HW3: (Neural) Language Modeling

Nicolas Drizard
nicolasdrizard@g.harvard.edu

Virgile Audi
vaudi@g.harvard.edu

March 11, 2016

1 Introduction

This assignment focuses on the task of language modeling, a crucial first-step for many natural language applications. In this report, we will present several count-based multinomial language models with different smoothing methods, an influential neural network based language model from the work of Bengio et al. (2003), and an extension to this language model which learns using noise contrastive estimation, as well as their implementation using Torch. We found this homework more challenging than the previous ones and encountered significant challenges that we will underline in this report.

2 Problem Description

The goal of the language models presented in this report is to learn a distributed representation for words as well as probability distribution for word sequences. Language models are usually represented as the probability of generating a new word conditioned on the preceeding words:

$$P(\boldsymbol{w}_{1:n}) = \prod_{i=1}^{n-1} P(w_{i+1}|w_i)$$

To simplify the analysis, it is common to make the assumption that a word is influenced only by the N words immediately preceeding it, which we call the context. Even with reasonably small values for N, building such models are extremely expensive computationally-wise as well as time-consuming if not ran on GPU. The joint proabibility of a sequence of 6 words taken from a vocabulary of 10 000 words could possibly imply training the model to fit up to $10^{4^6} - 1 = 10^{24} - 1$ parameters.

The objective of this homework was to predict a probability distribution over 50 words at a given place in the sentence and based on the previous words. To do so, we tried implementing N-grams models in an efficient manner. Due to computational limitations (no access to GPUs...), we faced difficulties training the Neural Network and focused our efforts on building strong count-based models to solve the problem of language modeling.

3 Model and Algorithms

We will now dive into more details of two types of models, i.e. count-based models and neural network models.

3.1 Count-based Models

3.2 Neural Networ Models

3.2.1 Regular Models

As in the neural networks build for previous homeworks, the model has for input a window of words preceding the wanted predicted word. It first convert the words in the window of size d_{win} my mapping them into a geometrical space of higher dimension d_{in} (30 in Bengio's paper). It then concatenates the words embeddings into a vector of size $d_{in} \times d_{win}$. This has for advantage of adding information about the position of the words in the window, as opposed to making a bag-of-words assumption. The higher dimensional representation of the window is then fed into a first linear model followed by a hyperbolic tangent layer to extract non- linear features. A second linear layer is then applied followed by a softmax to get a probability distribution over the vocabulary. We then train the model using a Negative Log-Likelihood criterion and stochastic gradient descent.

We can summarize the model in the following formula:

$$nnlm_1(\boldsymbol{x}) = \tanh(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})\boldsymbol{W'} + \boldsymbol{b'}$$

where we recall that:

- $\boldsymbol{x} \in \Re^{d_{in} \cdot d_{win}}$ is the concatenated word embeddings
- $\boldsymbol{W} \in \Re^{(d_{in} \cdot d_{win}) \times d_{hid}}$, and $\boldsymbol{b} \in \Re^{d_{hid}}$
- $\boldsymbol{W'} \in \Re^{d_{hid} \times |\mathcal{V}|}$, and $\boldsymbol{b'} \in \Re^{|\mathcal{V}|}$, where $|\mathcal{V}|$ is the size of the vocabulary.

We give a diagram of the model to better illustrate it:

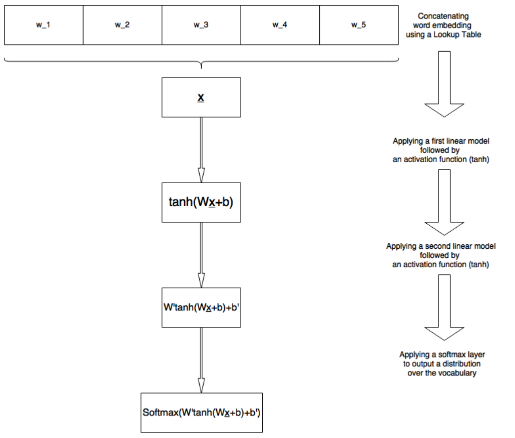

w_1
w_2
w_3
w_4
w_5
Concatenating word embedding using a Lookup Table
x
tanh(Wx+b)
W'tanh(Wx+b)+b'
Softmax(W'tanh(Wx+b)+b')
Applying a first linear model followed by an activation function (tanh)
Applying a second linear model followed by an activation function (tanh)
Applying a softmax layer to output a distribution over the vocabulary

Figure 1: Neural Language Model (Bengio,2003)

We then implemented a variant of the model using a skip-layer that concatenates the output of the tanh layer again with the original embeddings. The updated formula for the model is:

$$nnlm_2(\boldsymbol{x}) = [\tanh(\boldsymbol{xW} + \boldsymbol{b}), \boldsymbol{x}]\boldsymbol{W'} + \boldsymbol{b'}$$

where this time:

- $\boldsymbol{W'} \in \Re^{(d_{hid}+d_{in}\cdot d_{win})\times|\mathcal{V}|}$, and $\boldsymbol{b'} \in \Re^{|\mathcal{V}|}$

The updated diagram is as follows:

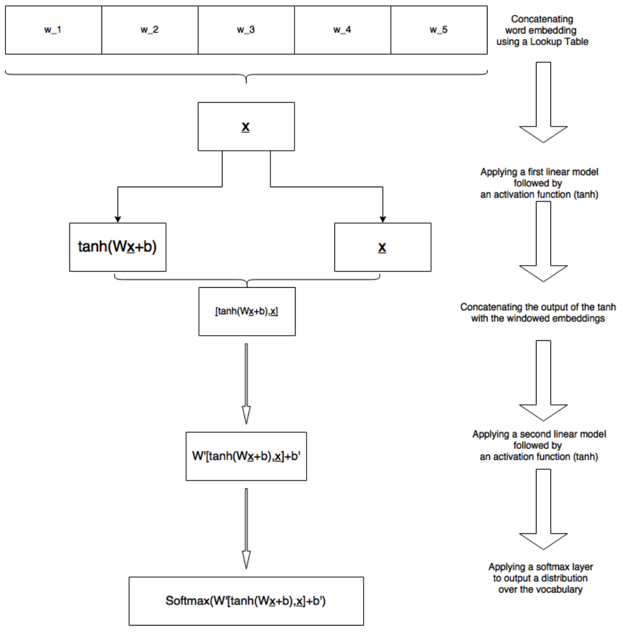

w_1
w_2
w_3
w_4
w_5
Concatenating word embedding using a Lookup Table
x
tanh(Wx+b)
x
[tanh(Wx+b),x]
W'[tanh(Wx+b),x]+b'
Softmax(W'[tanh(Wx+b),x]+b')
Applying a first linear model followed by an activation function (tanh)
Concatenating the output of the tanh with the windowed embeddings
Applying a second linear model followed by an activation function (tanh)
Applying a softmax layer to output a distribution over the vocabulary

Figure 2: Skip-Layer Model

We now show the pseudo code for training these NNLMs using batch stochastic gradient descent:

```
1: procedure $NNLM_i$($win_1, ..., win_n$,MaxEpoch,BatchSize,LearningRate)
2:     for epoch = 1,MaxEpoch do
3:         for batch = 1, |train|/BatchSize do
4:             for win in batch do
5:                 Call $NNLM_i$:forward(win)
6:                 Evalute the loss
7:                 Evaluate derivatives of the loss
8:                 Backprop through $NNLM_i$
9:             Update Parameters with LearningRate
```

3.2.2 Noise Contrastive Estimation

As mentioned earlier, training such model is extremely expensive in computation, especially on CPUs. The issue comes from the use of the softmax in the last layer of the model in order to obtain a distribution on a large vocabulary. In order to speed up the training time and reduce compution, we tried to implement NCE, which is a method used to fit unnomarlised method and therefore avoids using the last softmax layer.

The principle behind NCE is to sample for every context in the training data K wrong words that do not appear next in the windows using a noise distribution such a multinomial of the vocabulary simplex. We then apply a log-bilinear model. For a given context $\boldsymbol{x}$ and target y, the probability of y being a correct word for this context is given by:

$$p(D = 1|\boldsymbol{x}, \boldsymbol{y}) = \sigma(\log p(\boldsymbol{y}|D = 1, \boldsymbol{x}) - \log Kp(\boldsymbol{y}|D = 0, \boldsymbol{x}))$$

where

$$p(D = 1|\boldsymbol{x}, \boldsymbol{y}) = \sigma(logp(\boldsymbol{y}|D = 1, \boldsymbol{x}) - log(Kp(\boldsymbol{y}|D = 0, \boldsymbol{x})))$$

If the $p(y|D = 1, x)$ term still forces some normalisation in the model, thanks to the contribution of Mnih and Teh (2012), we can estimate the normalisation constant as a parameter in the model and even set to equal to 1. We can therefore replace this term by the score outpute by the linear model $z_{\boldsymbol{x}_i, w_i}$. The objective function that needs to be minimised becomes:

$$\mathcal{L}(\theta) = \sum_i \log \sigma(z_{\boldsymbol{x}_i, w_i}) - \log(Kp_{ML}(w_i))) + \sum_{k=1}^{K} \log(1 - \sigma(z_{\boldsymbol{x}_i, s_k}) - \log Kp_{ML}(s_k))$$

where p_{ML} is the pdf of the noise distribution, and s_k for $k \in \{1, ..., K\}$ are samples from the noise distribution.

4 Experiments

We now present the results of our experiments. We will first talk about the preprocessing and then continue with a comparison of the different models.

4.1 Data and Preprocessing

To complete this homework, we were given 3 datasets, one for training, one for validation and one for testing. The train set consisted of sentences with a total of over eight hundred thousands words from a vocabulary of ten thousands words. The validation set consisted of 70391 words. The particularity of the issue at hand consisted in the fact that we had to only predict a probability distribution over 50 words and not on the entire vocabulary. This is why we were provided the same validation set in the same format as for the test set. We could therefore predict 3370 words on the validation set to help us predict the 3361 words of the test set.

Most of the preprocessing was about building the N-grams for the different sets. We included in the preprocess.py file different functions to evaluate the windows as well as counting the different occurences of each N-grams. For instance, looking at 6-grams gave:

- 772 670 unique 6-grams on the training set,
- 887 522 6-grams in total,
- 70 391 6-grams on the validation set,
- 3 370 words to predict on the validation set,
- and 3 361 words of the test set

4.2 Evaluation

To evaluate the models, we will use the perplexity measure. For a set of m N-grams, $\boldsymbol{w}_1, ..., \boldsymbol{w}_m$, it is defined to be:

$$P(\boldsymbol{w}_1, ..., \boldsymbol{w}_m) = \exp\left(-\frac{1}{m}\sum_{i=1}^{m}\log P(w_N^i | w_{N-1}^i, ..., w_1^i)\right)$$

In other words, the perplexity translates how likely is the predicted word given the previous N-1 words. In this report, we will evaluated perplexity both on the entire vocabulary but also on the reduced 50 words to predict from. Values between these two "different" perplexities wil range from 3 to 1000.

4.3 Count-based Models

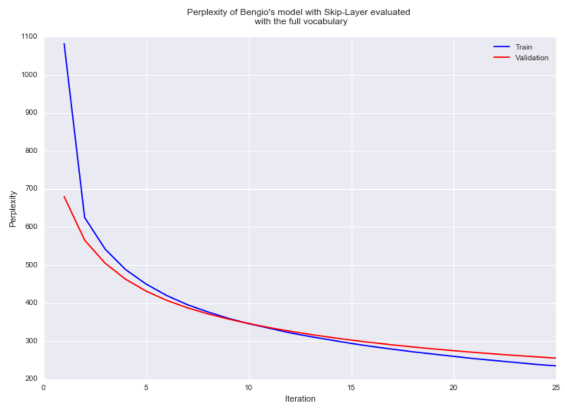
4.4 Neural Models

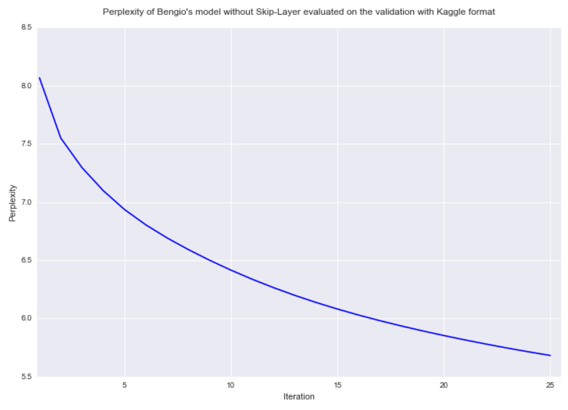
The main issue we faced with Bengio's model was training time. Even we managed to have a working model with a smaller vocabulary, we struggled at first to get a code that ran fast enough to experiment extensively with different paremetrisations. Our original code ran one epoch in about one hour for the paremetrisation. While trying to code the NCE approximation, we nevertheless managed to cut the training time to about 14-15 minutes.

We started to train the more simple neural network i.e. the one without the skip-layer, with the parametrisation suggested by Bengio:

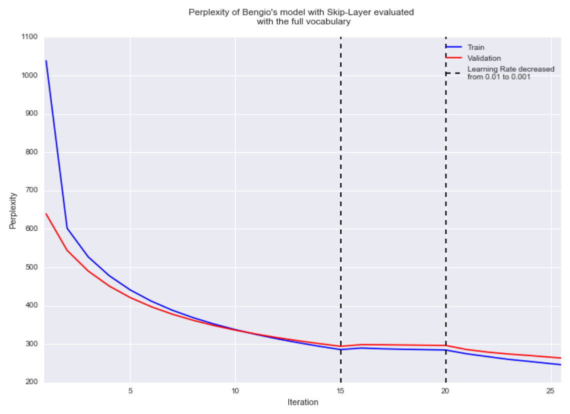
- Window size: $d_{win} = 5$
- Dimension of the embeddings: $d_{in} = 30$
- Hidden dimension: $d_{hid} = 100$

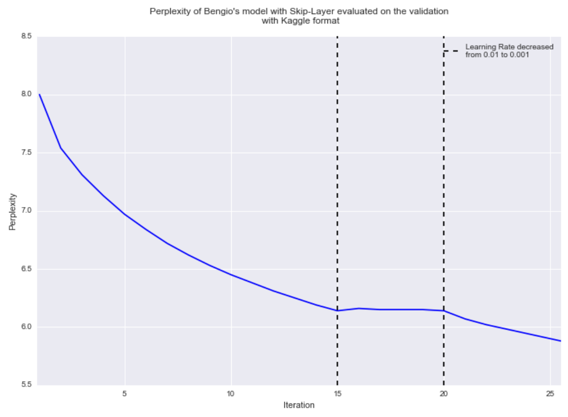
We summarize the results in the graphs below:





Before making a submission on kaggle, we decided to compare the encouraging results with the Skip-Layer model. We ran the experiment 5 epochs at a time to give us control on the learning rate. We thought that after the 15 epochs, the model was close to convergence, and decided to decrease the learning rate from 0.01 to 0.001. We obtain the following results:





As one can see, changing the learning rate negatively impacted the results. It plateaued but perplexity started decreasing again as soon as we re-up the learning rate. It also unclear how much improvement the skip-layer model brings compared to the original Bengio model, and this even without the change in learning rate. Based on these observation, we decided to submit to Kaggle the results of the simple model. We obtained:

$$Perp_{nnlm}^{test} = 5.47$$

Nevertheless, a final observation on the training of these NNLMs is that the models haven't seem to converge fully after 25 epochs. Running the algorithm for 5-10 extra epochs would most probably yielded better results and helped us reach the level of the count-base models.

4.5 NCE

We unfortunately did not succeed to implement a valid version of the Noise Contrastive Estimation. We did not managed to have a speed improvement and even worse observed that the perplexity on the validation was increasing instead of decreasing.

4.6 Mixtures of models

In order to increase our score, we decided to combine the differente approaches by averaging the results over the distributions outputted by various models.

5 Conclusion

End the write-up with a very short recap of the main experiments and the main results. Describe any challenges you may have faced, and what could have been improved in the model.

6 References

- Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:11371155.
- Mnih, A. and Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. *In Advances in Neural Information Processing Systems*, pages 22652273.
- Chen, S. and Goodman, J. (1998). An Empirical Study of Smo othing Techniques for Language. *Technical Report TR-10-98*, Computer Science Group, Harvard University.