CS224n: Natural Language Processing with Deep Learning [1]

Lecture Notes: Part VIII

Convolutional Neural Networks[2]

Winter 2019

[1] Course Instructors: Christopher Manning, Richard Socher

[2] Authors: Francois Chaubard, Richard Socher

1 CNNs (Convolutional Neural Networks)

1.1 Why CNNs?

Convolutional Neural Networks take in a sentence of word vectors and first create a phrase vector for all subphrases, not just grammatically correct phrases (as with Recursive Neural Network, addressed in the next set of notes). CNNs then group them together for the task at hand.

1.2 What is Convolution?

Let's start with the 1D case. Consider two 1D vectors, f and g with f being our primary vector and g corresponding to the **filter**. The convolution between f and g, evaluated at entry n is represented as $(f * g)[n]$ and is equal to $\sum_{m=-M}^{M} f[n-m]g[m]$.

Figure 1 shows the 2D convolution case. The 9×9 green matrix represents the primary matrix of concern, f. The 3×3 matrix of red numbers represents the filter g and the convolution currently being evaluated is at position $[2, 2]$. Figure 1 shows the value of the convolution at position $[2, 2]$ as 4 in the second table. Can you complete the second table?

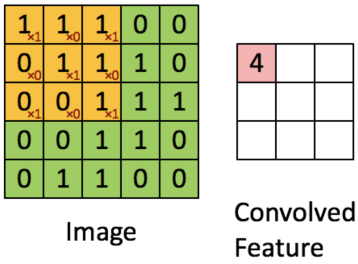

Figure 1: Convolution in the 2D case

1.3 A Single-Layer CNN

Consider word-vectors $x_i \in R^k$ and the concatenated word-vectors of a n-word sentence, $x_{1:n} = x_1 \oplus x_2 ... \oplus x_n$. Finally, consider a Convolutional filter $w \in R^{hk}$ i.e. over h words. For $k = 2$, $n = 5$ and $h = 3$, Figure 2 shows the Single-Layer Convolutional layer for NLP. We will get a single value for each possible combination of three consecutive words in the sentence, "the country of my birth". Note, the filter w is itself a vector and we will have $c_i = f(w^T x_{i:i|h-1} + b)$ to give $\mathbf{c} = [c_1, c_2 ... c_{n-h+1}] \in R^{n-h+1}$. For the last two time-steps, i.e. starting with the words "my" or "birth", we don't have enough word-vectors to multiply with the filter (since $h = 3$). If we necessarily need the convolutions associated with the last two word-vectors, a common trick is to pad the sentence with $h - 1$ zero-vectors at its

right-hand-side as in Figure 3.

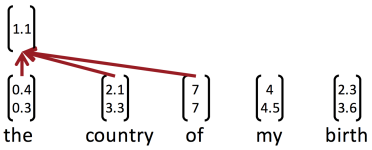

Figure 2: Single-Layer Convolution: one-step

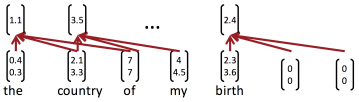

Figure 3: Single-Layer Convolution: all-steps

1.4 *Pooling*

Assuming that we don't use zero-padding, we will get a final convolutional output, $\mathbf{c}$ which has $n - h + 1$ numbers. Typically, we want to take the outputs of the CNN and feed it as input to further layers like a Feedforward Neural Network or a RecNN. But, all of those need a fixed length input while our CNN output has a length dependent on the length of the sentence, n. One clever way to fix this problem is to use max-pooling. The output of the CNN, $\mathbf{c} \in \mathbb{R}^{n-h-1}$ is the input to the max-pooling layer. The output of the max-pooling layer is $\hat{c} = max\{\mathbf{c}\}$, thus $\hat{c} \in \mathbb{R}$.

We could also have used min-pooling because typically we use ReLU as our non-linear activation function and ReLU is bounded on the low side to 0. Hence a min-pool layer might get smothered by ReLU, so we nearly always use max-pooling over min-pooling.

1.5 *Multiple-Filters*

In the example above related to Figure 2, we had $h = 2$, meaning we looked only at bi-gram with a single specific combination method i.e. filter. We can use multiple bi-gram filters because each filter will learn to recognize a different kind of bi-gram. Even more generally, we are not restricted to using just bi-grams, we can also have filters using tri-grams, quad-grams and even higher lengths. Each filter has an associated max-pool layer. Thus, our final output from the CNN layers will be a vector having length equal to the number of filters.

1.6 *Multiple-Channels*

If we allow gradients to flow into the word-vectors being used here, then the word-vectors might change significantly over training. This is desirable, as it specializes the word-vectors to the specific task at hand (away from say GloVe initialization). But, what about words that appear only in the test set but not in the train set? While other semantically related word vectors which appear in the train set will have moved significantly from their starting point, such words will still be at their initialization point. The neural network will be specialized for inputs which have been updated. Hence, we will get low performance on sentences with such words (words that are in test but not in train).

One work-around is to maintain two sets of word-vectors, one 'static' (no gradient flow into them) and one 'dynamic', which are updated via SGD. Both are initially the same (GloVe or other initial-

ization). Both sets are simultaneously used as input to the neural network. Thus, the initialized word-vectors will always play a role in the training of the neural network. Giving unseen words present in the test set a better chance of being interpreted correctly.

There are several ways of handling these two channels, most common is to simply average them before using in a CNN. The other method is to double the length of the CNN filters.

1.7 CNN Options

1. Narrow vs Wide
   Refer to Figure 4. Another way to ask this is should we not (narrow) or should we (wide) zero-pad? If we use Narrow Convolution, we compute a convolution only in those positions where all the components of a filter have a matching input component. This will clearly not be the case at the start and end boundaries of the input, as in the left side network in Figure 4. If we use Wide Convolution, we have an output component corresponding to each alignment of the convolution filter. For this, we will have to pad the input at the start and the end with $h-1$ zeros.
   In the Narrow Convolution case, the output length will be $n-h+1$ and in the Wide Convolution case, the length will be $n+h-1$.

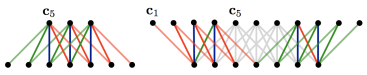

Figure 4: Narrow and Wide Convolution (from Kalchbrenner et al. (2014))

2. k-max pooling
   This is a generalization of the max pooling layer. Instead of picking out only the biggest (max) value from its input, the k-max pooling layer picks out the k biggest values. Setting $k=1$ gives the max pooling layer we saw earlier.