DLI Teaching Kit

Lab 4B (designed for individual work)

---

1 Batch Normalization [10 credits]

Batch Normalization [3] is a technique for reducing internal covariate shift and accelerating training speed. Assume that we are implementing a BN module with d-dimensional input $x = (x^1, x^2, \cdots, x^d)$ and we choose a mini-batch of size m.

1. Write down the expression of mean and variance of scaler features. If we want transform them into zero mean and unit standard deviation, how do we normalize the feature?
2. In order to avoid changing the representation of each layer, we need to add a pair of learning parameters γ^k and β^k in BN module. Write down the output of BN in terms of γ^k and β^k. Let us assume that y is the output of BN and E is the energy, please provide the formula of $\frac{\partial E}{\partial \gamma^k}$ and $\frac{\partial E}{\partial \beta^k}$ in terms of $\frac{\partial E}{\partial y_i^k}$.

2 Convolution [10 credits]

Table 1 depicts two matrices. One of the form (5×5 one) represents an image. The second (3×3 one) represents a convolution kernel. (Consider the bias term to be zero)

1. How many values will be generated if we forward propagate the image over the given convolution kernel?
2. Calculate these values.
3. Suppose the gradient backpropagated from the layers above this layer is a 3×3 matrix of all 1s. Write down the value of the gradient (with respect to the input) backpropagated out of this year.

3	1	4	1	5
9	2	6	5	3
5	8	9	7	9
3	2	3	8	4
6	2	6	4	3

3	8	3
2	7	9
5	0	2

Table 1: Image Matrix (5×5) and convolution filter (3×3)

3 Variants of pooling [10 credits]

1. List three different kinds of pooling, and their corresponding module implemented in PyTorch.
2. Write down the mathematical forms of these three pooling modules.
3. Pick one of the pooling listed and describe the reason for incorporating it into a deep learning system.

4 t-SNE [10 credits]

Go through [12] and answer the following questions.

1. What is the crowding problem and how does t-SNE alleviate it? Give details.
2. The cost function of t-SNE is given by:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

   Where:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$
$$p_{ij} = \frac{exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_{k \neq l} exp(-\|x_k - x_l\|^2/2\sigma^2)}$$

   Please derive $\frac{\partial C}{\partial y_i}$ in detail.

5 Sentence Classification [20 credits]

Sentence Classification is a common problem in NLP. There are many ways to solve this problem. As simplest, you could just use machine learning algorithm such as Logistic regression, or any you can think of using such as a multilayer Perceptron (MLP). You could even take help from ConvNets or RNNs to give you good sentence vector, which you can feed into linear classification layer. Goal of this question is to make sure that you have a clear picture in your mind about all these possible techniques.

Let's say you have a corpus of 50K words. For the simplicity of this question, assume you have the trained word embedding matrix of size [50K*300] available with you, which can give you a word vector of size [1*300]. Consider one sentence having 10 words for the classification task and describe your approach for below techniques.

a. ConvNet

1. Design a one layer ConvNet which first maps the sentence to a vector of length 5 (with the help of convolution and pooling), then feeds this vector to fully connected layer with softmax to get the probability values for possible 3 classes.
2. Clearly mention the sizes for your input, kernal, outputs at each step (till you get the final [3*1] output vactor from softmax)
3. Please describe the effect of small filter size vs large filter size during the convolution? What would be your approach to select the filter sizes for classification task?

b. RNN

1. How can a simple RNN which is trained for language modeling be used to get the sentence vector?
2. Design a simple RNN which first maps the sentence to a vector of length 50 (with the help of convolution and pooling), then feeds this vector to fully connected layer with softmax to get the probability values for possible 4 classes.
3. Clearly mention the sizes of all the RNN components such as your input vector, hidden layer weight matrix, hidden state vecotrs, cell state vector, output layers (RNN components sizes would be same at each time stamp).

c. Extra credit question - 5 points

This is the open ended question. Come up with your own neural approach to classify a sentence. As an example maybe think of some combination of a CNN and RNN. Think about pros and cons of the individual CNN or RNN approach. Please describe your approach with a figure if possible.

6 Language Modeling [40 credits]

Word-level Recurrent Language Model

Language modeling is a fundamental task of natural language processing (NLP), and language models play an important role in many other tasks such as machine translation and sentence generation. A language model, in general, tries to predict the next word w_{t+1} given preceding words $w_1, \ldots, w_t$ at each time step t. The forward pass of a recurrent language model involves the following calculations.

Embedding Layer: The input word w_t is projected into a high-dimensional space by a word lookup table $\mathbf{E} \in \mathbb{R}^{|V| \times d}$, where $|V|$ is the vocabulary size and d is the dimension of a word vector:

$$\mathbf{e}_{w_t} = \mathbf{E}^\top \mathbf{w}_{w_t}, \tag{1}$$

where $\mathbf{w}_{w_t} \in \mathbb{R}^{|V|}$ is a one-hot vector whose i-th element is 1, and other elements are 0. In the implementation, we simply slice the word lookup table by the word index.

RNN Layer: Next we compute a memory states $\mathbf{h}_t$ given the word input $\mathbf{e}_{w_t}$ and a memory state from the previous time step $\mathbf{h}_{t-1}$:

$$\mathbf{h}_t = f(\mathbf{e}_{w_t}, \mathbf{h}_{t-1}), \tag{2}$$

where f is a recurrent function such as LSTM[2] and GRU[1]. When $t = 0$, $\mathbf{h}_0$ is usually a zero vector.

Prediction: The memory state $\mathbf{h}_t$ is affine-transformed followed by a softmax function:

$$\mathrm{P}\left(w_{t+1} = k | w_{<t+1}\right) = \frac{\exp\left(\mathbf{v}_k^\top \mathbf{h}_t + b_k\right)}{\sum_{k'} \exp\left(\mathbf{v}_{k'}^\top \mathbf{h}_t + b_{k'}\right)}, \tag{3}$$

where $\mathbf{v}_k$ is the k-th column of a parameter matrix $\mathbf{V} \in \mathbb{R}^{d \times |V|}$ and b_k is the k-th element of a bias vector $\mathbf{b} \in \mathbb{R}^d$.

In this section, you investigate and analyze recurrent language models. The starter code is available at `https://github.com/jakezhaojb/DSGA-1008-Spring2017-A2`.

a. Data

The characteristics of language models heavily rely on the training corpus. We want you to train your language models on several different datasets to see the differences.

- **Penn Treebank (PTB)** The starter code provides the Penn Treebank Corpus [10] preprocessed by Mikolov et al [11]. This text data consists of WSJ articles. For all the training, validation, and test sets, unknown words have been replaced by `<unk>`, and the vocabulary size is limited to 10k. Also, numbers have been replaced by `N`. Thus, there are no unknown words in validation/testing time.
- **Gutenberg Corpus** This dataset consists of texts written by 142 authors, from Charles Darwin to Isaac Asimov. A small subset of the corpus is available at in the starter code. This is constructed by randomly choosing 100k sentences from the original corpus. The full dataset is available at `http://web.eecs.umich.edu/~lahiri/gutenberg_dataset.html` [8]. Feel free to construct your own dataset (by authors, by time etc.) and run experiments. For example, Kim et al.[6] analyze change in language across time through language models.
- **Your choice** You are welcome to use any publicly available datasets.

		Train	Validation	Test
PTB	# Sentences	42k	3k	4k
	# Words	888k	70k	79k
Gutenberg	# Sentences	80k	10k	10k
	# Words	2.5m	315k	314k

Table 2: The size of each dataset.

Please note that you may want to modify `data.py` to control the vocabulary size and the words in the vocabulary. It is common to use N most frequent words as the vocabulary words. You should carefully choose N since the computational complexity of a softmax function grows with respect to the vocabulary size. In general, N =10k is used for small datasets.

b. Improve your model

We expect everyone to investigate key hyperparameters of a recurrent language model such as:

1. RNN cells (LSTM, GRU...),
2. different optimizers,
3. the number of RNN layers,
4. the dimension of word vectors,

5. the dimension of hidden states,
6. parameter initialization,
7. the vocabulary size,
8. shuffling the training data at the beginning of each epoch.

Note that training is recommended to be done on NVIDIA GPUs because CPUs would take a loner time. There are some papers [13, 5] discussing about tricks to train recurrent language models.

c. Word Embeddings

Once you have a trained model, you can obtain word embeddings form it. Word embeddings are stored in a word lookup table $\mathbf{E} \in \mathbb{R}^{|V| \times d}$, where $|V|$ is the vocabulary size and d is the dimension of a word vector. Please plot the word vectors in the 2D space using t-SNE [12]. You can use a package like this: `http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`.

d. Variants of Recurrent Language Model (Optional)

If you are ambitious, you could try to implement more complex language models. For example:

- C2W [9],
- Character-Aware Neural Language Model [7].

Models listed above utilize both word and character inputs, and predictions are still made at the word-level.

Write-up

Your write-up should include:

1. a description of your models,
2. a discussion of improvements (hyperparameter settings, optimization methods etc.),
3. empirical evaluation (perplexity),
4. qualitative analysis (generated samples, different training data, t-SNE plots etc.)

Code

Please write a script `rnnlm_result.py` that is able to execute on the PTB test data and print the perplexity. This should be same as the perplexity that you report in your write-up.

Submission

Send your submission (write-up, `rnnlm_result.py`) to your corresponding TA in a single zip folder named as "YourTeamName.zip" by the deadline. Include a link to the trained model file in the email. Please use the following title for your email.

`[`*CourseName* `YOUR_TEAM_NAME] Submission Lab4B`

References

[1] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[4] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[5] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. *Journal of Machine Learning Research*, 2015.

[6] Yoon Kim, Yi-I Chiu, Kentaro Hanaki, Darshan Hegde, and Slav Petrov. Temporal analysis of language through neural language models. *CoRR*, abs/1405.3515, 2014.

[7] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. *CoRR*, abs/1508.06615, 2015.

[8] Shibamouli Lahiri. Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 96–105, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

[9] Wang Ling, Tiago Luís, Luís Marujo, Rámon Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. EMNLP, 2015.

[10] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[11] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010.

[12] L.J.P van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9: 25792605, Nov 2008.

[13] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.