Aditi - Problem 5

April 3, 2017

Problem 5

1 CNN

- Take an embedding matrix E: $d \times V$ where V is vocabulary size and d is embedding size.
- Limit the number of words in each sentence to some number, say M.
- For each word w_t, take the one-hot encoding $\tilde{x_t} \in \mathbb{R}^V$ and compute

$$x_t = E\tilde{x_t}$$

  Note $x_t \in \mathbb{R}^d$.
- Glue all M different x_t together so that you have a $M \times d$ matrix. If the input contains less than M words, add padding. If the input contains more than M words, truncate.
- Now we can use this matrix as input to a CNN.
- Apply a convolution on this matrix. For example, suppose you use a convolution filter which has shape $2 \times d$. Then this is captures bigram information from the input sentence. Suppose you apply several convolution filters of height n to contain different degrees of $n-$gram information. Notice that generally we would prefer to set the length of the convolution filters to d so that the filter looks at the full embedding vectors.
- At this stage, you have several vectors, each corresponding to the application of a single convolution filter over the input matrix. Apply max-pooling over the entire vectors separately - that is, take the maximum value from each of the vectors. Append the results together. The result will be a vector which is as long as the number of convolution filters. Let us say this vector is $v \in \mathbb{R}^m$.
- Apply an element-wise non-linearity to the vector.
- Pass the vector to a softmax layer. Let $W \in \mathbb{R}^{3 \times m}$ and $b \in \mathbb{R}^3$. Then compute:

$$s = Wv + b$$

  Note $s \in \mathbb{R}^3$. The three elements of s should contains scores corresponding to the compatibility of the input which each of the three classes. To transform these probabilities to scores, we use the soft-max function. Let p_i be the probability that the input corresponds to class i. Then:

$$p_i = \frac{e^{s_i}}{\sum_{i=1}^{3} e^{s_i}}$$

- This model can be trained using a gradient descent optimizer. The loss function should be cross entropy loss between the target distribution and the probability distribution p_1, p_2, p_3 of each input in the training data. The target distribution is zero everywhere except at the class corresponding to the correct label of each input.

- To select filter sizes, apply convolution filters which are as tall as the value of n-grams which you think will be helpful for the classification task. Since we apply max-pooling to the output of each filter and then apply softmax, if a single convolution filter is more or less important than others, then the column of the weight matrix which uses the output of max-pooling on that filter will be given smaller weights by a successful optimization algorithm.

2 RNN

- Let f represent a recursive neural net which computes the hidden state h_t of the network at time t. Let w_t be the t-th word in the input sentence and $\tilde{x_t}$ be its corresponding one-hot vector. The we define x_t as:

  $$x_t = E\tilde{x_t}$$

  where $E \in \mathbb{R}^{d \times V}$ is the embedding matrix. V is the vocabulary size and length of $\tilde{x_t}$, and $x_t \in \mathbb{R}^d$ is the word embedding corresponding to the input token at w_t.

  Then we can set $h_0 = \vec{0}$ and compute the hidden state at time t as:

  $$h_t = f(x_t, h_{t-1})$$

  We can recursively compute h_T which will contain a summary of the entire input sentence.

- Use an RNN to get the history vectors $h_1, ..., h_{50}$ for each time-step in the input sentence. Line up the vectors so that you have a matrix which has height 50 and width d. Then apply several convolution filters of width d. Apply max-pooling over the vector resulting from each convolution. Concatenate the results - the current vector will be as long as the number of convolutions. Then apply a non-linearity and pass through a soft-max layer. - Also need to include details about sizes of matrices and vectors in RNN.

Extra Credit - 5D

fastText is a popular and often-effective model for sentence classification. Let $w_1, ..., w_n$ be an input sentence that we would like to classify. Let $\tilde{x}_1, ..., \tilde{x}_N \in \mathbb{R}^V$ be n-gram features corresponding to the input sentence, where V is the total number of features and N is the total number of corresponding n-grams that appear in $w_1, ..., w_n$. Let $E \in \mathbb{R}^{d \times V}$ be an embedding matrix and define:

$$x_i = E\tilde{x}_i$$

Now $x_1, ..., x_N \in \mathbb{R}^d$ are word embeddings corresponding to the n-gram features of the input sentence. Then compute:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Now we pass the average of the word embeddings through a linear classifier to compute a score $p = p(y = 1|w_1, ..., w_n)$:

$$p = W\mu + b$$

Finally, we train the model to maximize the probability of the correct class, or equivalently, to minimize the negative log-likelihood of the correct class.

There are two main advantages to the fastText model - in addition to strong empirical performance. The first is that it is a fairly straightforward and efficient model, and can have fewer parameters to train compared to a large RNN or ConvNet model. Then second is that, unlike an ConvNet, the fastText model can accept variable-length input. Though many fastText implements only consider fixed windows of inputs, this is not necessary from a mathematical standpoint since we average the embedding vectors of the input sequence.

Unlike ConvNet and RNN, fastText takes a bag-of-words/bag-of-n-grams approach to word order, since

it averages the embeddings corresponding to all of the n-grams and tokens of the input sentence. In contrast, RNNs traverse input sequences in order, and are therefore attentive to word order. Similarly, ConvNets used for natural language tasks process input sentences by stacking word embedding vectors together in an ordered manner before applying convolution, pooling and fully-connected layers, which is also attentive to word order. With fastText, in order to capture longer-term relationships between far-apart tokens, our best approach is to consider larger-sized n-gram features. However, this can become problematic. As the size of the n-gram grows, the number of possible n-grams also grows and we begin to encounter issues of sparsity, where we have little data associated with many of the large n-grams. This increases our model complexity in a manner that is, at best, prone to overfitting and, at worst, uninformative. For a language modeling task, this can be especially challenging since word order - and specifically long-term dependencies - can indicate crucial information.

We trained a simple fastText language model on Penn Treebank data with the default parameters found in Facebook's fastText implementation for five epochs and a window size of 20. Since the fastText package computes accuracy and not perplexity, we cite this below:

Validation Accuracy	8.12 %
Test Accuracy	8.17 %

Below, we give some examples of how fastText predicts the missing next word in a sentence compared to what the actual word is:

Input	Predicted	Actual
consumers may want to move their telephones a little closer to the tv set UNK UNK watching abc 's monday	night	UNK
may want to move their telephones a little closer to the tv set UNK UNK watching abc 's monday night	football	UNK
want to move their telephones a little closer to the tv set UNK UNK watching abc 's monday night football	can	UNK

From these experiments, it is clear that we have a very poor language model, since it simply guesses a somewhat common token in the vocabulary for each input. These results also suggest that the model can be insensitive to word order and content, since it guesses the same value for each input even though every pair of inputs differ in at least two tokens. Finally, however, it is worth noting that this was a somewhat small model which was not trained for very long, so we should not make definitive claims about its potential.