l)} \in R^{N \times C}$, node feature matrix at lth layer
- $W^{(l)} \in R^{C \times F}$, weights at lth layer

Test measure

- **Prediction Accuracy** on each node's label in dataset

$$accuracy = \frac{number\ \ of\ \ correct\ \ predictions}{total\ \ number\ \ of\ \ nodes}$$

Outline

- ☑ Introduction
- ☑ Data
- ☑ Preprocessing Codes
- ☐ **Answers**

DNN network (1)

- Use DNN to compare with GCN
- You can use your own DNN structure e.g. number of layers and neurons in each layer
- Use only node features in classification

DNN network (2)

- Example codes are below

```
dnn = Sequential([
    ly.Dense(units=128, activation='relu', kernel_initializer='he_normal'),
    ly.Dense(units=64, activation='relu', kernel_initializer='he_normal'),
    ly.Dense(units=num_classes, kernel_initializer='he_normal')
])
```

DNN network (3)

- Training and testing of DNN is below

```
loss_fn = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
dnn.compile(optimizer='adam', loss=loss_fn, metrics=['acc'])
dnn.fit(x = x_train, y = y_train, batch_size=32, epochs=20, verbose=2,
        validation_data=(x_valid, y_valid))
dnn.summary()
```

```
train_loss, train_acc = dnn.evaluate(x_train, y_train, verbose=0)
valid_loss, valid_acc = dnn.evaluate(x_valid, y_valid, verbose=0)
test_loss, test_acc = dnn.evaluate(x_test, y_test, verbose=0)

print("Train accuracy: ", train_acc)
print("Valid accuracy: ", valid_acc)
print("Test accuracy: ", test_acc)
```

DNN network (4)

- Corresponding results are below

```
Epoch 19/20
56/56 - 0s - loss: 0.2950 - acc: 1.0000 - val_loss: 1.4818 - val_acc: 0.5714
Epoch 20/20
56/56 - 0s - loss: 0.2472 - acc: 1.0000 - val_loss: 1.4561 - val_acc: 0.5714
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                multiple                  183552
_________________________________________________________________
dense_1 (Dense)              multiple                  8256
_________________________________________________________________
dense_2 (Dense)              multiple                  455
=================================================================
Total params: 192,263
Trainable params: 192,263
Non-trainable params: 0

Train accuracy:  1.0
Valid accuracy:  0.5714286
Test accuracy:  0.38428572
```

GCN network (1)

- Recall, propagation rule of GCN

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

- We need each matrix to propagate over layer

GCN network (2)

- X is node feature matrix of shape (N, F)
- Y is one hot encoded label matrix of node label, and shape is (N, num_classes)
- A is norm matrix, that is $\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}$

GCN network (3)

- Mask is used when extract target nodes
- As transductive learning algorithms use all edges in graph, we do forward pass using the whole graph and evaluate only corresponding nodes in each dataset.
- We can calculate loss of each dataset using mask

GCN network (4)

- Define each model and call function

```
class GCN(Model):
    def __init__(self, A, input_dim=1433, hid_dim=64, num_classes=7, num_nodes=2708):
        super(GCN, self).__init__()
        self.A = tf.cast(A, dtype='float32')
        self.hid_dim = hid_dim
        w_init = tf.initializers.he_normal()

        self.W1 = self.add_weight(name='W1',
                                  shape=(input_dim, self.hid_dim),
                                 initializer=w_init,
                                 trainable=True)
        self.W2 = self.add_weight(name='W2',
                          shape=(self.hid_dim, num_classes),
                         initializer=w_init,
                         trainable=True)
        self.var_list = self.weights


    def call(self, x):
        x = tf.cast(x, "float32")
        L1 = tf.matmul(tf.matmul(self.A, x), self.W1)
        L1 = tf.nn.tanh(L1)

        L2 = tf.matmul(tf.matmul(self.A, L1), self.W2)
        return L2
```

GCN network (5)

- Define a loss function (inside the class GCN)
- Use mask to calculate loss of nodes in dataset
- tf.gather_nd extracts submatrix of given matrix

```
def loss_fn(self,logits, labels, indices):
    _labels = tf.gather_nd(labels, indices)
    _logits = tf.gather_nd(logits, indices)
    loss = tf.nn.softmax_cross_entropy_with_logits(labels=_labels, logits=_logits)
    return tf.reduce_mean(loss)
```

GCN network (6)

- Prepare norm matrix which is used at every layer

```python
# make adj matrix from citation information
def get_adj_matrix(ids):
    cora_cites = np.loadtxt('./cora/cora.cites', dtype=np.int32)
    N = ids.shape[0]
    adj_matrix = np.zeros(shape=(N, N), dtype=np.int32)

    # iterate over line
    for i in range(cora_cites.shape[0]):
        node1, node2 = cora_cites[i]
        idx1 = np.where(ids==node1)[0]
        idx2 = np.where(ids==node2)[0]

        # treat as undirected graph
        adj_matrix[idx1, idx2] = 1
        adj_matrix[idx2, idx1] = 1

    return adj_matrix

# make DAD(normalization) matrix
def get_norm_matrix(adj_matrix):
    a_tilda = adj_matrix + np.eye(adj_matrix.shape[0]) # A_ = A+I
    d_tilda = np.diag(1 / np.sqrt(np.sum(a_tilda, axis=1))) # D_^(-1/2)
    return np.matmul(np.matmul(d_tilda, a_tilda), d_tilda)
```

GCN network (7)

- Training and testing of GCN is below

```
def evaluate(self, x, labels, indices):
    logits = self.call(x)
    loss = self.loss_fn(logits, labels, indices)
    _logits = tf.gather_nd(logits, indices)
    _labels = tf.gather_nd(labels, indices)

    pred = tf.argmax(_logits, axis=1)
    ans = tf.argmax(_labels, axis=1)
    correct = tf.equal(pred, ans)
    acc = tf.reduce_mean(tf.cast(correct, tf.float32))
    return loss, acc

def train(self, x, labels, idx_train, idx_val, optimizer, max_epochs=20):
    for epoch in range(1, max_epochs+1):
        with tf.GradientTape() as tape:
            logits = self.call(x)
            train_loss = self.loss_fn(logits, labels, idx_train)

        grad_list = tape.gradient(train_loss, self.var_list)
        grads_and_vars = zip(grad_list, self.var_list)
        optimizer.apply_gradients(grads_and_vars)

        # Evaluation
        train_loss, train_acc = self.evaluate(x, labels, idx_train)
        valid_loss, valid_acc = self.evaluate(x, labels, idx_val)
        print(f"Epoch {epoch:3d}: {train_loss:.4f}, {train_acc*100:.2f},"
              ,f"{valid_loss:.4f}, {valid_acc*100:.2f}")
```

GCN network (8)

- Training and testing of GCN is below

```
num_nodes, input_dim = x.shape[0], x.shape[1]

adj_matrix = get_adj_matrix(ids)
norm_matrix = get_norm_matrix(adj_matrix)

gcn = GCN(A = norm_matrix, input_dim=input_dim, hid_dim=64, num_classes=num_classes, num_nodes=num_nodes)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
_idx_train = np.expand_dims(idx_train, axis=1)
_idx_val = np.expand_dims(idx_valid, axis=1)

gcn.train(x=x, labels=y, idx_train=_idx_train, idx_val=_idx_val, optimizer=optimizer, max_epochs=20)
```

```
test_loss, test_acc = gcn.evaluate(x, y, np.expand_dims(idx_test, axis=1))
print("Test accuracy: ", test_acc)
```

GCN network (9)

- Corresponding results are below

```
Epoch   1: 1.3616, 82.14, 1.6715, 57.14
Epoch   2: 0.9158, 91.07, 1.4557, 78.57
Epoch   3: 0.5902, 100.00, 1.2635, 78.57
Epoch   4: 0.3708, 100.00, 1.1073, 64.29
Epoch   5: 0.2303, 100.00, 0.9905, 64.29
Epoch   6: 0.1430, 100.00, 0.9073, 71.43
Epoch   7: 0.0896, 100.00, 0.8508, 71.43
Epoch   8: 0.0571, 100.00, 0.8148, 78.57
Epoch   9: 0.0371, 100.00, 0.7942, 78.57
Epoch  10: 0.0246, 100.00, 0.7848, 78.57
Epoch  11: 0.0167, 100.00, 0.7831, 78.57
Epoch  12: 0.0116, 100.00, 0.7866, 78.57
Epoch  13: 0.0083, 100.00, 0.7935, 78.57
Epoch  14: 0.0061, 100.00, 0.8025, 78.57
Epoch  15: 0.0045, 100.00, 0.8127, 78.57
Epoch  16: 0.0035, 100.00, 0.8236, 78.57
Epoch  17: 0.0027, 100.00, 0.8346, 78.57
Epoch  18: 0.0022, 100.00, 0.8457, 78.57
Epoch  19: 0.0018, 100.00, 0.8565, 78.57
Epoch  20: 0.0015, 100.00, 0.8671, 78.57

Test accuracy:  tf.Tensor(0.73285717, shape=(), dtype=float32)
```

Result

- After same epochs, DNN has accuracy of 0.38 and GCN has accuracy of 0.73

```
gcn_loss, gcn_acc = gcn.evaluate(x, y, np.expand_dims(idx_test, axis=1))
dnn_loss, dnn_acc = dnn.evaluate(x_test, y_test, verbose=0)
print(f"[GCN] test loss: {gcn_loss:.4f}, test acc: {gcn_acc*100:.2f}")
print(f"[DNN] test loss: {dnn_loss:.4f}, test acc: {dnn_acc*100:.2f}")
```

```
[GCN] test loss: 0.8967, test acc: 73.29
[DNN] test loss: 1.6441, test acc: 38.43
```

What You Need To Know

- How to construct neural networks other than CNN and RNN using tensorflow
- How to train GCN
  - Using mask for calculating loss and accuracy
  - Transductive learning

Questions?