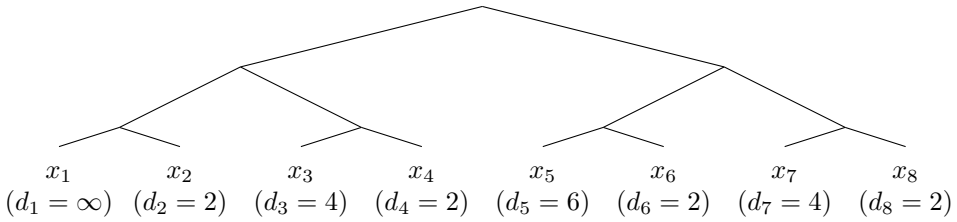
The ON-LSTM and PRPN Architectures*

Karl Stratos

1 Background

1.1 Top-Down Parsing

The traditional approach to inducing the tree structure of a sentence is bottom-up: we start from words and consider all possible ways subtrees compose using dynamic programming (e.g., the inside-outside algorithm [2] and its modern adaptation [1]). PRPN and ON-LSTM take a top-down approach. Given a sentence $x_1 \dots x_T$, we infer a value d_t at each $t = 1 \dots T$ that measures the "syntactic distance" between x_{t-1} and x_t. In the parsing context, d_t is most naturally viewed as the number of edges between (x_{t-1}, x_t) in the underlying tree. Here's an illustration:



If we infer such distance values $d_1 \dots d_T$, we can estimate a corresponding tree structure in various ways. In PRPN and ON-LSTM, we use a heurstic that is biased toward right-branching trees. In pseudocode: we call $\textbf{Parse}(x_1 \dots x_T)$ define as

Input: sequence $x = (x_i \dots x_j)$
Output: a binary tree over x

1. If $|x| \leq 1$, return x.
2. Find $k^* = \arg\max_{k=i+1}^{j} d_k$.
3. Return $(\textbf{Parse}(x_i \dots x_{k^*-1}), (x_{k^*}, \textbf{Parse}(x_{k^*} \dots x_j)))$

Note that this algorithm fails to recover the balanced tree in the above example.

*Parsing of Shen *et al.* [4] and Shen *et al.* [3].

1.2 The Stick-Breaking Process

Let p_I be a distribution over $\{1 \dots n+1\}$. We draw $i \sim p_I(\cdot)$ and for $j = 1 \dots n$ define $g_j := [[i \leq j]]$. The probability that $g_j = 1$ is the CDF of I evaluated at j:

$$p_{G_j}(1) = \sum_{i=1}^{j} p_I(i) \tag{1}$$

The expected value of I can be expressed as a function of (1):

$$\mathop{\mathbf{E}}_{i \sim p_I(\cdot)}[i] = \sum_{i=1}^{n+1} \sum_{j=1}^{i} p_I(i) = \sum_{j=1}^{n+1} \sum_{i=j}^{n+1} p_I(i) = \sum_{j=1}^{n+1} \left(1 - \sum_{i=1}^{j-1} p_I(i)\right) = n + 1 - \sum_{j=1}^{n} p_{G_j}(1) \tag{2}$$

In English: the binary vector $g \in \{0,1\}^n$ is viewed as a broken stick. The breaking point $i \in \{1 \dots n+1\}$ dictates that each $j < i$ is assigned to the left half and each $j \geq i$ to the right half of the stick. The smaller j is, the less likely it is assigned to the right half. We can calculate the expected value of i from the expected value of g.

Dynamic stick lengths. When we don't know the stick length n ahead, it's convenient to process the stick sequentially and continue with probability α_j at each position j (by convention $\alpha_{n+1} := 0$). In this case we define

$$p_I(i) = (1 - \alpha_i) \prod_{j=1}^{i-1} \alpha_j \qquad \forall i = 1 \dots n+1 \tag{3}$$

The expression has a nice telescoping property: $\sum_{i=1}^{j} p_I(i) = 1 - \prod_{k=1}^{j} \alpha_k$ for any $j = 1 \dots n+1$ (in particular $\sum_{i=1}^{n+1} p_I(i) = 1$). Furthermore, by (1) the probability that j is assigned to the left half of the stick is simply

$$p_{G_j}(0) = \prod_{k=1}^{j} \alpha_k \tag{4}$$

The stick-breaking process in (3) as $n \to \infty$ characterizes the Dirichlet process under certain statistical assumptions (Appendix A). However, for the purposes of this note it is unnecessary to understand this connection.

1.2.1 Applications to ON-LSTM and PRPN

In ON-LSTM, we view the LSTM state at each time step t as a fixed-length stick. We parameterize the distribution over breaking points by using the current LSTM inputs, calculate the expected value of the sticks using (1), and use these sticks to meta-gate the standard LSTM gates so that segments of the LSTM state correspond to different levels in a tree.

In PRPN, at each position t we have M memory slots corresponding to the previous M positions. We treat these slots as a stick going from right to left (away from t). For

$j = t - 1 \ldots$ we continue moving as long as the syntactic distance between $(j-1, j)$ is at most that between $(t-1, t)$. When we stop, we only use the memory slots assigned to the first half of the stick by calculating (4). This corresponds to conditioning on the local topological structure of the underlying tree over a sequence $x_1 \ldots x_t$. If x_t is not the leftmost child of any node, we condition only on its siblings to the left. Otherwise, letting ν denote the topmost node such that x_t is the leftmost child of, we condition on all the production of the left siblings of ν.

2 The ON-LSTM Architecture

Let $E \in \mathbb{R}^{d' \times |V|}$ denote the word embedding matrix where V is the vocabulary and $d' = 400$ is the dimension of word embeddings. Let $L \geq 2$ (default $L = 3$) denote the number of LSTM layers. At each position $t \geq 1$, for layer $l = 1 \ldots L$ the corresponding ON-LSTM cell takes an input vector $h_t^{(l-1)}$ and the previous state $\left(h_{t-1}^{(l)}, c_{t-1}^{(l)}\right)$ of that layer to compute the new state

$$\left(h_t^{(l)}, c_t^{(l)}\right) = \textbf{ON-LSTM}\left(h_t^{(l-1)}, \left(h_{t-1}^{(l)}, c_{t-1}^{(l)}\right)\right)$$

where $h_t^{(0)} = \textbf{LockedDrop}_{0.5}\left(E_{x_t}\right)$ is just the current word embedding after locked dropout (i.e., it uses the same dropout mask across t for the entire BPTT sequence). Let $d^{(l)}$ denote the state dimension at layer l: the parameters of the ON-LSTM cell are shaped so that $d^{(1)} = \cdots = d^{(L-1)} = 1150$ and $d^{(L)} = d'$. Then the model defines a distribution over words at position $t+1$ by

$$p(\cdot|x_1 \ldots x_t) = \text{softmax}\left(\textbf{LockedDrop}_{0.45}\left(E^\top h_t^{(L)}\right)\right)$$

2.0.1 The ON-LSTM Cell

The l-th LSTM layer first computes 6 affine transformations of the input vector and the previous hidden state using parameters $A^{(l)} \in \mathbb{R}^{(2p^{(l)}+4d^{(l)}) \times d^{(l-1)}}$, $B^{(l)} \in \mathbb{R}^{(2p^{(l)}+4d^{(l)}) \times d^{(l)}}$, $b^{(l)} \in \mathbb{R}^{2p^{(l)}+4d^{(l)}}$ where $p^{(l)} = d^{(l)}/C$ is the number of segments in the state vector divided into C-sized chunks (default $C = 10$):

$$\left(v_1^t, v_2^t, u_1^t, u_2^t, u_3^t, u_4^t\right) = A^{(l)} h_t^{(l-1)} + B^{(l)} h_{t-1}^{(l)} + b^{(l)}$$

where $v_i^t \in \mathbb{R}^{p^{(l)}}$ and $u_i^t \in \mathbb{R}^{d^{(l)}}$. The standard LSTM forget, input, output gates and the raw cell state are computed as $\left(f_t^{(l)}, i_t^{(l)}, o_t^{(l)}\right) = \sigma\left(u_1^t, u_2^t, u_3^t\right)$ and $\tilde{c}_t^{(l)} = \tanh\left(u_4^t\right)$. Now two distributions over $\left\{1 \ldots p^{(l)}\right\}$ are computed from v_1^t and v_2^t: the index i drawn from these distributions defines a breaking point for a stick of length $p^{(l)}$. We compute the expected value of the sticks $\tilde{f}_t^{(l)}, \tilde{i}_t^{(l)} \in \mathbb{R}^{p^{(l)}}$ opposite in directions by

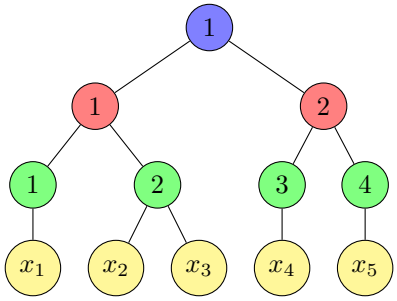
$$\left[\tilde{f}_t^{(l)}\right]_j = \sum_{i=1}^{j} \text{softmax}_j\left(v_1^t\right) \qquad \left[\tilde{i}_t^{(l)}\right]_j = 1 - \sum_{i=1}^{j} \text{softmax}_j\left(v_2^t\right) \qquad \forall j = 1 \ldots p^{(l)}$$

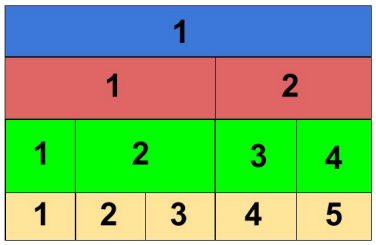
These sticks divide the state into $p^{(l)}$ conceptual segments corresponding to different tree levels as follows:

$$\begin{aligned}
\omega_t^{(l)} &= \tilde{f}_t^{(l)} \odot \tilde{i}_t^{(l)} \\
\hat{f}_t^{(l)} &= f_t^{(l)} \odot \text{stretch}\left(\omega_t^{(l)}\right) + \text{stretch}\left(\tilde{f}_t^{(l)} - \omega_t^{(l)}\right) \\
\hat{i}_t^{(l)} &= i_t^{(l)} \odot \text{stretch}\left(\omega_t^{(l)}\right) + \text{stretch}\left(\tilde{i}_t^{(l)} - \omega_t^{(l)}\right) \\
c_t^{(l)} &= \hat{f}_t^{(l)} \odot c_{t-1}^{(l)} + \hat{i}_t^{(l)} \odot \tilde{c}_t^{(l)} \\
h_t^{(l)} &= o_t^{(l)} \odot \tanh\left(c_t^{(l)}\right)
\end{aligned}$$

where $v = \text{stretch}\,(u) \in \mathbb{R}^{d^{(l)}}$ is defined as $v_{i+j} = u_i$ for $j = 0 \ldots C-1$. If $[\tilde{f}_t^{(l)}]_j = 0$, it means the j-th tree level is being forgotten, so a large breaking point in $\tilde{f}_t^{(l)}$ means lots of tree levels are being forgotten (x_t is starting a new high-level constituent). Thus the expected breaking point $p^{(l)} - \sum_{j=1}^{p^{(l)}-1} [\tilde{f}_t^{(l)}]_j$ (2) can serve as syntactic distance d_t in the previous top-down parsing algorithm (for a certain layer l). If $[\tilde{i}_t^{(l)}]_j = 1$, it means the j-th tree level is being updated by x_t. The overlap $\omega_t^{(l)}$ indicates tree levels that are both continued and updated: the hidden states in these levels, and only in these levels, are fine-controlled by the standard LSTM forget and input gates.

A picture is worth a thousand words here ($C = 1$ for simplicity):





$$\tilde{f}_2 = \begin{bmatrix}0\\0\\1\\1\end{bmatrix} \quad \tilde{i}_2 = \begin{bmatrix}1\\1\\1\\0\end{bmatrix} \quad \omega_2 = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad \hat{f}_2 = \begin{bmatrix}0\\0\\.3\\1\end{bmatrix} \quad \hat{i}_2 = \begin{bmatrix}1\\1\\.4\\0\end{bmatrix}$$

$$\tilde{f}_4 = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \quad \tilde{i}_4 = \begin{bmatrix}1\\1\\1\\0\end{bmatrix} \quad \omega_4 = \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \quad \hat{f}_4 = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \quad \hat{i}_4 = \begin{bmatrix}1\\1\\1\\0\end{bmatrix}$$

Remark. Note that the last index of $\tilde{i}_t$ is always 0 under this formulation (the authors' implementation). This seems wasteful: the last segment is never used. If this is unintended, we should adjust the length of v_i^t to be $p^{(l)} + 1$.

3 The PRPN Architecture

Each boldfaced function denotes a layer with its own set of trainable parameters (Appendix B), except $\mathbf{Drop}_r$ which is just dropout with rate r. The only other parameter is the word lookup matrix $E \in \mathbb{R}^{d' \times |V|}$ where V is the vocabulary and $d' = 800$ is the dimension of word embeddings.

Let $M = 15$ denote the number of memory slots, $L = 2$ the number of LSTM layers, and $Q = 5$ the number of words to look back. For each layer $l = 1 \dots L$, the model maintains a memory tape of the past M LSTM states of dimension $d = 1200$

$$\mu^{(l)} := \left(\left(h_{-M}^{(l)}, c_{-M}^{(l)} \right) \dots \left(h_{-1}^{(l)}, c_{-1}^{(l)} \right) \right)$$

(initialized to zero). At each position $t \geq 1$, the model computes $d_{t-M+1} \dots d_t \in [0, 1]$ as follows: if $i < 1$ set $d_i = \infty$, otherwise compute

$$d_i = \sigma \left(\mathbf{Conv}_{1,d}^{1} \left(\mathbf{Drop}_{0.5} \left(\text{ReLU} \left(\mathbf{BN}_d \left(\mathbf{Conv}_{Q+1,d'}^{d} \left(\mathbf{Drop}_{0.5} \left([E_{x_{i-Q}} \dots E_{x_i}] \right) \right) \right) \right) \right) \right) \right) \tag{5}$$

where E_{x_j} is zero for $j < 1$ and the convolutional layers are shared across all i. Each d_i represents the syntactic distance between (x_{i-1}, x_i). The model then computes $\alpha_{t-M+1}^t \dots \alpha_t^t \in [0, 1]$ by

$$\alpha_i^t = \frac{\text{hardtanh}\left((d_t - d_i)\tau + 1 \right) + 1}{2}$$

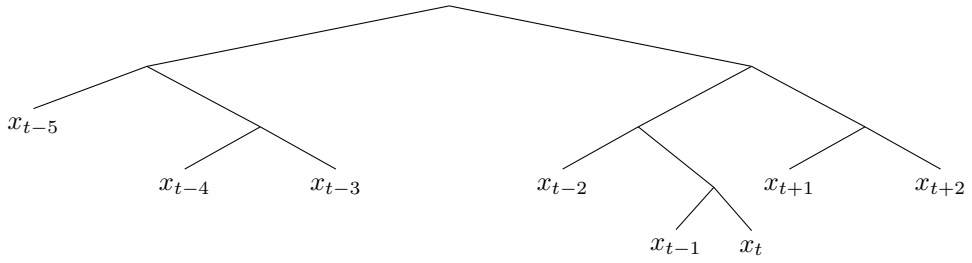
which is approximately the indicator function $[[d_t \geq d_i]]$ as $\tau \to \infty$ (the paper uses $\tau = 20$). α_i^t is small iff the syntactic distance between (x_{i-1}, x_i) is strictly larger than the syntactic distance between (x_{t-1}, x_t). The model uses these values to compute the M gates $g_{t-M+1}^t \dots g_t^t \in [0, 1]$:

$$g_i^t = \prod_{j=i}^{t} \alpha_j^t \approx \begin{cases} 1 & \text{if } i > \kappa \\ 0 & \text{if } i \leq \kappa \end{cases} \tag{6}$$

where $\kappa < t$ is the *rightmost* position that has a larger syntactic distance than t. The model uses these gates on the memory tape to focus on the past states at positions $[\kappa, t)$. Specifically, for each layer $l = 1 \dots L$ it computes (letting $h_t^{(0)} = E_{x_t}$)

$$\left(\tilde{h}_t^{(l)}, \tilde{c}_t^{(l)} \right) = \mathbf{GatedAttention} \left(\mu^{(l)}, h_{t-1}^{(l)} \oplus h_t^{(l-1)}, g_{t-M+1}^t \dots g_t^t \right)$$
$$\left(h_t^{(l)}, c_t^{(l)} \right) = \mathbf{LSTMCell} \left(\mathbf{Drop}_{r_l} \left(h_t^{(l-1)} \right), \left(\mathbf{Drop}_{0.5} \left(\tilde{h}_t^{(l)} \right), \tilde{c}_t^{(l)} \right) \right)$$

where the dropout rate r_l is 0.7 if $l = 1$ and 0.5 otherwise. **LSTMCell** is a standard LSTM cell with layer normalizations instead of dropout. Now the model defines a distribution $p(\cdot | x_1 \dots x_t)$ over words at position $t + 1$ by using the previous M top hidden states stored in the memory $\mu^{(L)}$ as well as the current top hidden state $h_t^{(L)}$. When using the memory, we want use gates $g_i^{t+1} \approx [[i > \kappa']]$ where $\kappa' < t$ is the rightmost position that has a larger syntactic distance than $t + 1$. The reason is a bit subtle. Consider the following tree with the assumption that d_i is the number of edges between (x_{i-1}, x_i).



As we compute the LSTM state at position t, we focus only on $h_{t-1}^{(L)}$ since $d_{t-1} > d_t$. As we predict x_{t+1}, we want to focus on $h_{t-2}^{(L)}$ also since $d_{t-1} < d_{t+1}$ and $d_{t-2} > d_{t+1}$. However, the formula in (5) requires the future word at position $t+1$ to compute d_{t+1}. As a hack, the model uses different parameters to compute $\hat{d}_{t+1}$ from $x_{t-Q} \dots x_t$:

$$\hat{d}_{t+1} = \sigma\left(\mathbf{Conv}_{1,d}^{1}\left(\mathbf{Drop}_{0.5}\left(\text{ReLU}\left(\mathbf{BN}_{d}\left(\mathbf{Conv}_{Q+1,d'}^{d}\left(\mathbf{Drop}_{0.5}\left([E_{x_{t-Q}} \dots E_{x_t}]\right)\right)\right)\right)\right)\right)\right)$$

(again the convolutional layers are shared across all i). The gates are calculated as $\hat{g}_i^{t+1} = \prod_{j=i}^{t} \hat{\alpha}_j^{t+1}$ for $i = t - M + 1 \dots t$ where

$$\hat{\alpha}_i^{t+1} = \frac{\text{hardtanh}\left(\left(\hat{d}_{t+1} - d_i\right)\tau + 1\right) + 1}{2}$$

The model uses these gates on $\mu^{(L)}$ attended with key $h_t^{(L)}$ to calculate a summary of the past M top hidden states (ignoring the cell states):

$$\tilde{h}_t = \mathbf{GatedAttention}\left(\mu^{(L)}, h_t^{(L)}, g_{t-M+1}^{t+1} \dots g_t^{t+1}\right)[0]$$

Finally, the model concatenates the summary and the current hidden state and adjusts the size using the word embedding dictionary E and additional parameters $W \in \mathbb{R}^{d' \times 2d}$, $b_1 \in \mathbb{R}^{d'}$, $b_2 \in \mathbb{R}^{|V|}$:

$$u^t = E^\top\left(\mathbf{Drop}_{0.7}\left(\tanh\left(\mathbf{BN}_{d'}\left(W\mathbf{Drop}_{0.5}\left(h_t^{(L)} \oplus \tilde{h}_t^{(L+1)}\right) + b_1\right)\right)\right)\right) + b_2$$

The distribution is defined by $p(\cdot|x_1 \dots x_t) = \text{softmax}\left(u^t\right)$.

A The Dirichlet Process

We consider a simplified version of the Dirichlet process (DP) equipped with the normal distribution $\mathcal{N}(0,1)$ and a scalar $\alpha > 0$. It is a distribution over distributions over $\mathbb{R}$ whose mean is $\mathcal{N}(0,1)$ and has the property

$$q \sim \text{DP} \qquad \xrightarrow{\text{a.s.}} \qquad q \text{ is a discrete distribution over } \mathbb{R}$$

The scalar α controls the support size of q: it is concentrated on a single $x \in \mathbb{R}$ as $\alpha \to 0$, and it spreads over $\mathbb{R}$ as $\alpha \to \infty$. The DP has a useful application to Bayesian nonparametric clustering models because it can yield a prior distribution over clusters even when the number of clusters is unspecified.

How do we define such a distribution? An explicit construction is given by the stick-breaking process: if $q \sim \text{DP}$, then for all $x \in \mathbb{R}$

$$q(x) = \sum_{i=1}^{\infty} \left(\beta_i \prod_{j=1}^{j-1} (1 - \beta_j) \right) [[x = x_i]]$$

where $x_i \sim \mathcal{N}(0,1)$ and $\beta_i \sim \text{Beta}(1, \alpha)$ are iid draws. Note that each indicator $[[x = x_i]]$ is weighted by an increasingly smaller value $\beta_i \prod_{j=1}^{j-1}(1-\beta_j)$ that tracks how much of the stick is left after breaking it i times. Smaller α leads to larger β_i's that shorten the stick faster, and q will be more concentrated.

Assume $\alpha = 1$ for simplicty. Consider an infinite mixture model that draws $q \sim \text{DP}$ and then draws $x_1 \dots x_N \sim q$. Consider the following alternative sampling procedure:

- Draw $y_1 \sim \mathcal{N}(0,1)$.
- For $i = 2 \dots N$, pick $j \in \{1 \dots i\}$ uniformly at random: if $j < i$ then set $y_i = y_j$, otherwise draw $y_i \sim \mathcal{N}(0,1)$.

Then $x_1 \dots x_N$ and $y_1 \dots y_N$ are distributed identically! This gives another characterization of the DP that illustrates the "rich gets richer" aspect: a repeated sample value is drawn with a higher probability. More generally, α is used as a smoothing value: we draw a new $y_i \sim \mathcal{N}(0,1)$ with probability $\alpha/(\alpha + i - 1)$.

B The Layers

B.1 GatedAttention

Parameters

- Affine transformation $A \in \mathbb{R}^{d \times p}$ and $b \in \mathbb{R}^d$ where p is the dimension of the key

Input

- Memory slots $(h_{-M}, c_{-M}) \dots (h_{-1}, c_{-1}) \in \mathbb{R}^d \times \mathbb{R}^d$
- Key vector $k \in \mathbb{R}^p$
- Gate values $g_{-M} \dots g_{-1} \in [0,1]^M$

Output

- $\left(\tilde{h}, \tilde{c}\right) \in \mathbb{R}^d \times \mathbb{R}^d$: gated combination of $(h_{-M}, c_{-M}) \dots (h_{-1}, c_{-1})$

Forward pass

$$\begin{aligned} \tilde{k} &= \mathbf{Drop}_{0.5}\left(A\mathbf{Drop}_{0.5}\left(k\right) + b\right) \\ s_{-i} &\propto g_{-i} \exp\left(\frac{h_{-i} \cdot \tilde{k}}{\sqrt{d}}\right) && \forall i = 1 \dots M \\ \left(\tilde{h}, \tilde{c}\right) &= \sum_{i=1}^{M} s_{-i}\left(h_{-i}, c_{-i}\right) \end{aligned}$$

B.2 $\mathbf{Conv}^d_{d',Q}$

Parameters

- d filters $F_1 \dots F_d \in \mathbb{R}^{d' \times Q}$

Input

- $X \in \mathbb{R}^{d' \times Q}$: Q vectors of dimension d'

Output

- $y \in \mathbb{R}^d$: d-dimensional representation of X

Forward pass

$$y_i = \sum_{j=1}^{d'} \sum_{k=1}^{Q} [F_i]_{j,k} \times X_{j,k} \qquad \forall i = 1 \dots d$$

B.3 $\mathbf{BN}_d$

Parameters

- $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$

Input

- Minibatch $x_1 \dots x_N \in \mathbb{R}^d$

Output

- "Whitened" minibatch $\bar{x}_1 \dots \bar{x}_N \in \mathbb{R}^d$

Forward pass

1. Compute the mean μ_j and standard deviation σ_j of $\{[x_1]_j \dots [x_N]_j\}$ for $j = 1 \dots d$.
2. Return

$$\bar{x}_i = \gamma \odot \frac{x_i - \mu}{\sigma + 0.001} + \beta \qquad \forall i = 1 \dots N$$

Remark

- The layer requires batch size > 1: pretend that this is the case in this note.
- The mean μ and variance σ of features are continuously estimated from the training data with momentum 0.1 and used at test time.

References

[1] Drozdov, A., Verga, P., Yadav, M., Iyyer, M., and McCallum, A. (2019). Unsupervised latent tree induction with deep inside-outside recursive autoencoders. *arXiv preprint arXiv:1904.02142*.

[2] Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech & language*, **4**(1), 35–56.

[3] Shen, Y., Lin, Z., wei Huang, C., and Courville, A. (2018). Neural language modeling by jointly learning syntax and lexicon. In *International Conference on Learning Representations*.

[4] Shen, Y., Tan, S., Sordoni, A., and Courville, A. (2019). Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*.