Deep Neural Networks for Natural Language Processing (AI6127)

JUNG-JAE KIM

TUTORIAL 3: WORD WINDOW CLASSIFICATION

Question 1: Implement Softmax classifier for the named entity recognition (NER) task

- Use the Softmax classifier implementation at http://web.stanford.edu/class/cs224n/materials/ww_classifier.ipynb
- Use the NER corpus of CoNLL 2002 dataset at https://drive.google.com/file/d/1Neuy76b_R6-OYevNpKAGumA8rJR7Udnj/view?usp=sharing
- Compare the performance with that of the hands-on “Named entity recognition by using CRF”

Recall

Hands-on: Word-window classification using SoftmaxClassifier

- Data preparation
- Build model
- Train model
- Predict with model

Codes: Word-window classification using SoftmaxClassifier – toy data

```
train_sents = [s.lower().split() for s in ["we 'll always have Paris",
                                           "I live in Germany",
                                           "He comes from Denmark",
                                           "The capital of Denmark is Copenhagen"]]
train_labels = [[0, 0, 0, 0, 1],
                [0, 0, 0, 1],
                [0, 0, 0, 1],
                [0, 0, 0, 1, 0, 1]]
test_sents = [s.lower().split() for s in ["She comes from Paris"]]
test_labels = [[0, 0, 0, 1]]
```

Use CoNLL data

Codes: Word-window classification using SoftmaxClassifier – Converting tokenized sentence lists to vocabulary indices

```
id_2_word = ["<pad>", "<unk>", "we", "always", "have", "paris",
             "i", "live", "in", "germany",
             "he", "comes", "from", "denmark",
             "the", "of", "is", "copenhagen"]
word_2_id = {w:i for i,w in enumerate(id_2_word)}


def convert_tokens_to_inds(sentence, word_2_id):
    return [word_2_id.get(t, word_2_id["<unk>"]) for t in sentence]
```

Collect vocab from training data

Codes: Word-window classification using SoftmaxClassifier – Padding for windows

```
def pad_sentence_for_window(sentence, window_size, pad_token="<pad>"):
    return [pad_token]*window_size + sentence + [pad_token]*window_size


for sent in train_sents:
    tok_idxs = convert_tokens_to_inds(pad_sentence_for_window(sent, window
_size), word_2_id)
    print([id_2_word[idx] for idx in tok_idxs])
```

Codes: Word-window classification using SoftmaxClassifier – Batching sentences together with a DataLoader

```
def my_collate(data, window_size, word_2_id):

    x_s, y_s = zip(*data)

    window_padded = [convert_tokens_to_inds(pad_sentence_for_window(sentence, window_size), word_2_id) for sentence in x_s]

    # append zeros to each list of token ids in batch so that they are all the same length

    padded = nn.utils.rnn.pad_sequence([torch.LongTensor(t) for t in window_padded], batch_first=True)



    # convert labels to one-hots

    labels, lengths = [], []

    for y in y_s:

        lengths.append(len(y))

        label = torch.zeros((len(y), 2))

        true = torch.LongTensor(y)

        false = ~true.byte()

        label[:, 0] = false

        label[:, 1] = true

        labels.append(label)

    padded_labels = nn.utils.rnn.pad_sequence(labels, batch_first=True)



    return padded.long(), padded_labels, torch.LongTensor(lengths)
```

Change to multi-class classification

Codes: Word-window classification using SoftmaxClassifier – Model

```
class SoftmaxWordWindowClassifier(nn.Module):

    def __init__(self, config, vocab_size, pad_idx=0):

        super(SoftmaxWordWindowClassifier, self).__init__()

        # Instance variables.

        self.window_size = 2*config["half_window"]+1

        self.embed_dim = config["embed_dim"]

        self.hidden_dim = config["hidden_dim"]

        self.num_classes = config["num_classes"]

        self.freeze_embeddings = config["freeze_embeddings"]

        # Embedding layer: model holds an embedding for our vocab. sets aside a special index in the embedding matrix for padding vector (of zeros). by default, embeddings are parameters (so gradients pass through them)

        self.embed_layer = nn.Embedding(vocab_size, self.embed_dim, padding_idx=pad_idx)

        if self.freeze_embeddings:

            self.embed_layer.weight.requires_grad = False
```

Codes: Word-window classification using SoftmaxClassifier – Model

Hidden layer: we want to map embedded word windows of dim (window_size+1)*self.embed_dim to a hidden layer. nn.Sequential allows you to efficiently specify sequentially structured models. First the linear transformation is evoked on the embedded word windows. Next the nonlinear transformation tanh is evoked.

```
self.hidden_layer = nn.Sequential(nn.Linear(self.window_size*self.embed_dim, self.hidden_dim), nn.Tanh())
```

Output layer: we want to map elements of the output layer (of size self.hidden dim) to a number of classes.

```
self.output_layer = nn.Linear(self.hidden_dim, self.num_classes)
```

Softmax: The final step of the softmax classifier: mapping final hidden layer to class scores. Pytorch has both logsoftmax and softmax functions (and many others). Since our loss is the negative LOG likelihood, we use logsoftmax. Technically you can take the softmax, and take the log but PyTorch's implementation is optimized to avoid numerical underflow issues.

```
self.log_softmax = nn.LogSoftmax(dim=2)
```

Codes: Word-window classification using SoftmaxClassifier – Model

```
def forward(self, inputs):

    # Let B:= batch_size, L:= window-padded sentence length, D:= self.embed_dim, S:= self.window_size, H:= self.hidden_dim

    # inputs: a (B, L) tensor of token indices

    B, L = inputs.size()



    # Fist, get our word windows for each word in our input.

    token_windows = inputs.unfold(1, self.window_size, 1)

    _, adjusted_length, _ = token_windows.size()

    # Good idea to do internal tensor-size sanity checks, at the least in comments!

    assert token_windows.size() == (B, adjusted_length, self.window_size)

    embedded_windows = self.embed_layer(token_windows)

    embedded_windows = embedded_windows.view(B, adjusted_length, -1)



    layer_1 = self.hidden_layer(embedded_windows)

    output = self.output_layer(layer_1)

    output = self.log_softmax(output)

    return output
```

Codes: Word-window classification using SoftmaxClassifier – Training

```
def loss_function(outputs, labels, lengths):
    """Computes negative LL loss on a batch of model predictions."""
    B, L, num_classes = outputs.size()
    num_elems = lengths.sum().float()

    # get only the values with non-zero labels
    loss = outputs*labels

    # rescale average
    return -loss.sum() / num_elems
```

Codes: Word-window classification using SoftmaxClassifier – Training

```
def train_epoch(loss_function, optimizer, model, tra
in_data):
    ## For each batch, we must reset the gradients
stored by the model.
    total_loss = 0
    for batch, labels, lengths in train_data:
        # clear gradients
        optimizer.zero_grad()
        # evoke model in training mode on batch
        outputs = model.forward(batch)
        # compute loss w.r.t batch
        loss = loss_function(outputs, labels, lengths)
        # pass gradients back, startiing on loss value
        loss.backward()
        # update parameters
        optimizer.step()
        total_loss += loss.item()
    # return the total to keep track of how you did thi
s time around
    return total_loss
```

Codes: Word-window classification using SoftmaxClassifier – Training

```
config = {"batch_size": 4,
          "half_window": 2,
          "embed_dim": 25,
          "hidden_dim": 25,
          "num_classes": 2,
          "freeze_embeddings": False,
         }
learning_rate = .0002
num_epochs = 10000
model = SoftmaxWordWindowClassifier(config, len(word_2_id))
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Change some parameters (e.g. num_epochs)

Codes: Word-window classification using SoftmaxClassifier – Training

```
train_loader = torch.utils.data.DataLoader(list(zip(train_sents, train_labels)),
    batch_size=2, shuffle=True,
    collate_fn=partial(my_collate, window_size=2, word_2_id=word_2_id))


losses = []
for epoch in range(num_epochs):
   epoch_loss = train_epoch(loss_function, optimizer, model, train_loader)
   if epoch % 100 == 0:
      losses.append(epoch_loss)
```

Codes: Word-window classification using SoftmaxClassifier – Prediction

```
test_loader = torch.utils.data.DataLoader(list(zip(test_sents, test_labels)),
    batch_size=1, shuffle=False,
    collate_fn=partial(my_collate, window_size=2, word_2_id=word_2_id))


for test_instance, labs, _ in test_loader:
    outputs = model.forward(test_instance)
```

Output what is required for evaluation (see next slides)

Recall

Hands-on: Named entity recognition by using CRF

- Download and preprocess NER corpus (CONLL 2002)
- Prepare CRF model for NER
- Run CRF for training and evaluation

https://drive.google.com/file/d/1Neuy76b_R6-OYevNpKAGumA8rJR7Udnj/view?usp=sharing

Codes: dataset, data preparation

```
import nltk

nltk.download('conll2002')

from nltk.corpus import conll2002


# get training/testing datasets

train_sents = list(conll2002.iob_sents('esp.train')) ## spain

test_sents = list(conll2002.iob_sents('esp.testb'))
```

Codes: Features

```
def word2features(sent, i): # skip

def sent2features(sent): # skip


def sent2labels(sent):
    return [label for token, postag, label in sent]

def sent2tokens(sent):
    return [token for token, postag, label in sent]
```

Codes: Feature extraction

- X_train = [sent2features(s) for s in train_sents]
- y_train = [sent2labels(s) for s in train_sents]
- X_test = [sent2features(s) for s in test_sents]
- y_test = [sent2labels(s) for s in test_sents]

Change sent2features to e.g. sent2wordembeddings

Codes: Training – replace with softmax classifier

```
# train CRF model
!pip install sklearn_crfsuite
import sklearn_crfsuite
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=True
)

crf.fit(X_train, y_train)
```

Codes: Evaluation

```
# get label set
labels = list(crf.classes_)
labels.remove('O')
print(labels)

# evaluate CRF model
from sklearn_crfsuite import metrics

y_pred = crf.predict(X_test)
metrics.flat_f1_score(y_test, y_pred, average='weighted', labels=labels)
```

Hands-on

Answer 1: Color scheme

- Hands-on: NER with CRF spacy
- Hands-on: Softmax classifier implementation
- Changes for this answer

Answer 1: Download CoNLL 2002 corpus (Spanish)

```
import nltk

nltk.download('conll2002')

from nltk.corpus import conll2002


# get training/testing datasets

train_sents = list(conll2002.iob_sents('esp.train')) ## spain

test_sents = list(conll2002.iob_sents('esp.testb'))
```

Answer 1: Data preparation – Functions of sentence representations for sequence labelling

```
print(train_sents[0]) #each tuple contains token, syntactic tag, ner label
## [('Melbourne', 'NP', 'B-LOC'), ('(', 'Fpa', 'O'), ('Australia', 'NP', 'B-LOC'), …

def sent2labels(sent): return [label for token, postag, label in sent]
def sent2tokens(sent): return [token for token, postag, label in sent]
def convert_labels_to_inds(sent_labels, label_2_id):
  return [label_2_id[label] for label in sent_labels]
```

Answer 1: Data preparation – Sentence representations for sequence labelling

```
train_sent_tokens = [sent2tokens(s) for s in train_sents]
train_labels = [sent2labels(s) for s in train_sents]

train_id_2_label = list(set([label for sent in train_labels for label in sent]))
train_label_2_id = {label:i for i, label in enumerate(train_id_2_label)}
print("Number of unique labels in training data:", len(train_id_2_label)) ## 9
train_label_inds = [convert_labels_to_inds(sent_labels, train_label_2_id) for sent_labels in train_labels]

test_sent_tokens = [sent2tokens(s) for s in test_sents]
test_labels = [sent2labels(s) for s in test_sents]
test_label_inds = [convert_labels_to_inds(s, train_label_2_id) for s in test_labels]
```

Answer 1: Data preparation – Converting tokenized sentence lists to vocabulary indices

```
id_2_word = list(set([token for sent in train_sent_tokens for token in sent])) + ["<pad>", "<unk>"]

word_2_id = {w:i for i,w in enumerate(id_2_word)}

def convert_tokens_to_inds(sentence, word_2_id):
    return [word_2_id.get(t, word_2_id["<unk>"]) for t in sentence]

# padding for windows
window_size = 2
def pad_sentence_for_window(sentence, window_size, pad_token="<pad>"):
    return [pad_token]*window_size + sentence + [pad_token]*window_size
```

Answer 1: Data preparation – Batching sentences together with a DataLoader

```
def my_collate(data, window_size, word_2_id):

    x_s, y_s = zip(*data)

    window_padded = [convert_tokens_to_inds(pad_sentence_for_window(sentence, window_size), word_2_id) for sentence in x_s]

    # append zeros to each list of token ids in batch so that they are all the same length

    padded = nn.utils.rnn.pad_sequence([torch.LongTensor(t) for t in window_padded], batch_first=True)
```

Answer 1: Data preparation – Batching sentences together with a DataLoader

```
# convert labels to one-hots
labels, lengths = [], []
for y in y_s:
    lengths.append(len(y))
    one_hot = torch.zeros(len(y), len(train_id_2_label))
    y = torch.tensor(y).unsqueeze(1)
    label = one_hot.scatter_(1, y, 1)
    labels.append(label)
padded_labels = nn.utils.rnn.pad_sequence(labels, batch_first=True)
return padded.long(), padded_labels, torch.LongTensor(lengths)
```

Answer 1: Data preparation – Data loading

```
batch_size = 4

# Shuffle True is good practice for train loaders.
train_loader = DataLoader(list(zip(train_sent_tokens, train_label_inds)),
    batch_size=batch_size, shuffle=True,
    collate_fn=partial(my_collate, window_size=2, word_2_id=word_2_id))
```

Answer 1: Modeling

```
class SoftmaxWordWindowClassifier(nn.Module):
        …
def loss_function(outputs, labels, lengths):
        …
def train_epoch(loss_function, optimizer, model, train_data):
        …
```

Answer 1: Config

```
config = {"batch_size": 4,
          "half_window": 2,
          "embed_dim": 25,
          "hidden_dim": 25,
          "num_classes": 9,
          "freeze_embeddings": False,
         }
learning_rate = .0002
num_epochs = 100
model = SoftmaxWordWindowClassifier(config, len(word_2_id))
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Answer 1: Training

```
for epoch in range(num_epochs):
    epoch_loss = train_epoch(loss_function, optimizer, model, train_loader)
    print(epoch, epoch_loss)
```

Answer 1: Evaluation

```
test_loader = DataLoader(list(zip(test_sent_tokens, test_label_inds)),
                         batch_size=batch_size, shuffle=False,
                         collate_fn=partial(my_collate, window_size=2, word_2_id=word_2_id))


test_outputs = []
for test_instance, labs, _ in test_loader:
    outputs_full = model.forward(test_instance)
    outputs = torch.argmax(outputs_full, dim=2)
    for i in range(outputs.size(0)):
      test_outputs.append(outputs[i].tolist())
```

Answer 1: Evaluation

```
y_test = test_labels
y_pred = []
for test, pred in zip(test_labels, test_outputs):
  y_pred.append([train_id_2_label[id] for id in pred[:len(test)]])


!pip install sklearn-crfsuite
from sklearn_crfsuite import metrics

metrics.flat_f1_score(y_test, y_pred, average='weighted', labels=train_id_2_label)
```

Answer 1: Evaluation comparison

- CRF: 79%
- Softmax classifier: 83%