Deep Neural Networks for Natural Language Processing (AI6127)

JUNG-JAE KIM

TUTORIAL 2: WORD VECTORS

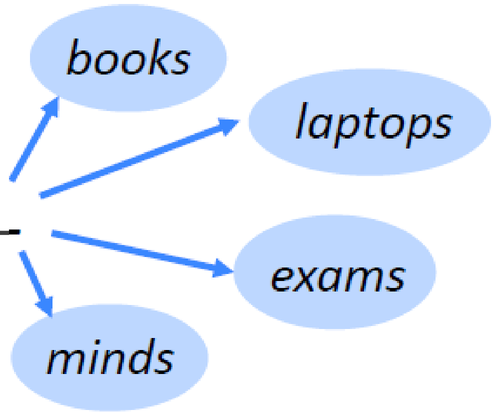
Question 1: Read and understand the example implementation of n-gram language modeling

- https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

Language Modeling

- Language Modeling is the task of predicting what word comes next.

E.g. the students opened their ______



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$p\left(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \cdots, \boldsymbol{x}^{(1)}\right)$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a Language Model.

Week 3 slide

n-gram Language Models

the students opened their ______

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!
- Definition: An *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”
- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

Question 2: Give answer codes for the exercise “Computing Word Embeddings: Continuous Bag-of-Words”

- Based on the example implementation of n-gram language modeling
- Filling up the __init__ and forward functions
- Giving codes for training and for displaying word vectors of selected words

Question 3 - Run the notebook “Gensim word vector visualization of various word vectors”

- http://web.stanford.edu/class/cs224n/materials/Gensim.zip
- Change glove file path as follows:

```
!wget http://nlp.stanford.edu/data/glove.6B.zip

!unzip glove.6B.zip

glove_file = datapath(‘/content/glove.6B.100d.txt')
```

Hands-on

Answer 1

```
CONTEXT_SIZE = 2
EMBEDDING_DIM = 10
test_sentence = "When forty winters shall besiege thy brow, …".split()
# build a list of tuples.  Each tuple is ([ word_i-2, word_i-1 ], target word)
trigrams = [([test_sentence[i], test_sentence[i + 1]], test_sentence[i + 2])
            for i in range(len(test_sentence) - 2)]
print(trigrams[:3]) # print the first 3, just so you can see what they look like

vocab = set(test_sentence)
word_to_ix = {word: i for i, word in enumerate(vocab)}
```

Answer 1 - model

```
class NGramLanguageModeler(nn.Module):
    def __init__(self, vocab_size, embedding_dim, context_size):
        super(NGramLanguageModeler, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear1 = nn.Linear(context_size * embedding_dim, 128)
        self.linear2 = nn.Linear(128, vocab_size)

    def forward(self, inputs):
        embeds = self.embeddings(inputs).view((1, -1))
        out = F.relu(self.linear1(embeds))
        out = self.linear2(out)
        log_probs = F.log_softmax(out, dim=1) # next slide
        return log_probs
```

Week 2 slide

Training with softmax and cross-entropy loss

- For each training example (x,y), our objective is to maximize the probability of the correct class y
- This is equivalent to minimizing the negative log probability of that class:

$$-\log p(y|x) = -\log\left(\frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)}\right)$$

- Using log probability converts our objective function to sums, which is easier to work with on paper and in implementation

Answer 1 – loss function / optimizer

losses = []

loss_function = nn.NLLLoss() # negative log likelihood loss

model = NGramLanguageModeler(len(vocab), EMBEDDING_DIM, CONTEXT_SIZE)

optimizer = optim.SGD(model.parameters(), lr=0.001) # stochastic gradient descent

Answer 1 – training

```
for epoch in range(10):
    total_loss = 0
    for context, target in trigrams:
        # Step 1. Prepare inputs to be passed to model (i.e, turn words into indices and wrap in tensors)
        context_idxs = torch.tensor([word_to_ix[w] for w in context], dtype=torch.long)

        # Step 2. Before passing in a new instance, zero out the gradients from the old instance
        model.zero_grad()

        # Step 3. Run the forward pass, getting log probabilities over next words
        log_probs = model(context_idxs)

        # Step 4. Compute your loss function
        loss = loss_function(log_probs, torch.tensor([word_to_ix[target]], dtype=torch.long))

        # Step 5. Do the backward pass and update the gradient
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    losses.append(total_loss)
print(losses)  # The loss decreased every iteration over the training data!
```

Answer 2 – data pre-processing

```
vocab_size = len(vocab)
word_to_ix = {word:ix for ix, word in enumerate(vocab)}
ix_to_word = {ix:word for ix, word in enumerate(vocab)}
data = []
for i in range(2, len(raw_text) - 2):
    context = [raw_text[i - 2], raw_text[i - 1],
               raw_text[i + 1], raw_text[i + 2]]
    target = raw_text[i]
    data.append((context, target))
```

https://github.com/FraLotito/pytorch-continuous-bag-of-words

Answer 2 – model

```
class CBOW(torch.nn.Module):
    def __init__(self, vocab_size, embedding_dim):
        super(CBOW, self).__init__()

        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear1 = nn.Linear(embedding_dim, 128)
        self.activation_function1 = nn.ReLU()

        self.linear2 = nn.Linear(128, vocab_size)
        self.activation_function2 = nn.LogSoftmax(dim = -1)
```

```
def forward(self, inputs):
    embeds = sum(self.embeddings(inputs)).view(1,-1)
    out = self.linear1(embeds)
    out = self.activation_function1(out)
    out = self.linear2(out)
    out = self.activation_function2(out)
    return out

def get_word_emdedding(self, word):
    word = torch.tensor([word_to_ix[word]])
    return self.embeddings(word).view(1,-1)
```

https://github.com/FraLotito/pytorch-continuous-bag-of-words

Answer 2 – loss function / optimizer

```
model = CBOW(vocab_size, EMDEDDING_DIM)


loss_function = nn.NLLLoss()

optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
```

Answer 2 – training

```
for epoch in range(50):
    total_loss = 0
    for context, target in data:
        context_vector = make_context_vector(context, word_to_ix)

        log_probs = model(context_vector)
        total_loss += loss_function(log_probs, torch.tensor([word_to_ix[target]]))

    #optimize at the end of each epoch
    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()
```

Answer 2 – testing

```
context = ['People','create','to', 'direct']

context_vector = make_context_vector(context, word_to_ix)

a = model(context_vector)


print(f'Raw text: {" ".join(raw_text)}\n')

print(f'Context: {context}\n')

print(f'Prediction: {ix_to_word[torch.argmax(a[0]).item()]}')
```

Answer 3 – Download and load GloVe word embeddings

!wget http://nlp.stanford.edu/data/glove.6B.zip

!unzip glove.6B.zip

glove_file = datapath('glove.6B.100d.txt')

word2vec_glove_file = get_tmpfile("glove.6B.100d.word2vec.txt")

glove2word2vec(glove_file, word2vec_glove_file) # convert glove format to word2vec format

model = KeyedVectors.load_word2vec_format(word2vec_glove_file)

Answer 3 – Find most similar words

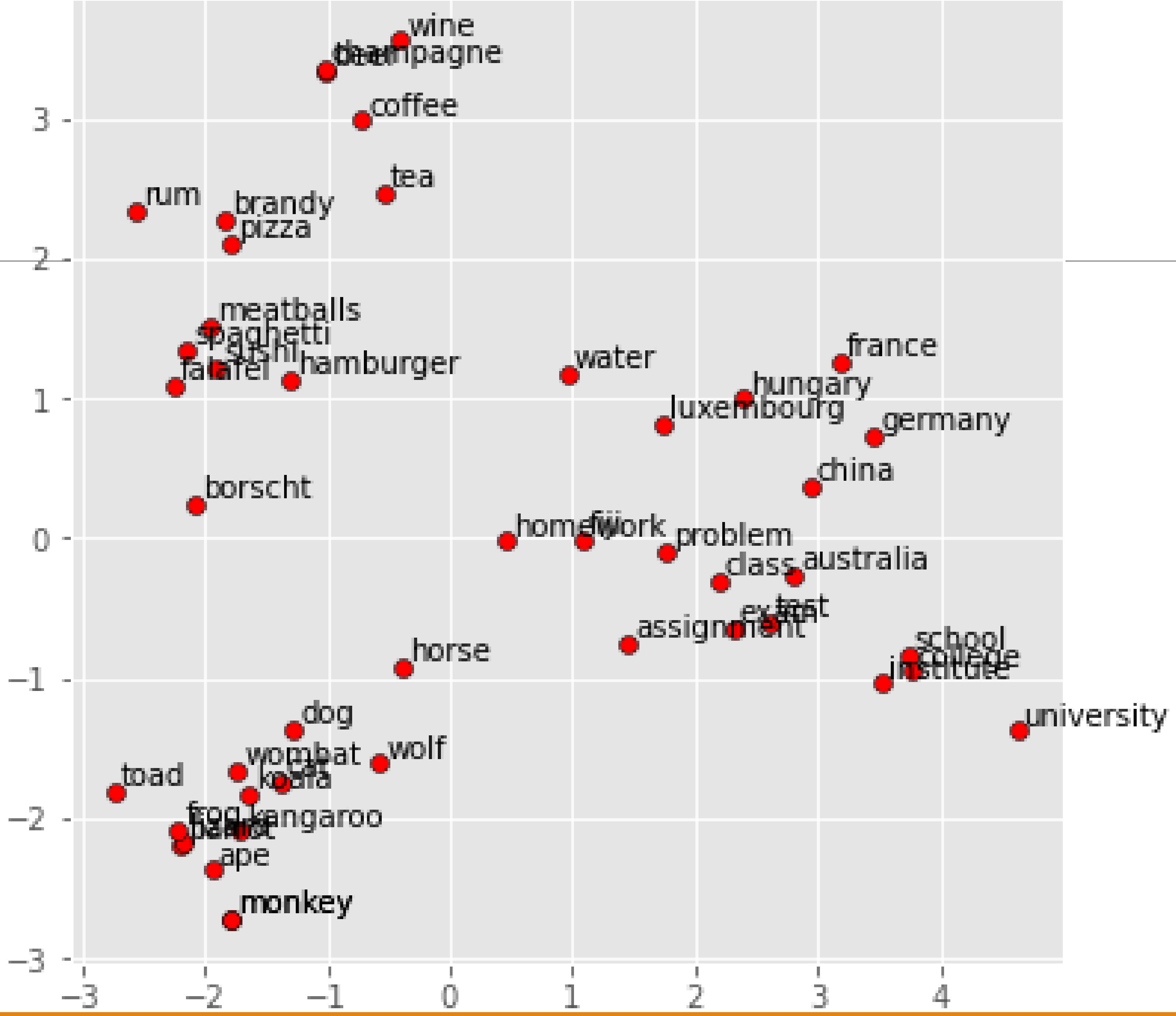
- model.most_similar('obama')
- model.most_similar('banana')
- model.most_similar(negative='banana')
  - Find the words most dissimilar to ‘banana’

Answer 3 - Analogy

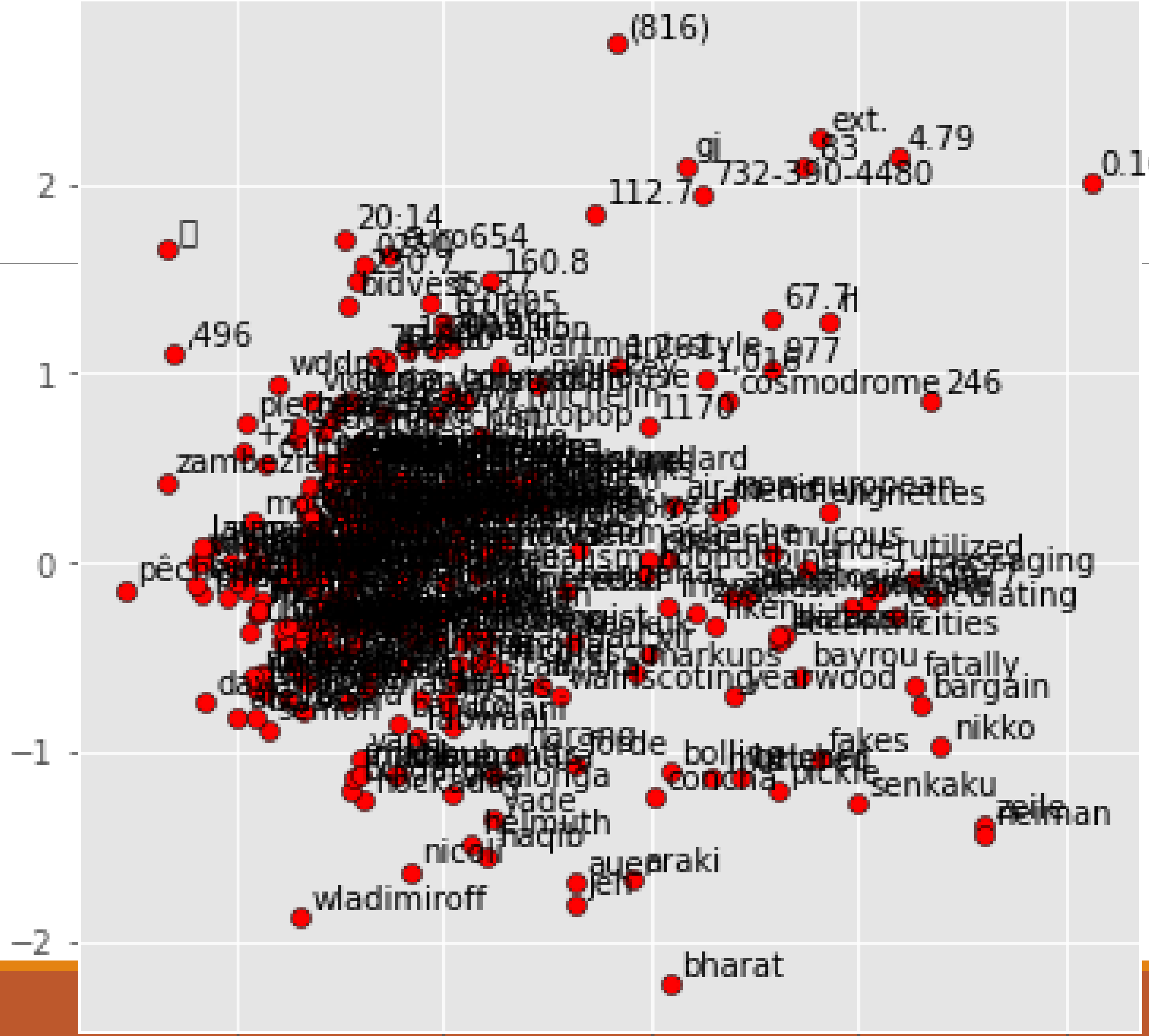
```
result = model.most_similar(positive=['woman', 'king'], negative=['man'])
print("{}: {:.4f}".format(*result[0]))

def analogy(x1, x2, y1):
    result = model.most_similar(positive=[y1, x2], negative=[x1])
    return result[0][0]

analogy('japan', 'japanese', 'australia')
analogy('australia', 'beer', 'france')
print(model.doesnt_match("breakfast cereal dinner lunch".split()))
```

Answer 3 – display_pca_scatterplot

```
def display_pca_scatterplot(model, words=None, sample=0):
    if words == None:
        if sample > 0: words = np.random.choice(list(model.vocab.keys()), sample)
        else: words = [ word for word in model.vocab ]
    word_vectors = np.array([model[w] for w in words])
    # principal component analysis: linear dimensionality reduction using singular value decomposition
    twodim = PCA().fit_transform(word_vectors)[:,:2]
    plt.figure(figsize=(6,6))
    plt.scatter(twodim[:,0], twodim[:,1], edgecolors='k', c='r')
    for word, (x,y) in zip(words, twodim): plt.text(x+0.05, y+0.05, word)
```



wine
coffee
tea
rum
brandy
pizza
meatballs
spaghetti
hamburger
water
france
hungary
luxembourg
germany
china
borscht
problem
class
australia
assignment
horse
school
university
dog
wombat
wolf
toad
kangaroo
ape
monkey
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4


(816)
ext.
4.79
0.16
gi
732-390-4480
112.7
20:14
160.8
.496
67.7
1,018
877
cosmodrome
246
1170
zambezia
air-menu
underutilized
mucous
circulating
eccentricities
markups
wainscoting
bayrou
fatally
bargain
nikko
fakes
senkaku
helmuth
naqib
nicol
araki
wladimiroff
bharat
2
1
0
−1
−2