NYU DATA SCIENCE

word2vec in theory and practice with tensorflow

https://github.com/bmtgoncalves/word2vec-and-friends/

Bruno Gonçalves

www.bgoncalves.com

Teaching machines to read!

- Computers are really good at crunching numbers but not so much when it comes to words.

- Perhaps can we represent words numerically?

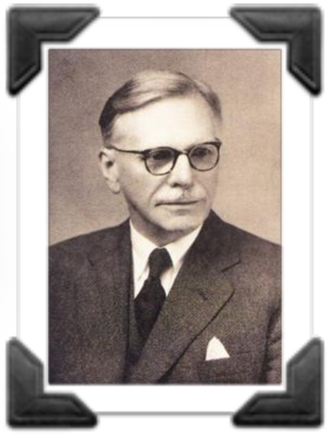
$$v_{after} = (0, 0, 0, 1, 0, 0, \cdots)^T$$
$$v_{above} = (0, 0, 1, 0, 0, 0, \cdots)^T$$

One-hot encoding

a	1
about	2
above	3
after	4
again	5
against	6
all	7
am	8
an	9
and	10
any	11
are	12
aren't	13
as	14
...	...

- Can we do it in a way that preserves semantic information?

"You shall know a word by the company it keeps"
(J. R. Firth)



- Words that have similar meanings are used in similar contexts and the context in which a word is used helps us understand it's meaning.

The red house is beautiful.
The blue house is old.
The red car is beautiful.
The blue car is old.

Teaching machines to read!

"You shall know a word by the company it keeps"
(J. R. Firth)

➡Words with similar meanings should have similar representations.

➡From a word we can get some idea about the context where it might appear

___ ___ house __ ____.
___ ___ car __ ______.

$$\max p\left(C|w\right)$$

➡And from the context we have some idea about possible words

The red _____ is beautiful.
The blue _____ is old.

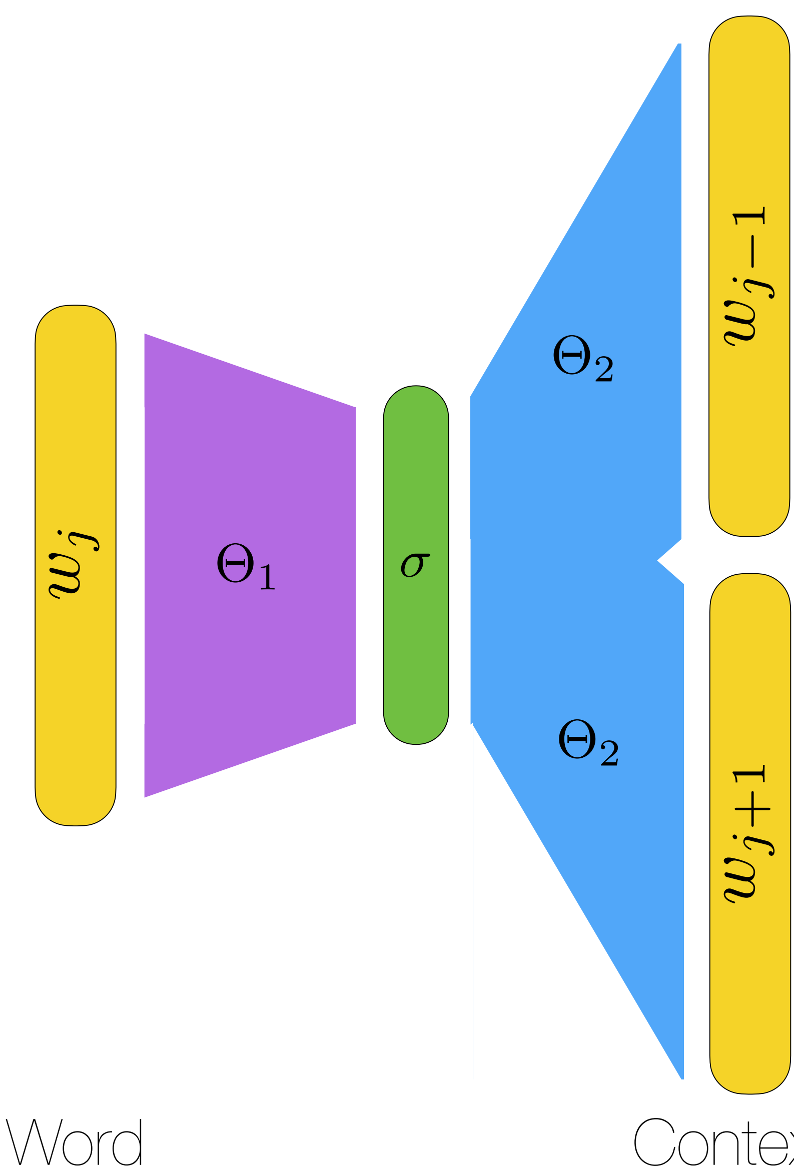
$$\max p\left(w|C\right)$$

word2vec

Mikolov 2013

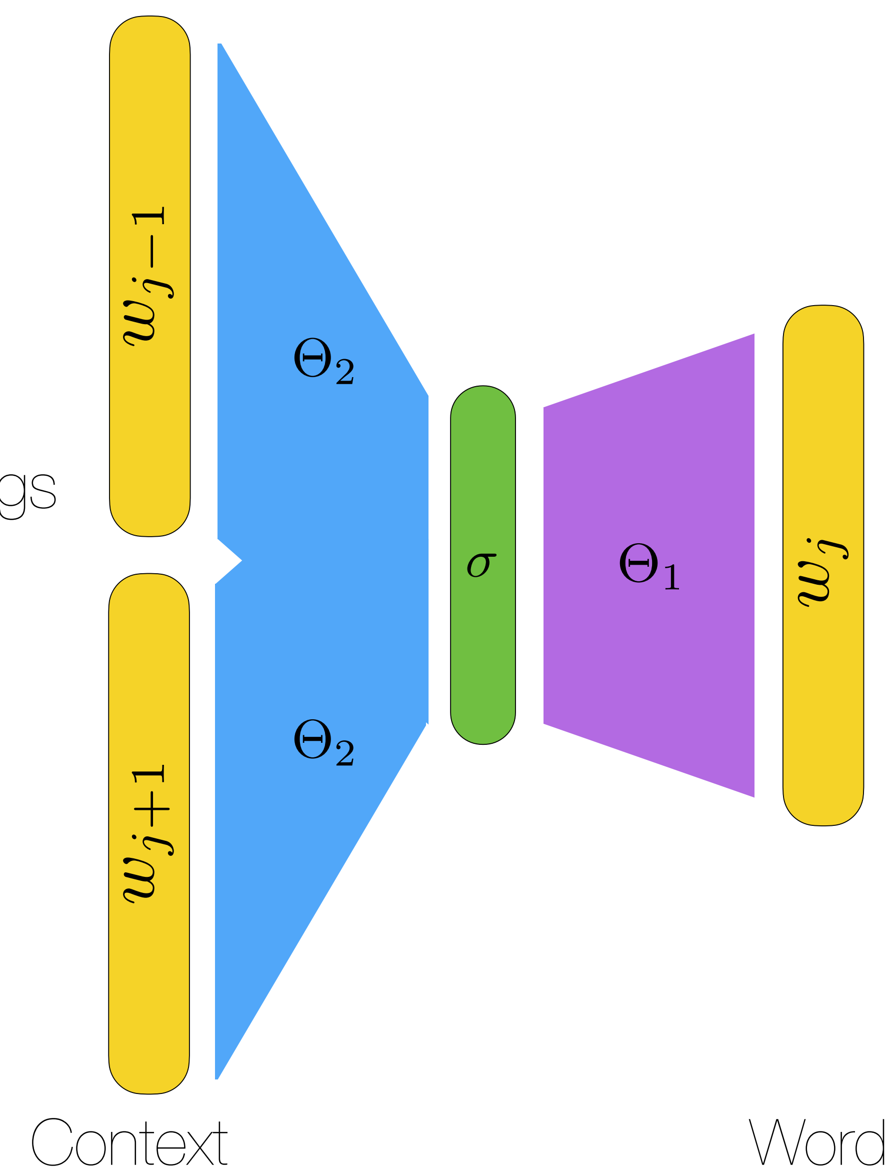
Skipgram

$$\max p\left(C|w\right)$$



Continuous Bag of Words

$$\max p\left(w|C\right)$$



Θ_1 word embeddings

Θ_2 context embeddings

w_j one hot vector

σ activation function

Skipgram

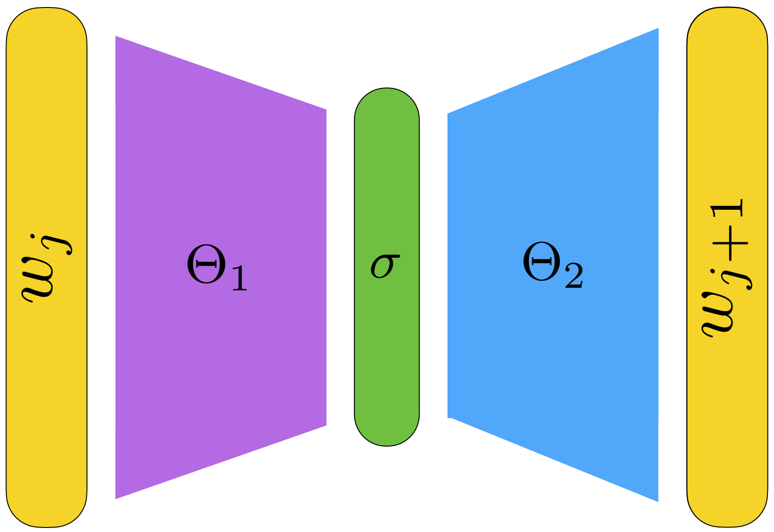
- Let us take a better look at a simplified case with a single context word
- Words are one-hot encoded vectors $w_j = (0, 0, 1, 0, 0, 0, \cdots)^T$ of length V
- Θ_1 is an $(M \times V)$ matrix so that when we take the product:

$$\Theta_1 \cdot w_j$$

- We are effectively selecting the j'th column of Θ_1:

$$v_j = \Theta_1 \cdot w_j$$

- The linear activation function simply passes this value along which is then multiplied by Θ_2, a $(V \times M)$ matrix.



- Each element k of the output layer its then given by:

$$u_k^T \cdot v_j$$

- We convert these values to a normalized probability distribution by using the softmax

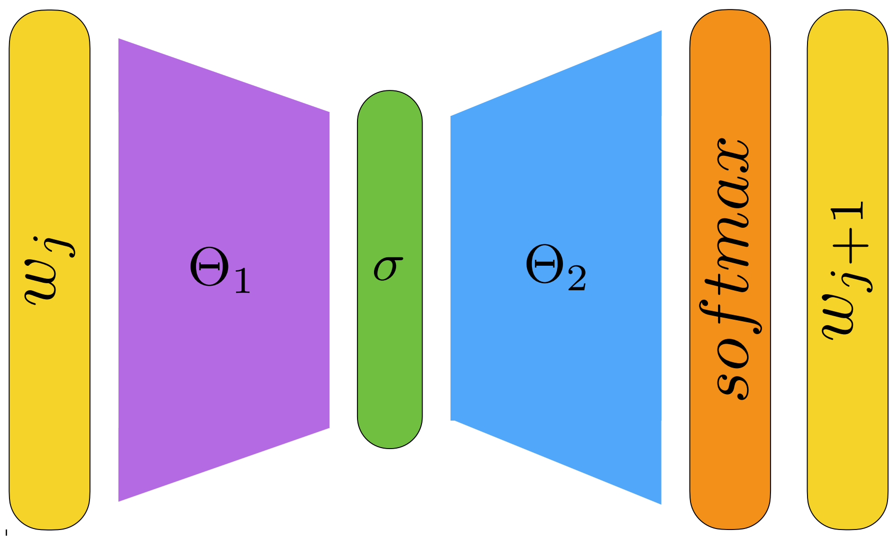
Softmax

- A standard way of converting a set of number to a normalized probability distribution:

$$softmax\left(x\right) = \frac{\exp\left(x_j\right)}{\sum_l \exp\left(x_l\right)}$$

- With this final ingredient we obtain:

$$p\left(w_k|w_j\right) \equiv softmax\left(u_k^T \cdot v_j\right) = \frac{\exp\left(u_k^T \cdot v_j\right)}{\sum_l \exp\left(u_l^T \cdot v_j\right)}$$



- Our goal is then to learn:

$$\Theta_1 \qquad \Theta_2$$

- so that we can predict what the next word is likely to be using:

$$p\left(w_{j+1}|w_j\right)$$

- But how can we quantify how far we are from the correct answer? Our error measure shouldn't be just binary (right or wrong)…

Cross-Entropy

- First we have to recall that what we are, in effect, comparing two probability distributions:

$$p\left(w_k|w_j\right)$$

- and the one-hot encoding of the context:

$$w_{j+1} = \left(0, 0, 0, 1, 0, 0, \cdots\right)^T$$

- The Cross Entropy measures the distance, in number of bits, between two probability distributions p and q:

$$H\left(p, q\right) = -\sum_k p_k \log q_k$$

- In our case, this becomes:

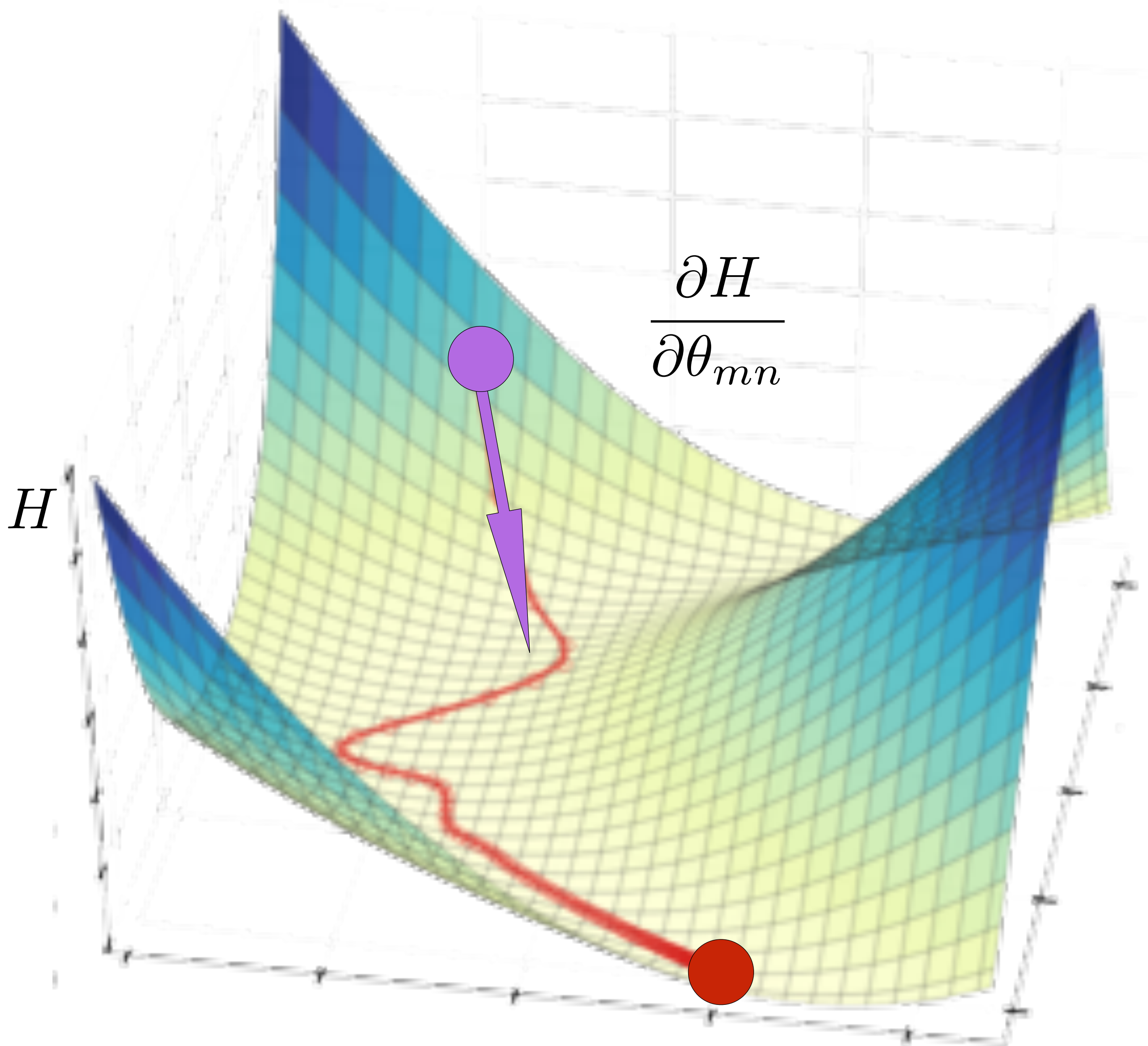
$$H\left[w_{j+1}, p\left(w_k|w_j\right)\right] = -\sum_k w_{j+1}^k \log p\left(w_k|w_j\right)$$

- So it's clear that the only non zero term is the one that corresponds to the "hot" element of w_{j+1}

$$H = -\log p\left(w_{j+1}|w_j\right)$$

- This is our Error function. But how can we use this to update the values of Θ_1 and Θ_2?

Gradient Descent



- Find the gradient for each training batch
- Take a step **downhill** along the direction of the gradient

$$\theta_{mn} \leftarrow \theta_{mn} - \alpha \frac{\partial H}{\partial \theta_{mn}}$$

- where α is the step size.
- Repeat until "convergence".

Chain-rule

- How can we calculate

$$\frac{\partial H}{\partial \theta_{mn}} = \frac{\partial}{\partial \theta_{mn}} \log p\left(w_{j+1} | w_j\right) \qquad \theta_{mn} = \left\{\theta_{mn}^{(1)}, \theta_{mn}^{(2)}\right\}$$

- we rewrite:

$$\frac{\partial H}{\partial \theta_{mn}} = \frac{\partial}{\partial \theta_{mn}} \log \frac{\exp\left(u_k^T \cdot v_j\right)}{\sum_l \exp\left(u_l^T \cdot v_j\right)}$$

- and expand:

$$\frac{\partial H}{\partial \theta_{mn}} = \frac{\partial}{\partial \theta_{mn}} \left[u_k^T \cdot v_j - \log \sum_l \exp\left(u_l^T \cdot v_j\right)\right]$$

- Then we can rewrite:

$$u_k^T \cdot v_j = \sum_q \theta_{kq}^{(2)} \theta_{qj}^{(1)}$$

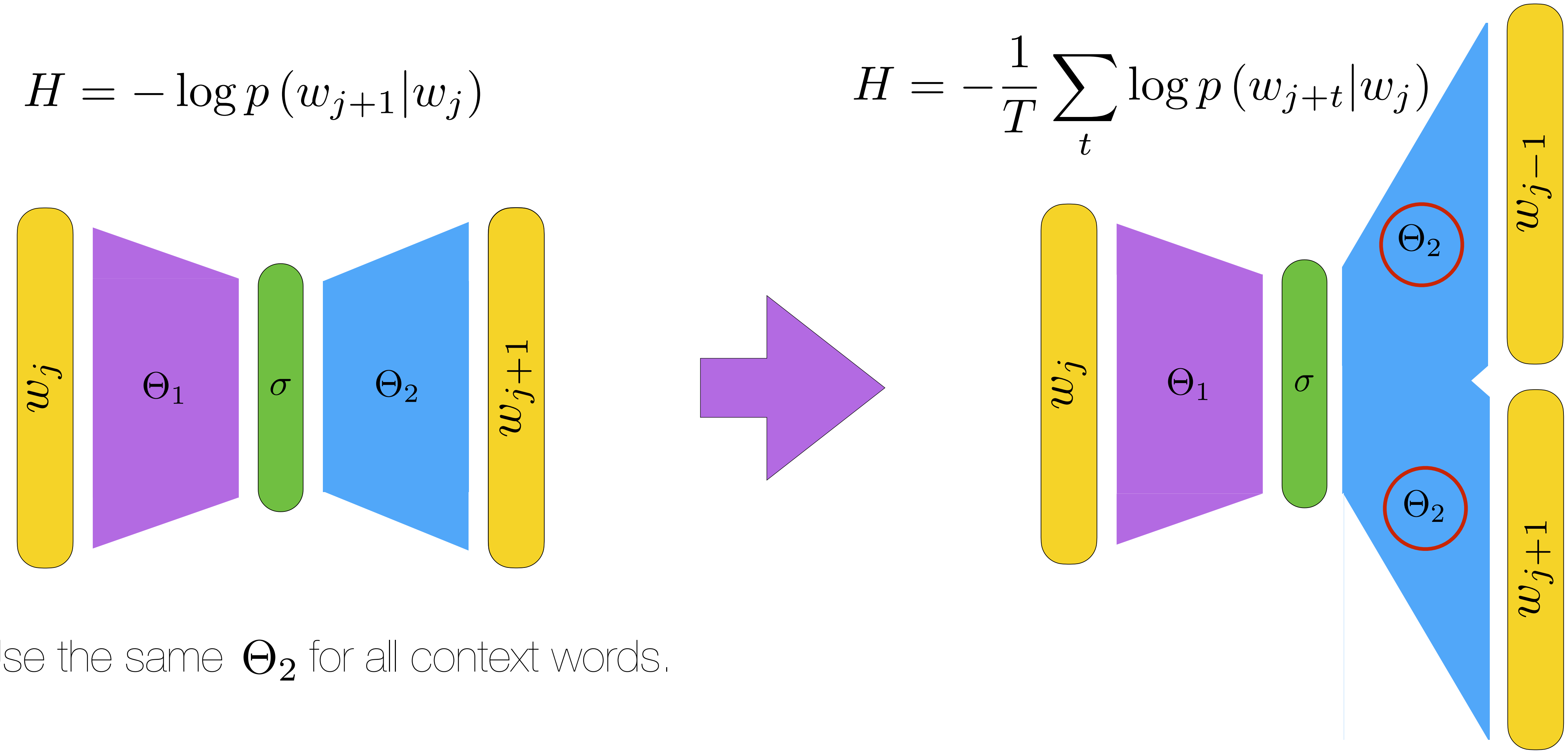
- and apply the chain rule:

$$\frac{\partial f\left(g\left(x\right)\right)}{\partial x} = \frac{\partial f\left(g\left(x\right)\right)}{\partial g\left(x\right)} \frac{\partial g\left(x\right)}{\partial x}$$

SkipGram with Larger Contexts

$$H = -\log p\left(w_{j+1}|w_j\right)$$

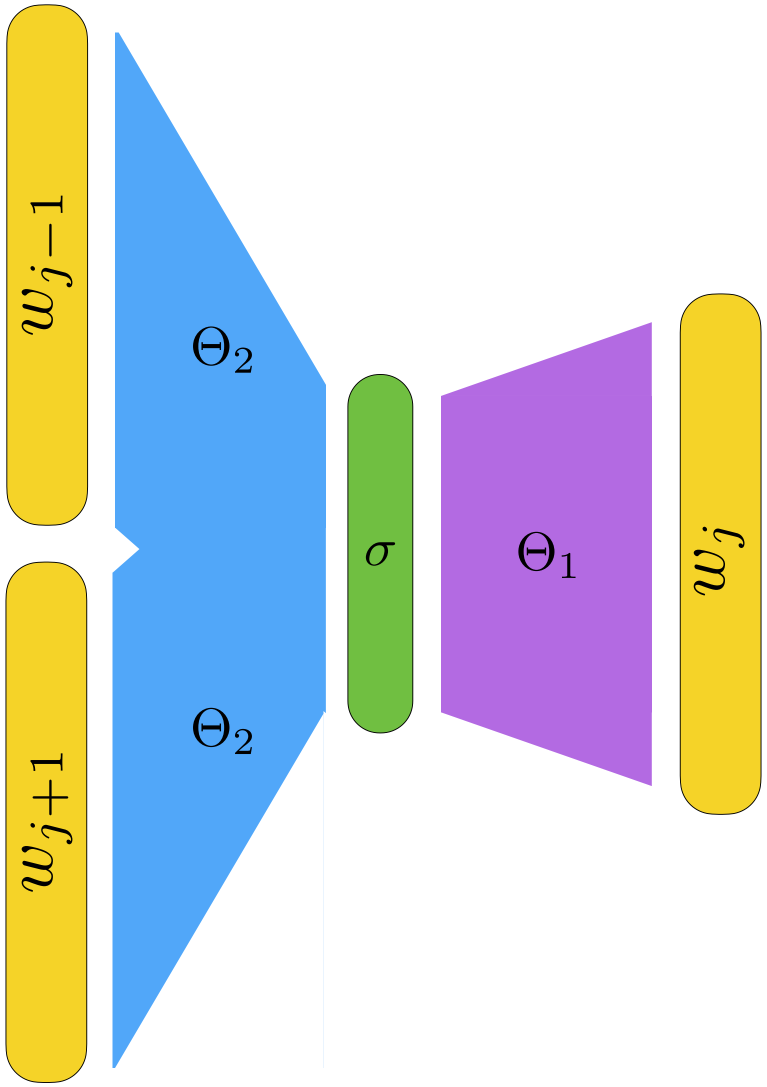
$$H = -\frac{1}{T}\sum_t \log p\left(w_{j+t}|w_j\right)$$



- Use the same Θ_2 for all context words.
- Use the average of cross entropy.
- word order is not important (the average does not change).

Continuous Bag of Words

- The process is essentially the same

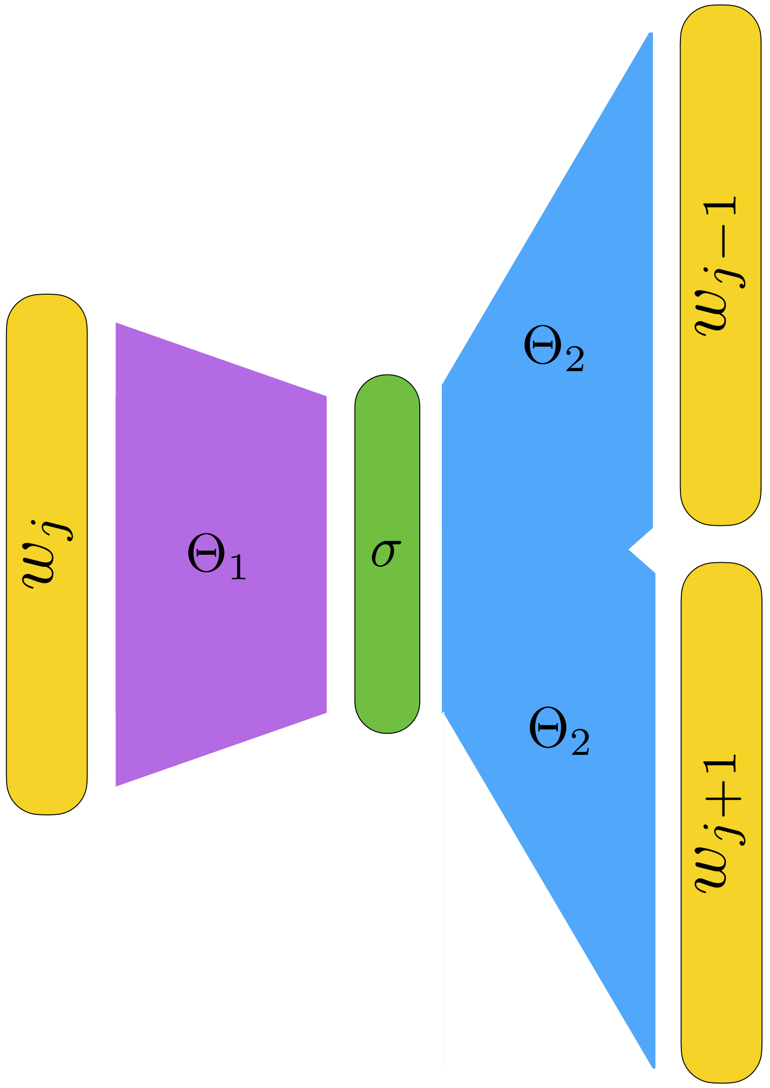


Variations

- Hierarchical Softmax:
  - Approximate the softmax using a binary tree
  - Reduce the number of calculations per training example from V to $\log_2 V$ and increase performance by orders of magnitude.
- Negative Sampling:
  - Under sample the most frequent words by removing them from the text before generating the contexts
  - Similar idea to removing stop-words — very frequent words are less informative.
  - Effectively makes the window larger, increasing the amount of information available for context

Comments

- word2vec, even in its original formulation is actually a family of algorithms using various combinations of:
  - Skip-gram, CBOW
  - Hierarchical Softmax, Negative Sampling
- The output of this neural network is deterministic:
  - If two words appear in the same context ("blue" vs "red", for e.g.), they will have similar internal representations in Θ_1 and Θ_2
  - Θ_1 and Θ_2 are vector embeddings of the input words and the context words respectively
- Words that are too rare are also removed.
- The original implementation had a dynamic window size:
  - for each word in the corpus a window size k' is sampled uniformly between 1 and k



Online resources

- C - https://code.google.com/archive/p/word2vec/ (the original one)
- Python/tensorflow - https://www.tensorflow.org/tutorials/word2vec
  - Both a minimalist and an efficient versions are available in the tutorial
- Python/gensim - https://radimrehurek.com/gensim/models/word2vec.html
- Pretrained embeddings:
  - 90 languages, trained using wikipedia: https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

Analogies

"You shall know a word by the company it keeps"
(J. R. Firth)

- The embedding of each word is a function of the context it appears in:

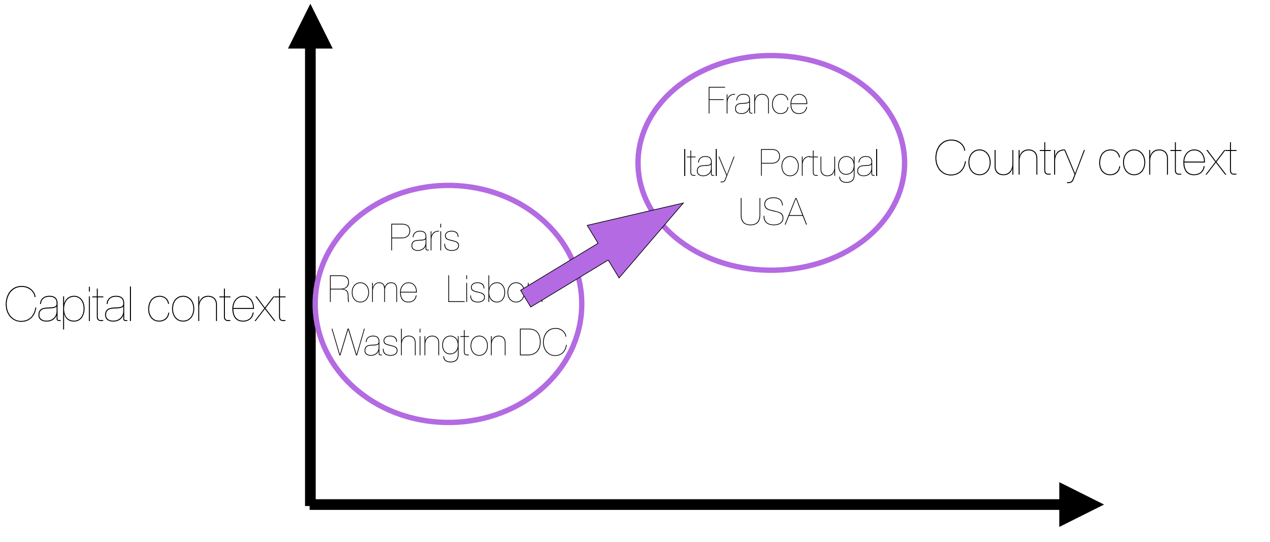
$$\sigma\left(red\right) = f\left(context\left(red\right)\right)$$

- words that appear in similar contexts will have similar embeddings:

$$context\left(red\right) \approx context\left(blue\right) \implies \sigma\left(red\right) \approx \sigma\left(blue\right)$$

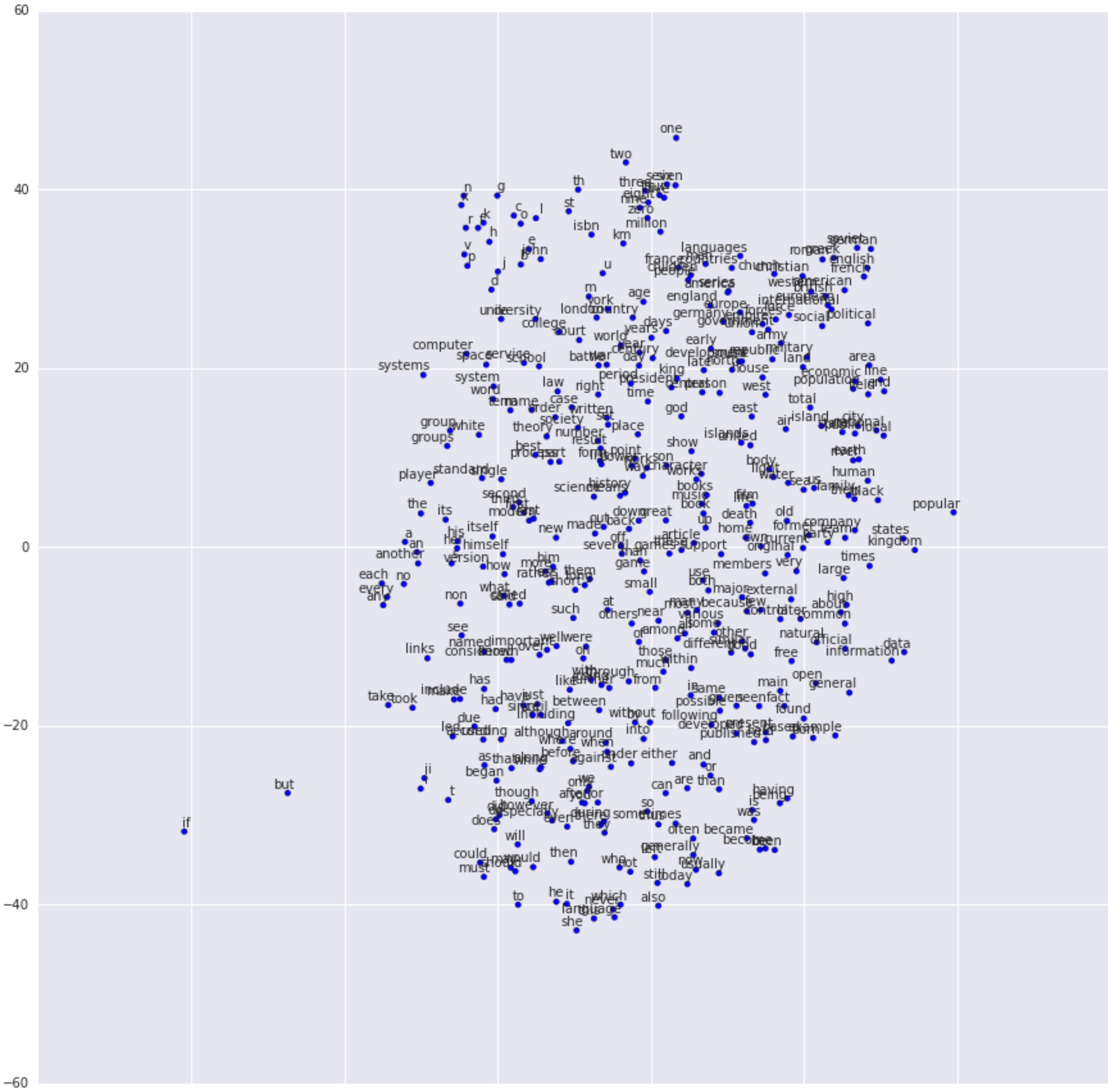
- "Distributional hypotesis" in linguistics

Geometrical relations between contexts imply semantic relations between words!



$$\sigma\left(France\right) - \sigma\left(Paris\right) + \sigma\left(Rome\right) = \sigma\left(Italy\right)$$

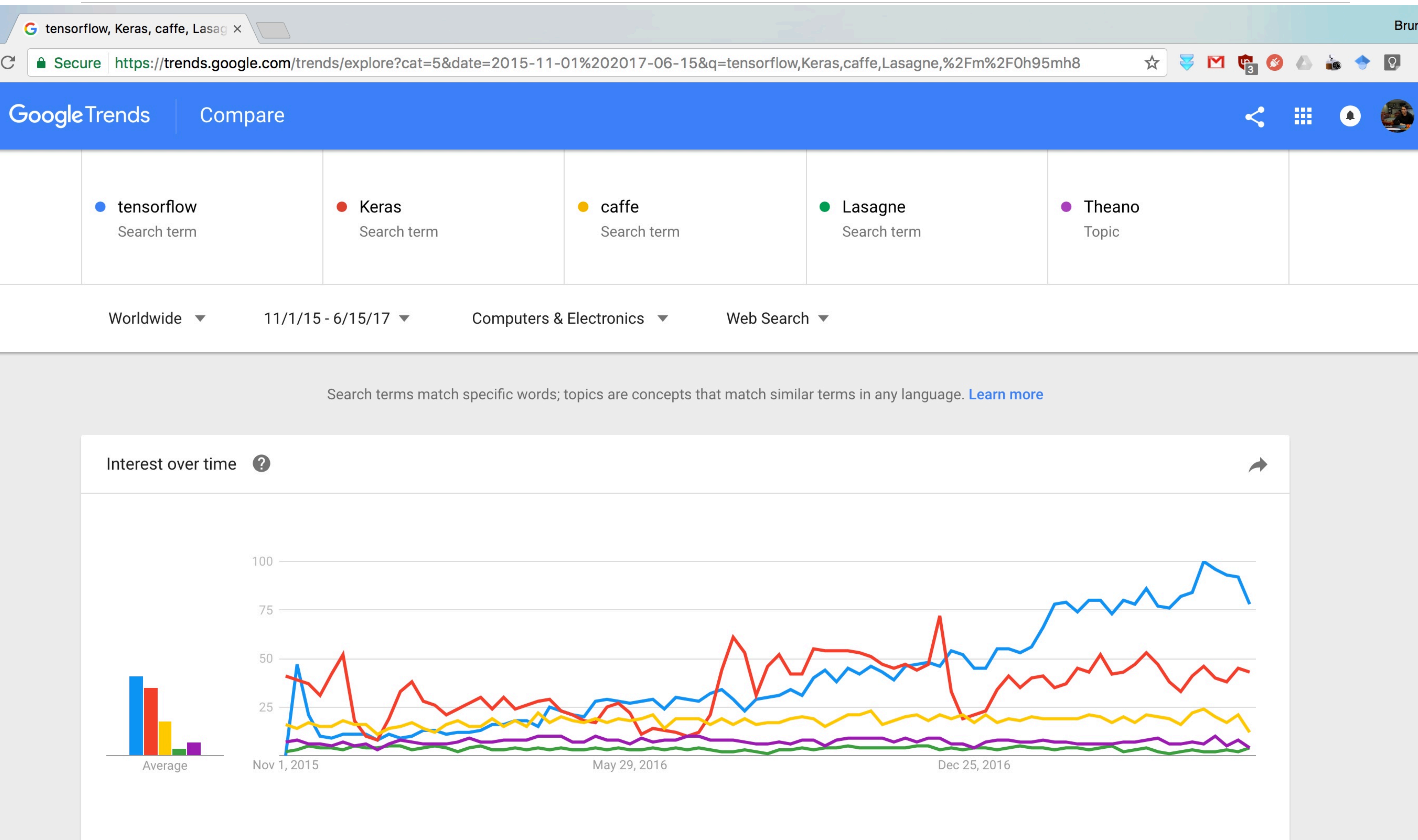
Visualization



TensorFlow

https://github.com/bmtgoncalves/word2vec-and-friends/

Tensorflow



tensorflow, Keras, caffe, Lasag
Bruno
Secure https://trends.google.com/trends/explore?cat=5&date=2015-11-01%202017-06-15&q=tensorflow,Keras,caffe,Lasagne,%2Fm%2F0h95mh8
Google Trends
Compare
tensorflow
Search term
Keras
Search term
caffe
Search term
Lasagne
Search term
Theano
Topic
Worldwide
11/1/15 - 6/15/17
Computers & Electronics
Web Search
Search terms match specific words; topics are concepts that match similar terms in any language. Learn more
Interest over time
100
75
50
25
Average
Nov 1, 2015
May 29, 2016
Dec 25, 2016

A diversion…

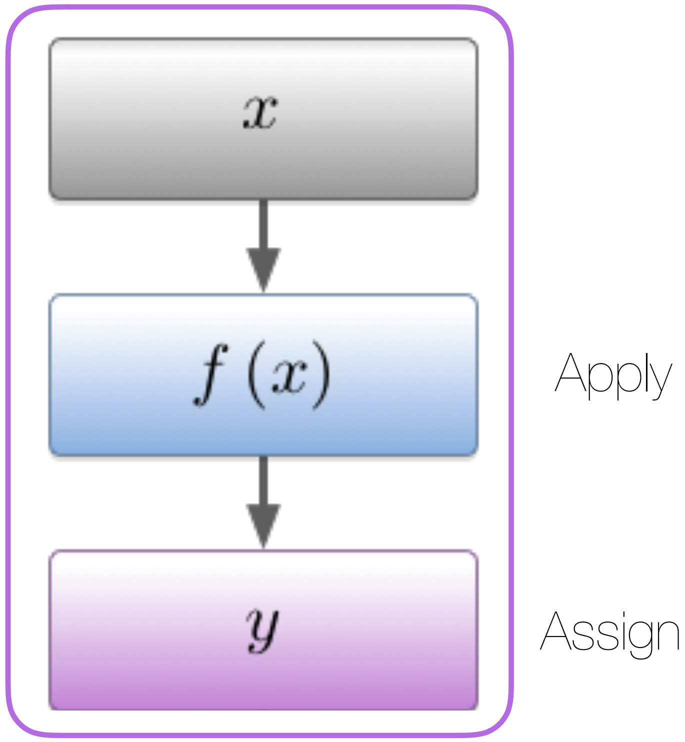
https://www.tensorflow.org/

- Let's imagine I want to perform these calculations:

$$y = f\left(x\right)$$

$$z = g\left(y\right)$$

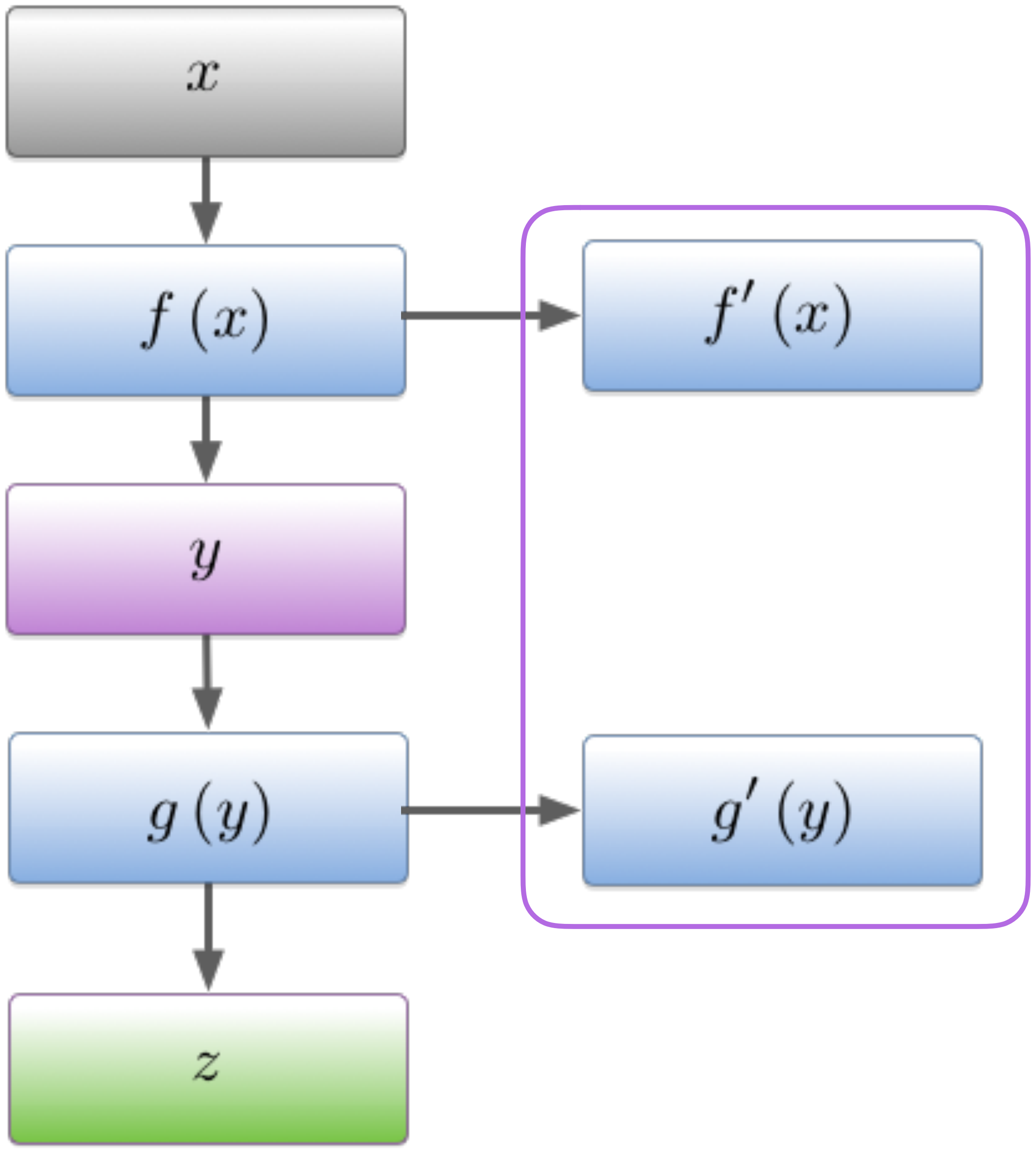
- for some given x .
- To calculate z we must follow a certain sequence of operations.
- Which can be shortened if we are interested in just the value of y
- In Tensorflow, this is called a Computational Graph and it's the most fundamental concept to understand
- Data, in the form of tensors, flows through the graph from inputs to outputs
- Tensorflow, is, essentially, a way of defining arbitrary computational graphs in a way that can be automatically distributed and optimized.



Computational Graphs

https://www.tensorflow.org/

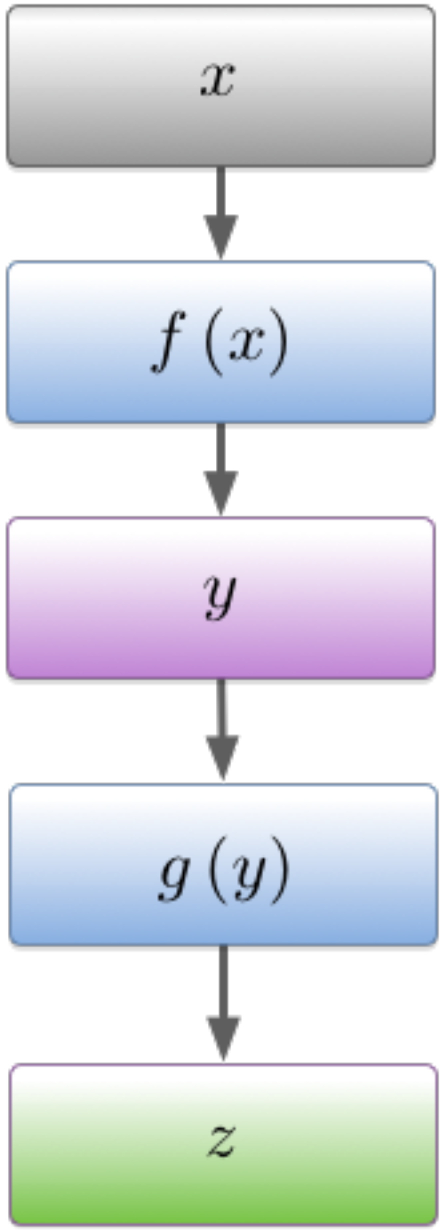
- If we use base functions, tensorflow knows how to automatically calculate the respective gradients
- Automatic BackProp
- Graphs can have multiple outputs
    - Predictions
    - Cost functions
    - etc…



Sessions

https://www.tensorflow.org/

- After we have defined the computational graph, we can start using it to make calculations
- All computations must take place within a "session" that defines the values of all required input values
- Which values are required for a specific computation depend on what part of the graph is actually being executed.
- When you request the value of a specific output, tensorflow determines what is the specific subgraph that must be executed and what are the required input values.
- For optimization purposes, it can also execute independent parts of the graph in different devices (CPUs, GPUs, TPUs, etc) at the same time.



A basic Tensorflow program

TensorFlow
https://www.tensorflow.org/

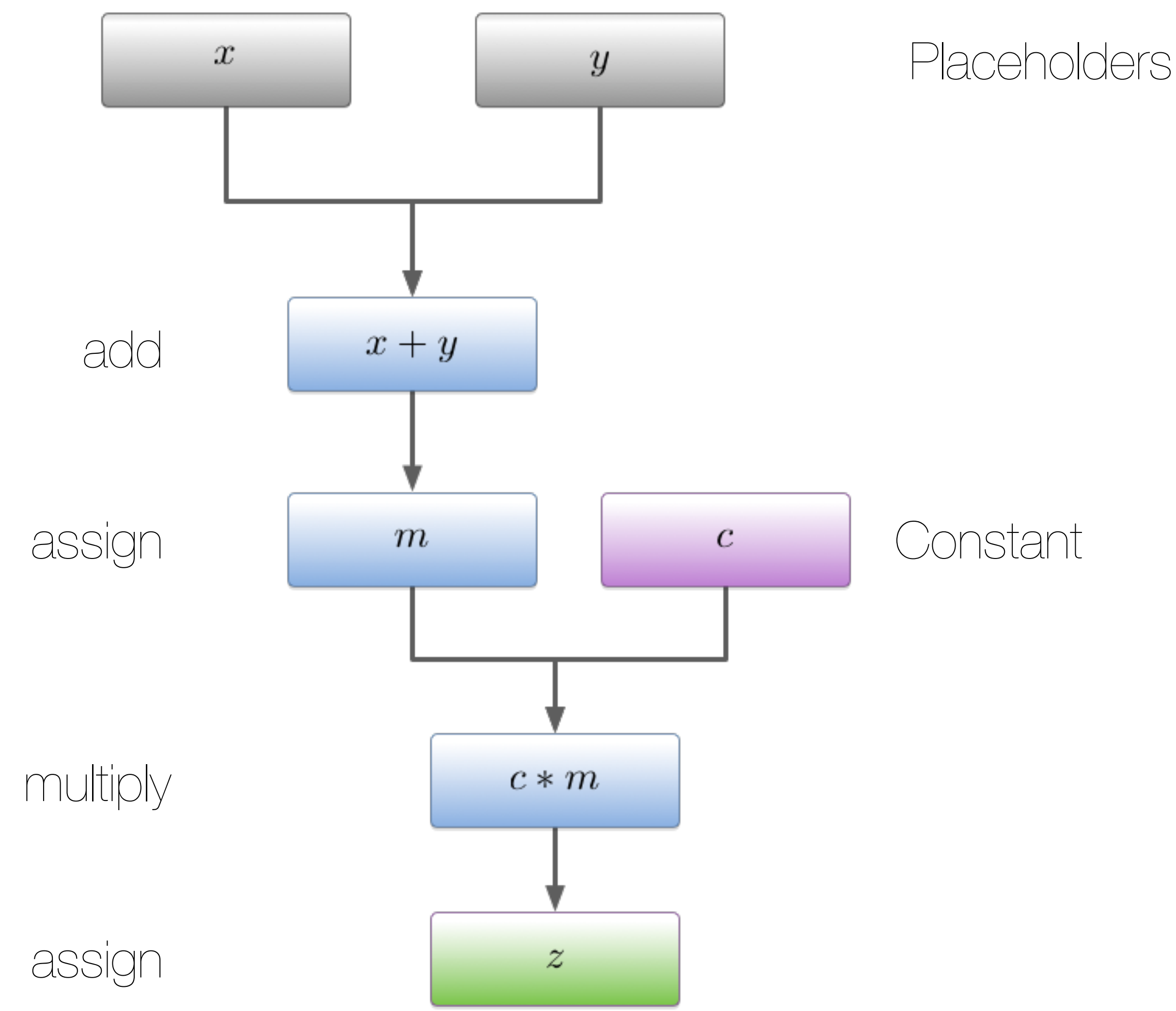
$$z = c * (x + y)$$

```python
import tensorflow as tf

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
c = tf.constant(3.)

m = tf.add(x, y)
z = tf.multiply(m, c)

with tf.Session() as sess:
    output = sess.run(z, feed_dict={x: 1., y: 2.})
    print("Output value is:", output)
```



@bgoncalves

basic.py

A basic Tensorflow program

https://www.tensorflow.org/

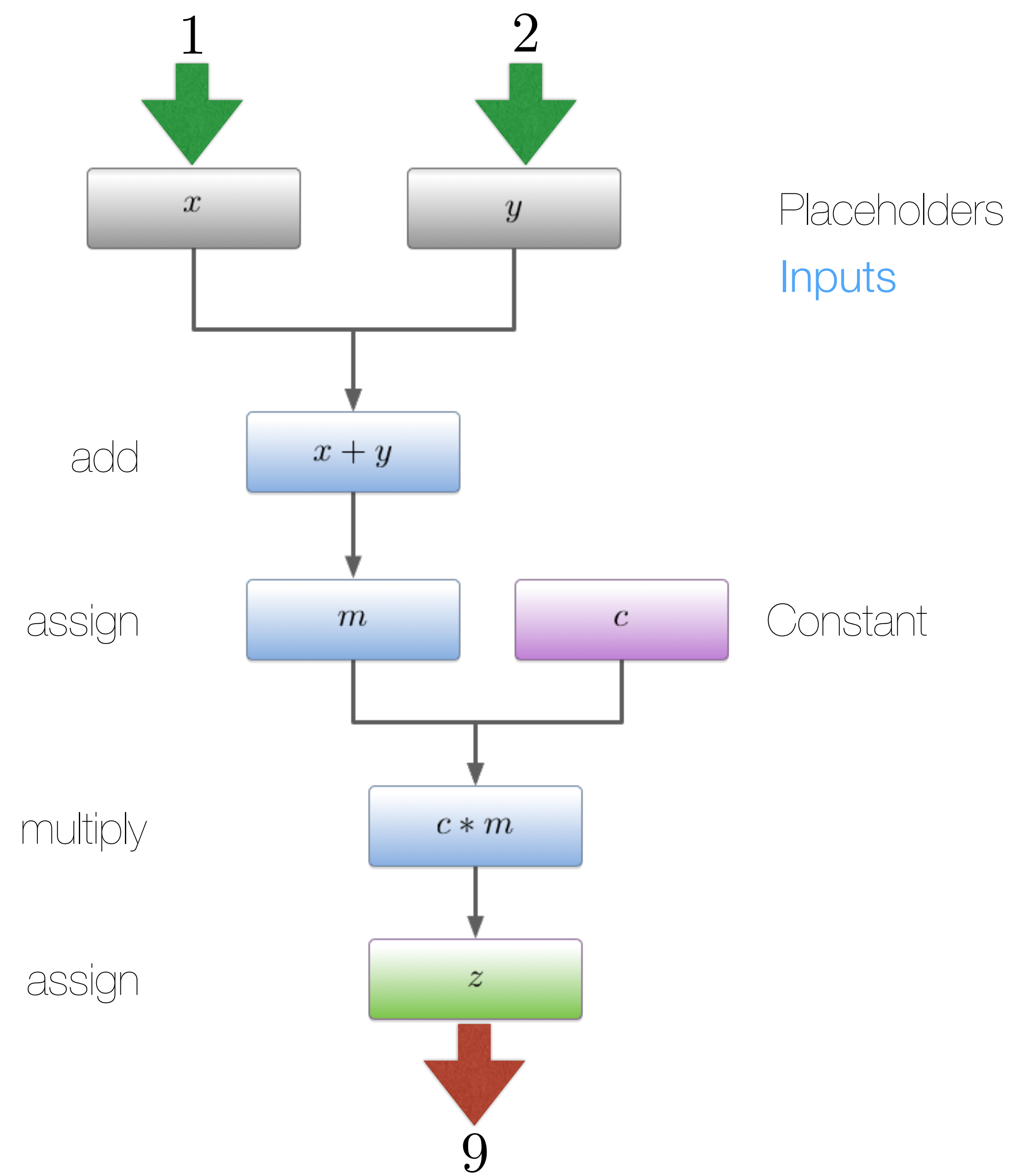
$$z = c * (x + y)$$

```
import tensorflow as tf

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
c = tf.constant(3.)

m = tf.add(x, y)
z = tf.multiply(m, c)

with tf.Session() as sess:
    output = sess.run(z, feed_dict={x: 1., y: 2.})
    print("Output value is:", output)
```



@bgoncalves

Linear Regression

```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

learning_rate = 0.01
N = 100
N_steps = 300

# Training Data
train_X = np.linspace(-10, 10, N)
train_Y = 2*train_X + 3 + 5*np.random.random(N)

# Computational Graph
X = tf.placeholder("float")
Y = tf.placeholder("float")

W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")
y = tf.add(tf.multiply(X, W), b)

cost = tf.reduce_mean(tf.pow(y-Y, 2))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

    for step in range(N_steps):
        sess.run(optimizer, feed_dict={X: train_X, Y: train_Y})
        cost_val, W_val, b_val = sess.run([cost, W, b], feed_dict={X: train_X, Y:train_Y})
        print("step", step, "cost", cost_val, "w", W_val, "b", b_val)
```

$$y = W * x + b$$

$$cost = \frac{1}{N}\sum_i (y_i - Y_i)^2$$

linear.py

@bgoncalves

Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

learning_rate = 0.01
N = 100
N_steps = 300

# Training Data
train_X = np.linspace(-10, 10, N)
train_Y = 2*train_X + 3 + 5*np.random.random(N)

# Computational Graph
X = tf.placeholder("float")
Y = tf.placeholder("float")

W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")
y = tf.add(tf.multiply(X, W), b)

cost = tf.reduce_mean(tf.pow(y-Y, 2))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

    for step in range(N_steps):
        sess.run(optimizer, feed_dict={X: train_X, Y: train_Y})
        cost_val, W_val, b_val = sess.run([cost, W, b], feed_dict={X: train_X, Y:train_Y})
        print("step", step, "cost", cost_val, "w", W_val, "b", b_val)
```

$$y = W * x + b$$

$$cost = \frac{1}{N} \sum_i (y_i - Y_i)^2$$

linear.py

@bgoncalves

Jupyter Notebook