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.

## Read file

def read\_file(filename):

with open(filename, encoding="utf8") as file:

text = file.read()

return text

## 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

## 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

## 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

## 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)**

## 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)

## 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

## 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