Programming Assignment 2

Welcome to the second programming assignment for CS 4650! In this project, we will train LSTM POS-taggers, which take in a sentence and outputs part-of-speech labels for every word in the sentence.

We will use English text from the Wall Street Journal, marked with POS tags such as `NNP` (proper noun) and `DT` (determiner).

(Instructor: Wei Xu; TAs: Marcus Ma, Mounica Maddela, Ben Podrazhansky, Rahul Katre)

To begin this project, make a copy of this notebook and save it to your local drive so that you can edit it.

If you want GPU's (which will improve training speed), you can always change your instance type to GPU by going to Runtime -> Change runtime type -> Hardware accelerator.

If you're new to PyTorch, or simply want a refresher, we recommend you start by looking through these [Introduction to PyTorch] slides and this interactive [PyTorch Basics notebook]. Additionally, this [Text Sentiment] notebook will provide some insight into working with PyTorch for NLP specific problems.

Part 0 Colab Setup [DO NOT MODIFY]

Below, we will import our libraries and check for GPU usage.

```
# DO NOT MODIFY #
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import random

RANDOM_SEED = 42
torch.manual_seed(RANDOM_SEED)
random.seed(RANDOM_SEED)

# this is how we select a GPU if it's avalible on your computer or in the Colab environment.
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

You can check to make sure a GPU is available using the following code block.

If the below message is shown, it means you are using a CPU.

```
/bin/bash: nvidia-smi: command not found
```

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Select the Runtime > "Change runtime type" menu to enable a GPU accelerator, ')
```

```
  print('and then re-execute this cell.')
else:
  print(gpu_info)
```

```
Wed Mar  8 00:51:01 2023       
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   74C    P0    33W /  70W |      3MiB / 15360MiB |      5%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
                                                                               
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
!curl https://raw.githubusercontent.com/cocoxu/CS4650_projects_spring2023/master/p2_train.txt >
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2775k  100 2775k    0     0  5843k      0 --:--:-- --:--:-- --:--:-- 5843k
```

Part 1 Data Preparation [10 points]

Part 1.1 Loading Data [DO NOT MODIFY]

`train.txt`: The training data is present in this file. This file contains sequences of words and their respective tags. The data is split into 80% training and 20% development to train the model and tune the hyperparameters, respectively. See `load_tag_data` for details on how to read the training data.

```
# DO NOT MODIFY

def load_tag_data(tag_file):
    all_sentences = []
    all_tags = []
    sent = []
    tags = []
    with open(tag_file, 'r') as f:
        for line in f:
            if line.strip() == "":
                all_sentences.append(sent)
                all_tags.append(tags)
                sent = []
                tags = []
            else:
                word, tag, _ = line.strip().split()
```

```
                sent.append(word)
                tags.append(tag)
    return all_sentences, all_tags

train_sentences, train_tags = load_tag_data('train.txt')

unique_tags = set([tag for tag_seq in train_tags for tag in tag_seq])

# Create train-val split from train data
train_val_data = list(zip(train_sentences, train_tags))
random.shuffle(train_val_data)

print("Data Length: ", len(train_val_data))
print("Total tags: ", len(unique_tags))
```

```
Data Length:  8935
Total tags:  44
```

Part 1.2 Training-Validation Splits

We need to split the data into training and validation splits. We will not be using a test split for this project. Implement `train_validation_split` in the cell below.

```
# train_validation_split
# This method takes in a list of features and labels and splits them into train/val splits.
# Note how we are not creating a test set for this project.
#
# args:
# data - list of the tuple (sentence, tags)
# labels - list of POS tags for each corresponding sentence
# split - split proportion for training and validation
#
# returns:
# train_split, test_split
def train_validation_split(data, split=0.8):
    train_split, test_split = None, None
    ############################################################################
    # TODO: Implement the train-validation split
    # Hint: Referencing Project 1 for this function and the subsequent functions
    # could prove useful.
    ############################################################################
    length = len(data)
    split_len = int(length * split)
    train_split = data[0:split_len]
    test_split = data[split_len:]

    ############################################################################
    #                             END OF YOUR CODE                             #
    ############################################################################
    return train_split, test_split
```

Testing our function:

```
# testing train_validation_split
training_data, val_data = train_validation_split(train_val_data)
print(f'Training data proportion: {len(training_data) / len(train_val_data)}')
print(f'Validation data proportion: {len(val_data) / len(train_val_data)}')
```

```
Training data proportion: 0.8
Validation data proportion: 0.2
```

Part 1.3 Word-to-Index and Tag-to-Index mapping

In order to work with text in Tensor format, we need to map each word and each tag to a unique index. Implement `create_word_and_tag_dicts` in the cell below.

```
# create_word_and_tag_dicts
# This method takes a collection of sentences and tags and produces three separate
# dictionaries that will be used later on.
#
# args:
# data - tuple of (sentences, tags) that we will use to build our dictionary.
#
# returns:
# word_to_idx - dict[str] -> int
#        dictionary that maps all of the words in the vocabulary to a unique integer
#        representation.
#
# tag_to_idx - dict[str] -> int
#       dictionary that maps each tag to a unique integer representation.
#
# idx_to_tag - dict[int] -> str
#       dictionary that maps each integer from tag_to_idx to its original tag.
#       essentially, the inverse of tag_to_idx.
def create_word_and_tag_dicts(sentences, unique_tags):
    word_to_idx, tag_to_idx, idx_to_tag = {}, {}, {}
    #############################################################################
    # TODO: Implement create_word_and_tag_dicts
    #############################################################################
    tags = list(unique_tags)
    tag_length = len(tags)
    # update word_to_idx
    idx = 0
    sents=[]
    for s in sentences:
      for w in s:
        sents.append(w)
    sents = list(set(sents))
    sen_length = len(sents)
    for i in range(sen_length):
      word_to_idx[sents[i]] = i
    # update tag_to_idx
    # update idx_to_tag
    for i in range(tag_length):
      tag_to_idx[tags[i]] = i
      idx_to_tag[str(i)] = tags[i]

    #############################################################################
```

```
    #                              END OF YOUR CODE                              #
    ############################################################################
    return word_to_idx, tag_to_idx, idx_to_tag
```

Testing our function:

```
word_to_idx, tag_to_idx, idx_to_tag = create_word_and_tag_dicts(train_sentences, unique_tags)
print(word_to_idx)
print( tag_to_idx)
print( idx_to_tag)
print("Total tags", len(tag_to_idx))
print("Vocab size", len(word_to_idx))
```

```
{'confirmation': 0, 'buffs': 1, 'questionable': 2, 'Lockheed': 3, 'casualty': 4, 'Operatin
{'RB': 0, 'JJR': 1, '.': 2, 'CD': 3, 'VBD': 4, 'RP': 5, 'FW': 6, 'RBR': 7, 'VB': 8, '$': 9
{'0': 'RB', '1': 'JJR', '2': '.', '3': 'CD', '4': 'VBD', '5': 'RP', '6': 'FW', '7': 'RBR',
Total tags 44
Vocab size 19121
```

Part 1.4 Prepare Sequence

Now we'll put everything together! `prepare_sequence` takes in a sentence and its corresponding tags, and returns the data transformed into index Tensors to be used for training in our model.

```
# prepare_sequence
# This method takes a sentence-tag pair and returns two Long-Tensors of the indices
# to be used for the LSTM model.
#
# returns:
# sentence_tensor - torch.LongTensor where each element in the tensor corresponds to
# the index of the word in the sentence
# tag_tensor - torch.LongTensor where each element in the tensor corresponds to
# the index of the tag
def prepare_sequence(sentence, tags, word_to_idx, tag_to_idx):
    sentence_tensor = torch.empty(len(sentence), dtype=torch.long)
    tag_tensor = torch.empty(len(tags), dtype=torch.long)
    ############################################################################
    # TODO: Implement prepare_sequence
    ############################################################################

    # update sentence_tensor
    sentence_tensor = torch.tensor([word_to_idx[word] for word in sentence]).long()

    # update tag_tensor
    tag_tensor = torch.tensor([tag_to_idx[t] for t in list(tags)]).long()

    ############################################################################
    #                              END OF YOUR CODE                              #
    ############################################################################
    return sentence_tensor, tag_tensor
```

```
prepare_sequence(train_sentences[0], train_tags[0], word_to_idx, tag_to_idx)
```

```
(tensor([13353,  3484,  5564, 11839,  1969, 11544, 10187,  6073, 16969, 15240,
          4815, 15763,  2127, 14931, 14955, 11935, 17416,  1985,   917, 11935,
         17023, 13667,  1985, 16015,  6073,  7720,  3193, 14259,  1638, 15981,
         10768, 14779,  7327, 16769,  5612,  5609,  5144]),
 tensor([29, 25, 27, 29, 33,  0, 11, 19,  8, 27, 40, 29, 25, 29, 23, 25, 15, 38,
         40, 25, 29, 29, 38,  8, 19,  8, 27, 40, 29, 25, 15, 31, 15, 32, 40, 23,
          2]))
```

Part 2 Word-Level POS Tagger [20 points]

Part 2.1 Set up model

We will build and train a Basic POS Tagger which is an LSTM model to tag the parts of speech in a given sentence using word-level information.

First we need to define some default hyperparameters.

```
EMBEDDING_DIM = 4
HIDDEN_DIM = 8
LEARNING_RATE = 0.1
LSTM_LAYERS = 1
DROPOUT = 0
EPOCHS = 10
```

Part 2.2 Define Model

The model takes as input a sentence as a tensor in the index space. This sentence is then converted to embedding space where each word maps to its word embedding. The word embeddings is learned as part of the model training process. These word embeddings act as input to the LSTM which produces a representation for each word. Then the representations of words are passed to a Linear layer.

```
class BasicPOSTagger(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size):
        super(BasicPOSTagger, self).__init__()
        #############################################################################
        # TODO: Define and initialize anything needed for the forward pass.
        # You are required to create a model with:
        # an embedding layer: that maps words to the embedding space
        # an LSTM layer: that takes word embeddings as input and outputs hidden states
        # a linear layer: maps from hidden state space to tag space
        #############################################################################
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers = LSTM_LAYERS, dropout=DROPOU
        self.linear = nn.Linear(hidden_dim, tagset_size)

        #############################################################################
        #                              END OF YOUR CODE                             #
        #############################################################################

    def forward(self, sentence):
        tag_scores = None
```

```
        ##############################################################################
        # TODO: Implement the forward pass.
        # Given a tokenized index-mapped sentence as the argument,
        # compute the corresponding raw scores for tags (without softmax)
        # returns:: tag_scores (Tensor)
        ##############################################################################
        em = torch.unsqueeze(self.embedding(sentence),1)
        ls, _ = self.lstm(em)
        tag_scores = self.linear(ls.squeeze(dim=1))

        ##############################################################################
        #                               END OF YOUR CODE                             #
        ##############################################################################
        return tag_scores
```

Part 2.3 Training

We define train and evaluate procedures that allow us to train our model using our created train-val split.

```
def train(epoch, model, loss_function, optimizer):
    model.train()
    train_loss = 0
    train_examples = 0
    for sentence, tags in training_data:
        ##############################################################################
        # TODO: Implement the training method
        # Hint: you can use the prepare_sequence method for creating index mappings
        # for sentences. Find the gradient with respect to the loss and update the
        # model parameters using the optimizer.
        ##############################################################################

        #zero out the parameter gradients
        optimizer.zero_grad()
        #prepare input data (sentences and gold labels)
        sentence_tensor, tag_tensor = prepare_sequence(sentence, tags, word_to_idx, tag_to_idx)
        #do forward pass with current input
        out = model(sentence_tensor)
        #get loss with model predictions and true labels
        loss = loss_function(out, tag_tensor)
        loss.backward()
        #update model parameters
        optimizer.step()
        #increase running total loss and the number of past training samples
        train_loss += loss.item()
        train_examples+= len(tags)

        ##############################################################################
        #                               END OF YOUR CODE                             #
        ##############################################################################

    avg_train_loss = train_loss / train_examples
    avg_val_loss, val_accuracy = evaluate(model, loss_function)

    print("Epoch: {}/{}\tAvg Train Loss: {:.4f}\tAvg Val Loss: {:.4f}\t Val Accuracy: {:.0f}".fo
                                                                      EPOCHS,
                                                                      avg_train_loss,
```

```
                                                                      avg_val_loss,
                                                                      val_accuracy))

def evaluate(model, loss_function):
  # returns:: avg_val_loss (float)
  # returns:: val_accuracy (float)
    model.eval()
    correct = 0
    val_loss = 0
    val_examples = 0
    with torch.no_grad():
        for sentence, tags in val_data:
            #############################################################################
            # TODO: Implement the evaluate method
            # Find the average validation loss along with the validation accuracy.
            # Hint: To find the accuracy, argmax of tag predictions can be used.
            #############################################################################

            #prepare input data (sentences and gold labels)
            sentence_tensor, tag_tensor = prepare_sequence(sentence, tags, word_to_idx, tag_to_
            #do forward pass with current batch of input
            out = model(sentence_tensor)
            #get loss with model predictions and true labels
            loss = loss_function(out, tag_tensor)
            #get the predicted labels
            pred = torch.argmax(out, dim=1)
            #get number of correct prediction
            correct += (pred == tag_tensor).sum().item()
            #increase running total loss and the number of past valid samples
            val_loss += loss.item()
            val_examples += len(tags)

            #############################################################################
            #                             END OF YOUR CODE                              #
            #############################################################################
    val_accuracy = 100. * correct / val_examples
    avg_val_loss = val_loss / val_examples
    return avg_val_loss, val_accuracy
```

```
#############################################################################
# TODO: Initialize the model, optimizer and the loss function
#############################################################################
model = BasicPOSTagger(EMBEDDING_DIM, HIDDEN_DIM, len(word_to_idx), len(tag_to_idx))
loss_function = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), LEARNING_RATE, momentum=0.9)

#############################################################################
#                             END OF YOUR CODE                              #
#############################################################################
for epoch in range(1, EPOCHS + 1):
    train(epoch, model, loss_function, optimizer)
```

```
Epoch: 1/10     Avg Train Loss: 0.0622  Avg Val Loss: 0.0492     Val Accuracy: 63
Epoch: 2/10     Avg Train Loss: 0.0419  Avg Val Loss: 0.0399     Val Accuracy: 72
Epoch: 3/10     Avg Train Loss: 0.0353  Avg Val Loss: 0.0362     Val Accuracy: 76
Epoch: 4/10     Avg Train Loss: 0.0305  Avg Val Loss: 0.0326     Val Accuracy: 79
Epoch: 5/10     Avg Train Loss: 0.0267  Avg Val Loss: 0.0309     Val Accuracy: 81
```

```
Epoch: 6/10     Avg Train Loss: 0.0254  Avg Val Loss: 0.0349     Val Accuracy: 79
Epoch: 7/10     Avg Train Loss: 0.0293  Avg Val Loss: 0.0383     Val Accuracy: 78
Epoch: 8/10     Avg Train Loss: 0.0294  Avg Val Loss: 0.0346     Val Accuracy: 81
Epoch: 9/10     Avg Train Loss: 0.0272  Avg Val Loss: 0.0334     Val Accuracy: 81
Epoch: 10/10    Avg Train Loss: 0.0276  Avg Val Loss: 0.0327     Val Accuracy: 82
```

Sanity Check! Under the default hyperparameter setting, after 5 epochs you should be able to get at least 75% accuracy on the validation set.

Part 2.4 Error analysis

In this step, we will analyze what kind of errors it was making on the validation set.

Step 1, write a method to generate predictions from the validation set. For every sentence, get its words, predicted tags (model_tags), and the ground truth tags (gt_tags). To make the next step easier, you may want to concatenate words from all sentences into a very long list, and same for model_tags and gt_tags.

Step 2, analyze what kind of errors the model was making. For example, it may frequently label NN as VB. Let's get the top-10 most frequent types of errors, each of their frequency, and some example words. One example is at below. It is interpreted as the model predicts NNP as VBG for 626 times, with five random example words of this error being shown.

```
['VBG', 'NNP', 626, ['Rowe', 'Livermore', 'Parker', 'F-16', 'HEYNOW']]
```

```
#############################################################################
# TODO: Generate predictions for val_data
# Create lists of words, tags predicted by the model and ground truth tags.
# Hint: It should look very similar to the evaluate function.
#############################################################################
def generate_predictions(model, val_data):
    # returns:: word_list (str list)
    # returns:: model_tags (str list)
    # returns:: gt_tags (str list)
    # Your code here
    model_tags = []
    gt_tags = []
    word_list = []
    model.train()
    with torch.no_grad():
        for sentence, tags in val_data:
          sentence_tensor, tag_tensor = prepare_sequence(sentence, tags, word_to_idx, tag_to_id
          out = model(sentence_tensor)
          pred = torch.argmax(out, dim=1)

          model_tag = [idx_to_tag[str(int(x))] for x in pred]

          gt_tag = [idx_to_tag[str(int(i))] for i in tag_tensor]
          words = sentence
          model_tags.append(model_tag)
          gt_tags.append(gt_tag)
          word_list.append(words)
```

```
    ##########################################################################
    #                             END OF YOUR CODE                           #
    ##########################################################################
    return word_list, model_tags, gt_tags

##########################################################################
# TODO: Carry out error analysis
# From those lists collected from the above method, find the
# top-10 tuples of (model_tag, ground_truth_tag, frequency, example words)
# sorted by frequency
##########################################################################
def error_analysis(word_list, model_tags, gt_tags):
    # returns: errors (list of tuples)
    # Your code here
    errors = []
    errors_list = []
    counts = {}
    example_words = {}

    for i in range(len(word_list)):
      sentence = word_list[i]
      mt = model_tags[i]
      gt = gt_tags[i]
      for s, m ,g in zip(sentence, mt, gt):
        pair = (m, g)
        if m != g and pair not in counts.keys():
          counts[pair] = 1
          example_words[pair] = [s]
        elif m != g and pair in counts.keys():
          counts[pair] +=1
          example_words[pair].append(s)

    for p in counts.keys():
      model_tag, gt_tag = p
      count = counts[p]
      words = example_words[p]
      e = (model_tag, gt_tag, count, words)
      errors_list.append(e)

    fre = []
    for t in errors_list:
      fre.append(t[2])

    fre = torch.argsort(torch.tensor(fre), descending=True)
    for i in fre:
      errors.append(errors_list[i])
    ##########################################################################
    #                             END OF YOUR CODE                           #
    ##########################################################################

    return errors

word_list, model_tags, gt_tags = generate_predictions(model, val_data)
errors = error_analysis(word_list, model_tags, gt_tags)
```

```
for i in errors[:10]:
  print(i)
```

```
('NNS', 'JJ', 406, ['gullible', 'black', 'double', 'soft', 'lengthy', 'fixed-rate', 'Other
('NN', 'NNP', 289, ['mature', 'Stock', 'Mattel', 'Hot', 'Spain', 'Sala', 'Ostrager', 'Nyne
('NN', 'JJ', 276, ['greedy', 'young', 'Initial', 'ever-narrowing', 'huge', 'great', '60-in
('NNP', 'JJ', 252, ['snake-oil', 'Callable', 'possible', 'one-year', 'good', 'cold', 'barg
('NN', 'NNS', 210, ['receipts', 'foes', 'loopholes', 'mortgages', 'mortgages', 'compatriot
('NNP', 'NN', 209, ['boiler-room', 'hotdog', 'freshman', 'treasury', 'saying', 'Energy', '
('NNS', 'NN', 201, ['medicine', 'humor', 'landing', 'depressant', 'novelist', 'Market', 'n
('NNS', 'NNP', 188, ['Bateman', 'Wenz', 'Hickman', 'Allenport', 'Ownership', 'Sala', 'Bent
('WDT', 'IN', 180, ['that', 'that', 'that', 'that', 'that', 'that', 'that', 'that', 'that'
('VBN', 'VBD', 176, ['found', 'held', 'made', 'permitted', 'topped', 'offered', 'sold', 's
```

Report your findings in the cell below.
What kinds of errors did the model make and why do you think it made them? Write a short paragraph (4-5 sentences) in the cell below.

Explaination:
The model seems more likely to wrongly tag 'JJ' and 'NN.' For 'JJ,' the model predicted 'NNS,' 'NN,' or 'NNP' instead because something we would like to use a noun as an adjective. For 'NN,' the model messes up with 'NNP' and 'NNS.' 'NNP' are also predicted as 'NN' or 'NNS.' 'NNS' are also predicted as 'NN.' It seems like the model cannot distinguish singular nouns from plural nouns and singular proper nouns because they are similar, and the model cannot determine the 's' belongs to a singular noun itself or a plural noun to represent more than two items. The model also messes up when using 'that' as an 'IN' and predicted as 'WDT.' If 'that' is used to connect a restrictive clause, it functions as subordinating conjunction and is thus tagged as 'IN,' but if adding a ',' before 'that,' the model may consider the connected clause does not restrict the subject and tagged as 'WDT.' The last one is 'VBD' as a past tense verb and 'VBD' as a past participle. The model was confused because some words have the same past participle and past tense.

Error Analysis:
('NNS', 'JJ', 406,
['gullible', 'black', 'double', 'soft', 'lengthy', 'fixed-rate', 'Other', 'automatic', 'much-beloved', 'plain', 'net', 'well-known', 'top-10', 'dense', 'fastest-growing', 'fourth', 'wholesale', 'pro-choice', 'much', 'competitive', 'undeveloped', 'procedural', 'African', 'Other', 'complete', 'safe', 'Other', 'consumer-advocacy', 'total', 'net', 'Structural', 'sixfold', 'disabled', 'adequate', 'second-quarter', 'undemocratic', 'historical', 'hefty', 'consumer-price', 'tentative', 'reluctant', 'central', 'criminal', 'seismic', 'outlying', 'pervasive', 'internal', 'inter-city', 'fourth', 'negative', 'illegal', 'bold', 'double-decker', 'Philippine', 'emotional', 'ready', 'human', 'wholesale', 'Money-fund', 'high', 'auxiliary', 'monthly', 'sole', 'sole', 'complete', 'unfair', 'aware', 'influential', 'bearish', 'laden', 'disenchanted', 'precious', 'agrarian-reform', 'difficult', 'eerie', 'full-power', '30-year', 'high', 'nonstrategic', 'electrical', 'Soviet-style', 'enough', 'prickly', 'unwelcome', 'much', 'dilutive', 'heady', 'higher-priced', 'producer-price', 'Last', 'HEAVY', 'impossible', 'lucrative', 'prime', 'fourth', 'adequate', 'sure', 'net', 'free', 'business-class', 'international', 'necessary', 'indomitable', 'single', 'discordant', 'Third-quarter', 'high', 'technological', 'typical', 'unfair', 'effective', 'necessary', 'fixed-rate', 'town-house', 'free', 'overseas', '20-year', 'after-tax', 'Political', 'pliant', 'tire-patching', 'Negotiable', 'semiconductor-depreciation', 'negative', 'top', 'well-servicing', 'confident', 'pro-choice', 'international', 'human', 'long-range', 'median', 'takeover-proof', 'post-quake', 'perfect', 'high', 'robust', 'Last', 'monthly', 'payable', 'Small', 'lucrative', 'high-yield', 'hush-hush', 'five-and-dime', 'particular', 'sore', 'on-line', 'potent', 'tiny', 'sure', 'adequate', 'responsive', 'different', 'Other', 'further', 'overall',

'double-deck', 'Other', 'northern', 'steady', 'capable', '40-point', 'great', 'secondary', '30-pound', 'superficial', 'difficult', 'rational', 'two-tier', 'relative', 'meaningless', 'five-hour', 'Foreign', 'Garpian', 'much', 'mandatory', 'global', 'top', 'payable', 'stepped-up', 'soot-stained', 'RTC-appointed', 'touchy', 'top', 'covert', 'sympathetic', 'comfortable', 'striking', 'pink', 'medical', 'dizzying', 'safe', 'particular', 'total', 'striking', 'antiquated', 'newspaper-industry', 'contrary', 'eastern', 'aware', 'dangerous', 'inter-company', 'Other', 'substantial', 'raw', 'one-page', 'usual', 'open', 'preferred', 'human-rights', 'much', 'Last', 'free', 'hefty', 'much', 'intense', 'fastest-growing', 'reconstructed', 'much', 'different', 'top', 'poor', 'regulatory', 'savings-and-loan', 'unregistered', 'high-quality', 'liquid', 'well-known', 'educational', 'dry', 'preferred', 'four-year', 'different', 'fewer-than-expected', 'natural', 'mild', '190.58-point', 'old-line', 'weeklong', 'sure', 'battery-operated', 'top', 'dizzying', 'lower-than-expected', 'human', 'pork-barrel', 'mild', 'unwary', 'high-risk', 'eastern', 'powdered', 'unregulated', 'net', 'overseas', 'fourth', 'net', 'inferior', 'total', 'different', 'high', 'York-based', 'black', 'widespread', 'exclusive', 'top', 'Chinese', 'sloppy', 'outright', 'monthly', 'legendary', 'made-for-TV', 'tall', 'five-year', 'utilitarian', 'daunting', 'technical', 'OK', 'historical', 'historical', 'great', 'Spanish', 'Korean', 'Last', 'regional', 'world-class', 'Early', 'medical', 'high-yield', 'top', 'unwilling', 'worthy', 'prickly', 'free', 'white-spirits', 'useful', 'Network-access', 'long-distance', 'Negotiable', 'undeveloped', 'black', 'net', 'median', 'interesting', 'special-interest', 'widespread', 'Short-term', 'pregnant', 'fat', 'Junior', 'yellow', 'debatable', 'statewide', 'bleak', 'nasty', 'Arkansas-based', 'Other', 'net', 'after-tax', 'Other', 'medical', 'raccoon-skin', 'preferred', 'top', 'undue', 'white-walled', 'socalled', 'tame', 'medical', '505-455', 'net', 'open', 'median', 'Malaysian', 'alive', '10-year', 'free', 'freight-transport', 'different', 'different', 'stock-for-debt', 'medical', 'enough', 'all-too-familiar', 'single', 'Last', 'lift-ticket', 'computer-related', 'unfair', 'Vietnamese', 'distinct', 'fourth', '20-year', 'right', 'much', 'company-owned', 'company-owned', 'galvanized', 'net', 'powerful', 'Third-quarter', 'top', 'multimillion-dollar', 'U.S.-backed', 'smart', 'double', 'international', 'fourth', 'unsettling', 'open', 'black', 'soft', 'medical', 'international', 'creative', 'soft', 'five-cent', 'pessimistic', 'high', 'much', 'double', 'High-end', 'unfortunate', 'reluctant', 'pro-choice', 'sudden', '190.58-point', 'fantastic', 'usual', 'all-day', 'different', 'powerful', 'widespread', 'emotional', 'alive', 'precious', 'a.k.a', 'human', 'Other', 'wholesale', 'net', 'fourth', 'sudden', 'extraordinary', 'long-distance', 'Chinese'])

('NN', 'NNP', 289,
['mature', 'Stock', 'Mattel', 'Hot', 'Spain', 'Sala', 'Ostrager', 'Nynex', 'Cie', 'Block', 'Trinova', 'Vietnam', 'Roper', 'Jack', 'Income', 'Mercantile', 'Royal', 'Olympia', 'Torstar', 'Toronto', 'Carat', 'Day', 'Exodus', 'Stock', 'View', 'Tire', 'Privatization', 'Lombardi', 'Calisto', 'Richards', 'Richards', 'Group', 'Coopers', 'Blackstone', 'Lufkin', 'N.H.', 'Joan', 'Tharp', 'Mercury', 'DAYAC', 'Heineken', 'Artois', 'mature', 'Laphroaig', 'Antique', 'Stock', 'Westmoreland', 'Stock', 'Industry', 'Western', 'Buksbaum', 'Noxell', 'Block', 'Windflower', 'Hawaiian', 'Hans-Dietrich', 'Tharp', 'Ford', 'Aldrich', 'Campbell', 'Majority', 'Parretti', 'Stock', 'Price', 'Riverside', 'Mexico-United', 'Group', 'CSC', 'Mitsubishi', 'Kaolin', 'Pride', 'Oliver', 'Donahue', 'J.C.', 'Strum', 'Wine', 'KLM', 'Royal', 'Elrick', 'HHS', 'De', 'Chernobyl', 'HLR', 'Amityville', 'Rafael', 'Morris', 'Murray', 'Shaevitz', 'Renzas', 'Bond', 'Perth', 'Rafael', 'III', 'Erie', 'Parker', 'Construction', 'Equipment', 'Crane', 'Elgin', 'Electronics', 'Erie', 'NASAA', 'Textron', 'Avdel', 'Kuala', 'Price', 'Parker', 'F-16', 'ALQ-178', 'Rapport', 'III', 'Yoshiaki', 'Stanley', 'Stock', 'Housing', 'Madson', 'Royce', 'Lufkin', 'Joseph', 'Sullivan', 'Helane', 'Iraq', 'Parker', 'Story', 'Fault', 'Hyman', 'Stores', 'Stores', 'Mattel', 'Segundo', 'Fine', 'Stores', 'Radio', 'Vietnam', 'Keg', 'Jack', 'MacAllister', 'NAS', 'NH', 'Twenties', 'Vietnam', 'Stock', 'Tandy', 'Fluor', 'BBDO', 'Worldwide', 'Cellular', 'Carl', 'Mercantile', 'Seidman', 'Cynthia', 'Turk', 'Kajima', 'Stock', 'Jiotto', 'Oracle', 'Anaheim-Santa', 'Ana', 'Assurances', 'Nynex', 'Stock', 'Jacobs', '79-year-old', 'Mississippian', 'Tennessee', 'Daniel', 'Computer', 'Golomb', 'Bond', 'Thevenot', 'Concord', 'Finance', 'Eagleton-Newark', 'Gatos', 'Circuit', 'Stanley', 'IMS', 'Della', 'View', 'Radio', 'Voice', 'Carrion', 'Sterling', 'Sailing', 'Joseph', 'Vittoria', 'Thunderbird', 'Mercury', 'Cougar', 'Ford', 'O.', 'Honduras', 'V.', 'Myron', 'Diebel', 'Jolla', 'Trinova', 'Kathy', 'Stanwick', 'Scandinavian', 'Base', 'Coopers', 'KPMG', 'Marwick', 'Private', 'Ripper', 'IMA', 'Lufkin',

'Conning', 'Hartford', 'Olsen', 'Boddington', 'Quayle', 'Jack', 'Lido', 'Gintel', 'Manufacturers', 'Renault', 'Flying', 'Tiger', 'Lowry', 'Consulting', 'Group', 'Mitsubishi', 'Mitsubishi', 'Sterling', 'Contract', 'Stock', 'Indochina', 'Putnam', 'III', 'Block', 'Tandy', 'Philips', 'Renault', 'Renault', 'Story', '20th', 'Springs', 'Sunday', 'Stock', 'Meridian', 'Club', 'Covert', 'Disneyland', 'Morris', 'Morris', 'Richmond', 'Stock', 'Stores', 'Dillard', 'Stores', 'Quotron', 'Silver', 'McDonald', 'Wames', 'Sharps', 'Pixley', 'Stock', 'Westmoreland', 'Acton', 'Tom', 'S&L', 'Coopers', 'Coopers', 'Shioya', 'Kursk', 'Crandall', 'U.S.S.R.', 'Hoffman', 'Scandinavian', 'Jupiter', 'Morris', 'Morris', 'Brewing', 'Motorola', 'Newswire', 'Trading', 'Stock', 'PLO', 'Hiroyuki', 'Club', 'Stock', 'DJ', 'Mercury', 'ASSETS', 'Honeywell', 'Rafael', 'Carrion'])

('NN', 'JJ', 276,
['greedy', 'young', 'Initial', 'ever-narrowing', 'huge', 'great', '60-inch', 'world-wide', 'solar', '30-year', 'chief', 'chief', 'chief', 'homeless', 'great', 'chief', 'same-store', 'crazy', 'preliminary', '17-member', 'impending', '44-cent-a-barrel', 'Lucullan', 'onetime', 'unsecured', 'average', 'federal-local', 'secret', 'positive', 'ultimate', 'DC-9', 'chief', 'Corp.-Toyota', 'hourly', 'inflation-adjusted', 'conservative', 'fine', 'chief', 'cumulative', 'prospective', 'steep', 'average', 'indirect', 'resettable', 'professional', 'mid-afternoon', 'narrow', 'current', 'multibillion-dollar', 'electrogalvanized', 'one-time', 'nightly', 'flower-bordered', 'subordinate', 'sensitive', 'electrolytic', 'net', 'tax-and-budget', 'single-malt', 'certain', 'full-time', 'Afrikaner', 'chief', 'numerous', 'ad-supported', 'net', 'conservative', 'first-class', 'average', 'net', 'mutual', 'untold', 'deleterious', 'necessary', 'Empty', 'Left-stream', 'structural', 'hard-hit', 'Houston-based', 'industrial', 'gas-station', 'fine', 'on-site', 'flagging', 'trendy', 'huge', 'full-page', 'executive', 'intellectual', 'certain', 'net', 'necessary', 'official', 'sexual', 'sour', 'occasional', 'effective', 'so-so', 'positive', 'EGA-VGA', 'potential', 'chief', 'nonperforming', 'certain', 'world-wide', 'short', 'LEBANESE', 'one-time', 'Health-care', 'industrial', 'industrial', 'quarterly', 'elusive', 'thick', 'modern', 'brief', 'contemporary', 'peculiar', 'potential', 'short', 'yearly', 'modest', 'preliminary', 'modest', 'chief', 'excess', 'psychological', 'galvanized', 'Short', 'Thermal', 'mutual', 'certain', 'net', 'alert', 'ready', 'cumulative', 'quarterly', 'effective', 'chief', 'dramatic', 'public', 'weekly', 'early-morning', 'pork-barrel', 'huge', 'rapid', 'still-daylighted', 'conservative', 'chief', 'disproportionate', 'intellectual', 'glorious', 'huge', 'certain', 'net', 'industry-funded', 'modest', 'Silver', 'drought-ravaged', 'secret', 'hard-hit', 'variable-rate', 'certain', 'appellate', 'chief', 'hazardous', 'seductive', 'prospective', 'official', 'last-minute', 'conservative', 'negotiable', 'naval', 'hourly', 'title-insurance', 'search-and-examination', 'choppy', 'short', 'foreign-exchange', 'quick', 'chief', 'advisory', 'chief', 'conservative', 'conservative', 'chief', 'precise', 'chief', 'average', 'chief', 'principal', 'western-style', 'famous', 'two-step', '25-cent-a-share', 'certain', 'official', 'non-financial', 'satirical', 'cellular', 'western', 'positive', 'potential', 'class-action', 'flawed', 'conservative', 'dual', 'official', 'conservative', 'Retail', 'Crude', 'preliminary', 'decent', 'non-dischargable', 'narrow', 'secret', 'collective', 'modest', 'great', 'mutual', 'official', 'self-explanatory', 'applicable', 'distinctive', 'preferred', 'startling', 'abnormal', 'wonderful', 'average', 'chief', 'huge', 'quarterly', 'federal-systems', 'probable', 'primordial', 'definite', 'worth', 'conscientious', 'three-year', 'same-store', 'peculiar', 'higher-than-expected', 'organizational', 'overwhelming', 'vast', 'coal-fired', 'stress-provoking', 'world-wide', 'synthetic-leather', 'interactive', 'Long-term', 'overseas', 'short', 'great', 'one-way', 'highest-volume', 'steep', 'fill-or-kill', 'quiescent', 'disadvantaged', 'four-year-old', 'ready', 'mutual', 'certain', 'chief', 'two-day', 'media-buying', 'municipal', 'opposite', 'chief', 'official', 'certain', 'huge', 'one-time', 'electric', 'Sino-foreign'])

('NNP', 'JJ', 252,
['snake-oil', 'Callable', 'possible', 'one-year', 'good', 'cold', 'bargain-basement', 'California', 'arched', 'young', 'white', 'South', 'diplomatic', 'rough', 'other', 'complete', 'independent', 'good', 'proverbial', 'inflation-adjusted', 'veto-proof', 'immediate', 'sluggish', 'potential', '20th', 'stable', 'two-part', 'American', 'top-level', 'American', 'British', '52-week', 'cash-hungry', 'ambitious', 'costly', 'Gargantuan', 'Victorian', 'Mass.-based', 'Western', 'alma', 'tear-jerking',

'removable', 'other', 'world-wide', 'Western', 'leveraged', 'Soviet', 'military', 'nuclear-power', 'corrosion-resistant', 'FEDERAL', 'volatile', 'volatile', 'optimum', 'Argentine', '30th', 'sluggish', 'Polish', 'lively', 'Big', 'commemorative', 'British', '500-stock', 'standard', 'California', 'pro-abortion', 'American', 'Italian', 'one-year', 'West', 'then-pending', 'tidal', 'weak', 'fresh', 'scary', 'bank-backed', 'Blue', 'costly', 'Swedish', 'potential', 'British', 'Armenian', 'beholden', 'other', 'stylistic', 'costly', 'low-budget', 'Chemical', 'current', 'British', 'unreported', 'personal', 'rebellious', 'low', 'other', 'two-year', 'public', 'other', 'Longtime', 'intellectual', 'significant', 'American', 'world-wide', 'additional', 'ill-suited', 'British', 'American', 'mutual-fund', 'U.K.', 'senior', 'normal', 'other', 'other', 'British', 'other', '50-story', 'red', 'convenient', 'time-strapped', 'World-wide', 'British', 'good', 'several', 'Frequent', 'short', 'interim', 'several', 'unexpected', 'quiet', 'multi-agency', 'good', 'MUTUAL', 'official', 'insolvent', 'unlawful', 'hazardous', 'young', 'Baltic', 'apparent', 'cumulative', 'cumulative', 'scientific', 'Long', 'official', 'assorted', 'disciplinary', 'composite', 'other', 'iced', 'suburban', 'mid-range', 'South', 'American', 'American', 'orthodox', '40-a-share', 'unsure', 'other', 'liquified', 'self-regulatory', 'Second', 'executive', 'New', 'unable', 'economic', 'British', 'New', 'separate', 'other', 'Year-earlier', 'possible', 'British', 'radiophonic', 'short', 'fetal-tissue', 'Honduran', 'other', 'South', 'volatile', 'low', 'all-time', 'wireline', 'several', 'quiet', 'wrong', 'British', 'British', 'bank-backed', 'American', 'similiar', 'other', 'U.S.-built', 'sober', 'parental-consent', 'public', 'recession-wary', 'well-entrenched', 'restrictive', 'Year-earlier', 'public', 'cumulative', 'cumulative', 'British', 'government-plus', 'rough', 'incompetent', 'other', '52-week', 'hopeful', 'affluent', 'executive', 'modern', 'public', 'common-stock', 'Next', 'favorable', 'flip-flopped', 'Second', 'communist', 'East', 'start-up', 'inflation-adjusted', 'other', 'certain', 'German', 'unsecured', 'Personal', 'Canadian', 'analogous', 'good', 'indoor', 'British', 'West', 'other', 'variable-rate', 'unsold', 'crucial', 'gawky', 'MIG-1', 'blind-sided', 'cold', 'unsuccessful', '23-5', 'fine', 'idle', 'plentiful', 'British', 'disappointing', 'warm-weather', 'other', 'second-story', 'southern'])

('NN', 'NNS', 210,
['receipts', 'foes', 'loopholes', 'mortgages', 'mortgages', 'compatriots', 'capital-gains', 'capital-gains', 'bailouts', 'gases', 'laptops', 'seven-eighths', 'notes', 'tons', 'rides', 'transactions', 'lords', 'hinterlands', 'staffers', 'disposals', 'notes', 'unions', 'targets', 'chemicals', 'sections', 'greats', 'materials', 'obligations', 'tons', 'tons', 'debts', 'hundreds', 'transactions', 'Partnerships', 'CDs', 'shelves', 'vaccines', 'deliveries', 'troubles', 'drawbacks', 'charges', 'countrymen', 'thistles', 'pears', 'capital-gains', 'times', 'Workers', 'mid-1970s', 'consumer-electronics', 'neighbhorhoods', 'sons', 'communities', 'pledges', 'reprisals', 'abrasives', 'ceramics', 'times', 'ones', 'Charlestonians', 'capital-gains', 'times', 'processors', 'giants', 'materials', 'mortgages', 'mortgages', 'sources', 'economics', 'communities', 'ones', 'CDs', 'repairs', 'charges', 'patients', 'Strategies', 'write-downs', 'goals', 'spirits', 'giants', 'PENCILS', 'reformers', 'promotions', 'uses', 'sources', 'notes', 'notes', 'increases', 'counties', 'approaches', 'forces', 'bottles', 'expressions', 'mutters', 'materials', 'chemicals', 'computers', 'communities', 'ministers', 'amounts', 'capital-gains', 'fighters', 'Places', 'families', 'components', 'switchers', 'disposals', 'lips', 'cosmetics', 'debts', 'counties', 'spills', 'stilts', 'Republicans', 'communists', 'mothers', 'unions', 'hundreds', 'chemicals', 'chemicals', 'reinforcements', 'deliberations', 'amounts', 'Skeptics', 'Readers', 'commentaries', 'times', 'notes', 'ministers', 'computers', 'times', 'piers', 'piers', 'processors', 'spirits', 'charges', 'criticisms', 'times', 'orphans', 'Executives', 'ventures', 'charges', 'forces', 'notes', 'notes', 'notes', 'essays', 'trivia', 'envelopes', 'ideas', 'troubles', 'buffs', 'families', 'materials', 'families', 'fundamentals', 'ventures', 'amounts', 'odds', 'hearts', 'PACS', 'ventures', 'materials', 'fixtures', 'collars', 'tons', 'tons', 'sunflowers', 'periods', 'devices', 'charges', 'pacemakers', 'amenities', 'notes', 'times', 'gases', 'multimedia', 'ways', 'releases', 'sources', 'ways', 'ways', 'charges', 'commuters', 'woes', 'Stores', 'foes', 'deliveries', 'charges', 'interiors', 'growers', 'sidelines', 'tons', 'rentals', 'inflows', 'Lines', 'repairs', 'notes', 'releases', 'tags', 'times', 'times', 'notes', 'fundamentals', 'transactions', 'amounts', 'fundamentals', 'inflows', 'releases', 'transports', 'ventures'])

('NNP', 'NN', 209,
['boiler-room', 'hotdog', 'freshman', 'treasury', 'saying', 'Energy', 'countryside', 'f', 'access', 'B.A.T', 'defamation', 'silver', 'group', 'fear', 'leniency', 'chlorine', 'Trading', 'front', 'receivables', 'hierarchy', 'wedge', 'producer', 'regime', 'rhythm', 'nonpriority', 'surface', 'court', 'instant', 'influx', 'computer-maintenance', 'reunion', 'maneuvering', 'producer', 'pie', 'break', 'producer', 'Insurance', 'lounge', 'Earth-quake', '5', 'city', 'rigor', 'exchange', 'shop', 'front', 'presidency', 'operating', 'motorbike', 'official', 'weighting', 'daughter', 'embarrassment', 'poker', 'Newsprint', 'postage', 'simplicity', 'drug-industry', 'lifestyle', 'State', 'furrier', 'assistance', 'talent', 'mailing', 'halt', 'radiation', 'accounting', 'Transportation', 'boilerplate', 'dog', 'PLASTIC', 'impetus', 'fixed-income', 'prevention', 'phone-company', 'basket', 'relocation', 'overdependence', 'utilization', 'ft.', 'Transport', 'sum', 'cocaine', 'front', 'bullet', 'sum', 'processing', 'octane', 'prosecution', 'facade', 'supplier', 'ideologist', 'managing', 'selling', 'jail', 'Man', 'royalty', 'copper', 'marine', 'receivables', 'ocean', 'disturbance', 'nervousness', 'Interest', 'sponsorship', 'witha', 'seafood', 'monitor', 'reunification', 'decree', 'date', 'flight', 'arbitrage', 'parity', 'disposal', 'eclectic', 'scrutiny', 'date', 'Trading', 'flashlight', 'correspondence', 'pressure', 'binge', 'assistance', 'magnitude', 'east', 'copper', 'default', 'default', 'asbestos', 'cinematographer', 'ploy', 'feel', 'help', 'Defense', 'misrepresentation', 'trash', 'victor', 'BEAT', 'conquest', 'B.A.T', 'murderer', 'stock-market', 'State', 'soybean', 'actor', 'overflow', 'restriction', 'brewer', 'leasing', 'hotel\/casino', 'volcano', 'Trading', 'analysis', 'industry', 'keep', 'carry', 'allocation', 'deficit', 'fluoride', 'City', 'fate', 'reconstruction', 'starvation', 'honeymoon', 'operating', 'Administration', 'surgery', 'highway', 'constituency', 'Journal', 'transport', 'patron', 'default', 'flatness', 'today', 'ownership', 'greenfield', 'revolutionary', 'regime', 'cogeneration', 'cogeneration', 'east', 'sedan', 'plume', 'break-up', 'pineapple', 'minicomputer', 'investing', 'palm', 'clerk', 'front', 'pong', 'lady', 'transport', 'building', 'fear', 'treatment', 'treatment', 'safety', 'consideration', 'constituency', 'producer', 'billing', 'productivity', 'motel', 'excitement', 'Oil', 'operating', 'peso'])

('NNS', 'NN', 201,
['medicine', 'humor', 'landing', 'depressant', 'novelist', 'Market', 'net', 're-election', 'duck', 'duck', 'duck', 'evaluation', 'challenge', 'round', 'net', 'glory', 'proviso', 'tolerance', 'coddling', 'crude', 'much', 'steakhouse', 'bribe', 'chloride', 'creature', 'formula', 'much', 'early-retirement', 'spokesperson', 'hamburger', 'destruction', 'post', 'total', 'wool', 'epic', 'subcompact', 'kicker', 'golf', 'anthem', 'pretext', 'Overhead', 'malnourishment', 'border', 'element', 'injection', 'bull-market', 'bail', 'landing', 'bus', 'booze', 'gig', 'mania', 'admission', 'audience', 'prohibition', 'faculty', 'manner', 'challenge', 'concept', 'net', 'hawk', 'perjury', 'reversal', 'veto', 'medication', 'total', 'hassle', 'artist', 'portrait', 'inability', 'tissue', 'Bonfire', 'restatement', 'destruction', 'artist', 'humor', 'shrinkage', '345-47', 'boost', 'dialing', 'Trim', 'voice', 'brain', 'flair', 'dare', 'lease', 'annuity', 'upsurge', 'fragment', 'post', 'randomness', 'anonymity', 'crude', 'similarity', 'rendition', 'angora', 'DEPOSIT', 'newsman', 'veto', 'border', 'allure', 'extermination', 'much', 'daze', 'shantytown', 'steak', 'drummer', 'probation', 'transit', 'mound', 'medicine', 'souvenir', 'fish', 'audacity', 'outage', 'Someone', 'command', 'net', 'staffing', 'staffing', 'border', 'encounter', 'strongman', 'photo', 'boundary', 'top', 'reversal', 'much', 'guardian', 'guardian', 'eagerness', 'detective', 'salespeople', 'Poverty', 'insider', 'behest', 'round', 'R2-D2', 'brewery', 'leasing', 'super-charger', 'vector', 'briefcase', 'souvenir', 'binding', 'landing', 'libel', 'top', 'audience', 'populism', 'retreat', 'mania', 'VIDEO', 'right', 'legislating', 'larceny', 'artist', 'notch', 'right', 'lumber', 'grave', 'spokesperson', 'boost', 'overseas', 'language', 'order-taking', 'skin', 'cent', 'sheep', 'roadblock', 'misdemeanor', 'signature', 'twist', 'affidavit', 'WORLD', 'outage', 'epicenter', 'verdict', 'bushel', 'bushel', 'round', 'thickness', 'net', 'mode', 'telex', 'crude', 'drug-policy', 'destruction', 'desire', 'cousin', 'horizon', 'stock-appreciation', 'total', 'rub', 'concept', 'sewage', 'terrorism', 'salespeople', 'topic', 'village', 'Vacation'])

('NNS', 'NNP', 188,
['Bateman', 'Wenz', 'Hickman', 'Allenport', 'Ownership', 'Sala', 'Benton', 'Fogg', 'Prizm', 'Corolla', 'Financiere', 'Pizza', 'Glaser', 'Daggs', 'Little', 'WCRS', 'Susie', 'Last', 'Small', 'Investors', 'Penn', 'N.A.', 'Mace', 'Stan', 'Himebaugh', 'Himebaugh', 'Thai', 'Rules', 'Stella', 'EMPIRE', 'PENCIL', 'Empire-Berol', 'Yardeni', 'Stanford', 'Sotheby', 'Securities', 'Interpublic', 'Lisa', 'Lockheed', 'Lubar', 'Lubar', 'Norton', 'Filmworks', 'Deep', 'Foreign', 'Giancarlo', 'Ferdinand', 'Marcos', 'Radical', 'Pollin', 'Isle', 'Doosan', 'Unimin', 'Petroleum', 'Parkshore', 'Kann', 'Rune', 'Prague', 'Marlo', 'Phil', 'Penney', 'Little', 'Sleeping', 'Lavidge', 'Palma', 'Ruffel', 'MC', 'Watson', 'Kaitaia', 'Medical', 'Charter', 'Halloween', 'Livermore', 'Freightways', 'Willens', 'Aoun', 'Amfac', 'Petersburg', 'Isuzu', 'Daiwa', 'ENGLAND', 'HASTINGS', 'Market-If-Touched', 'Rolls', 'Brent', 'Alcatraz', 'Morton', 'Ryan', 'Becker', 'Toensing', 'Triad', 'Slater', 'CDC', 'Beta', 'Veterans', 'Veterans', 'Oberstar', 'Aviation', 'Arkansas', 'Futures', 'PANDA', 'Accident', 'Arkansas', 'Autodesk', 'Novell', 'Mateyo', 'Carnegie', 'Declaration', 'Zane', 'Mann', 'Municipal', 'Advisor', 'Ledger', 'A.F.', 'Cohen', 'Cotton', 'Helliesen', 'Jaguar', 'Average', 'Fox', 'Banco', 'Avis', 'Willens', 'Courter', 'Beat', 'Laurance', 'NWA', 'Courter', 'Stanford', 'Peat', 'Deerfield', 'Cologne', 'Schloss', 'Securities', 'Gilder', 'Mercedes', 'Isle', 'Kenton', 'Accident', 'Advertising', 'Saatchi', 'Saatchi', 'Advertising', 'Advertising', 'AON', 'Chubb', 'Banco', 'CSC', 'Philippe', 'Felipe', 'Oswald', 'Metromedia', 'Qintex', 'Corroon', 'Lynn', 'Hollister', 'Fisher', 'KTXL', 'Salerno', 'Kumagai-Gumi', 'Electronic', 'Mayer', 'Platt', 'Banks', 'Accounting', 'Budweiser', 'MADD', 'Judge', 'Mississippi', 'Fox', 'Sidorenko', 'NWA', 'Michele', 'MeraBank', 'PR', 'Midway', 'Gregory', 'Bessemer', 'Courter', 'Rohrer', 'Investors', 'Securities', 'Madrid', 'NWA', 'Ginn', 'Cadillac', 'Underseas', 'Banco'])

('WDT', 'IN', 180,
['that', 'that'])

('VBN', 'VBD', 176,
['found', 'held', 'made', 'permitted', 'topped', 'offered', 'sold', 'stopped', 'based', 'appointed', 'expected', 'SURGED', 'stopped', 'involved', 'called', 'played', 'flew', 'set', 'gathered', 'tried', 'expected', 'received', 'made', 'perceived', 'provided', 'became', 'ruled', 'indicated', 'raised', 'developed', 'discovered', 'ignored', 'headed', 'moved', 'showed', 'reduced', 'called', 'acquired', 'moved', 'made', 'issued', 'played', 'teamed', 'produced', 'received', 'used', 'proposed', 'brought', 'interviewed', 'argued', 'set', 'fought', 'charged', 'defended', 'set', 'filled', 'sought', 'forced', 'reacted', 'became', 'grew', 'passed', 'called', 'made', 'increased', 'declared', 'meant', 'made', 'urged', 'stemmed', 'showed', 'won', 'sought', 'heard', 'flocked', 'authorized', 'became', 'suggested', 'financed', 'found', 'revised', 'showed', 'created', 'offered', 'favored', 'displayed', 'became', 'disclosed', 'offered', 'stopped', 'grew', 'showed', 'indicated', 'designed', 'gathered', 'borrowed', 'headed', 'grew', 'sold', 'ranked', 'found', 'spotted', 'became', 'pointed', 'caused', 'offered', 'beefed', 'found', 'slept', 'became', 'offered', 'waited', 'uncovered', 'fined', 'caught', 'represented', 'resumed', 'moved', 'showed', 'showed', 'heard', 'upheld', 'reacted', 'tried', 'ruled', 'killed', 'subordinated', 'opened', 'dumped', 'made',

'increased', 'filed', 'dumped', 'dispatched', 'aimed', 'drove', 'found', 'established', 'tried', 'compared', 'narrowed', 'opposed', 'showed', 'caused', 'made', 'suggested', 'surveyed', 'topped', 'authorized', 'left', 'sold', 'highlighted', 'topped', 'collapsed', 'moved', 'pictured', 'made', 'pressed', 'planned', 'discovered', 'signed', 'fought', 'became', 'pulled', 'sped', 'measured', 'damaged', 'disclosed', 'tightened', 'trimmed', 'called', 'grew', 'realized', 'led', 'owned', 'protected'])

Part 2.5 Hyper-parameter Tuning

In order to improve your model performance, try making some modifications on `EMBEDDING_DIM`, `HIDDEN_DIM`, and `LEARNING_RATE`. You will receive 50%/75%/100% credit for this section if your model, after being trained for 10 epochs, is able to achieve 80%/85%/90% accuracy on the validation set.

```
YOUR_EMBEDDING_DIM = 8
YOUR_HIDDEN_DIM = 16
YOUR_LEARNING_RATE = 0.001

##############################################################################
# TODO: Set three hyper-parameters. Initialize the model, optimizer and the loss function
# Hint, you may want to use reduction='sum' in the CrossEntropyLoss function
##############################################################################
model = BasicPOSTagger(YOUR_EMBEDDING_DIM, YOUR_HIDDEN_DIM, len(word_to_idx), len(tag_to_idx))
loss_function = nn.CrossEntropyLoss(reduction='sum')
optimizer = optim.Adam(model.parameters(), YOUR_LEARNING_RATE)
##############################################################################
#                              END OF YOUR CODE                              #
##############################################################################
for epoch in range(1, EPOCHS + 1):
    train(epoch, model, loss_function, optimizer)
```

```
Epoch: 1/10     Avg Train Loss: 1.7239  Avg Val Loss: 1.2019     Val Accuracy: 65
Epoch: 2/10     Avg Train Loss: 0.9858  Avg Val Loss: 0.8537     Val Accuracy: 76
Epoch: 3/10     Avg Train Loss: 0.7139  Avg Val Loss: 0.6694     Val Accuracy: 82
Epoch: 4/10     Avg Train Loss: 0.5541  Avg Val Loss: 0.5599     Val Accuracy: 85
Epoch: 5/10     Avg Train Loss: 0.4510  Avg Val Loss: 0.4914     Val Accuracy: 87
Epoch: 6/10     Avg Train Loss: 0.3794  Avg Val Loss: 0.4461     Val Accuracy: 88
Epoch: 7/10     Avg Train Loss: 0.3266  Avg Val Loss: 0.4145     Val Accuracy: 89
Epoch: 8/10     Avg Train Loss: 0.2863  Avg Val Loss: 0.3918     Val Accuracy: 90
Epoch: 9/10     Avg Train Loss: 0.2544  Avg Val Loss: 0.3758     Val Accuracy: 90
Epoch: 10/10    Avg Train Loss: 0.2284  Avg Val Loss: 0.3639     Val Accuracy: 91
```

Part 3 Character-level POS Tagger [15 points]

Use the character-level information to augment word embeddings. For example, words that end with -ing or -ly give quite a bit of information about their POS tags. To incorporate this information, run a character-level LSTM on every word to create a character-level representation of the word. Take the last hidden state from the character-level LSTM as the representation and concatenate with the word embedding (as in the BasicPOSTagger) to create a new word representation that captures more information.

```
# Create char to index mapping
char_to_idx = {}
```

```
unique_chars = set()
MAX_WORD_LEN = 0

for sent in train_sentences:
    for word in sent:
        for c in word:
            unique_chars.add(c)
        if len(word) > MAX_WORD_LEN:
            MAX_WORD_LEN = len(word)

for c in unique_chars:
    char_to_idx[c] = len(char_to_idx)
char_to_idx[' '] = len(char_to_idx)

print(char_to_idx)
```

```
{'n': 0, '*': 1, 'p': 2, '$': 3, 'Y': 4, '8': 5, 'D': 6, 'N': 7, 'G': 8, 'R': 9, '6': 10,
```

```
def prapareChar(sentence, char_to_idx):
    char_tensor = []
    for s in sentence:
      for w in s:
        ch = w
        if len(w) < MAX_WORD_LEN:
          pad_num = MAX_WORD_LEN - len(w)
          ch = ""
          for i in range(pad_num):
            ch += " "
          ch += w
        w_tensor = [char_to_idx[ch[i]] for i in range(MAX_WORD_LEN)]
      char_tensor.append(w_tensor)
    char_tensor = torch.tensor(char_tensor).long()
    return char_tensor

prapareChar(train_sentences, char_to_idx)



# char_tensor, _ = prepare_sequence(chars, train_tags, char_to_idx, tag_to_idx)
# print(char_tensor)
```

```
tensor([[80, 80, 80,  ..., 80, 80, 18],
        [80, 80, 80,  ..., 80, 80, 18],
        [80, 80, 80,  ..., 80, 80, 18],
        ...,
        [80, 80, 80,  ..., 80, 80, 18],
        [80, 80, 80,  ..., 80, 80, 18],
        [80, 80, 80,  ..., 80, 80, 18]])
```

Aside: Padding

For this project, we are not coding in batches (as you can see, each training loop runs on a single sentence per iteration). However, padding is a very important aspect of training, so we describe it in the section below.

How to do padding correctly for the characters?

Assume we have got a sentence ["We", "love", "NLP"]. You are supposed to first prepend a certain number of blank characters to each of the words in this sentence.

How to determine the number of blank characters we need? The calculation of MAX_WORD_LEN is here for help (which we already provide in the starter code). For the given sentence, MAX_WORD_LEN equals 4. Therefore we prepend two blank characters to "We", zero blank character to "love", and one blank character to "NLP". So the resultant padded sentence we get should be [" We", "love", " NLP"].

Then, we feed all characters in [" We", "love", " NLP"] into a char-embedding layer, and get a tensor of shape (3, 4, char_embedding_dim). To make this tensor's shape proper for the char-level LSTM (nn.LSTM), we need to transpose this tensor, i.e. swap the first and the second dimension. So we get a tensor of shape (4, 3, char_embedding_dim), where 4 corresponds to seq_len and 3 corresponds to batch_size.

The last thing you need to do is to obtain the last hidden state from the char-level LSTM, and concatenate it with the word embedding, so that you can get an augmented representation of that word.

This is an illustration for left padding characters.

Why doing the padding?

Someone may ask why we want to do such a kind of padding, instead of directly passing each of the character sequences of each word one by one through an LSTM, to get the last hidden state. The reason is that if you don't do padding, then that means you can only implement this process using "for loop". For CharPOSTagger, if you implement it using "for loop", the training time would be approximately 150s (GPU) / 250s (CPU) per epoch, while it would be around 30s (GPU) / 150s (CPU) per epoch if you do the padding and feed your data in batches. Therefore, we strongly recommend you learn how to do the padding and transform your data into batches. In fact, those are quite important concepts which you should get yourself familar with, although it might take you some time.

Why doing left padding?

Our hypothesis is that the suffixes of English words (e.g., -ly, -ing, etc) are more indicative than prefixes for the part-of-speech (POS). Though LSTM is supposed to be able to handle long sequences, it still lose information along the way and the information closer to the last state (which you use as char-level representations) will be retained better.

How to understand the dimention change?

Assume we have got a sentence with 3 words ["We", "love", "NLP"], and assume the dimension of character embedding is 2, the dimension of word embedding is 4, the dimension of word-level LSTM's hidden layer is 5, the dimension of character-level LSTM's hidden layer is 6.

In BasicPOSTagger, the dimension change would be (3x1x4) ----word-level LSTM----> (3x1x5) ----linear layer----> (3x1x44).

In CharPOSTagger, after padding, character embedding, and swapping, the dimension change would be (MAX_WORD_LEN, 3, 2) ----character-level LSTM----> (MAX_WORD_LEN, 3, 6) ----Take the last hidden state----> (3, 6) ----concatenate with word embedings----> (3x1x10) ----word-level LSTM----> (3x1x5) ----linear layer----> (3x1x44).

Part 3.1 Define CharPOSTagger Model

```
# New Hyperparameters
EMBEDDING_DIM = 4
HIDDEN_DIM = 8
LEARNING_RATE = 0.1
LSTM_LAYERS = 1
DROPOUT = 0
EPOCHS = 10
CHAR_EMBEDDING_DIM = 4
CHAR_HIDDEN_DIM = 4
```

```
from torch.nn.modules import dropout
class CharPOSTagger(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, char_embedding_dim,
                 char_hidden_dim, char_size, vocab_size, tagset_size):
        super(CharPOSTagger, self).__init__()
        #############################################################################
        # TODO: Define and initialize anything needed for the forward pass.
        # You are required to create a model with:
        # an embedding layer for word: that maps words to their embedding space
        # an embedding layer for character: that maps characters to their embedding space
        # a character-level LSTM layer: that finds the character-level embedding for a word
        # a word-level LSTM layer: that takes the concatenated representation per word (word emb
        # a linear layer: maps from hidden state space to tag space
        #############################################################################
        self.embd_w = nn.Embedding(vocab_size, embedding_dim)
        self.embd_c = nn.Embedding(char_size, char_embedding_dim)
        self.lstm_c = nn.LSTM(char_embedding_dim, char_hidden_dim, num_layers=LSTM_LAYERS, drop
        self.lstm_w = nn.LSTM(embedding_dim + char_hidden_dim, hidden_dim, num_layers=LSTM_LAYE
        self.fc = nn.Linear(hidden_dim, tagset_size)

        #############################################################################
        #                             END OF YOUR CODE                              #
        #############################################################################

    def forward(self, sentence, chars):
        tag_scores = None
        #############################################################################
        # TODO: Implement the forward pass.
        # Given a tokenized index-mapped sentence and a character sequence as the arguments,
        # find the corresponding raw scores for tags (without softmax)
        # returns:: tag_scores (Tensor)
        #############################################################################
        emw = self.embd_w(sentence)
        # print(emw.shape)
        emc = self.embd_c(chars)
        # print(emc.shape)
        lc, _ = self.lstm_c(emc)
        # print(lc.shape)
        lc_last = lc[:, -1, :]
        # print(lc_last.shape)
        w_c_cat = torch.cat((lc_last, emw), dim=1)
        ls, _ = self.lstm_w(w_c_cat)
        tag_scores = self.fc(ls)
```

```
        ############################################################################
        #                              END OF YOUR CODE                            #
        ############################################################################
        return tag_scores
```

Part 3.2 Training and Evaluation

```
def train_char(epoch, model, loss_function, optimizer):
    model.train()
    train_loss = 0
    train_examples = 0
    for sentence, tags in training_data:
        ############################################################################
        # TODO: Implement the training method
        # Hint: you can use the prepare_sequence method for creating index mappings
        # for sentences. For constructing character input, you may want to left pad
        # each word to MAX_WORD_LEN first, then use prepare_sequence method to create
        # index  mappings.
        ############################################################################
        optimizer.zero_grad()
        #zero out the parameter gradients
        sentence_tensor, tag_tensor = prepare_sequence(sentence, tags, word_to_idx, tag_to_idx)
        char_tensor = prapareChar(sentence, char_to_idx)
        #prepare input data (sentences, characters, and gold labels)
        out = model(sentence_tensor, char_tensor)
        #do forward pass with current batch of input
        loss = loss_function(out, tag_tensor)
        #get loss with model predictions and true labels
        loss.backward()
        optimizer.step()
        #update model parameters
        train_loss += loss.item()
        train_examples+= len(tags)
        #increase running total loss and the number of past training samples

        ############################################################################
        #                              END OF YOUR CODE                            #
        ############################################################################

    avg_train_loss = train_loss / train_examples
    avg_val_loss, val_accuracy = evaluate_char(model, loss_function)

    print("Epoch: {}/{}\tAvg Train Loss: {:.4f}\tAvg Val Loss: {:.4f}\t Val Accuracy: {:.0f}".fo
                                                                      EPOCHS,
                                                                      avg_train_loss,
                                                                      avg_val_loss,
                                                                      val_accuracy))


def evaluate_char(model, loss_function):
    # returns:: avg_val_loss (float)
```

```
    # returns:: val_accuracy (float)
    model.eval()
    correct = 0
    val_loss = 0
    val_examples = 0
    with torch.no_grad():
        for sentence, tags in val_data:
            #############################################################################
            # TODO: Implement the evaluate method
            # Find the average validation loss along with the validation accuracy.
            # Hint: To find the accuracy, argmax of tag predictions can be used.
            #############################################################################

            #prepare input data (sentences, characters, and gold labels)
            sentence_tensor, tag_tensor = prepare_sequence(sentence, tags, word_to_idx, tag_to_
            char_tensor = prapareChar(sentence, char_to_idx)
            #do forward pass with current batch of input
            out = model(sentence_tensor, char_tensor)
            #get loss with model predictions and true labels
            loss = loss_function(out, tag_tensor)
            #get the predicted labels
            pred = torch.argmax(out, dim=1)
            #get number of correct prediction
            correct += (pred == tag_tensor).sum().item()
            #increase running total loss and the number of past valid samples
            val_loss += loss.item()
            val_examples += len(tags)

            #############################################################################
            #                             END OF YOUR CODE                              #
            #############################################################################
    val_accuracy = 100. * correct / val_examples
    avg_val_loss = val_loss / val_examples
    return avg_val_loss, val_accuracy
```

```
#############################################################################
# TODO: Initialize the model, optimizer and the loss function
# Hint, you may want to use reduction='sum' in the CrossEntropyLoss function
#############################################################################
model = CharPOSTagger(EMBEDDING_DIM, HIDDEN_DIM, CHAR_EMBEDDING_DIM, CHAR_HIDDEN_DIM, len(char_
loss_function = nn.CrossEntropyLoss(reduction='sum')
optimizer = optim.Adam(model.parameters(), YOUR_LEARNING_RATE)

#############################################################################
#                             END OF YOUR CODE                              #
#############################################################################
for epoch in range(1, EPOCHS + 1):
    train_char(epoch, model, loss_function, optimizer)
```

```
Epoch: 1/10     Avg Train Loss: 2.0730  Avg Val Loss: 1.4653     Val Accuracy: 59
Epoch: 2/10     Avg Train Loss: 1.1918  Avg Val Loss: 1.0049     Val Accuracy: 71
Epoch: 3/10     Avg Train Loss: 0.8583  Avg Val Loss: 0.7722     Val Accuracy: 78
Epoch: 4/10     Avg Train Loss: 0.6565  Avg Val Loss: 0.6146     Val Accuracy: 84
Epoch: 5/10     Avg Train Loss: 0.5228  Avg Val Loss: 0.5169     Val Accuracy: 86
Epoch: 6/10     Avg Train Loss: 0.4360  Avg Val Loss: 0.4539     Val Accuracy: 88
Epoch: 7/10     Avg Train Loss: 0.3768  Avg Val Loss: 0.4107     Val Accuracy: 89
Epoch: 8/10     Avg Train Loss: 0.3328  Avg Val Loss: 0.3787     Val Accuracy: 90
```

```
Epoch: 9/10     Avg Train Loss: 0.2989  Avg Val Loss: 0.3538     Val Accuracy: 90
Epoch: 10/10    Avg Train Loss: 0.2715  Avg Val Loss: 0.3348     Val Accuracy: 91
```

Sanity Check! Under the default hyperparameter setting, after 5 epochs you should be able to get at least 85% accuracy on the validation set.

Part 3.3 Error analysis

Write a method to generate predictions for the validation set. Create lists of words, tags predicted by the model and ground truth tags.

Then use these lists to carry out error analysis to find the top-10 types of errors made by the model.

This part is very similar to part 1.7. You may want to refer to your implementation there.

```
#############################################################################
# TODO: Generate predictions for val_data
# Create lists of words, tags predicted by the model and ground truth tags.
# Hint: It should look very similar to the evaluate function.
#############################################################################
def generate_predictions(model, val_data):
    # returns:: word_list (str list)
    # returns:: model_tags (str list)
    # returns:: gt_tags (str list)
    # Your code here
    model_tags = []
    gt_tags = []
    word_list = []
    model.train()
    with torch.no_grad():
        for sentence, tags in val_data:
          sentence_tensor, tag_tensor = prepare_sequence(sentence, tags, word_to_idx, tag_to_id
          char_tensor = prapareChar(sentence, char_to_idx)
          out = model(sentence_tensor, char_tensor)
          pred = torch.argmax(out, dim=1)

          model_tag = [idx_to_tag[str(int(x))] for x in pred]

          gt_tag = [idx_to_tag[str(int(i))] for i in tag_tensor]
          words = sentence
          model_tags.append(model_tag)
          gt_tags.append(gt_tag)
          word_list.append(words)


    #############################################################################
    #                             END OF YOUR CODE                              #
    #############################################################################


    return word_list, model_tags, gt_tags

#############################################################################
# TODO: Carry out error analysis
```

```
# From those lists collected from the above method, find the
# top-10 tuples of (model_tag, ground_truth_tag, frequency, example words)
# sorted by frequency
#############################################################################
def error_analysis(word_list, model_tags, gt_tags):
    # returns: errors (list of tuples)
    # Your code here
    errors = []
    errors_list = []
    counts = {}
    example_words = {}

    for i in range(len(word_list)):
      sentence = word_list[i]
      mt = model_tags[i]
      gt = gt_tags[i]
      for s, m ,g in zip(sentence, mt, gt):
        pair = (m, g)
        if m != g and pair not in counts.keys():
          counts[pair] = 1
          example_words[pair] = [s]
        elif m != g and pair in counts.keys():
          counts[pair] +=1
          example_words[pair].append(s)

    for p in counts.keys():
      model_tag, gt_tag = p
      count = counts[p]
      words = example_words[p]
      e = (model_tag, gt_tag, count, words)
      errors_list.append(e)

    fre = []
    for t in errors_list:
      fre.append(t[2])

    fre = torch.argsort(torch.tensor(fre), descending=True)
    for i in fre:
      errors.append(errors_list[i])

    #############################################################################
    #                             END OF YOUR CODE                              #
    #############################################################################

    return errors

word_list, model_tags, gt_tags = generate_predictions(model, val_data)
errors = error_analysis(word_list, model_tags, gt_tags)

for i in errors[:10]:
  print(i)
```

```
('NN', 'JJ', 311, ['slick-talking', 'snake-oil', 'gullible', 'Callable', 'ever-narrowing',
('VBN', 'VBD', 248, ['held', 'made', 'permitted', 'completed', 'Warned', 'fled', 'topped',
('NN', 'NNP', 226, ['Bateman', 'Bryan', 'Egon', 'Harrison', 'IRA', 'Wheeling-Pittsburgh',
('JJ', 'NN', 187, ['commercial', 'depository', 'medicine', 'humor', 'net', 'many', 'shape'
('NNP', 'NN', 181, ['yacht', 'agility', 'Market', 'reflection', 'tandem', 're-election', '
('JJ', 'NNP', 163, ['League', 'mature', 'British', 'Ownership', 'Beau', 'Manion', 'Commerc
```

```
('NNP', 'JJ', 154, ['Initial', 'uncomfortable', 'fixed-rate', 'automatic', 'California', '
('NNS', 'VBZ', 126, ['HAS', 'requires', 'assists', 'targets', 'sells', 'follows', 'shows',
('VBD', 'VBN', 101, ['ended', 'continued', 'brought', 'formed', 'formed', 'approved', 'got
('NN', 'VBP', 82, ['use', 'vanish', 'pay', 'note', 'tax', 'plant', 'point', 'look', 'engag
```

Report your findings in the cell below.

What kinds of errors does the character-level model make as compared to the original model, and why do you think it made them?

Explaination:

The character-level model didn't have much 'IN' and 'WDT' compared to the Basic POS model. But the model still cannot distinguish well between 'NN' and 'JJ' because some adjectives can also be used as nouns, 'VBD' and 'VBN' because some verbs have the same past tense and past participle, and 'NN' and'NNP' because the model didn't detect the first capital letter in 'NNP.' It also messes up with 'JJ' and 'NNP' also because some adjectives are nouns, and some nouns can be used as an adjective. Considering 'VBP' as 'NN' is because 'VBZ' is the original form of verbs, and some of them can also be used as singular nouns. Similarly, for 'VBZ' and 'NNS,' 3rd person singular present verbs sometimes have the same spelling as plural nouns.

Error Analysis:

('NN', 'JJ', 311,

['slick-talking', 'snake-oil', 'gullible', 'Callable', 'ever-narrowing', 'literary', 'bargain-basement', 'net', 'capitalist', 'Jovian', 'solar', 'wholesale', 'Plump', 'rough', 'Past', 'same-store', 'preliminary', 'Structural', 'impending', '44-cent-a-barrel', 'second-quarter', 'Lucullan', 'undemocratic', 'historical', '20th', 'favorable', 'two-part', 'reluctant', '52-week', 'secret', 'wholesale', 'inter-city', 'minimal', 'ultimate', 'executive', 'hourly', 'wrong', 'bold', 'Gargantuan', 'Victorian', 'wholesale', 'Money-fund', 'auxiliary', 'puzzling', 'indirect', 'resettable', '20-year', 'alma', 'nonresident', 'selective', 'rapid', 'removable', 'mid-afternoon', 'influential', 'bearish', 'laden', 'home-building', 'multibillion-dollar', 'eerie', 'nuclear-power', 'full-power', 'so-called', 'corrosion-resistant', 'nonstrategic', 'optimum', 'unsuspected', 'electrical', 'Soviet-style', 'Argentine', 'dilutive', '30th', 'single-malt', 'full-time', 'executive', 'lively', 'lucrative', 'ballistic', 'antithetical', 'indomitable', 'nonperforming', 'CORPORATE', 'technological', 'untold', 'town-house', 'then-pending', 'tidal', '20-year', 'Political', 'hard-hit', 'tire-patching', 'Negotiable', 'implausible', 'on-site', 'non-telephone', 'highly-confident', 'median', 'Dutch', 'Armenian', 'leveraged', 'fetal', 'post-quake', 'stylistic', 'perfect', 'marginal', 'horrible', 'undemocratic', 'official', 'pre-reform', 'payable', 'sour', 'lucrative', 'five-and-dime', 'sore', 'image-building', 'forthcoming', 'angry', 'troubling', 'two-year', 'executive', 'deep', '40-point', 'nonperforming', 'superficial', 'rational', 'quake-prone', 'LEBANESE', 'five-hour', 'Health-care', 'Garpian', 'foregone', 'complex', 'payable', 'RTC-appointed', 'permanent', 'touchy', 'passive', 'perfect', 'dizzying', 'elusive', 'red', 'modern', 'newspaper-industry', 'contemporary', 'dangerous', 'peculiar', 'severe', 'interim', 'preliminary', 'executive', 'official', 'alert', 'Thermal', 'unregistered', 'time-share', 'net', 'unmet', 'dry', 'alert', 'leveraged', 'four-year', 'unprepared', 'four-year', 'Long', 'official', 'assorted', 'pork-barrel', 'rapid', 'unstylish', 'plentiful', 'dizzying', 'bond-trading', 'disproportionate', 'majority-party', 'orthodox', 'pork-barrel', 'powdered', 'net', 'savvy', 'freemarket', 'Silver', 'secret', 'valid', 'inferior', 'total', 'hard-hit', 'bruising', 'official', 'primary', 'Chinese', 'radiophonic', 'hourly', 'title-insurance', 'outright', 'foreign-exchange', 'legendary', 'NEW', 'theatrical', 'wide', 'executive', 'five-year', 'fetal-tissue', 'utilitarian', 'daunting', 'optional', 'precise', 'executive', 'executive', 'principal', 'western-style', 'two-step', '25-cent-a-share', 'official', 'historical', 'historical', 'unsettling', 'satirical', 'wireline', 'cellular', 'western', 'overhead', 'Anti-nuclear', 'contiguous', 'alternate', 'official', 'primary', 'wide', 'Negotiable', 'Crude', 'median', 'theatrical', 'preliminary', 'decent', 'interesting', 'compulsive', 'primary', 'stock-quote', 'similiar', 'non-dischargable', 'secret', 'Junior', 'U.S.-built', 'verbal', 'collective', 'statewide', 'bleak', 'Arkansas-based', 'official', 'multi-family', 'raccoon-skin', 'short-changing', 'startling', 'elective', 'incompetent', 'burlesque', 'white-walled', 'abnormal', 'wonderful', 'compulsive', 'tame', '52-week', 'hopeful', 'median', 'primordial', 'stock-for-debt', 'jubilant', 'flip-flopped', 'worth', 'high-tech', 'communist', 'distinct', '20-year', 'right', 'same-store', 'peculiar', 'organizational', 'multimillion-dollar', 'U.S.-backed', 'vast', 'stress-provoking', 'primary', 'unsettling', 'Personal', 'creative', 'Long-term', 'five-cent', 'outdoor', 'many', 'wrong', 'highest-volume', 'wrong', 'reluctant', 'quiescent', 'gawky', 'ebullient', 'all-day', 'unsuccessful', 'media-buying', 'opposite', 'first-hand', 'executive', 'precious', 'a.k.a', 'idle', 'plentiful', 'drab', 'official', 'wholesale', 'net', 'warm-weather', 'second-story', 'Chinese'])

('VBN', 'VBD', 248,
['held', 'made', 'permitted', 'completed', 'Warned', 'fled', 'topped', 'featured', 'offered', 'sold', 'stopped', 'based', 'passed', 'pushed', 'expected', 'heaved', 'believed', 'reached', 'conspired', 'involved', 'called', 'played', 'set', 'gathered', 'bounced', 'tried', 'expected', 'introduced', 'started', 'remanded', 'sweetened', 'faced', 'settled', 'set', 'perceived', 'provided', 'lost', 'raised', 'developed', 'predicted', 'predicted', 'stayed', 'discovered', 'ignored', 'headed', 'moved', 'reduced', 'called', 'ended', 'acquired', 'made', 'issued', 'played', 'missed', 'teamed', 'produced', 'received', 'slipped', 'used', 'estimated', 'proposed', 'brought', 'glanced', 'smiled', 'interviewed', 'argued', 'pleaded', 'reminded', 'set', 'suspended', 'sustained', 'estimated', 'charged', 'seemed', 'set', 'spurned', 'divested', 'filled', 'bolstered', 'raided', 'forced', 'reacted', 'believed', 'carried', 'passed', 'called', 'made', 'increased', 'slipped', 'failed', 'lost', 'lost', 'served', 'settled', 'lost', 'heard', 'authorized', 'financed', 'revised', 'reached', 'requested', 'created', 'offered', 'annoyed', 'favored', 'edged', 'predicted', 'displayed', 'disclosed', 'materialized', 'rambled', 'outnumbered', 'commented', 'offered', 'transmogrified', 'stopped', 'changed', 'lost', 'occupied', 'proved', 'designed', 'gathered', 'warned', 'crowded', 'retreated', 'flew', 'borrowed', 'headed', 'sold', 'spotted', 'placed', 'alerted', 'caused', 'sent', 'offered', 'beefed', 'offered', 'completed', 'waited', 'uncovered', 'fined', 'caught', 'multipled', 'joined', 'represented', 'survived', 'failed', 'resumed', 'reached', 'seemed', 'slumped', 'changed', 'renewed', 'died', 'conspired', 'heard', 'ended', 'estimated', 'restored', 'reflected', 'tried', 'staged', 'introduced', 'boosted', 'killed', 'subordinated', 'dumped', 'made', 'increased', 'aided', 'schmumpered', 'worked', 'filed', 'eluded', 'dumped', 'dispatched', 'aimed', 'proved', 'released', 'pleaded', 'established', 'tried', 'completed', 'compared', 'opposed', 'referred', 'undervalued', 'scammed', 'lacked', 'reached', 'caused', 'narrowed', 'made', 'slipped', 'surveyed', 'topped', 'authorized', 'died', 'left', 'outnumbered', 'sold', 'highlighted', 'feared', 'topped', 'completed', 'firmed', 'moved', 'emerged', 'suspended', 'pictured', 'led', 'joined', 'suspended', 'slipped', 'girded', 'planned', 'discovered', 'completed', 'contested', 'slipped', 'tacked', 'signed', 'pulled', 'stayed', 'reflected', 'measured', 'qualified', 'damaged', 'disclosed', 'eased', 'ended', 'lagged', 'called', 'reflected', 'consisted', 'realized', 'led', 'abounded', 'reached', 'provided', 'owned', 'teemed', 'triggered', 'retained', 'warned', 'revealed', 'experienced', 'protected'])

('NN', 'NNP', 226,
['Bateman', 'Bryan', 'Egon', 'Harrison', 'IRA', 'Wheeling-Pittsburgh', 'Hot', 'SIA', 'Financiere', 'Madison', 'Consumer', 'Greenberg', 'Petrochemical', 'Ex-Im', 'Presidents', 'Kennedy', 'Nixon', 'Supplemental', 'SSI', 'Pizza', 'Hut', 'Corporation', 'Herman', 'Torstar', 'Glaser', 'Enforcement', 'Angrist', 'Examiner', 'Smalling', 'Elders', 'Caddyshack', 'Kramer', 'Holliston', 'Unincorporated', 'Seagram', 'Stan', 'Cohen', 'Crandall', 'Rhona', 'Laboratory', 'EMPIRE', 'Yardeni', 'Metatrace', 'Assistant', 'Revenue', 'Women', 'Buksbaum', 'Sheldon', 'BankWatch', 'Neil', 'Polygram', 'Deep', 'Midnight', 'Windflower', 'Foreign', 'Shelton', 'Rahill', 'Olshan', 'Majority', 'Parretti', 'Luerssen', 'Pollin', 'Riverside', 'Powder', 'Stuart', 'Doosan', 'Denise', 'Marketing', 'Kaolin', 'Unimin', 'Pride', 'Parkshore', 'Tower', 'Schrager', 'Prague',

'Donahue', 'Sleeping', 'Adam', 'Wolfe', 'Television', 'Rafael', 'Location', 'Perth', 'Rafael', 'Stuart', 'Brown-Forman', 'Erie', 'Sol', 'Construction', 'Crane', 'Charter', 'Livermore', 'Laboratory', 'Financing', 'Durney', 'Petersburg', 'Professional', 'Isuzu', 'Yoshiaki', 'NEW', 'CARE', 'Greece', 'Coach', 'Madson', 'Royce', 'Brent', 'Galveston-Houston', 'Shattuck', 'Oakland-Berkeley', 'Becker', 'Spenser', 'Fault', 'Toensing', 'Deal', 'Gary', 'Kansas', 'Slater', 'Streetspeak', 'Hawley', 'Hawley', 'Hale', 'Wendy', 'Roaring', 'Ameron', 'Kennedy', 'Traverse', 'Capcom', 'PANDA', 'Alliance', 'Kajima', 'Jiotto', 'Autodesk', 'Oracle', 'Anaheim-Santa', 'Ana', 'Cheney', 'Carnegie', 'Whittle', 'Sirrine', 'Engineering', 'Greiner', 'Engineering', 'Engineering', 'Mississippian', 'Mississippi', 'Advisor', 'Wayne', 'Eagleton-Newark', 'Cohen', 'Liability', 'Helliesen', 'Voice', 'Carrion', 'Banco', 'Storer', 'Kroger', 'Sailing', 'Thunderbird', 'Beat', 'Streisand', 'Marvin', 'Nielsen', 'Marketing', 'Nielsen', 'Marketing', 'Diamond-Star', 'Abortion', 'KPMG', 'Marwick', 'Ripper', 'Lawrence', 'Kristol', 'Gilder', 'Emshwiller', 'Alliance', 'Advertising', 'Advertising', 'Advertising', 'Banco', 'Consulting', 'Diamond-Star', 'Contract', 'Philips', 'Marion', '20th', 'Strip', 'Book-of-the-Month', 'Core', 'Walt', 'Anaheim', 'Salerno', 'Capcom', 'Wendy', 'Silver', 'Platt', 'Accounting', 'Jacki', 'Ragan', 'Adamski', 'Lesley', 'Edgar', 'Mississippi', 'Viktor', 'Sidorenko', 'Kursk', 'LSX', 'Howard', 'Crandall', 'Soup', 'Shevardnadze', 'Hoffman', 'Brewing', 'Merill', 'PR', 'Midway', 'Bessemer', 'Trading', 'Hibler', 'Arafat', 'Hiroyuki', 'Ginn', 'Cadillac', 'Janesville', 'Cavalier', 'Duy', 'Carrion'])

('JJ', 'NN', 187,
['commercial', 'depository', 'medicine', 'humor', 'net', 'many', 'shape', 'weight', 'net', 'withdrawal', 'weight', 'crude', 'Bond', 'much', 'leniency', 'few', 'much', 'early-retirement', 'commercial', 'governorship', 'jetliner', 'pizza', 'past', 'physical', '5', 'much', 'sweetheart', 'commercial', 'context', 'last', 'conscript', 'malaria', 'sidewalk', 'current', 'theme', 'past', 'lounge', 'replay', 'pride', 'know-how', 'pop', 'other', 'many', 'slogan', 'sidewalk', 'calendar', 'motorbike', 'creamer', 'mail', 'net', 'total', 'public', 'hassle', 'pride', 'lifestyle', 'portrait', 'mail', 'debacle', 'halt', 'soft-drink', 'mail', 'mail', 'quarterly', 'fun', 'multiparty', 'humor', 'ball', 'past', 'aspect', 'many', 'quest', 'many', 'fine', 'mail', 'octane', 'pride', 'Mail-order', 'facade', 'convention', 'ideologist', 'falloff', 'crude', 'Bond', 'sun', 'rendition', 'Candy', 'marine', 'many', 'newsman', 'ice', 'disturbance', 'ball', 'much', 'ideology', 'dark', 'general', 'resiliency', 'Bond', 'Ad', 'many', 'concrete', 'mound', 'much', 'disposal', 'withdrawal', 'flashlight', 'many', 'winner', 'disqualification', 'drink', 'reliance', 'enterprise', 'net', 'general', 'photo', 'top', 'much', 'other', 'detective', 'picture', 'insider', 'behest', 'misrepresentation', 'round', 'tailspin', 'BEER', 'convention', 'brewery', 'intelligence', 'jetliner', 'briefcase', 'past', 'net', 'merchant', 'top', 'psychology', 'actor', 'overflow', 'contest', 'theme', 'hose', 'notch', 'tip', 'built-in', 'dependence', 'sheep', 'preadmission', 'twist', 'alternative', 'aspect', '5', 'friend', 'psychology', 'theme', 'round', 'boutique', 'scene', 'average', 'past', 'logic', 'net', 'crude', 'chief', 'general', 'revolutionary', 'cogeneration', 'Japanese', 'cogeneration', 'REPLICATION', 'importer', 'ballplayer', 'plume', 'total', 'public', 'German', 'dial-tone', 'shape', 'psychology', 'mail', 'aspect', 'withdrawal', 'constituency', 'average', 'Net', 'productivity', 'motel', 'poet'])

('NNP', 'NN', 181,
['yacht', 'agility', 'Market', 'reflection', 'tandem', 're-election', 'treasury', 'Soybean', 'evaluation', 'Computer', 'Energy', 'B.A.T', 'defamation', 'tolerance', 'steakhouse', 'sigh', 'Homerun', 'vinyl', 'chloride', 'department-store', 'hamburger', 'cover', 'prospectus', 'receivables', 'perjury', 'wool', 'intensity', 'subcompact', 'anthem', 'pretext', 'dictatorship', 'malnourishment', 'rhythm', 'drill', 'divestiture', 'nonpriority', 'surface', 'E-mail', 'flotilla', 'DEBT', 'Commonwealth', 'declaration', 'Insurance', 'temblor', 'mania', 'quiz', '5', 'dignity', 'rigor', 'anybody', 'raiser', 'tie', 'Newsprint', 'hawk', 'stand', 'drug-industry', 'State', 'inability', 'furrier', 'bullion', '5', 'smell', 'Transportation', 'loft', 'PLASTIC', 'impetus', 'shrinkage', 'foundation', 'stress', 'Trim', 'diet', 'controller', 'ft.', 'pier', 'Transport', 'upsurge', 'cocaine', 'zip', '5', 'temblor', 'duty', 'seniority', 'honor', 'Ski', 'guy', 'watch', 'cotton', 'allure', 'extermination', 'daybreak', 'sponsorship', 'edge', 'Mortgage', 'trip', 'performer', 'brush', 'tornado', 'medicine', 'cartridge', 'souvenir', 'dissemination', 'scrutiny',

'pence', '5', 'pence', 'Someone', 'outcry', 'stockholder', 'notice', 'strongman', '5', 'condominium', 'divestiture', 'Government', 'drumroll', 'Basketball', 'singer', 'ploy', 'feel', 'Defense', 'duty', 'loan-loss', 'BEAT', 'Panic', '5', 'souvenir', 'scorecard', 'B.A.T', 'pence', 'pence', 'defection', 'libel', 'murderer', 'Term', 'State', 'residence', 'mania', 'TIP', 'DISCOUNT', 'blanket', 'City', '5', 'glass', 'reconstruction', 'eel', 'skin', 'Administration', 'roadblock', 'fleet', 'Journal', 'verdict', 'embroidery', 'caseload', 'uniform', 'guerrilla', '5', 'stand', 'telex', 'bullion', 'drug-policy', 'rash', 'MANAGER', 'break-up', 'palm', 'ozone', 'rub', 'temblor', 'Mortgage', 'snowsuit', 'Life', 'zenith', 'vault', 'and\/or', 'uproar', 'salespeople', 'Nightlife', 'trip', 'village', 'money-transfer', 'Oil', 'peso'])

('JJ', 'NNP', 163,
['League', 'mature', 'British', 'Ownership', 'Beau', 'Manion', 'Commerciale', 'Maumee', 'Roper', 'Bradford', 'Olympia', 'Magazine', 'Connaught', 'Reebok', 'Last', 'Small', 'Vic', 'Caldwell', 'Legal', 'Tire', 'SciMed', '30-year', 'Texan', 'Nader', 'Connaught', 'Nghe', 'LME', 'Thai', 'Nuclear', 'Joan', 'Lawrence', 'Empire-Berol', 'mature', 'Laphroaig', 'Christie', 'Crutcher', 'Industrials', 'Dallas', 'Trident', 'Procter', 'Noxell', 'Noxell', 'Hawaiian', 'Tower', 'Roebuck', 'Karen', 'Giancarlo', 'Corazon', 'Ferdinand', 'Radical', 'Nestle', 'British', 'HRH', 'Rune', 'Alcee', 'British', 'Hajak', 'Phil', 'Little', 'Enthusiast', 'Elrick', 'Lavidge', 'Houston-based', 'Willamette', 'Ukraine', 'Chernobyl', 'Food', 'Amityville', 'Shealy', 'Commerciale', 'Kaitaia', 'Reebok', 'Springfield', 'McClelland', 'Halloween', 'Oxford', 'Lawrence', 'Omaha', 'Avdel', 'Avdel', 'Kuala', 'Lumpur', 'Kurt', 'ENGLAND', 'Denmark', 'Solow', 'Market-If-Touched', 'Susan', 'Rogin', 'Alcatraz', 'Roebuck', 'Morton', 'Story', 'Hyman', 'Foster', 'Subcommittee', 'Cynthia', 'Turk', 'Ashton-Tate', 'Westin', 'Assurance', 'Beau', 'Science', '79-year-old', 'Pinpoint', 'Embarcadero', 'Municipal', 'Baby', 'British', 'Playback', 'Tort', 'Jordan', 'Simat', 'Mountain', 'VOA', 'Cardinal', 'Vittoria', 'Pauline', 'Laurance', 'Eclipse', 'Stanwick', 'Base', 'Peat', 'Fred', 'Indiana', 'Olsen', 'Boddington', 'Vice', 'Connaught', 'Eclipse', 'Indochina', 'Commerciale', 'Philippe', 'Story', 'Bince', 'Mirage', 'Bonanza', 'Metromedia', 'Clanahan', 'Jacqueline', 'Electronic', 'Kobe', 'German', 'Quotron', 'Investigation', 'Manion', 'STREET', 'Reebok', 'Nuclear', 'Bailit', 'Michele', 'Miller', 'Motorola', 'FM', 'Virgin', 'Legal', 'Gregory', 'Nikko', 'Pinpoint', 'British', 'Vauxhill', 'Nghe', 'Rafael'])

('NNP', 'JJ', 154,
['Initial', 'uncomfortable', 'fixed-rate', 'automatic', 'California', 'snooty', 'Canadian', 'South', 'African', 'consumer-advocacy', 'Colombian', 'veto-proof', '17-member', 'consumer-price', 'American', 'top-level', 'American', 'British', 'federal-local', 'cash-hungry', 'dismal', 'Corp.-Toyota', 'preferred-stock', 'costly', 'First', 'extreme', 'Western', 'Western', 'Soviet', 'agrarian-reform', 'FEDERAL', 'nightly', 'prickly', 'heady', 'tax-and-budget', 'Afrikaner', 'HEAVY', 'Hispanic', 'Hispanic', 'third-largest', 'Canadian', 'Big', 'commemorative', 'unfair', '500-stock', 'deleterious', 'California', 'American', 'Italian', 'fixed-rate', 'overseas', 'Left-stream', 'fragile', 'scary', 'neat', 'gas-station', 'Blue', 'costly', 'Swedish', 'long-range', 'British', 'beholden', 'costly', 'low-budget', 'generic', 'Chemical', 'robust', 'British', 'competent', 'double-deck', 'Longtime', 'American', 'irrational', 'Foreign', 'American', 'mandatory', 'U.K.', 'costly', 'upward', 'Primary', 'pink', 'British', '50-story', 'inter-company', 'yearly', 'human-rights', 'multi-agency', 'psychological', 'MUTUAL', 'unlawful', 'Baltic', 'scientific', 'Electrical', 'Pretax', 'metropolitan', 'South', 'year-ago', 'American', 'Hispanic', 'American', 'self-regulatory', 'overseas', 'yearly', 'New', 'New', 'seductive', 'Year-earlier', 'choppy', 'costly', 'made-for-TV', 'mandatory', 'advisory', 'scientific', 'OK', 'Honduran', 'Canadian', 'irrational', 'South', 'Korean', 'non-financial', 'Early', 'class-action', 'prickly', 'British', 'white-spirits', 'American', 'shrewd', 'Short-term', 'sober', 'recession-wary', 'Western', 'Year-earlier', 'British', 'Western', 'socalled', 'modern', 'definite', 'lift-ticket', 'conscientious', 'East', 'quick-fix', 'Swiss', 'year-ago', 'now-standard', 'lifelong', 'emergency-medical', 'residual', 'synthetic-leather', 'Canadian', 'British', 'West', 'disadvantaged', 'two-day', 'human'])

('NNS', 'VBZ', 126,
['HAS', 'requires', 'assists', 'targets', 'sells', 'follows', 'shows', 'serves', 'weights', 'exercises', 'handles', 'closes', 'speaks', 'focuses', 'receives', 'estimates', 'rises', 'warns', 'diminishes', 'fits', 'advises', 'retires', 'develops', 'receives', 'charges', 'helps', 'regrets', 'declines', 'plans', 'buys', 'shows', 'consoles', 'agrees', 'claims', 'results', 'instructs', 'enjoys', 'spawns', 'reduces', 'reduces', 'tires', 'keeps', 'prices', 'wraps', 'reads', 'sells', 'cites', 'fancies', 'lives', 'fits', 'leads', 'hails', 'plans', 'implies', 'Says', 'points', 'creates', 'manufactures', 'conducts', 'plans', 'sells', 'increases', 'hangs', 'removes', 'fits', 'tracks', 'sells', 'threatens', 'heads', 'claims', 'alarms', 'admits', 'collapses', 'lists', 'suffers', 'hours', 'values', 'follows', 'figures', 'arrives', 'reduces', 'funds', 'lags', 'estimates', 'follows', 'results', 'grovels', 'concedes', 'plans', 'reaches', 'falls', 'follows', 'happens', 'plows', 'sows', 'feeds', 'penalizes', 'rewards', 'divides', 'shares', 'isolates', 'chides', 'concedes', 'marks', 'produces', 'Says', 'denies', 'complements', 'uses', 'regrets', 'concedes', 'Says', 'hours', 'uses', 'carries', 'plows', 'creates', 'kills', 'buys', 'imposes', 'confers', 'roars', 'produces', 'relies', 'uses', 'shows'])

('VBD', 'VBN', 101,
['ended', 'continued', 'brought', 'formed', 'formed', 'approved', 'got', 'added', 'sued', 'kept', 'finished', 'closed', 'accused', 'voiced', 'ended', 'rebounded', 'agreed', 'sent', 'failed', 'announced', 'earned', 'attributed', 'released', 'contributed', 'sentenced', 'accepted', 'undermined', 'entrenched', 'agreed', 'closed', 'recommended', 'agreed', 'received', 'declined', 'closed', 'had', 'produced', 'declined', 'made', 'ended', 'advanced', 'excited', 'had', 'decided', 'sidelined', 'approved', 'had', 'approved', 'announced', 'made', 'continued', 'felt', 'closed', 'had', 'collapsed', 'received', 'found', 'prolonged', 'resigned', 'reported', 'accepted', 'accepted', 'made', 'faded', 'attributed', 'entrenched', 'promised', 'lowered', 'turned', 'agreed', 'ended', 'reported', 'suffered', 'prompted', 'raised', 'expressed', 'went', 'extended', 'announced', 'advanced', 'failed', 'marked', 'included', 'turned', 'ordered', 'reported', 'polled', 'decided', 'collapsed', 'failed', 'accused', 'hit', 'agreed', 'ended', 'concluded', 'followed', 'accused', 'announced', 'recommended', 'lost', 'turned'])

('NN', 'VBP', 82,
['use', 'vanish', 'pay', 'note', 'tax', 'plant', 'point', 'look', 'engage', 'show', 'cut', 'blame', 'court', 'agree', 'point', 'underscore', 'close', 'continue', 'run', 'range', 'simplify', 'cut', 'point', 'report', 'decline', 'display', 'concern', 'buy', 'cost', 'tax', 'concern', 'speak', 'show', 'mark', 'show', 'decline', 'court', 'qualify', 'face', 'march', 'put', 'work', 'credit', 'eat', 'run', 'call', 'agree', 'point', 'crack', 'refer', 'read', 'vanish', 'stay', 'contend', 'play', 'trust', 'favor', 'forecast', 'work', 'lie', 'pay', 'range', 'cut', 'care', 'stem', 'concern', 'pitch', 'estimate', 'range', 'block', 'cost', 'Do', 'plan', 'expose', 'buy', 'lap', 'fear', 'rest', 'concentrate', 'cost', 'afford', 'favor'])

Part 4: Submit Your Homework

This is the end. Congratulations!

Now, follow the steps below to submit your homework in [Gradescope]():

1. Rename this ipynb file to 'CS4650_p2_GTusername.ipynb'. We recommend ensuring you have removed any extraneous cells & print statements, clearing all outputs, and using the Runtime --> Run all tool to make sure all output is update to date. Additionally, leaving comments in your code to help us understand your operations will assist the teaching staff in grading. It is not a requirement, but is recommended.
2. Click on the menu 'File' --> 'Download' --> 'Download .py'.
3. Click on the menu 'File' --> 'Download' --> 'Download .ipynb'.

4. Download the notebook as a .pdf document. Make sure the output from your training loops are captured so we can see how the loss and accuracy changes while training.
5. Upload all 3 files to GradeScope.

1 秒 完成时间：20:35