MONASH University

FIT3181/5215 Deep Learning

Word Embedding with Word2Vec

Trung Le and Teaching Team
Department of Data Science and AI
Faculty of Information Technology, Monash University
Email: **trunglm@monash.edu**

GROUP OF EIGHT AUSTRALIA

Outline

- ❑ Word2Vec Characteristics and Word2Vec for Feature Extraction (Tute 8a) (*****)
- ❑ Word2Vec for Initializing Embedding Matrix (Tute 8b) (*****)

Train Word2Vec on a dataset

```
import gensim.downloader as api
from gensim.models import Word2Vec
```

```
dataset = api.load("text8")
model = Word2Vec(dataset)
model.save("./text8-word2vec.bin")
```

Train Word2Vec on text8 dataset (CBOW, window size=5)

```
from gensim.models import KeyedVectors
model = KeyedVectors.load("./text8-word2vec.bin")
word_vectors = model.wv
```

Load pretrained Word2Vec from a file

```
word_vectors.get_vector('king')
```

```
array([ 1.0615952 ,  3.374791  , -2.4529877 , -0.89189297,  2.0385118 ,
       -1.0867552 , -1.2593621 ,  0.20547654, -0.70700854,  0.29547456,
       -0.9151951 ,  0.99069464,  1.7152157 ,  0.64989454,  0.33458185,
        2.499265  , -1.0269971 ,  4.957024  , -6.161608  , -0.10745641,
        0.10324214,  1.1219409 ,  0.98975873, -0.08191033, -0.7929074 ,
       -0.28150806, -1.0557121 ,  0.27056807,  0.31582335,  2.9731138 ,
       -1.4136707 ,  0.93965536,  1.1514933 ,  0.38530475, -1.5722595 ,
       -0.08922919, -1.2710185 , -0.7054481 ,  0.7354161 ,  1.4659075 ,
        1.2870685 , -1.2874846 , -1.6638854 ,  0.20794497, -0.1928033 ,
       -3.8513193 , -0.0873706 ,  0.43098506,  0.12324328, -1.6535882 ,
       -0.6248446 , -0.28294212,  1.4047468 , -0.42495435,  0.7049425 ,
       -0.26330888, -1.7225645 , -0.866658  ,  1.3149631 , -0.5719914 ,
       -1.3960481 ,  1.7349594 ,  2.8976836 ,  2.233186  ,  0.905698  ,
        0.24419262,  1.7447696 ,  2.4310687 , -0.6564301 ,  2.1977458 ,
       -0.28740513, -0.0529648 ,  1.8151288 ,  1.2035793 ,  0.51843506,
        2.2382748 , -1.7706063 , -1.7169152 , -3.8160467 ,  0.2048373 ,
        1.1777579 ,  2.9256532 ,  0.7214914 , -3.804784  , -0.3797294 ,
       -1.3870562 , -1.8468527 ,  0.96608454, -0.51972026, -1.4571909 ,
       -2.1815338 , -1.7526524 , -2.4643364 , -0.5413108 , -0.6252542 ,
        0.33478758, -0.27308032, -2.7191868 , -2.4398658 , -1.7016346 ],
      dtype=float32)
```

Get vector representation of a word

```
word_vectors.cosine_similarities(word_vectors.get_vector('king'), [word_vectors.get_vector('queen'), word_vectors.get_vector('australia')])
```

```
array([0.7218381 , 0.09479931], dtype=float32)
```

Compute cosine similarity

Advanced operations in Word2Vec

```
def print_most_similar(word_conf_pairs, k):
    for i, (word, conf) in enumerate(word_conf_pairs):
        print("{:.3f} {:s}".format(conf, word))
        if i >= k-1:
            break
    if k < len(word_conf_pairs):
        print("...")
```

Print returned results in better form

```
print_most_similar(word_vectors.most_similar(positive=["france", "berlin"], negative=["paris"]), 1)
```

```
0.802 germany
```

france – paris + berlin = germany

```
print(word_vectors.doesnt_match(["hindus", "parsis", "singapore","christians"]))
```

```
singapore
```

Not matched word

```
print_most_similar(word_vectors.most_similar("king"), 10)
```

```
0.759 prince
0.722 queen
0.712 vii
0.698 emperor
0.682 kings
0.669 elector
0.668 regent
0.666 constantine
0.663 throne
0.663 pope
```

Top 10 similar words of king

```
print_most_similar(word_vectors.most_similar("china"), 10)
```

```
0.795 japan
0.760 taiwan
0.746 india
0.667 thailand
0.661 indonesia
0.651 pakistan
0.647 tibet
0.645 afghanistan
0.643 burma
0.639 kazakhstan
```

Top 10 similar words of China

Word2Vec for Feature Extraction

```python
def get_vector(word, model):
    try:
        vec = model.get_vector(word)
    except KeyError:
        vec = np.zeros([model.size])
    return vec
```

(1) Get vector representation for a word

```python
# This function returns the word vector for an input review. It is computed as the average of all vectors,
# if existed, for all words in the reviews
def get_avg_vector(review, model):
    tokens = review.split()
    vecs = [get_vector(word, model) for word in tokens if word in model]
    if len(vecs) > 0:
        vecs = np.asarray(vecs).sum(axis = 0)/len(vecs)
    return vecs
```

(2) Get vector representation for a sentence

```python
word2vect = api.load("glove-twitter-25")
```

(3) Load Word2Vec

```python
# Getting movie reviews data
X_train = []
y_train = []
for line in open('./datasets/imdb/train-pos.txt'):
    vec = get_avg_vector(line, word2vect)
    if len(vec)>0:
        X_train += [vec]
        y_train += [1]
for line in open('./datasets/imdb/train-neg.txt'):
    vec = get_avg_vector(line, word2vect)
    if len(vec)>0:
        X_train += [vec]
        y_train += [0]
print("[INFO]Finish loading reviews to the training set!")
```

```
[INFO]Finish loading reviews to the training set!
```

(4) Extract feature vectors for training set

```python
X_test = []
y_test = []
for line in open('./datasets/imdb/test-pos.txt'):
    vec = get_avg_vector(line, word2vect)
    if len(vec)>0:
        X_test += [vec]
        y_test += [1]
for line in open('./datasets/imdb/test-neg.txt'):
    vec = get_avg_vector(line, word2vect)
    if len(vec)>0:
        X_test += [vec]
        y_test += [0]
print("[INFO]Finish loading reviews to the testing set!")
```

```
[INFO]Finish loading reviews to the testing set!
```

(5) Extract feature vectors for test set

```python
classifiers = [
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    GradientBoostingClassifier()
]
```

(6) Declare a list of classifiers

```python
X_train, X_test, y_train, y_test = np.asarray(X_train), np.asarray(X_test), np.asarray(y_train), np.asarray(y_test)

for clf in classifiers:
    clf.fit(X_train, y_train)
    name = clf.__class__.__name__
    test_predictions = clf.predict(X_test)
    acc = accuracy_score(y_test, test_predictions)
    print("Classifier: {}, Accuracy: {:.4%}".format(name, acc))
```

```
Classifier: DecisionTreeClassifier, Accuracy: 63.9680%
Classifier: RandomForestClassifier, Accuracy: 74.2200%
Classifier: GradientBoostingClassifier, Accuracy: 74.1560%
```

(7) Train classifiers on training sets and evaluate on test set

DataManager

```python
from torch.nn.utils.rnn import pad_sequence
from transformers import BertTokenizer
```

BertTokenizer

```python
class DataManager:
    def __init__(self, url= None):
        self.url = url
        self.max_seq_len = None        # store the max sequence length
        self.num_sentences = None      # store number of sentences
        self.texts = None              # store all sentences
        self.labels = None             # store all labels
        self.num_seqs = None          # store sequences of indices
        self.vocab_size = None
        self.tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

BertTokenizer

```python
    def read_data(self, file_path):
        df = pd.read_csv(file_path, encoding = "ISO-8859-1")
        df['label'] = df['v1'].apply(lambda x: 1 if x == 'spam' else 0)
        labels, texts = df['label'].to_numpy(), df['v2'].tolist()
        self.texts= texts
        print("Number of sentences: ", len(texts))
        self.labels = torch.from_numpy(labels)
```

Read data from csv file

```python
def build_vocabulary(self):
    vocab = self.tokenizer.get_vocab()
    self.word2idx = {w: i for i, w in enumerate(vocab)}
    self.idx2word = {i:w for w,i in self.word2idx.items()}
    self.vocab_size = len(self.word2idx)
    self.min_index = min(self.word2idx.values())
    self.max_index = max(self.word2idx.values())

def process_data(self):
    self.build_vocabulary()
    self.transform_to_numbers()
```

```python
def transform_to_numbers(self):
  token_ids = []
  num_seqs = []

  for text in self.texts:  # iterate over the list of text
    text_seqs = self.tokenizer.tokenize(str(text))  # tokenize each text individually
    # Convert tokens to IDs
    token_ids = self.tokenizer.convert_tokens_to_ids(text_seqs)
    # Convert token IDs to a tensor of indices using your word2idx mapping
    seq_tensor = torch.LongTensor(token_ids)
    num_seqs.append(seq_tensor)  # append the tensor for each sequence

# Pad the sequences and create a tensor
  if num_seqs:
    self.num_seqs = pad_sequence(num_seqs, batch_first=True)  # Pads to max length of the sequences
    self.num_sentences, self.max_seq_len = self.num_seqs.shape
    print(self.num_seqs.shape)
```

```python
def train_valid_test_split(self, train_ratio= 0.8, test_ratio=0.1):
    train_size = int(self.num_sentences*train_ratio) +1
    test_size = int(self.num_sentences*test_ratio) +1
    valid_size = self.num_sentences - (train_size + test_size)
    data_indices = list(range(self.num_sentences))
    random.shuffle(data_indices)
    train_set_data = self.num_seqs[data_indices[:train_size]]
    train_set_labels = self.labels[data_indices[:train_size]]
    train_set = torch.utils.data.TensorDataset(train_set_data, train_set_labels)
    test_set_data = self.num_seqs[data_indices[-test_size:]]
    test_set_labels = self.labels[data_indices[-test_size:]]
    test_set = torch.utils.data.TensorDataset(test_set_data, test_set_labels)
    valid_set_data = self.num_seqs[data_indices[train_size:-test_size]]
    valid_set_labels = self.labels[data_indices[train_size:-test_size]]
    valid_set = torch.utils.data.TensorDataset(valid_set_data, valid_set_labels)
    self.train_loader = DataLoader(train_set, batch_size=64, shuffle=True)
    self.test_loader = DataLoader(test_set, batch_size=64, shuffle=False)
    self.valid_loader = DataLoader(valid_set, batch_size=64, shuffle=False)
```

Word2Vec for initializing embedding matrix

```
class RNN_Spam_Detection:
    def __init__(self, run_mode="scratch", embed_model="glove-wiki-gigaword-300", embed_size=128,
             hidden_size=128, data_manager=None):
        self.embed_path = "embeddings/E.npy"
        self.embed_model = embed_model
        self.embed_size = embed_size
        self.run_mode = run_mode
        if run_mode != 'scratch':
            self.embed_size = int(self.embed_model.split("-")[-1])
        self.data_manager = data_manager
        self.vocab_size = self.data_manager.vocab_size
        self.word2idx = self.data_manager.word2idx
        self.embed_matrix = np.zeros((self.vocab_size, self.embed_size))
        self.run_mode = run_mode
        self.hidden_size = hidden_size
        self.model = None
```

→ Hold an RNN model (SpamDetectionModel)

```
def build_embedding_matrix(self):
    if os.path.exists(self.embed_path): # file existed
        self.embed_matrix = np.load(self.embed_path) # Load the file for embe
    else: # file not existed or first-time run
        self.word2vect = api.load(self.embed_model) # load embedding model
        for word, idx in self.word2idx.items():
            try:
                self.embed_matrix[idx] = self.word2vect.word_vec(word) # assi
            except KeyError: # word cannot be found
                pass
        np.save(self.embed_path, self.embed_matrix)
```

```
def build(self):
  if self.run_mode != 'scratch':
    embed_matrix = torch.from_numpy(self.embed_matrix)
  else:
    self.build_embedding_matrix()
    embed_matrix = None
```

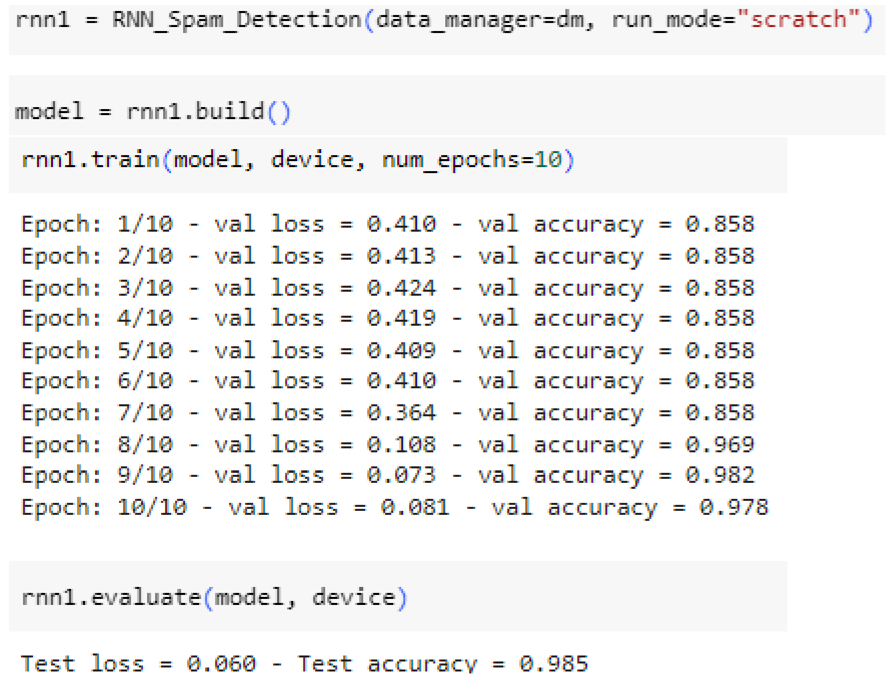
→ Build embedding matrix

```
  model = SpamDetectionModel(self.vocab_size, self.embed_size, self.hidden_size, embed_matrix)
  self.criterion = nn.CrossEntropyLoss(reduction="mean")
  return model
```

```
def train(self, model, device, num_epochs):
  optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
  for epoch in range(1, num_epochs + 1):
    train_loss, train_acc = train_epoch(model, optimizer, self.data_manager.train_loader,
                                        self.criterion, device)
    val_loss, val_acc = test_epoch(model, self.data_manager.valid_loader,
                                   self.criterion, device)
    msg = f"Epoch: {epoch}/{num_epochs} - train loss = {train_loss:.3f} - train accuracy = {train_acc*100:.3f}%"
    msg =  msg + f"- val loss = {val_loss:.3f} - val accuracy = {val_acc*100:.3f}%"
    print(msg)

def evaluate(self, model, device):
  loss, acc = test_epoch(model, self.data_manager.test_loader, self.criterion, device)
  print(f"Test loss = {loss:.3f} - Test accuracy = {acc*100:.3f}%")
```

```
class SpamDetectionModel(nn.Module):
  def __init__(self, vocab_size, embedding_dim, hidden_dim, embed_matrix=None):
    super(SpamDetectionModel, self).__init__()

    if embed_matrix is not None:
      self.embedding_layer = nn.Embedding.from_pretrained(embed_matrix)
    else:
      self.embedding_layer = nn.Embedding(vocab_size, embedding_dim)
    self.rnn_layer = nn.GRU(embedding_dim, hidden_dim, num_layers=1, batch_first=True)
    self.dense_layer = nn.Linear(hidden_dim, 2)

  def forward(self, x):
    e = self.embedding_layer(x)
    h = self.rnn_layer(e)
    if isinstance(h, tuple):
      h = h[0]
    h = h[:, -1, :]
    y = self.dense_layer(h)
    return y
```

Word2Vec for initializing embedding matrix



```
rnn1 = RNN_Spam_Detection(data_manager=dm, run_mode="scratch")
```

```
model = rnn1.build()
```

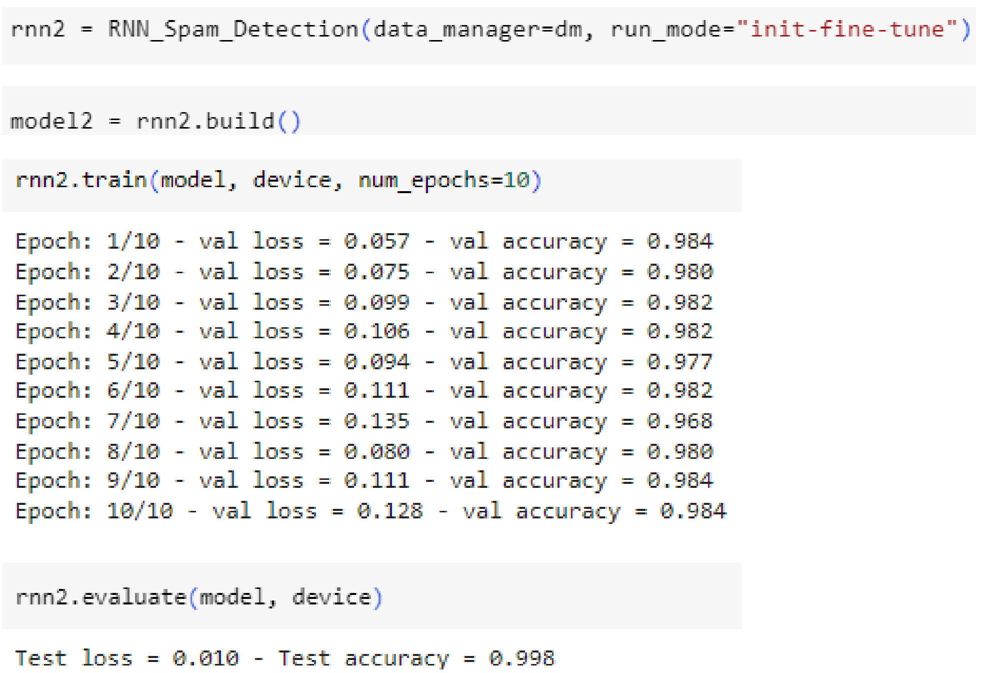
```
rnn1.train(model, device, num_epochs=10)
```

```
Epoch: 1/10 - val loss = 0.410 - val accuracy = 0.858
Epoch: 2/10 - val loss = 0.413 - val accuracy = 0.858
Epoch: 3/10 - val loss = 0.424 - val accuracy = 0.858
Epoch: 4/10 - val loss = 0.419 - val accuracy = 0.858
Epoch: 5/10 - val loss = 0.409 - val accuracy = 0.858
Epoch: 6/10 - val loss = 0.410 - val accuracy = 0.858
Epoch: 7/10 - val loss = 0.364 - val accuracy = 0.858
Epoch: 8/10 - val loss = 0.108 - val accuracy = 0.969
Epoch: 9/10 - val loss = 0.073 - val accuracy = 0.982
Epoch: 10/10 - val loss = 0.081 - val accuracy = 0.978
```

```
rnn1.evaluate(model, device)
```

```
Test loss = 0.060 - Test accuracy = 0.985
```

Training embedding matrix from scratch



```
rnn2 = RNN_Spam_Detection(data_manager=dm, run_mode="init-fine-tune")
```

```
model2 = rnn2.build()
```

```
rnn2.train(model, device, num_epochs=10)
```

```
Epoch: 1/10 - val loss = 0.057 - val accuracy = 0.984
Epoch: 2/10 - val loss = 0.075 - val accuracy = 0.980
Epoch: 3/10 - val loss = 0.099 - val accuracy = 0.982
Epoch: 4/10 - val loss = 0.106 - val accuracy = 0.982
Epoch: 5/10 - val loss = 0.094 - val accuracy = 0.977
Epoch: 6/10 - val loss = 0.111 - val accuracy = 0.982
Epoch: 7/10 - val loss = 0.135 - val accuracy = 0.968
Epoch: 8/10 - val loss = 0.080 - val accuracy = 0.980
Epoch: 9/10 - val loss = 0.111 - val accuracy = 0.984
Epoch: 10/10 - val loss = 0.128 - val accuracy = 0.984
```

```
rnn2.evaluate(model, device)
```

```
Test loss = 0.010 - Test accuracy = 0.998
```

Using Word2Vec to initialize embedding matrix and fine tune

Thanks for your attention!