import matplotlib.pyplot as plt

import seaborn as sns

import matplotlib as mpl

import matplotlib.pylab as pylab

import numpy as np

%matplotlib inline

#Data Prepration

import re

sentences = """We are about to study the idea of a computational process.

Computational processes are abstract beings that inhabit computers.

As they evolve, processes manipulate other abstract things called data.

The evolution of a process is directed by a pattern of rules

called a program. People create programs to direct processes. In effect,

we conjure the spirits of the computer with our spells."""

# remove special characters

sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)

# remove 1 letter words

sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()

# lower all characters

sentences = sentences.lower()

words = sentences.split()

vocab = set(words)

vocab\_size = len(vocab)

embed\_dim = 10

context\_size = 2

word\_to\_ix = {word: i for i, word in enumerate(vocab)}

ix\_to\_word = {i: word for i, word in enumerate(vocab)}

# data - [(context), target]

data = []

for i in range(2, len(words) - 2):

context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]

target = words[i]

data.append((context, target))

print(data[:5])

embeddings = np.random.random\_sample((vocab\_size, embed\_dim))

def linear(m, theta):

w = theta

return m.dot(w)

def log\_softmax(x):

e\_x = np.exp(x - np.max(x))

return np.log(e\_x / e\_x.sum())

def NLLLoss(logs, targets):

out = logs[range(len(targets)), targets]

return -out.sum()/len(out)

def log\_softmax\_crossentropy\_with\_logits(logits,target):

out = np.zeros\_like(logits)

out[np.arange(len(logits)),target] = 1

softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)

return (- out + softmax) / logits.shape[0]

def forward(context\_idxs, theta):

m = embeddings[context\_idxs].reshape(1, -1)

n = linear(m, theta)

o = log\_softmax(n)

return m, n, o

def backward(preds, theta, target\_idxs):

m, n, o = preds

dlog = log\_softmax\_crossentropy\_with\_logits(n, target\_idxs)

dw = m.T.dot(dlog)

return dw

def optimize(theta, grad, lr=0.03):

theta -= grad \* lr

return theta

#Genrate training data

theta = np.random.uniform(-1, 1, (2 \* context\_size \* embed\_dim, vocab\_size))

epoch\_losses = {}

for epoch in range(80):

losses = []

for context, target in data:

context\_idxs = np.array([word\_to\_ix[w] for w in context])

preds = forward(context\_idxs, theta)

target\_idxs = np.array([word\_to\_ix[target]])

loss = NLLLoss(preds[-1], target\_idxs)

losses.append(loss)

grad = backward(preds, theta, target\_idxs)

theta = optimize(theta, grad, lr=0.03)

epoch\_losses[epoch] = losses

ix = np.arange(0,80)

fig = plt.figure()

fig.suptitle('Epoch/Losses', fontsize=20)

plt.plot(ix,[epoch\_losses[i][0] for i in ix])

plt.xlabel('Epochs', fontsize=12)

plt.ylabel('Losses', fontsize=12)

def predict(words):

context\_idxs = np.array([word\_to\_ix[w] for w in words])

preds = forward(context\_idxs, theta)

word = ix\_to\_word[np.argmax(preds[-1])]

return word

# (['we', 'are', 'to', 'study'], 'about')

predict(['we', 'are', 'to', 'study'])

def accuracy():

wrong = 0

for context, target in data:

if(predict(context) != target):

wrong += 1

return (1 - (wrong / len(data)))

accuracy()

predict(['processes', 'manipulate', 'things', 'study'])