Programming Assignment 2: Caption Generation

Deadline: Monday, Feb. 20, at 11:59pm

Submission: You must submit two files through MarkUs[1]: a PDF file containing your writeup, titled `a2-writeup.pdf`, and your code file `mlbl.py`. Your writeup must be typeset using LaTeX.

The programming assignments are individual work. See the Course Information handout[2] for detailed policies.

You should attempt all questions for this assignment. Most of them can be answered at least partially even if you were unable to finish earlier questions. If you were unable to run the experiments, please discuss what outcomes you might hypothetically expect from the experiments. If you think your computational results are incorrect, please say so; that may help you get partial credit.

Setting Up

The first step is to install Python, autograd (`https://github.com/HIPS/autograd`), SciPy, NumPy and scikit-learn. Check out the websites of the course and relevant packages for more details.

Neural Language Model

In this assignment, we will train a multimodal log bilinear language model. In particular, we will deal with a dataset which contains data of two modalities, i.e., image and text. An instance of the dataset consists of an image and several associated sentences. Each sentence is a so-called *caption* of the image which describe its content. The overall goal of the neural language model is to generate the caption given an image. Note that a caption (sentence) is generated word by word conditioned on both the image and a fixed size context. The context of the word just means a fixed-size contiguous sequence of words ahead of it.

To understand the model, we first clarify the notations. An image is represented as a feature vector $\mathbf{x} \in \mathbb{R}^M$. A sentence is a sequence of words where each word w in the vocabulary is represented as a D-dimensional real valued vector $r_w \in \mathbb{R}^D$. Let $\mathbf{R}$ denote the $K \times D$ matrix of word representation vectors where K is the vocabulary size.

We now describe the multimodal log bilinear language model which is slightly different from the one described in section 4.1 of [4]. Given the image feature, we first predict the word representation as follows,

$$\hat{\mathbf{r}} = \left(\sum_{i=1}^{n-1} \mathbf{C}^{(i)} \mathbf{r}_{w_i} \right) + \mathbf{C}^{(m)} g\left(\mathbf{J}\mathbf{x} + h \right), \tag{1}$$

where n is the context size and is a hyperparameter of the model. $\mathbf{r}_{w_i}$ is the representation of i-th word, i.e., w_i-th row of matrix $\mathbf{R}$. $\mathbf{J}$ and h are weight matrix and bias vector respectively. g is the Rectified Linear Unit (ReLU) [3] function. $\{\mathbf{C}^{(i)} | i = 1, ..., n-1\}$ are $D \times D$ context matrices

[1] `https://markus.teach.cs.toronto.edu/csc321-2017-01`
[2] `http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/syllabus.pdf`

of text modality. $\mathbf{C}^{(m)}$ is a $D \times M$ context matrix of image modality. The conditional probability $P(w_n|w_1, \ldots, w_{n-1}, \mathbf{x})$ is then computed as

i looping over vocabulary

$$P(w_n|w_1, \ldots, w_{n-1}, \mathbf{x}) = \frac{\exp(\hat{\mathbf{r}}^\top \mathbf{r}_i + b_i)}{\sum_{j=1}^{K} \exp(\hat{\mathbf{r}}^\top \mathbf{r}_j + b_j)}, \tag{2}$$

where b_i is a bias vector associated with the i-th word. The overall set of parameters to be learned is summarized as $\{\mathbf{J}, \mathbf{C}^{(i)}, \mathbf{C}^{(m)}, h, b_i, \mathbf{R}|i = 1, \ldots, n\}$.

Based on the above conditional probability, we use beam search to generate the caption. To briefly explain how it works, we assume the vocabulary size and beam width are V and N respectively. For the first word of the caption, we sort the vocabulary in descending order based on their conditional probabilities and take the first N as candidates. For the second one, we compute the conditional probabilities given the first word is one of the N candidates from last step. In this way, we will have $V \times N$ possible sequences of length 2. We then take the first N sequences of which the probabilities are top N. This procedure is repeatedly applied until the generated sequences are of desired length.

We train the model using the cross-entropy criterion, which is equivalent to maximizing the probability it assigns to the targets in the training set. Hopefully it will also learn to make sensible predictions for sequences it hasn't seen before.

Dataset

Download and extract the archive from the CSC321 homework page `http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/`. The dataset inside the archive is a randomly sampled subset of the Microsoft COCO dataset (`http://mscoco.org/`). It contains 1000, 250 and 250 instances for training, validation and testing respectively.

For each image, we provide the hidden representation of last fully connected layer of VGG-16 as the feature vector which is of size 4096. Feature vectors are stored in:

```
/data/train.npy
/data/val.npy
/data/test.npy
```

The corresponding image lists are stored in:

```
/data/train_img_id.txt
/data/val_img_id.txt
/data/test_img_id.txt
```

The associated sentences are stored in:

```
/data/sentences_coco_train.json
/data/sentences_coco_val.json
```

We also provide initial word embedding, i.e., initial values of $\mathbf{R}$, in the file `/data/word_embeddings.p`. The size of word embedding is 100. The vocabulary consists of 858 words.

Questions

Visualize Word Embedding (10%)

To get the feel of what word embeddings are, you can use t-SNE [2] to visualize these high dimensional vectors in a 2-dimensional space. We already provide the implementation in the file `visual_word_embed.py`. By running it, you can find how the words are distributed. In this question, you are asked to pick up some subset of the whole vocabulary and explain what you have found inside the visualization (**Hint**: you can select words such as nouns and verbs which can easily form clusters in the visualization). Pick up several pairs of words, e.g., (man, woman), (man, sugar) and then compute the cosine similarity between word embeddings of each pair. Explain what you have found from these similarity scores.

Implement Forward Pass of Model (40%)

In this question, you will be asked to implement the forward pass of neural language model. Specifically, looking into the `mlbl.py`, you should fill in the blanks of the member function `forward` based on your understanding of the model. In the code file your implementation should follow after the comment as below.

```
##############################################################################
# You should implement forward pass here!
##############################################################################
```

In terms of programming, forward pass only involves basic `NumPy` operations. Since we will use autograd for the backward pass of the model, you will have to use the customized NumPy of autograd for the forward pass. You should be careful with the restrictions of the customized NumPy operations. For example, use `np.dot(A, B)` instead of `A.dot(B)`. Please refer to the tutorial of autograd (`https://github.com/HIPS/autograd/blob/master/docs/tutorial.md`) for more details.

Implement Backward Pass of Model (20%)

In this question, you will be asked to implement the backward pass of neural language model. Looking into the `mlbl.py`, you should fill in the blanks of the member function `backward` based on your understanding of the model. You will use `grad` function of autograd for this question. Please refer to the document of autograd for detailed instructions. In the code file your implementation should follow after the comment as below.

```
##############################################################################
# You should implement backward pass here!
##############################################################################
```

Train and Evaluate Model (30%)

Once you correctly implemented the forward and backward passes, you can train the model by simply running `run_trainer.py`. During training, the program will periodically open a webpage which displays 2 example validation images and the corresponding generated captions. All hyperparameters, e.g., learning rate, momentum, are set in the file `trainer.py`. You can play around with different values of hyperparameters to see what happens.

We use BLEU [1] score to quantitatively evaluate our generated caption against the ground truth caption. By training the model for around 15 to 20 minutes, you should be able to see some sensible captions and get around 6.0 bleu score on the validation set. Note the generated word '`unk`' means unknown which is a dummy element added into the vocabulary.

To test the model, you can just run `run_tester.py`. Since we only use a subset of the whole dataset and model is fairly simple, the performance is far from perfect. However, it should be able to generate some reasonable sentences.

In this question, you will be asked to:

- Try different choices of learning rate and momentum and compare the corresponding validation performance. You should at least experiment with two different values of learning rate and momentum respectively (totally 4 settings including the one already in the code). Show the curve of BLEU scores on the validation set during training (**Hint**: you can change `prog['_bleu']` in the `trainer.py` to modify how often the program computes the BLEU score). Explain why one set of hyperparameters might work better than another.

- Choose the best model based on the validation performance. Test the model and report the BLEU score on the testing set.

- To understand what the model has learned, you can run `run_interpreter.py` to obtain the top-K probable candidates out of the whole vocabulary for each word in the sentence, where K is the beam width. You can tweak the beam width and explain what you found from the candidate words. For example, you may find that word 'am' follows word 'I' at a very high probability.

- Look through the results of good or bad captions and trying to explain why they occurred.

What To Submit

For reference, here is everything you need to hand in:

- A PDF file `a2-writeup.pdf`, typeset using LaTeX, containing your answers to the conceptual questions.

- Your implementation of `mlbl.py`.

References

[1] Papineni, K., Roukos, S., Ward, T. and Zhu, W.J., 2002, July. BLEU: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting on association for computational linguistics (pp. 311-318). Association for Computational Linguistics.

[2] Maaten, L.V.D. and Hinton, G.E., 2008. Visualizing data using t-SNE. Journal of Machine Learning Research, 9(Nov), pp.2579-2605.

[3] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[4] Kiros, R., Salakhutdinov, R. and Zemel, R.S., 2014, June. Multimodal Neural Language Models. In International conference on machine learning (Vol. 14, pp. 595-603).

[5] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollr, P. and Zitnick, C.L., 2014, September. Microsoft coco: Common objects in context. In European Conference on Computer Vision (pp. 740-755). Springer International Publishing.