Deep Learning for Natural Language Processing

Assignement 1, Stanford NLP course CS 224d

Benjamin Muller

1 Softmax

a

The softmax function is defined as follows :

$$softmax(x) = \left(\frac{e^{x_i}}{\sum_j e^{x_i}} \right)_{i \in 0..d-1} \text{ for any } x \in R^d$$

We indexed vectors that will actually be row-vector in the code from 0 to d-1 in order to be consistent with python

Therefore, let c be a vector in R^d equal to the constant c_0 at each of its component :

$$softmax(x+c) = \frac{e^{x_i + c_0}}{\sum_j e^{x_i + c_0}} = \frac{e^{x_i} e^{c_0}}{\sum_j e^{x_i} e^{c_0}} = \frac{e^{x_i}}{\sum_j e^{x_i}} = softmax(x)$$

2 Neural Network Basics

a

The sigmoid is given by :

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Here x is a real number. We get easily :

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

In our code, we'll consider the sigmoid as a function from R^d to R^d applying the above transformation element-wise

The idea of a basic *FeedForward Neural Network* with a single hidden layer is the following. We input row-observations to our first and unique hidden layer. The hidden layer compute successively a linear transformation and a non-linear transformation. In our case the non-linear transformaton will be an element-wise sigmoid. Then the output of the hidden layer is transformed with a new linear transformation and a softmax.
The process I just described is called a *forward pass.* At this step, an error measure is computed comparing the prediction to the observed target data. In our case, we'll use as usually for classification, a cross entropy loss function.
Given this forward pass that lead to a prediction, we're going to compute a *backward* pass which consists in updating all the parameters of the network (that we call *weights*) in order to reduce the predicted error. In this purpose, we have to compute the gradient of our error function in regard to each single weight of the network.

b

Let's first compute the gradient of the error function regarding the input of the softmax that we denote θ.
We assume that the observed target vector has the form of a one-hot vector. Let's o be the index for which the element

of the target vector is one. Our cross-entropy error is then :

$$CE(y, \hat{y}) = -log(\hat{y}_o) = -log(softmax(\theta)_o) = -log(\frac{e^{\theta_o}}{\sum_i e^{\theta_i}})$$

Therefore, the gradient regarding the component o :

$$\nabla_{\theta_o} CE(y, \hat{y}) = -\frac{softmax(\theta)_o(1 - softmax(\theta)_o)}{softmax(\theta)_o} = \hat{y}_o - 1$$

And the gradient regarding all other components

$$\nabla_{\theta_i} CE(y, \hat{y}) = -\frac{softmax(\theta)_o(-softmax(\theta)_i)}{softmax(\theta)_o} = \hat{y}_i \text{ for any } i \neq o$$

We can be more concise, as y is a one hot vector, we get a vectorial expression :

$$\nabla_\theta CE(y, \hat{y}) = \hat{y} - y$$

c

We compute now the gradient of the cross entropy error regarding the input denoted $x \in R^{D_x}$ of our 1 hidden layer Feed Forward Neural Network with sigmoid hidden activation and softmax output.
We recall the two equation linking our input x and output ŷ. We also recall that we consider row-vectors in order to be consistent with the code representation.

$$h = \sigma\,\underbrace{(xW_1 + b_1)}_{a_1}\ ,\ \hat{y} = softmax\,\underbrace{(hW_2 + b_2)}_{a_2} \text{ and } J = CE(y, \hat{y})$$

with $x \in R^{D_x}$ the row-input vector, $W_1 \in R^{D_x H}$ and $W_2 \in R^{H D_y}$ the weights, $b_1, b_2 \in R$ the biases of each layer, $h \in R^H$ and $\hat{y} \in R^{D_y}$ our output.
Therefore, by applying the chain rule we get,

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial a_2}\frac{\partial a_2}{\partial h}\frac{\partial h}{\partial a_1}\frac{\partial a_1}{\partial x}$$

By using the results we prooved in section 1, and the expression of a_1 and a_2 we get :

$$\frac{\partial J}{\partial a_2} = \hat{y} - y$$

$$\frac{\partial a_2}{\partial h} = W_2'$$

$$\frac{\partial h}{a_1} = \sigma(a_1)(1 - \sigma(a_1))$$

$$\frac{\partial a_1}{\partial x} = W_1'$$

d

The parameters of the network are the weights and the biases. Then our network has (D$_x$+1)$.H+(H+1).D_y parameters.$

3 Word2Vec

Let's now dig into the word2vec model. The idea of the word2vec is to learn the best vectorial representation of words by modelling in some way and according to this representation the probability of getting a word conditionned on other words.
For instance, the skip-gram, conditionnaly to a centered word, model the probability of getting each word in its context (i.e included in a given interval) as a softmax of the center word dot the entire vocabulary.

As a such definition is very heavy computionnaly, because it implies to compute over the entire vocabulary for each single probability, we'll intoduce a simplification called *negative sampling.*
We also implement a *Continous Bag of Word* (CBOW) model that reverse the center word - context words relationship. These models belong to the paradigm of *Neural Language Model* (NLM) as it can be seen as Feed Forward Neural Networks. More precisely, NLM leverage the efficiency in terms of computational complexity and modeling power, of Neural Networks in order to extract the best representation for the language. Skip-Gram and CBOW are two very successful examples of Neural Language Models.

a

Given one output word indexed by o (in the one-hot encoding) and embedded by u_o, and one center word indexed by c and embedded by v_c, we define the probability as follows :

$$\hat{y}_o = p(o/c) = \frac{e^{u_o^T v_c}}{\sum_{w=0}^{W-1} e^{u_w^T v_c}}$$

u_w will be named as "output" vectors as they correspond to outputs in the neural representation, whereas v_c will be named as "input", predicted and center word vector as it represents the input in the neural representation

As in the previous section, error is measured using the cross-entropy. Hence, we define the error of the predicted output vector $\hat{y}_o$ regarding the true observation y_o (one hot encoded) as

$$J_{softmax-CE}(o, v_c) = CE(y_o, \hat{y}_o) = -log(p(o/c))$$

Therefore, we get :

$$\nabla_{v_c} J_{softmax-CE}(o, v_c) = -u_o + \frac{\sum_{w=0}^{W-1} e^{u_w^T v_c} u_w}{\sum_{w=0}^{W-1} e^{u_w^T v_c}}$$

We recall that in order to be consistent with the code, each written vector here is actually a row-vector

b

Then we can compute the gradient regarding output vector u_w.

$$\nabla_{u_w} J_{softmax-CE}(o, v_c) = \frac{e^{u_w^T v_c}}{\sum_{w=0}^{W-1} e^{u_w^T v_c}} v_c \text{ for } w \neq o$$

and

$$\nabla_{u_o} J_{softmax-CE}(o, v_c) = (\frac{e^{u_w^T v_c} v_c}{\sum_{w=0}^{W-1} e^{u_w^T v_c}} - 1) v_c$$

c

The main issue about using softmax as the output layer is it requires to iterate about the entire vocabulary. This step is computationnaly very heavy. In order to overcome this problem, a solution is to use *Negative Sampling.*
Negative sampling defines the loss function as follows : We assume to have drawn K words distinct from our output word o (for simpicity we index them by 1,..,K) we defined the loss as

$$J_{neg-sample}(o, v_c) = -log(\sigma(u_o^T v_c)) - \sum_{k=1}^{K} log(\sigma(-u_k^T v_c))$$

We compute as for the softmax the gradient regarding u_w and v_c :

$$\nabla_{v_c} J_{neg-sample}(o, v_c) = -(1 - \sigma(u_o^T v_c)) u_o + \sum_{k=1}^{K} (1 - \sigma(-u_k^T v_c)) u_k$$

and

$$\nabla_{u_w} J_{neg-sample}(o, v_c) = (1 - \sigma(u_w^T v_c))v_c \text{ for } w \neq o$$
$$\nabla_{u_o} J_{neg-sample}(o, v_c) = -(1 - \sigma(u_o^T v_c))v_c$$

In terms of run time, both for forward and backward pass, if we count the number of exponential function to compute for both solutions (as in both solutions the computation within the exponential has the same weight as it's dense vector of same dimension) we get :

$$\text{Speed-up-ratio}_{forward+backward}(\frac{\text{neg-sample}}{\text{softmax}}) \text{ at least } 2\frac{W+1}{K+1}$$

d

The skip gram and continous model consist in a specific sum of the losses function that we defined above.
The skip-gram model defines the loss as follows :

$$J_{skip-gram} = \sum_{-m \leq j_{\neq 0} \leq m} F(w_{c+j}, v_c)$$

As we computed in both cases (CE-softmax and Negative Sampling) the gradient for any word-vector we can write that :

$$\nabla_w J_{skip-gram} = \sum_{-m \leq j_{\neq 0} \leq m} \nabla_w F(w_{c+j}, v_c) \text{ . for any w in the vocabulary reusing (a),(b) and (c)}$$

$$\nabla_{v_c} J_{skip-gram} = \sum_{-m \leq j_{\neq 0} \leq m} \nabla_{v_c} F(w_{c+j}, v_c) \text{ reusing (a),(b) and (c)}$$

For any other word input v_j, the gradient is 0 as the loss doesn't depend on them.

For the CBOW model, we kind of reverse the skip gram model. Indeed, for the CBOW, we use the context words agregated by summing them to predict the center word whereas the skip gram consist in the opposite. Hence, the predicted vector is now (it replaces v_c):

$$\hat{v} = \sum_{-m \leq j_{\neq 0} \leq m} v_{c+j}$$

The loss is defined as follows

$$J_{CBOW}(w_{c-m...c+m}) = F(w_c, \hat{v})$$

As F is whether the Negative Sampling loss or the softmax-CE loss, we adapt the computation of the gradient that we made above and we get :

$$\nabla_w J_{CBOW}(w_{c-m...c+m}) = \nabla_w F(w_c, \hat{v}) \text{ for any output w in the vocabulary}$$

$$\nabla_{v_i} J_{CBOW}(w_{c-m...c+m}) = \nabla_{v_i} F(w_c, \hat{v}) \text{ for any input } v_i \text{ with } i \in \{\text{-m,..m}\}\text{-}\{0\}$$

And the gradient is equal to 0 for any other input vector.

g

Skip-gram model

We followed the assignement hyperparameters advices. We chose $K = 10$ for the negative sampling, a context size of $C = 5$, an embedded space of 10 dimensions and we ran 40000 iterations for training. We visualize the embedded space after a straight *Singular Value Decomposition*. We get figure 1. We notice a few relevant facts about this projected embedded space.

- The prepositions "a", "the" are located together far from other words and close to very usual punctuation as "," and "." .
- "good" and "bad" are close to each other
- Names like "Trump" and "Goldman" are close to each other

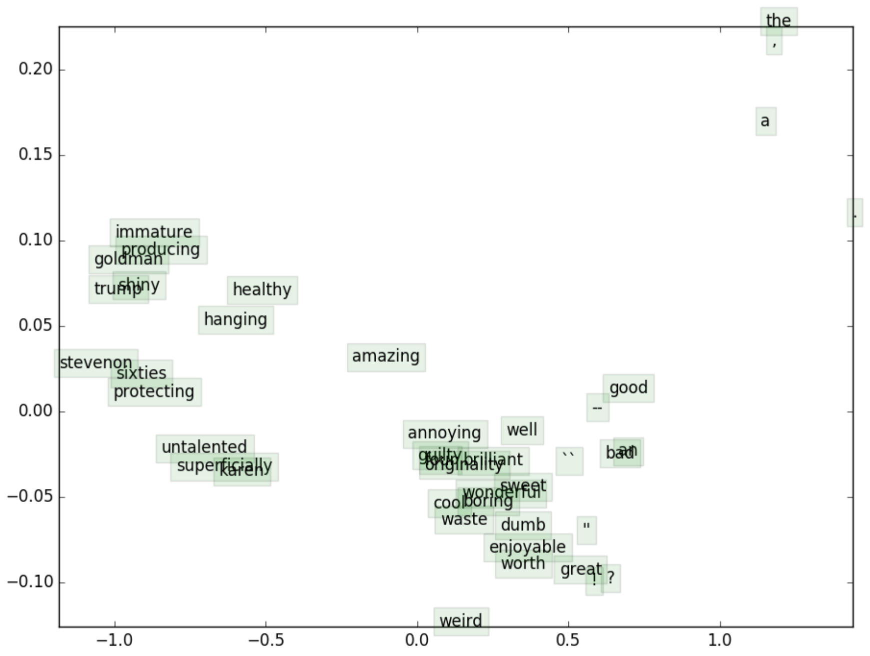

Figure 1: Skip-gram embedded space after SVD

Of course, as this visualization is only a few elements of a 2 dimensions projections of a 10 dimensions space , making sense of a such restricted view is a tricky job.
The task that is usually tested on a such embedded space in order to analysis its relevancy in its full complexity is the so-called *analogical reasoning* task which test the linear combination of words and compare them to what a human would expect. This is not the purpose of this project so we won't test it here.

Continous Bag of Words

As for the Skip-gram model, we plot the embedded space after doing a singular value decomposition we get the Figure 2. The hyperparameters choices are the same than the skip-gram. In the same way as for the Skip-gram, interpreting a such plot is tricky.
We can notice some invariance with the skip gram model, for example in the way the model treated tokens like "the", "a", "," , "." which is reassuiring.

4 Sentiment Analysis

We are now going to use the embedded word-vectors to build a supervised sentiment analysis model. We are going to train a softmax regression in order to predict a sentiment level caracterized by 5 level (from 0 to 4).

b

Regularization, by adding a term in the objective that takes in account a measure of the complexity of the parameters (here a norm 2) allows to reduce overfitting and perform some kind of features selection. Indeed this term in the objective controls the complexity of the weight from being to high and then constraint the parameters from fitting too much the observations.

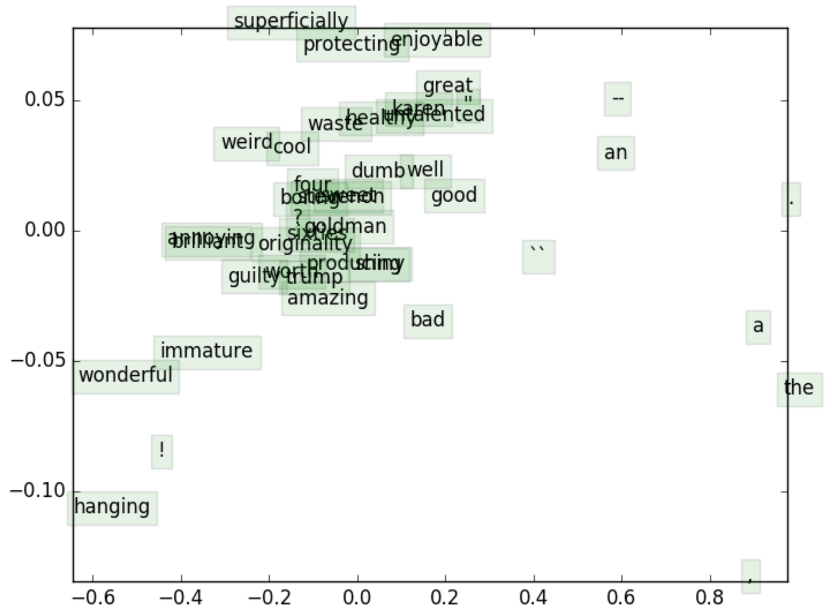

Figure 2: CBOW embedded space after SVD

c

For both CBOW and Skip-gram, we did a straight grid search for finding the best regularization parameter of our softmax regression (from 10^{-5} to 10^{-1}). We selected the following final models :

Result				
	$\|\|.\|\|_2$ Regularization	Train accuracy	Dev accuracy	Test accuracy
Skip-Gram	10^{-4}	29.11%	29.28%	27.21
CBOW	$1/2 * 10^{-4}$	29.46%	29.23%	26.87%

d

For the Skip-gram model, the grid search is plotted on figure 3 and on figure 4 for the CBOW model.

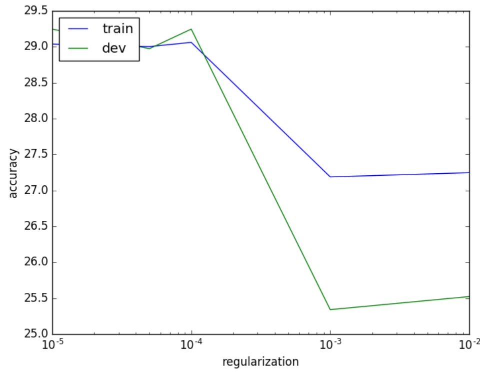

Figure 3: Regularization Search - Skip-Gram model

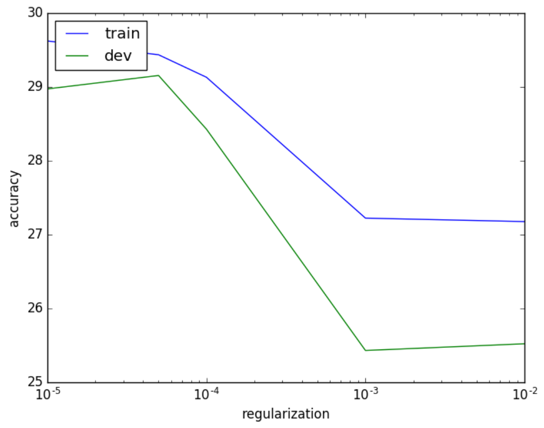

Figure 4: Regularization Search - CBOW model