Scalable Data Engineering, Exercise 9

Henry Haustein

Task 1

(a) Python

```python
print(torch.sum(x, 0))
print(torch.sum(x, 1))
print(torch.sum(x, 2))
```

(b) Python

```python
ones = torch.ones([1,2,4,5])
random = torch.randn(10,4)
random + ones.reshape([10, 4])
```

Task 2

Right answers are:

- Matrix Multiplication Backwards
- Backward Step of the linear layer

Task 3

Right answer is [0.27, 0.73]

Task 4

(a) Python

```python
class NGramLanguageModeler(nn.Module):

  def __init__(self, vocab_size, embedding_dim, context_size,
      hidden_size):
    super(NGramLanguageModeler, self).__init__()
    self.embeddings = nn.Embedding(vocab_size, embedding_dim)
    self.linear1 = nn.Linear(context_size * embedding_dim,
        hidden_size)
    self.linear2 = nn.Linear(hidden_size, 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)
      return log_probs
```

(b) Python, only the loop

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

    # Step 2. Recall that torch *accumulates* gradients.
        Before passing in a new instance, you need to 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. (Again, Torch wants
        the target word wrapped in a tensor)
    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()

    # Get the Python number from a 1-element Tensor by calling
         tensor.item()
    total_loss += loss.item()

  #print("\t", total_loss)
  losses.append(total_loss)
```

(c) Python, only the second function

```
def most_similar(word_to_test, word_to_ix):
  test_embedding = get_word_embedding_for_word(word_to_test,
      word_to_ix)

  # get embeddings for all other possible words like aaa bbb
      ccc
  cos = torch.nn.CosineSimilarity()
```

```
    results = {}
    for c in string.ascii_lowercase:
      c_embedding = get_word_embedding_for_word(c+c+c,
          word_to_ix)
      cosine_similarity = cos(test_embedding, c_embedding)
      results[c+c+c] = cosine_similarity.item()

    sorted_results =  dict(sorted(results.items(), key=lambda
        item: -item[1]))
    return sorted_results
```

Task 5

```
# Let's load the model
model = gensim.models.KeyedVectors.load_word2vec_format("
    word2vec_embeddings.bin", binary=True)

# Print the length fo the whole vocabulary
print(len(model.wv.vocab))

# Print the embedding of the  word  good
print(model["good"])

# Get the 10 most similar words of "good"
print(model.most_similar("good", topn = 10))

# Check whether our embeddings are good at the analogy task, what
    is the result to "?-man" when looking at "queen-king"?
print(model.most_similar(positive = ["woman", "king"], negative =
    ["man"], topn = 1))
```