Dependency parser NN model

Jun Kang

March 2018

1 Dependency parser NN model

Below adds more detail to Chen and Manning (2014) paper to help you undetstand it better.

1.1 Input to Hidden

The mapping from input layer to hidden layer h is defined as below:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3 \tag{1}$$

Let's start by looking atthe embeddings.

- d: embedding size.
- $e_i^w \in \mathbb{R}^d$: an embedding of a word i
- $e_j^t \in \mathbb{R}^d$: an embedding of a POS tag j
- $e_k^l \in \mathbb{R}^d$: an embedding of a arc label k

Each embedding (e_i^w, e_j^t and e_k^l) is a vector and would look as below.

Now we have to build an input data. Suppose we are adding following number of word/POS/label embeddings as input data: n_w, n_t, n_l.

We group these embeddings by their type: x^w, x^t, x^l.

$$x^w = \left[e^w_{w_1}; e^w_{w_2}; ...; e^w_{w_{n_w}}\right]$$
$$= \begin{bmatrix} e^w_{w_1} \\ e^w_{w_2} \\ ... \\ e^w_{w_{n_w}} \end{bmatrix}$$

$$x^t = \left[e^t_{t_1}; e^t_{t_2}; ...; e^t_{t_{n_t}}\right]$$
$$= \begin{bmatrix} e^t_{t_1} \\ e^t_{t_2} \\ ... \\ e^t_{t_{n_t}} \end{bmatrix}$$

$$x^l = \left[e^l_{l_1}; e^t_{l_2}; ...; e^l_{t_{n_l}}\right]$$
$$= \begin{bmatrix} e^l_{t_1} \\ e^l_{t_2} \\ ... \\ e^l_{t_{n_l}} \end{bmatrix}$$

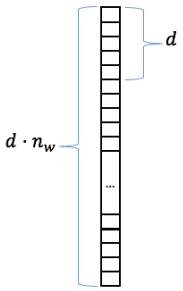
In fact, we can treat these as large column vectors. Then, the dimensions are:

$$x^w \in \mathbb{R}^{(d \cdot n_w)}$$

$$x^t \in \mathbb{R}^{(d \cdot n_t)}$$

$$x^l \in \mathbb{R}^{(d \cdot n_l)}$$

For example, x^w would look as below:



Then, the input weight, W_1 should have to following dimensions (d_h is the size of hidden layer):

$$W_1^w \in \mathbb{R}^{d_h \times (d \cdot n_w)}$$
$$W_1^t \in \mathbb{R}^{d_h \times (d \cdot n_t)}$$
$$W_1^l \in \mathbb{R}^{d_h \times (d \cdot n_l)}$$

The matrix product $W_1 x^w$, then, would yield a $d_h \times 1$ matrix or a vector of size d_h. (If you are not familiar with matrix product, look at section 2.)

Similarly, all matrix product terms in (1) produce vectors of size d_h. Since b_1 is also a vector of size d_h, (1) becomes:

$$h = r^3 \tag{2}$$
$$\tag{3}$$

where $r = W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1$ and $r \in \mathbb{R}^{d_h}$

Mind that the cube here is a element-wise computation. That is, each element is cubed.

1.2 Hidden to Output

Here we map hidden layer to output layer (which will tell us which "transition(s)" we should take.)

$$p = softmax(W_2 h)$$

where W_2 is a matrix of $|T| \times d_h$. Here $|T|$ is the number of transitions.

Then, the resulting p will be a vector of size $|T|$.

2 Matrix Product

If you are not familiar with this or studied this years ago, it would be helpful to refresh some basic linear algebra topics.

* Matrix Product and element-wise multiplication is totally different. In tensorflow, former is tf.matmul and the latter is tf.multiply.

Below is the one example I used to explain NCE, which also shows a matrix product between V and U^T

So, you have two tensors as below:

$$inputs = V = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{pmatrix} [batch\_size, emb\_size]$$

and

$$neg\_samples = U = \begin{pmatrix} u_{x1} \\ u_{x2} \\ u_{x3} \end{pmatrix} = \begin{pmatrix} u_{x1,1} & u_{x1,2} \\ u_{x2,1} & u_{x2,2} \\ u_{x3,1} & u_{x3,2} \end{pmatrix} [sample\_size, emb\_size]$$

If you compute:

$$\begin{aligned} V \times U^T &= \begin{pmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{pmatrix} \times \begin{pmatrix} u_{x1,1} & u_{x1,2} \\ u_{x2,1} & u_{x2,2} \\ u_{x3,1} & u_{x3,2} \end{pmatrix}^T \\ &= \begin{pmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{pmatrix} \times \begin{pmatrix} u_{x1,1} & u_{x2,1} & u_{x3,1} \\ u_{x1,2} & u_{x2,2} & u_{x3,2} \end{pmatrix} \\ &= \begin{pmatrix} v_{1,1}u_{x1,1} + v_{1,2}u_{x1,2} & v_{1,1}u_{x2,1} + v_{1,2}u_{x2,2} & v_{1,1}u_{x3,1} + v_{1,2}u_{x3,2} \\ v_{2,1}u_{x1,1} + v_{2,2}u_{x1,2} & v_{2,1}u_{x2,1} + v_{2,2}u_{x2,2} & v_{2,1}u_{x3,1} + v_{2,2}u_{x3,2} \end{pmatrix} \\ &= \begin{pmatrix} v_1 \cdot u_{x1} & v_1 \cdot u_{x2} & v_1 \cdot u_{x3} \\ v_2 \cdot u_{x1} & v_2 \cdot u_{x2} & v_2 \cdot u_{x3} \end{pmatrix} \end{aligned}$$

Then, you will get

$$v_c$$

against all negative samples. Finding a way to do this with tensorflow is just a matter of finding right API functions and carefully feeding arguments. I will leave that to you.

References

Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.