Recurrent Neural Networks

CS 287

(Based on Yoav Goldberg's notes)

Review: Continuous Bag-of-Bigrams Features?

Representation is counts of input bigrams,

- $\mathcal{F}$; the vocabulary of the bigram language.
- $\mathbf{x} = \sum_i \delta(f_i)$

Example: Movie review input,

`A sentimental mess`

$$\begin{aligned} \mathbf{x} &= v(\texttt{word:A}) + v_2(\texttt{bigram:A:sentimental}) \\ &+ v(\texttt{word:sentimental}) + v_2(\texttt{bigram:sentimental:mess}) \\ &+ v(\texttt{word:mess}) \end{aligned}$$

Review: Convolution Formally

Let our input be the embeddings of the full sentence, $\mathbf{X} \in \mathbb{R}^{n \times d^0}$

$$\mathbf{X} = [v(w_1), v(w_2), v(w_3), \dots, v(w_n)]$$

Define a window model as $NN_{window} : \mathbb{R}^{1 \times (d_{\mathrm{win}} d^0)} \mapsto \mathbb{R}^{1 \times d_{\mathrm{hid}}}$,

$$NN_{window}(\mathbf{x}_{win}) = \mathbf{x}_{win}\mathbf{W}^1 + \mathbf{b}^1$$

The convolution is defined as $NN_{conv} : \mathbb{R}^{n \times d^0} \mapsto \mathbb{R}^{(n - d_{\mathrm{win}} + 1) \times d_{\mathrm{hid}}}$,

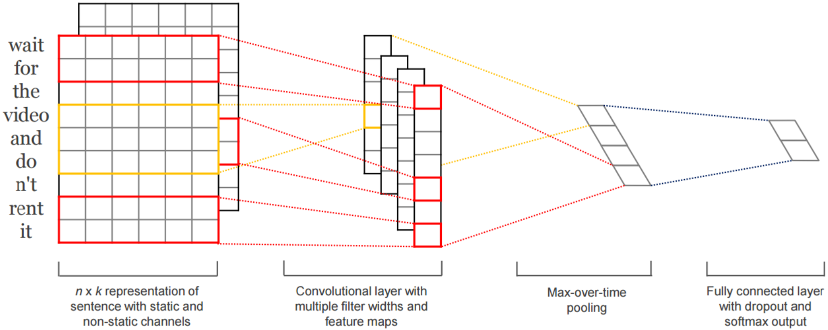
$$NN_{conv}(\mathbf{X}) = \tanh \begin{bmatrix} NN_{window}(\mathbf{X}_{1:d_{\mathrm{win}}}) \\ NN_{window}(\mathbf{X}_{2:d_{\mathrm{win}}+1}) \\ \vdots \\ NN_{window}(\mathbf{X}_{n-d_{\mathrm{win}}:n}) \end{bmatrix}$$

Review: Pooling

- Unfortunately $NN_{conv} : \mathbb{R}^{n \times d^0} \mapsto \mathbb{R}^{(n - d_{\mathrm{win}} + 1) \times d_{\mathrm{hid}}}$.
- Need to map down to d_{out} for different n
- Recall pooling operations.
- Pooling "over-time" operations $f : \mathbb{R}^{n \times m} \mapsto \mathbb{R}^{1 \times m}$
  1. $f_{max}(\mathbf{X})_{1,j} = \max_i X_{i,j}$
  2. $f_{min}(\mathbf{X})_{1,j} = \min_i X_{i,j}$
  3. $f_{mean}(\mathbf{X})_{1,j} = \sum_i X_{i,j} / n$

$$f(\mathbf{X}) = \begin{bmatrix} \Downarrow & \Downarrow & \dots \\ \Downarrow & \Downarrow & \dots \\ & \vdots & \\ \Downarrow & \Downarrow & \dots \end{bmatrix} = \begin{bmatrix} & \dots & \end{bmatrix}$$

Review: Convolution Diagram (Kim, 2014)



- $n = 9$, $d_{\text{hid}} = 4$, $d_{\text{out}} = 2$
- red- $d_{\text{win}} = 2$, blue- $d_{\text{win}} = 3$, (ignore back channel)

Quiz

Normally when we use a convolution layer we set d_{win} to a small constant. However you could also set it to degenerate values. Describe what model you get when you use the following variants on the standard convolution layer.

- $d_{\mathrm{win}} = 1$ with sparse word features and no pooling or non-linearity.
- same as above with sum-over-time pooling
- $d_{\mathrm{win}} = n$ (length of sentence) and no pooling.

Answer

- This is simply an embedding layer! Here, the number of filters is the same as the embedding size d_{emb}.
- This is a continuous bag-of-words model. The convolution acts as the embedding and then the pooling is the sum of the embeddings
- This is the same as a concatenation of the embedding features followed by a linear layer. The linear layer has different values for each position.

Representation of Sequence

- Many tasks in NLP involve sequences

$$w_1, \ldots, w_n$$

- Representations as matrix dense vectors $\mathbf{X}$
  (Following YG, slight abuse of notation)

$$\mathbf{x}_1 = \mathbf{x}_1^0 \mathbf{W}^0, \ldots, \mathbf{x}_n = \mathbf{x}_n^0 \mathbf{W}^0$$

- Would like fixed-dimensional representation.

Pooling over time?

- Pooling-over-time gives a fixed-dimensional value.
- However has issues.
- How does convolution help here? What doesn't it do?

Text Classification

Consider this (contrived) example:

`How can you not see this movie?`

`You should not see this movie.`

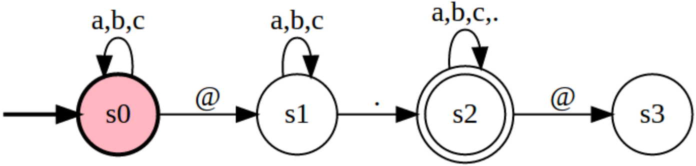
- Would like to classify them differently, despite similar bigrams
- Generally want to have memory when making decisions.

Contents

Finite State Models

- Simple, classical way of representing memory
- Current state representation saves necessary past information.

Example: Email Address Parsing



Deterministic Finite State Machine Formally

- $\mathcal{S}$; set of possible states
- Σ; vocabulary
- $s_0 \in \mathcal{S}$; start state
- $R : (\mathcal{S}, \Sigma) \rightarrow \mathcal{S}$; transition function

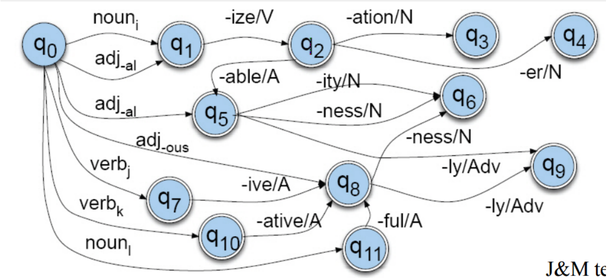
- Maps input $w_1, \ldots, w_n$ to states $s_1, \ldots s_n$
- For all $i \in \{1, \ldots, n\}$

$$s_i = R(s_{i-1}, w_i)$$

Finite State Machines in NLP

- words to phonemes in speech
- n-gram language models
- manual part-of-speech taggers
- word morphology

Example: Morphology



Variants of State Machines

- Acceptors; make decision based on final state s_n
- Transducers; apply function $y_i = O(s_i)$ to produce output at each intermediary state
- Encoders; utilize last state s_n in another model

Also interesting:

- Ways to learn finite state machine structure
- Learning weighted finite state machines

Contents

Recurrent Neural Networks

- Motivation is to maintain history in the model
- Neural network models with “memory”
- However no longer finite in the same sense.

Hidden State

- $\mathcal{S} = \mathbb{R}^{d_{\text{hid}}}$; hidden state space
- $\Sigma = \mathbb{R}^{d_{\text{in}}}$; input state space
- $s_0 \in \mathcal{S}$; initial state vector
- $R : (\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{hid}}}) \mapsto \mathbb{R}^{d_{\text{hid}}}$; parameterized transition function
- How might we define R?

$$NN_{elman}(\mathbf{x}, \mathbf{s}) = \tanh([\mathbf{x}, \mathbf{s}]\mathbf{W} + \mathbf{b})$$

Hidden State

- $\mathcal{S} = \mathbb{R}^{d_{\text{hid}}}$; hidden state space
- $\Sigma = \mathbb{R}^{d_{\text{in}}}$; input state space
- $s_0 \in \mathcal{S}$; initial state vector
- $R : (\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{hid}}}) \mapsto \mathbb{R}^{d_{\text{hid}}}$; parameterized transition function
- How might we define R?

$$NN_{elman}(\mathbf{x}, \mathbf{s}) = \tanh([\mathbf{x}, \mathbf{s}]\mathbf{W} + \mathbf{b})$$

Sequence Recurrence

- Can map from dense sequence to dense representation.
- $\mathbf{x}_1, \dots, \mathbf{x}_n \mapsto \mathbf{s}_1, \dots, \mathbf{s}_n$
- For all $i \in \{1, \dots, n\}$

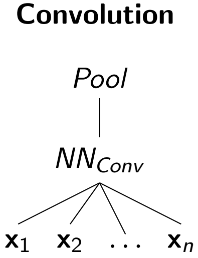
$$\mathbf{s}_i = R(\mathbf{s}_{i-1}, \mathbf{x}_i; \theta)$$

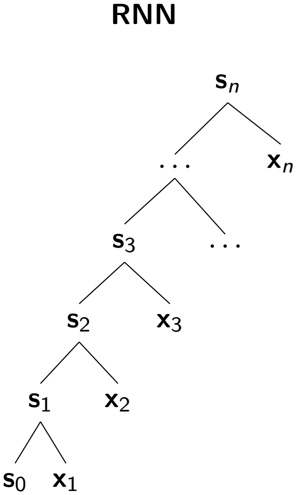
- θ is shared by all R

Example:

$$\begin{aligned}
\mathbf{s}_4 &= R(\mathbf{s}_3, \mathbf{x}_4) \\
&= R(R(\mathbf{s}_2, \mathbf{x}_3), \mathbf{x}_4) \\
&= R(R(R(R(\mathbf{s}_0, \mathbf{x}_1), \mathbf{x}_2), \mathbf{x}_3), \mathbf{x}_4)
\end{aligned}$$

RNN versus Convolution and Pooling





Using Recurrent Neural Networks

- Hidden states can be applied in different ways.
- Can be used similarly to finite machines
  - Acceptor
  - Transducer
  - Encoder

Using RNNs: Acceptor

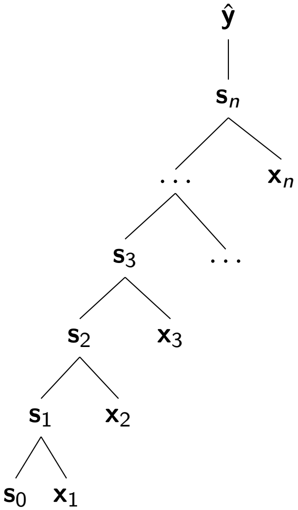
- Simplest case, sentence acceptor:

$$\hat{boldy} = O(\mathbf{s}_n) = \text{softmax}(\mathbf{s}_n\mathbf{W} + \mathbf{b})$$

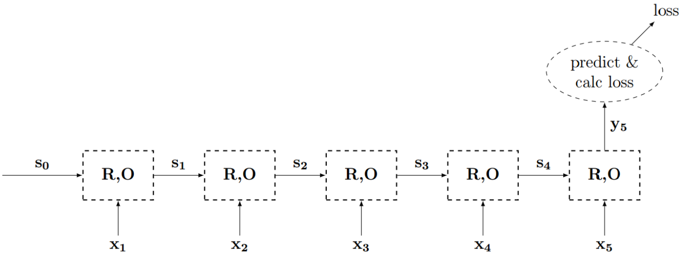
- $O : \mathbb{R}^{d_{\text{hid}}} \mapsto \mathbb{R}^{d_{\text{out}}}$; final layer

- Can be applied to text classification-like tasks

Using RNNs: Acceptor Architecture



Using RNNs: Acceptor (LR version, YG)



Acceptor Versus Convolution

- In theory, acceptor can model arbitrarily long sequences.
- Memory allows it to incorporate long-range info.
- Convolution can be run in parallel, multiple dimensions
- Convolution is much shallower, easier to train

Contents

How do we learn the model?

- RNNs are trained with SGD and Backprop (surprise)
- Implementation can be complicated, mainly for efficiency.
- Called *backpropagation through time* (BPTT).

Training Acceptors

Training process:

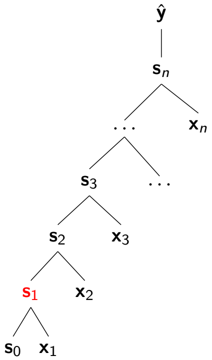
- Run forward propagation.
- Run backward propagation.
- Update all weights

Weights θ of R are shared:

$$\frac{\partial L}{\partial \theta} = \sum_{i=1}^{n} \frac{\partial L(\ldots R(\mathbf{x}_i, \mathbf{s}_{i-1}))}{\partial \theta}$$

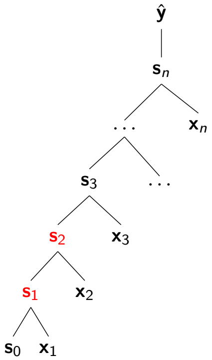
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



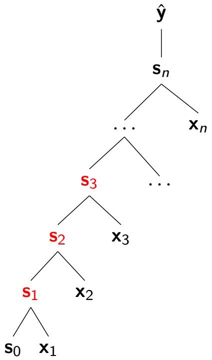
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



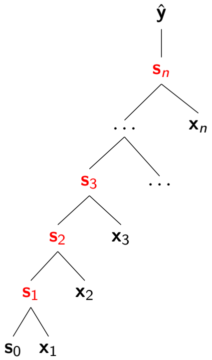
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



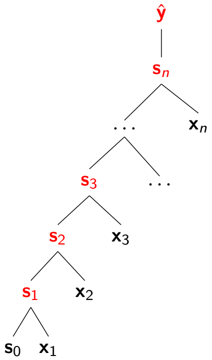
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



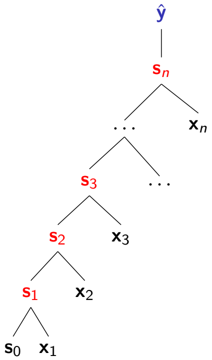
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



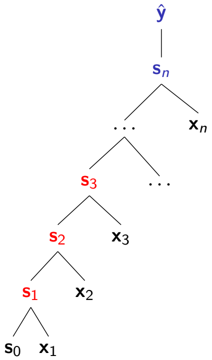
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



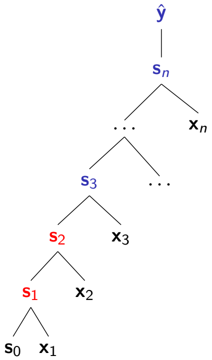
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



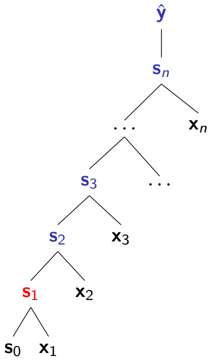
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



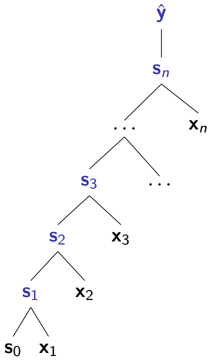
BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



BPTT (Acceptor)

- Run forward propagation.
- Run backward propagation.
- Update all weights (shared)



Issues

- Can be inefficient, but batch/GPUs help.
- Model is much deeper than previous approaches.
  - This matters a lot, focus of next class.
- Variable-size model for each sentence.
  - Have to be a bit more clever in Torch.

Contents

RNN for Language Modeling

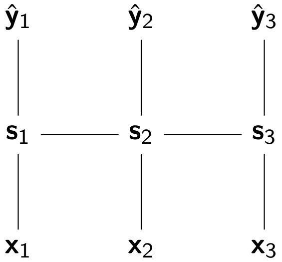
- Recent popularization of RNNs has been based on language modeling (Mikolov, 2012)
- In particular RNNs allow for non-Markovian models

$$p(w_i | w_1, \ldots, w_{i-1}; \theta) = O(\mathbf{s}_i)$$

- Compare this to the feed-forward windowed approach.

$$p(w_i | w_{i-n+1}, \ldots, w_{i-1}; \theta) = O(\mathbf{s}_i)$$

RNN as Transducer



- Can reuse hidden state each time

$$p(w_i | w_1, \ldots, w_{i-1}; \theta) = O(\mathbf{s}_i) = O(R(\mathbf{s}_{i-1}, \mathbf{x}_i))$$

$$p(w_{i+1} | w_1, \ldots, w_i; \theta) = O(R(\mathbf{s}_i, \mathbf{x}_{i+1}))$$

Transducers Formally

- Prediction next $\hat{\mathbf{y}}_i$ as we go
- For all $i \in \{1, \ldots, n\}$

$$\hat{\mathbf{y}}_i = O(\mathbf{s}_i) = \mathrm{softmax}(\mathbf{s}_i \mathbf{W} + \mathbf{b})$$

- $O : \mathbb{R}^{d_{\mathrm{hid}}} \mapsto \mathbb{R}^{d_{\mathrm{out}}}$

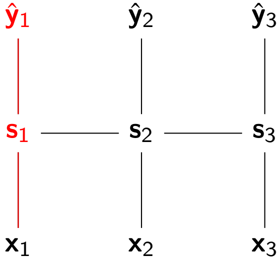
BPTT Transducer Training

- Run forward propagation.
- Run backward propagation
- Update all weights (shared)

$$
\begin{tikzcd}
\hat{\mathbf{y}}_1 & \hat{\mathbf{y}}_2 & \hat{\mathbf{y}}_3 \\
\mathbf{s}_1 \arrow[u, dash] \arrow[r, dash] & \mathbf{s}_2 \arrow[u, dash] \arrow[r, dash] & \mathbf{s}_3 \arrow[u, dash] \\
\mathbf{x}_1 \arrow[u, dash] & \mathbf{x}_2 \arrow[u, dash] & \mathbf{x}_3 \arrow[u, dash]
\end{tikzcd}
$$

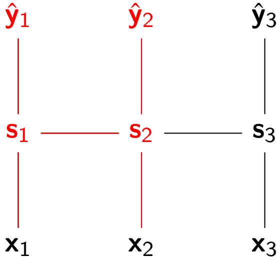
BPTT Transducer Training

- Run forward propagation.
- Run backward propagation
- Update all weights (shared)



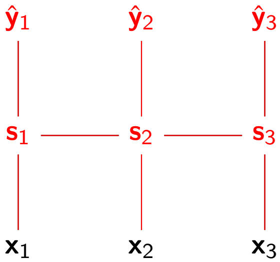
BPTT Transducer Training

- Run forward propagation.
- Run backward propagation
- Update all weights (shared)



BPTT Transducer Training

- Run forward propagation.
- Run backward propagation
- Update all weights (shared)



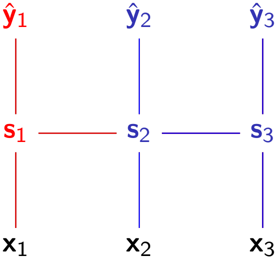
BPTT Transducer Training

- Run forward propagation.
- Run backward propagation
- Update all weights (shared)

$$\begin{array}{ccc} \hat{\mathbf{y}}_1 & \hat{\mathbf{y}}_2 & \hat{\mathbf{y}}_3 \\ | & | & | \\ \mathbf{s}_1 & - \mathbf{s}_2 - & \mathbf{s}_3 \\ | & | & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{array}$$

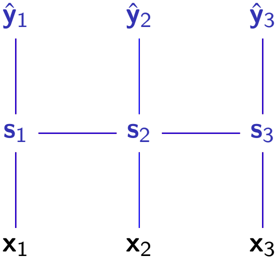
BPTT Transducer Training

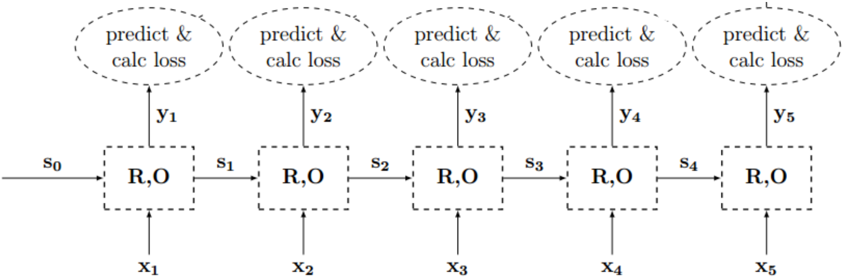
- Run forward propagation.
- Run backward propagation
- Update all weights (shared)



BPTT Transducer Training

- Run forward propagation.
- Run backward propagation
- Update all weights (shared)




predict & calc loss
predict & calc loss
predict & calc loss
predict & calc loss
predict & calc loss
$\mathbf{y_1}$
$\mathbf{y_2}$
$\mathbf{y_3}$
$\mathbf{y_4}$
$\mathbf{y_5}$
$\mathbf{s_0}$
R,O
$\mathbf{s_1}$
R,O
$\mathbf{s_2}$
R,O
$\mathbf{s_3}$
R,O
$\mathbf{s_4}$
R,O
$\mathbf{x_1}$
$\mathbf{x_2}$
$\mathbf{x_3}$
$\mathbf{x_4}$
$\mathbf{x_5}$

Bidirectional RNNs

- RNNs compute a prefix representation.
- But for tagging we used a bidirectional window.
- How can we get a postfix representation?

$$w_1 \, w_2 \left[w_3 \, w_4 \, w_5 \, w_6 \, w_7 \, w_8 \right]$$

Bidirectional Models

- For all $i \in \{1, \ldots, n\}$

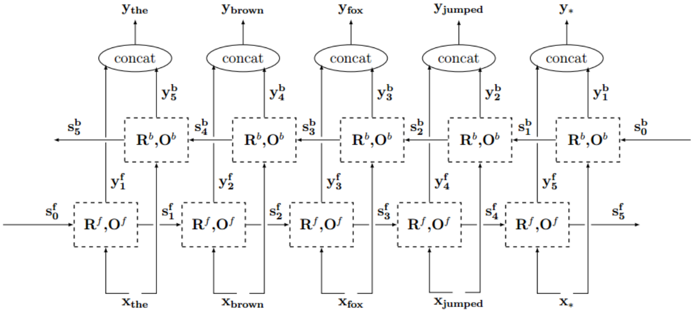
$$\mathbf{s}_i^f = R^f(\mathbf{s}_{i-1}, \mathbf{x}_i)$$

- For all $i \in \{1, \ldots, n\}$

$$\mathbf{s}_i^b = R^b(\mathbf{s}_{i+1}, \mathbf{x}_i)$$

- For all $i \in \{1, \ldots, n\}$

$$\hat{\mathbf{y}}_i = O([\mathbf{s}_i^b, \mathbf{s}_i^f]) = [\mathbf{s}_i^b, \mathbf{s}_i^f]\mathbf{W} + \mathbf{b}$$



$\mathbf{y_{the}}$
$\mathbf{y_{brown}}$
$\mathbf{y_{fox}}$
$\mathbf{y_{jumped}}$
$\mathbf{y_{*}}$
concat
concat
concat
concat
concat
$\mathbf{y^b_5}$
$\mathbf{y^b_4}$
$\mathbf{y^b_3}$
$\mathbf{y^b_2}$
$\mathbf{y^b_1}$
$\mathbf{s^b_5}$
$\mathbf{s^b_4}$
$\mathbf{s^b_3}$
$\mathbf{s^b_2}$
$\mathbf{s^b_1}$
$\mathbf{s^b_0}$
$\mathbf{R}^b,\mathbf{O}^b$
$\mathbf{y^f_1}$
$\mathbf{y^f_2}$
$\mathbf{y^f_3}$
$\mathbf{y^f_4}$
$\mathbf{y^f_5}$
$\mathbf{s^f_0}$
$\mathbf{s^f_1}$
$\mathbf{s^f_2}$
$\mathbf{s^f_3}$
$\mathbf{s^f_4}$
$\mathbf{s^f_5}$
$\mathbf{R}^f,\mathbf{O}^f$
$\mathbf{x_{the}}$
$\mathbf{x_{brown}}$
$\mathbf{x_{fox}}$
$\mathbf{x_{jumped}}$
$\mathbf{x_{*}}$

Bidirection Models

Many applications:

- Tagging
- Handwriting Recognition (given full sentence)
- Speech Recognition (given full utterance)
- Machine Translation