POLITECNICO
MILANO 1863

Neural Graph Machines:
Learning Neural Networks Using Graphs

with an application in Sentiment Analysis

Advanced Algorithms and Parallel Programming

Gerlando Savio Scibetta 863511
scibetta.savio@gmail.com

What this project is about

- Semi-Supervised Learning with Neural Graph Machines
- Deep Learning applied to Natural Language Processing
- Binary Sentiment Analysis of Movie Reviews

Tools: python™ TensorFlow™

https://github.com/gssci/neural-graph-machine-sentiment-analysis

Intro

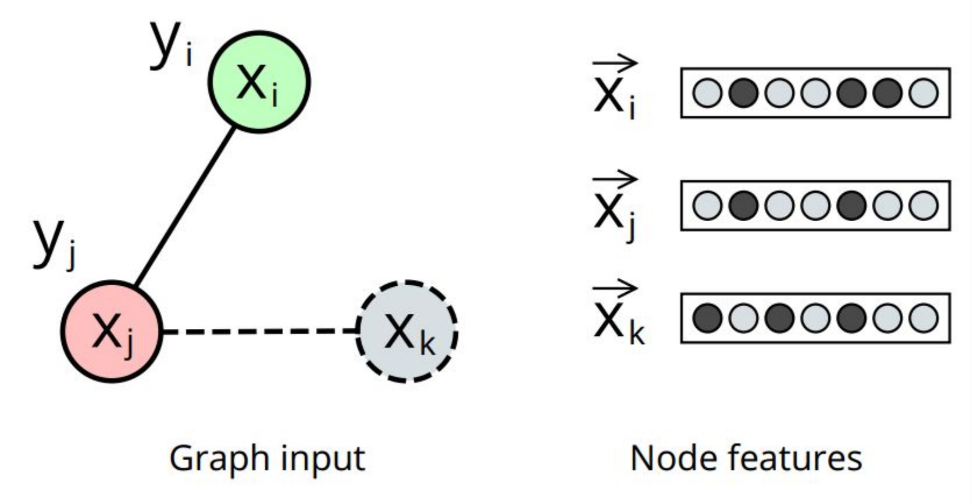
The proposed paper illustrates a semi-supervised learning technique for a variety of neural networks using both labeled and unlabeled training data, by using an algorithm that combines maximum likelihood learning with Label Propagation.

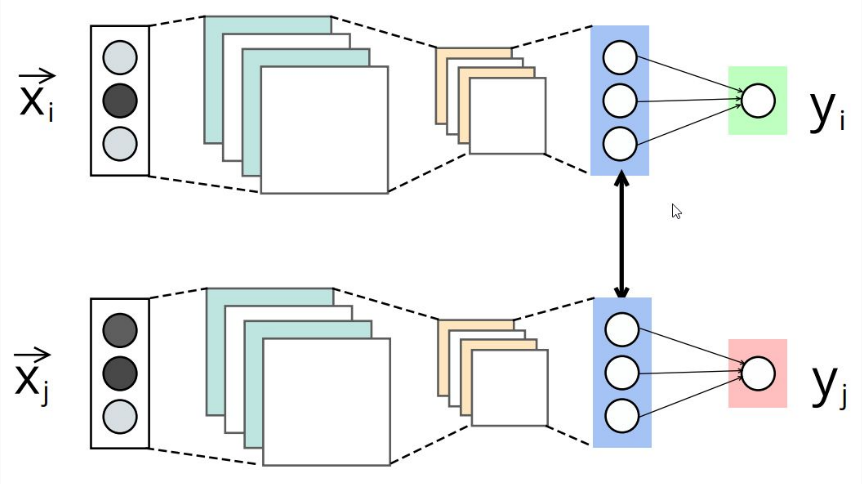
- The graph describes relationships between nodes, such as similarities between embeddings, phrases or images.

The new objective allows the neural networks to harness both labeled and unlabeled data by:

- allowing the network to train using labeled data as in the supervised setting,
- biasing the network to learn similar hidden representations for neighboring nodes on a graph, in the same vein as label propagation.

Neural Graph Machines





An example of a graph and feature inputs. There are two labeled nodes (xi, xj) and one unlabeled node (xk) and two edges. The feature vectors are used as neural network inputs.
The second figure illustrates a NGM on a CNN.
The training flow ensures the neural net to make accurate node-level predictions and biases the hidden representations of neighbouring nodes to be similar.

Objective Function

$$\mathcal{C}_{\text{NN}}(\theta) = \sum_n c(g_\theta(x_n), y_n),$$

Generic Neural Network Cost Function
Where g represents the mapping, parametrized by θ, of the inputs, and c is the cost function (typically l-2 for regression, cross-entropy for classification)

Cost Function for Label Propagation,
By optimizing this function we find the optimal soft label distribution $\hat{Y}$ for each node. The function encourages that:

1. the label distribution of seed nodes should be close to the ground truth
2. the label distribution of neighbouring nodes should be similar
3. if relevant, the label distribution should stay close to our prior belief (U - prior distribution)

$$\begin{aligned}\mathcal{C}_{\text{LP}}(\hat{Y}) = \mu_1 \sum_{v \in V_l} \left\| \hat{Y}_v - Y_v \right\|_2^2 \\ + \mu_2 \sum_{v \in V, u \in \mathcal{N}(v)} w_{u,v} \left\| \hat{Y}_v - \hat{Y}_u \right\|_2^2 \\ + \mu_3 \sum_{v \in V} \left\| \hat{Y}_v - U \right\|_2^2,\end{aligned}$$

Objective Function (2)

$$
\begin{aligned}
\mathcal{C}_{\mathrm{NGM}}(\theta) = & \sum_{n=1}^{V_l} c(g_\theta(x_n), y_n) \\
& + \alpha_1 \sum_{(u,v)\in\mathcal{E}_{LL}} w_{uv} d(h_\theta(x_u), h_\theta(x_v)) \\
& + \alpha_2 \sum_{(u,v)\in\mathcal{E}_{LU}} w_{uv} d(h_\theta(x_u), h_\theta(x_v)) \\
& + \alpha_3 \sum_{(u,v)\in\mathcal{E}_{UU}} w_{uv} d(h_\theta(x_u), h_\theta(x_v)),
\end{aligned}
$$

The proposed objective function is a
weighted
sum of the neural network cost and the
label propagation.

Training instances (either labeled or unlabeled) that are connected in a graph should have similar predictions. This can be done by encouraging neighboring data points to have a similar hidden representation learnt by a neural network, resulting in a modified objective function for training neural network architectures using both labeled and unlabeled datapoints.

We call architectures trained using this objective Neural Graph Machines

Dataset

The dataset used in the project is the Large Movie Review Dataset, from

Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan and Ng, Andrew Y. and Potts, Christopher - Learning Word Vectors for Sentiment Analysis - 2011

The dataset is ideal for our experiments, since it is split into:

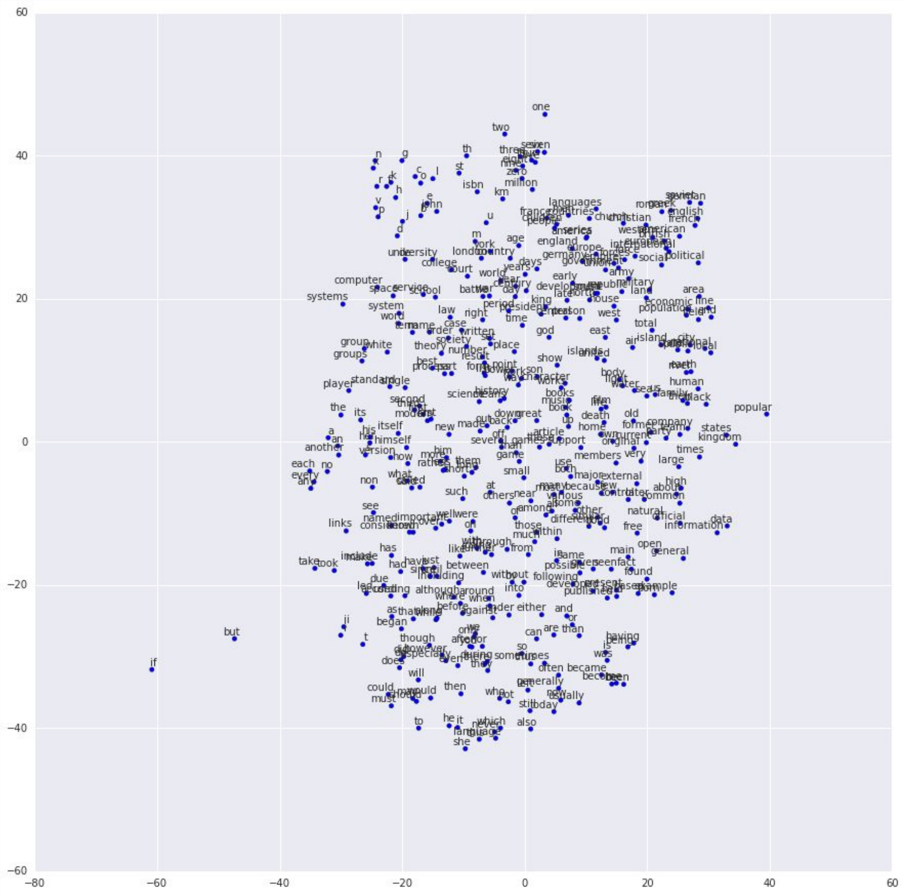
- 25000 samples for supervised training
- 25000 samples for testing
- 50000 unlabeled samples to exploit with the graph

Graph Construction - Word2Vec

With Word2Vec we are able to represent a word as a vector of floating point numbers that represent its meaning.

In our model the output vector has 300 features

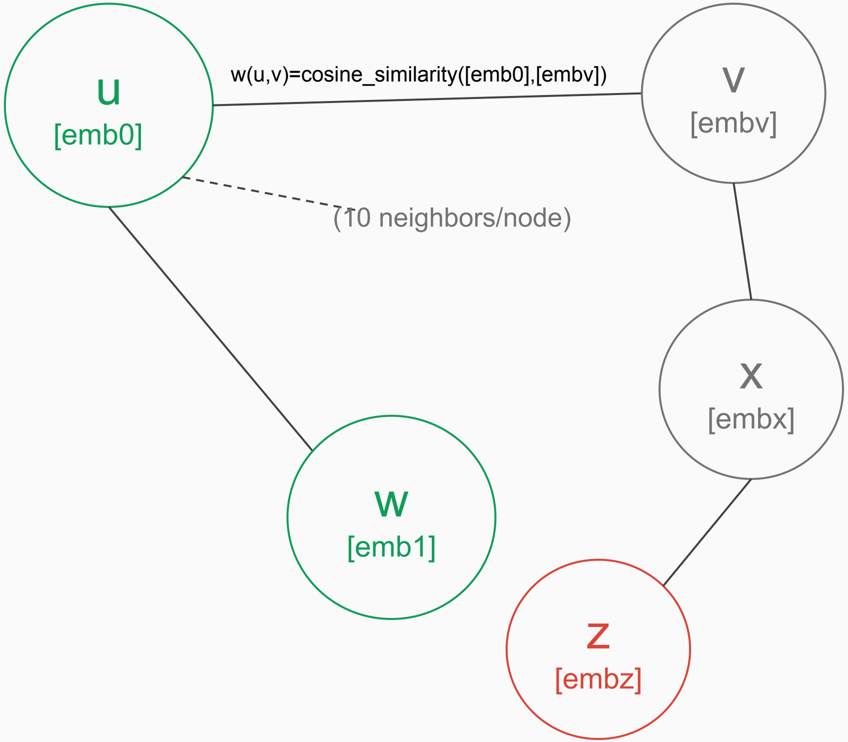
The embedding of a moview review is computed as the average of the embedding of all the words.



Graph Construction - Distance

The generated data structures are:

- Graph: dict: node -> [nodes]
- Edges_weight: dict: (u,v) -> w(u,v)
- E_{LL}
- E_{LU}
- E_{LL}
- Indices_dict: dict: u -> review_u.txt



The Model - Character Level Conv-NN

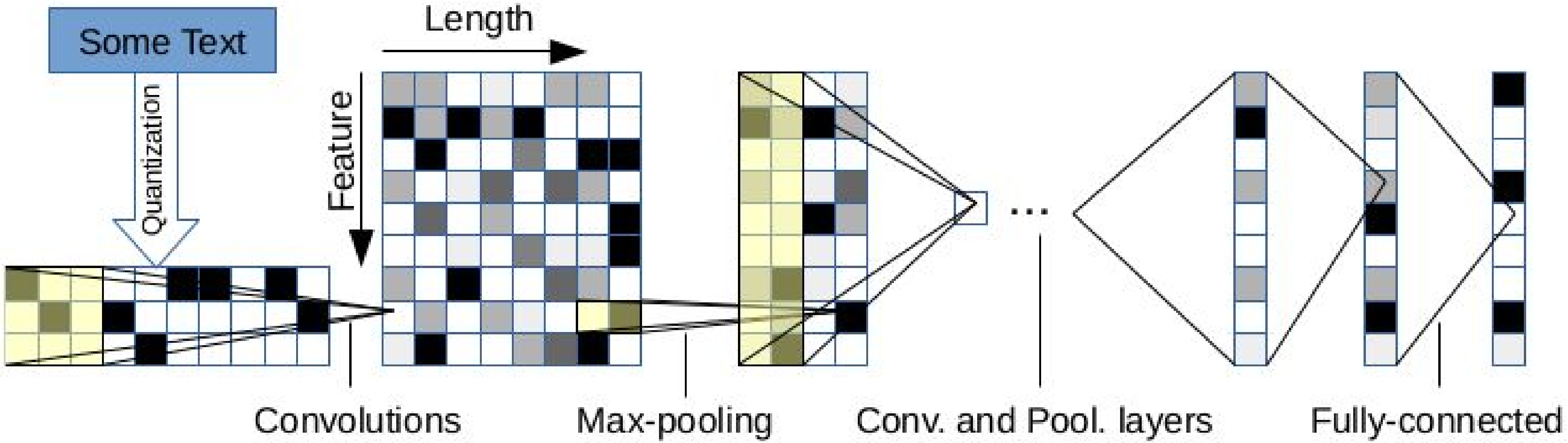


Figure 1: Illustration of our model

The Model - Inputs

The Character-Level ConvNN obviously can’t process text data in raw form.

The model accepts a sequence of encoded characters as input. The encoding is done by transforming a review in a one-hot encoding of the text of each character.

Max_length = 1014 characters
Feature ↔ Character in Alphabet

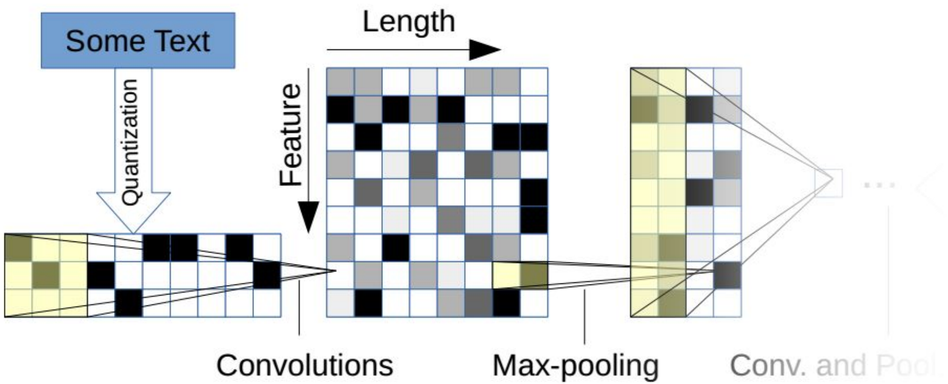
In order to reduce batch generation time, I pre-encoded all the inputs and created an iterator.

Ref. [Zhang 2015, Character-level Convolutional Networks for Text Classification]

Alphabet of 70 possible chars

```
abcdefghijklmnopqrstuvwxyz0123456789
-,;.!?:'''/\|_@#$%^&*~`+-=<>()[]{}
```

Illustration of input layers



The Model - Prediction

Much of the effort of the project went into creating the actual layers of the neural network and defining a function that returns the current scores for one input. That is:

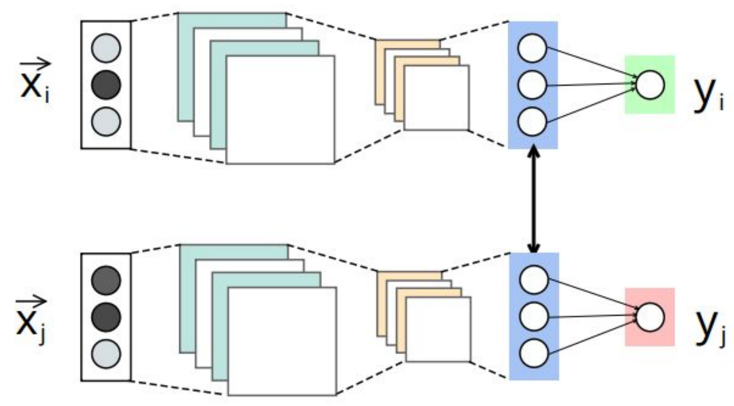
g: X $\rightarrow R^2$ [P(x is pos), P(x is neg)]

Key contribution: Danny Britz's blog. wildml.com

```
def g(input_x,num_classes=2, filter_sizes=(7, 7, 3), frame_size=32, num_hidden_units=256,
      num_quantized_chars=70, dropout_keep_prob=0.5):

    with tf.device('/cpu:0'):
        a = tf.one_hot(
            indices=input_x,
            depth=70,
            axis=1,
            dtype=tf.float32
        )

    a = tf.expand_dims(a, 3)

    # Convolutional Layer 1
    with tf.name_scope("conv-maxpool-1"):
        filter_shape = [num_quantized_chars, filter_sizes[0], 1, frame_size]
        W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.05), name="W")
        b = tf.Variable(tf.constant(0.1, shape=[frame_size]), name="b")
        conv = tf.nn.conv2d(a, W, strides=[1, 1, 1, 1], padding="VALID", name="conv1")
        h = tf.nn.relu(tf.nn.bias_add(conv,b),name="relu")
        pooled = tf.nn.max_pool(
```

Table 2. Settings of CNNs for the text classification experiment, including the number of convolutional layers and their sizes. The baseline model is the *small CNN* from (Zhang et al., 2015) and is significantly larger than our model.

Setting	Baseline	Our "tiny CNN"
# of conv. layers	6	3
Frame size in conv. layers	256	32
# of FC layers	3	3
Hidden units in FC layers	1024	256



Training

$$
\begin{aligned}
\mathcal{C}_{\mathrm{NGM}}(\theta) = & \sum_{(u,v)\in\mathcal{E}_{LL}} \alpha_1 w_{uv} d(h_\theta(x_u), h_\theta(x_v)) + c_{uv} \\
& + \sum_{(u,v)\in\mathcal{E}_{LU}} \alpha_2 w_{uv} d(h_\theta(x_u), h_\theta(x_v)) + c_u \\
& + \sum_{(u,v)\in\mathcal{E}_{UU}} \alpha_3 w_{uv} d(h_\theta(x_u), h_\theta(x_v)), \qquad (4)
\end{aligned}
$$

where

$$
\begin{aligned}
c_{uv} &= \frac{1}{|u|} c(g_\theta(x_u), y_u) + \frac{1}{|v|} c(g_\theta(x_v), y_v) \\
c_u &= \frac{1}{|u|} c(g_\theta(x_u), y_u),
\end{aligned}
$$

The paper proposes an alternative, optimized Cost/Loss Function that is better suited for training.
In particular this approach allows us to train the network using mini batches of 128 edges.

Given the Embeddings Graph G=(V,E), the time complexity for each epoch of the training is O(M) where M = |E|

```
pred_u1 = g(in_u1)
pred_v1 = g(in_v1)
pred_u2 = g(in_u2)

loss_function = tf.reduce_sum(alpha1 * weights_ll * tf.nn.softmax_cross_entropy_with_logits(logits=pred_u1, labels=pred_v1) \
                + cu1 * tf.nn.softmax_cross_entropy_with_logits(logits=pred_u1, labels=labels_u1) \
                + cv1 * tf.nn.softmax_cross_entropy_with_logits(logits=pred_v1, labels=labels_v1)) \
                + tf.reduce_sum(alpha2 * weights_lu * tf.nn.softmax_cross_entropy_with_logits(logits=pred_u2, labels=g(in_v2)) \
                + cu2 * tf.nn.softmax_cross_entropy_with_logits(logits=pred_u2, labels=labels_u2)) \
                + tf.reduce_sum(alpha3 * weights_uu * tf.nn.softmax_cross_entropy_with_logits(logits=g(in_u3), labels=g(in_v3)))

optimizer = tf.train.AdamOptimizer().minimize(loss_function)
```

A caveat of this approach is that it requires that each batch is split in an appropriate set of tensors of different size.

Also note how the loss function depends of the output layer values of multiple inputs at a time.
≠usual Supervised Learning approach

Optimized Loss Function

g(x): predicted distribution of sample x

|u| : number of incident edges of node u

L(x,y) : cross-entropy between distributions x and y

y_u : correct labels for node u

$$\alpha_1 * \sum_{(u,v)\in\_E\_LL} L(g(u),g(v)) + 1/|u| * L(g(u), y_u) + 1/|v| * L(g(v), y_v) +$$

$$+ \ \alpha_2 * \sum_{(u,v)\in E\_LU} L(g(u),g(v)) + 1/|u| * L(g(u), y_u) +$$

$$+ \alpha_3 * \sum_{(u,v)\in E\_UU} L(g(u),g(v))$$

I experimented with $\alpha_1 = 0.1$ $\alpha_2 = 0.1$ $\alpha_3 = 0.5$, increasing them over 0.2 (max value of 1/|u|) keeps the network in a training loop

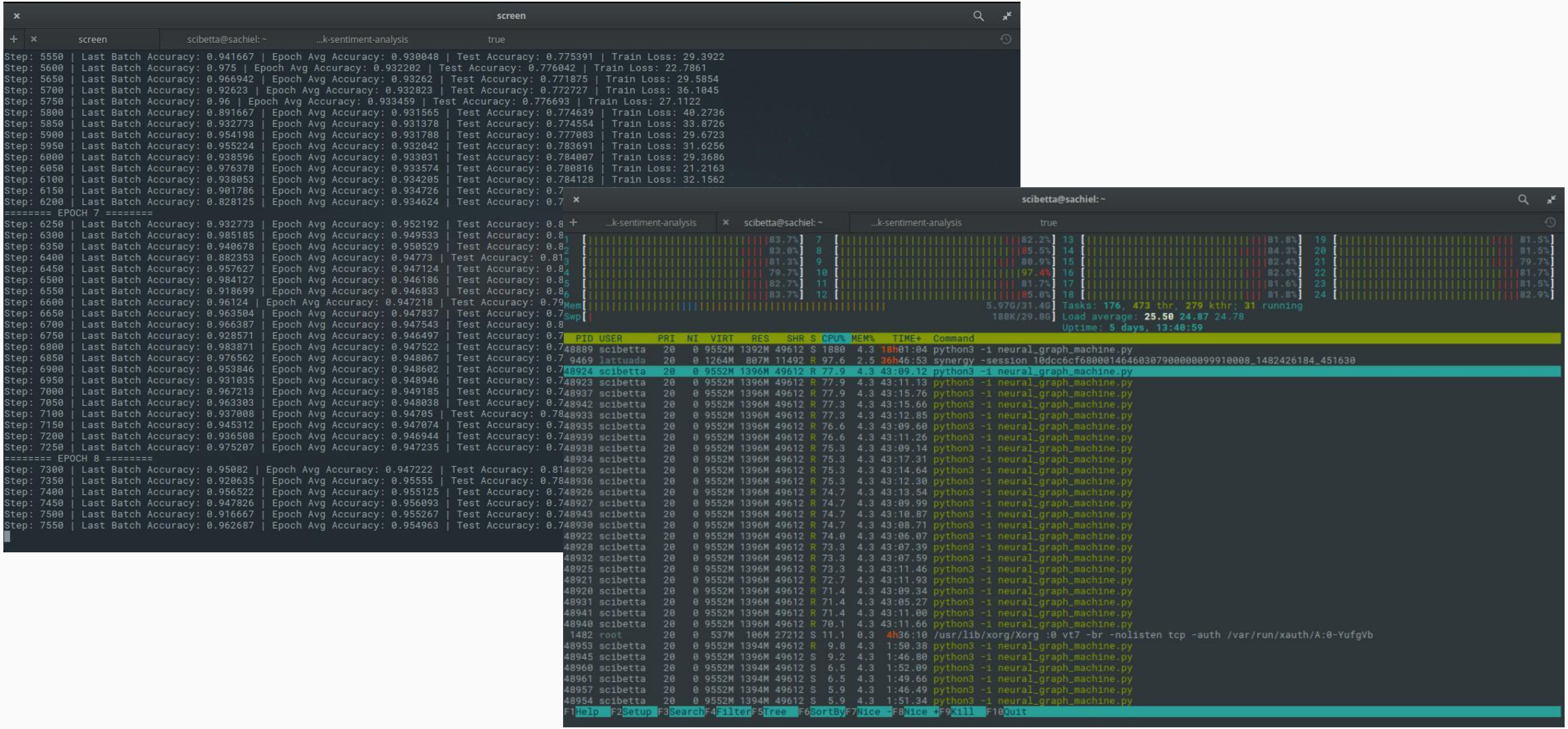
Meaning of hyperparameters

α_1: Hint from unsupervised learning for speed-up

α_2: Prediction of unlabeled nodes should be similar to the labeled prediction (which will eventually be correct)

α_3: Prediction of unlabeled nodes should be similar to the unlabeled nodes prediction, since they probably are both positive since they are similar, and also they were corrected in the α_2 term

After a few exciting sleepless nights staring at this...



Results

We compare the Accuracy of a Char-Level Conv NN trained using a classic Supervised Learning approach and an alternative Semi-Supervised “Neural Graph Machine” loss function. The training of the model without the graph shows lower results on an Accuracy metric.

Average Accuracy On Test Set	
Small-Char-CNN + NGM	Small-Char-CNN
82.16%	77.50%

But, more importantly in my opinion, by playing around with it, the NGM shows much better performance in classifying text from any user input than the original network. This is most likely due to the fact that NGM is trained on much more data, thus it is able to generalize on more varied input.

How it could be improved - in order of importance

- Tweak hyperparameters. Unfortunately the ones used by the publishers were not reported in the paper. To compute the ideal contributions, a cross-validation technique should be implemented, but it would have proved to be too computationally complex to handle, even for Sachiel. (It requires training the network 10x times)
- Improve performance on Unsupervised Learning approach, i.e. the one that is used to build the Graph.
  - For example, better data preprocessing, using word lemmatization, using more sophisticated Doc2Vec algorithms to compute embedding of a review.
- Increase the number of convolutional layers and of neurons in fully connected layers. This decision would be supported by the larger amount of input data provided by the graph. But again, it would require weeks to be trained.

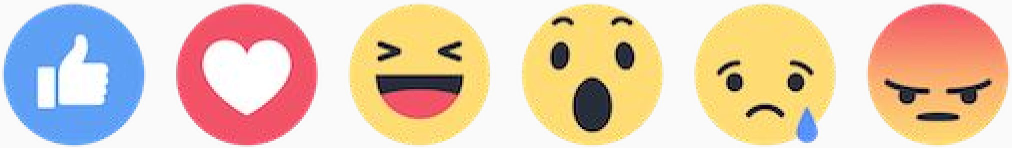
Closing remarks

The result is remarkable, since the technique shows performance comparable to Bag-Of-Words technique or Word Embedding, usually used in the task of text classification, but without the need to supply the model with a corpus of words associated with a sentiment, or in general with one of the classes.

The model learns by itself to find the discriminating constructs in the input text to classify them, (automatically learns words) and is resistant to misspelling.

Future Development

This project could prove very useful in my chatbot experiments. By extracting public data from Facebook, and leveraging on the number of “reactions” of the posts, I could enable my chatbot to ‘react’ emotionally to user input, without the need to use carefully constructed corpora!



End

All the code, including the trained model and a demo is freely available on my GitHub:

https://github.com/gssci/neural-graph-machine-sentiment-analysis

Now let's go to the **demo**.