In [1]:
```
import re
import numpy as np
import string
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

from subprocess import check_output
from wordcloud import WordCloud, STOPWORDS

stopwords = set(STOPWORDS)
data ="""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 d
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In ef
we conjure the spirits of the computer with our spells."""

wordcloud = WordCloud(
                            background_color='white',
                            stopwords=stopwords,
                            max_words=200,
                            max_font_size=40,
                            random_state=42
                           ).generate(data)

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(24, 24))
axes[0].imshow(wordcloud)
axes[0].axis('off')
axes[1].imshow(wordcloud)
axes[1].axis('off')
axes[2].imshow(wordcloud)
axes[2].axis('off')
fig.tight_layout()
```

Definition

Word2vec

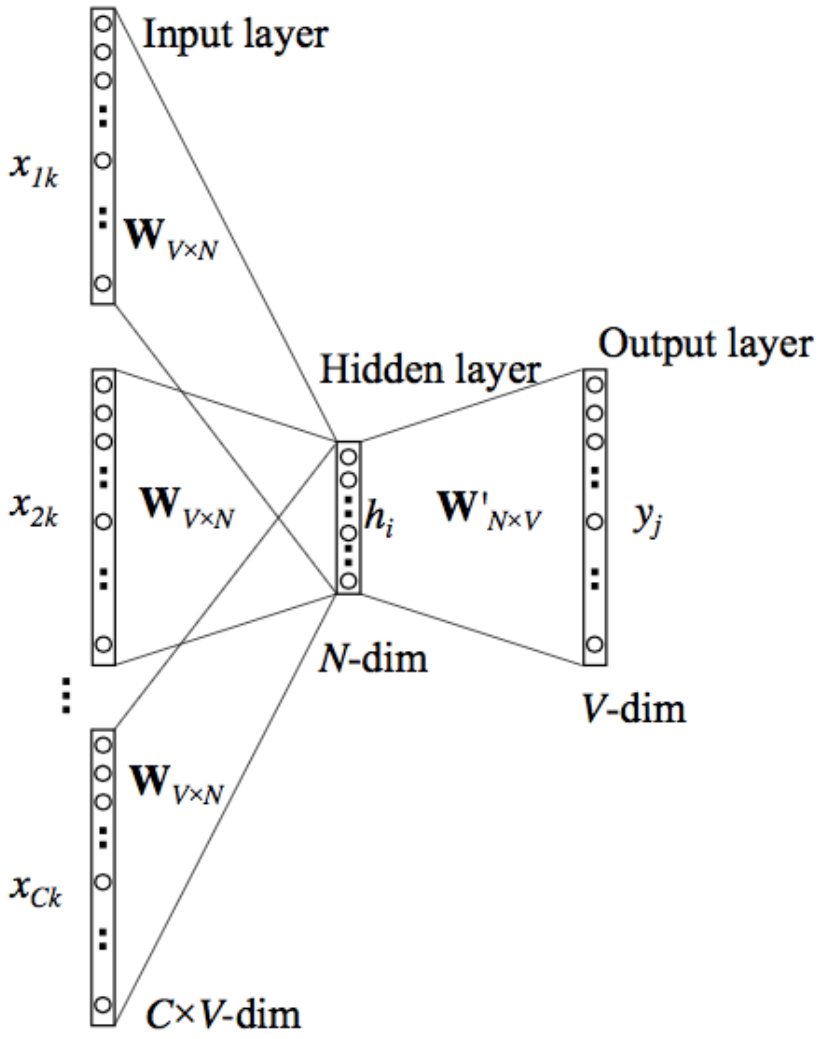
Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space.

CBOW

CBOW or Continous bag of words is to use embeddings in order to train a neural network where the context is represented by multiple words for a given target words.

For example, we could use “cat” and “tree” as context words for “climbed” as the target word.
This calls for a modification to the neural network architecture.
The modification, shown below, consists of replicating the input to hidden layer connections C times, the number of context words, and adding a divide by C operation in the hidden laver neurons.

CBOW Architecture



The CBOW architecture is pretty simple contains :
- the word embeddings as inputs (idx) - the linear model as the hidden layer - the log_softmax as the output

Dataset

In [2]:
```
sentences = """We are about to study the idea of a computational pr
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called d
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In ef
we conjure the spirits of the computer with our spells."""
```

Clean Data

In [3]:
```
# 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()
```

Vocabulary

In [4]:
```
words = sentences.split()
vocab = set(words)
```

In [5]:
```
vocab_size = len(vocab)
embed_dim = 10
context_size = 2
```

Implementation

Dictionaries

In [6]:
```
word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
```

Data bags

In [7]:
```
# data - [(context), target]

data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i +
    target = words[i]
    data.append((context, target))
print(data[:5])
```

```
[(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'stud
```

Embeddings

In [8]:
```
embeddings = np.random.random_sample((vocab_size, embed_dim))
```

Linear Model

In [9]:
```
def linear(m, theta):
    w = theta
    return m.dot(w)
```

Log softmax + NLLloss = Cross Entropy

In [10]:
```
def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
```

In [11]:
```
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)
```

In [12]:
```
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=

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

Forward function

In [13]:
```
def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)

    return m, n, o
```

Backward function

In [14]:
```
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
```

Optimize function

In [15]:
```
def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta
```

Training

In [16]:
```
theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, voc
```

In [17]:
```
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
```

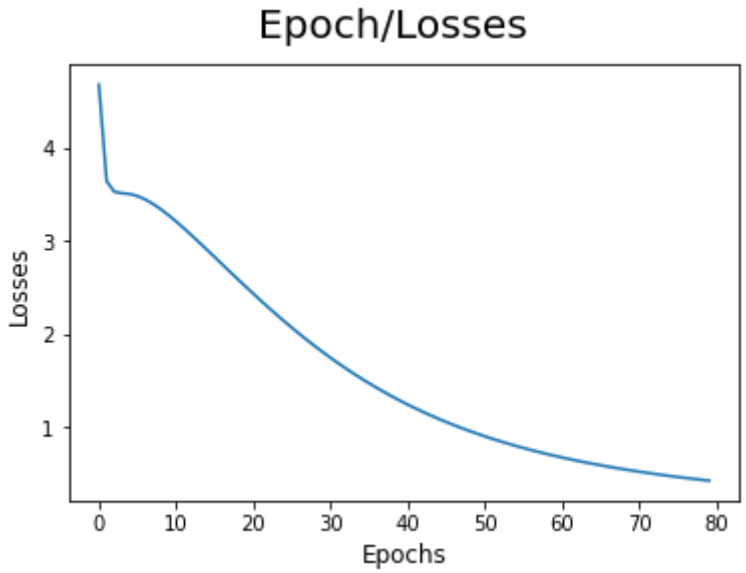
Analyze

Plot loss/epoch

In [18]:
```
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)
```

Out[18]:
```
Text(0, 0.5, 'Losses')
```



Predict function

In [19]:
```
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
```

In [20]:
```
# (['we', 'are', 'to', 'study'], 'about')
predict(['we', 'are', 'to', 'study'])
```

Out[20]:
```
'about'
```

Accuracy

In [21]:
```
def accuracy():
    wrong = 0

    for context, target in data:
        if(predict(context) != target):
            wrong += 1

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

In [22]:
```
accuracy()
```

Out[22]:
```
1.0
```

80 epochs and 100% accuracy

If you enjoyed this post, don't forget to up-vote!

In [ ]:

In [ ]:

In [ ]: