EROL ÖZKAN – HW3

- In this assignment, a neural language model built using Dynet framework. For dataset, Trumph's speeches are used.
- Input and output vector sizes are simply vocabulary size. So, input is just a single word vector, output is a single vector.
- Implemented model predicts the next word vector based on the current word vector.
- 64 hidden units are used.
- The conclusion: sequential models like RNN (more specifically LSTM) should give better results.

1. Read file

```
def read_file(filename):
    with open(filename, encoding="utf8") as file:
        text = file.read()
    return text
```

2. Proprocess text (clean text)

```
def preprocess(text):
    text.replace('SPEECH', ' ')
    text = text.lower()
    tokens = text.split()
    table = str.maketrans('', '', string.punctuation)
    tokens = [token.translate(table) for token in tokens]
    tokens = [token for token in tokens if token.isalpha()]
    return tokens
```

3. Vectorize text

```
def get_vectors(tokens):
    word_to_id = dict()
    id_to_word = []
    counter = Counter(tokens)
    for word, count in counter.items():
        if count >= TRESHOLD:
            id_to_word.append(word)
            word_to_id[word] = len(word_to_id)
    return id_to_word, word_to_id
```

4. Extract bigrams - (x, y) means "x is followed by y"

```
def generate_bigram_corr(clean_text, word_to_id):
    corr = []
    for index in range(0, len(clean_text) - 1):
        if clean_text[index] in word_to_id and clean_text[index + 1] in word_to_id:
            input = word_to_id[clean_text[index]]
            output = word_to_id[clean_text[index + 1]]
            corr.append((input, output))
    return corr
```

5. Create neural network with 64 hidden units

```
HIDDEN_SIZE = 64
INPUT_VEC_SIZE = len(word_to_id)
OUTPUT_VECTOR_SIZE = len(word_to_id)

W = m.add_parameters((HIDDEN_SIZE, INPUT_VEC_SIZE))
b = m.add_parameters(HIDDEN_SIZE)
V = m.add_parameters((OUTPUT_VECTOR_SIZE, HIDDEN_SIZE))
a = m.add_parameters(OUTPUT_VECTOR_SIZE)

x = dy.vecInput(INPUT_VEC_SIZE)
y = dy.vecInput(OUTPUT_VECTOR_SIZE)
h = dy.tanh((W * x) + b)

y_pred = (V * h) + a
loss = dy.squared_distance(y_pred, y)
```

6. Train network for each seen instance

```
for iter in range(ITERATIONS):
  mloss = 0.0
  seen_instances = 0
  for word_pair in bigram_corr:
    x.set(get_vector(word_pair[0], INPUT_VEC_SIZE))
    y.set(get_vector(word_pair[1], OUTPUT_VECTOR_SIZE))
    seen_instances += 1
    mloss += loss.value()
    loss.backward()
    trainer.update()

    if (seen_instances > 1 and seen_instances % 1000 == 0):
      print(seen_instances, "/", len(bigram_corr), "***average loss is:", mloss / seen_instances)

  print("loss: %0.9f" % mloss)
```

7. Create new sentences (recursively)

```
def generate_sentence(word_to_id, id_to_word):
  crated_string = []
  start_word = random.choice(list(word_to_id))
  crated_string.append(start_word)

  def generate_word(word):
    start_word_id = word_to_id[word]
    start_word_vector = get_vector(start_word_id, INPUT_VEC_SIZE)
    x.set(start_word_vector)
    prediction = y_pred.value()
    index, value = max(enumerate(prediction), key=operator.itemgetter(1))
    generated_word = id_to_word[index]
    crated_string.append(generated_word)

    if len(crated_string) < 20:
      generate_word(generated_word)

  generate_word(start_word)
  return crated_string
```

8. Generated Sentences

```
itself thank you to thank you to thank you to thank you to thank you to thank you to thank
wherever thank you to thank you to thank you to thank you to thank you to thank you to thank
releasing thank you to thank you to thank you to thank you to thank you to thank you to thank
gee thank you to thank you to thank you to thank you to thank you to thank you to thank
bridges thank you to thank you to thank you to thank you to thank you to thank you to thank
```