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
```

$$\mathbf{x} = v(\text{word:A}) + v_2(\text{bigram:A:sentimental})$$

$$+ v(\text{word:sentimental}) + v_2(\text{bigram:sentimental:mess})$$

$$+ v(\text{word:mess})$$
Let our input be the embeddings of the full sentence, \( X \in \mathbb{R}^{n \times d^0} \)

\[
X = [v(w_1), v(w_2), v(w_3), \ldots, v(w_n)]
\]

Define a window model as \( \mathcal{N}\mathcal{N}_{\text{window}} : \mathbb{R}^{1 \times (d_{\text{win}}d^0)} \mapsto \mathbb{R}^{1 \times d_{\text{hid}}} \),

\[
\mathcal{N}\mathcal{N}_{\text{window}}(x_{\text{win}}) = x_{\text{win}}W^1 + b^1
\]

The convolution is defined as \( \mathcal{N}\mathcal{N}_{\text{conv}} : \mathbb{R}^{n \times d^0} \mapsto \mathbb{R}^{(n-d_{\text{win}}+1) \times d_{\text{hid}}} \),

\[
\mathcal{N}\mathcal{N}_{\text{conv}}(X) = \tanh \left[ \begin{array}{c}
\mathcal{N}\mathcal{N}_{\text{window}}(X_{1:d_{\text{win}}}) \\
\mathcal{N}\mathcal{N}_{\text{window}}(X_{2:d_{\text{win}}+1}) \\
\vdots \\
\mathcal{N}\mathcal{N}_{\text{window}}(X_{n-d_{\text{win}}:n})
\end{array} \right]
\]
Review: Pooling

- Unfortunately $\mathcal{NN}_{\text{conv}} : \mathbb{R}^{n \times d^0} \mapsto \mathbb{R}^{(n-d_{\text{win}}+1) \times d_{\text{hid}}}$.

- Need to map down to $d_{\text{out}}$ for different $n$

- Recall pooling operations.

- Pooling “over-time” operations $f : \mathbb{R}^{n \times m} \mapsto \mathbb{R}^{1 \times m}$
  1. $f_{\text{max}}(X)_{1,j} = \max_i X_{i,j}$
  2. $f_{\text{min}}(X)_{1,j} = \min_i X_{i,j}$
  3. $f_{\text{mean}}(X)_{1,j} = \sum_i X_{i,j} / n$

$$f(X) = \begin{bmatrix} \downarrow & \downarrow & \cdots \\ \downarrow & \downarrow & \cdots \\ & & \vdots \\ \downarrow & \downarrow & \cdots \end{bmatrix} = \begin{bmatrix} \cdots \end{bmatrix}$$
- \( n = 9, \ d_{\text{hid}} = 4 \), \( d_{\text{out}} = 2 \)

- \( \text{red-} \ d_{\text{win}} = 2, \ \text{blue-} \ d_{\text{win}} = 3 \), (ignore back channel)
Normally when we use a convolution layer we set $d_{\text{win}}$ 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.

- $d_{\text{win}} = 1$ with sparse word features and no pooling or non-linearity.
- same as above with sum-over-time pooling
- $d_{\text{win}} = n$ (length of sentence) and no pooling.
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)
  \[
  x_1 = x_1^0 \mathbf{W}^0, \ldots, x_n = x_n^0 \mathbf{W}^0
  \]

- 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 Machines

Recurrent Neural Networks

Training RNNs

RNN Variants
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
- $\Sigma$; 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

Finite State Machines

Recurrent Neural Networks

Training RNNs

RNN Variants
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 $R$?

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

- \( S = \mathbb{R}^{d_{\text{hid}}}; \) hidden state space
- \( \Sigma = \mathbb{R}^{d_{\text{in}}}; \) input state space
- \( s_0 \in S; \) 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 \( R? \)

\[
NN_{\text{elman}}(x, s) = \tanh([x, s]W + b)
\]
Sequence Recurrence

- Can map from dense sequence to dense representation.
  \[ x_1, \ldots, x_n \mapsto s_1, \ldots, s_n \]

- For all \( i \in \{1, \ldots, n\} \)
  \[ s_i = R(s_{i-1}, x_i; \theta) \]

- \( \theta \) is shared by all \( R \)

Example:

\[
\begin{align*}
  s_4 &= R(s_3, x_4) \\
        &= R(R(s_2, x_3), x_4) \\
        &= R(R(R(s_0, x_1), x_2), x_3), x_4)
\end{align*}
\]
RNN versus Convolution and Pooling

Convolution

Pool

NN_{Conv}

x_1 \ x_2 \ldots \ x_n

RNN

\ldots

s_n

\ldots

s_3

\ldots

s_2

\ldots

s_1

\ldots

s_0

x_1 \ x_2 \ x_3 \ x_n
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:

\[
\hat{b}o\text{ldy} = O(s_n) = \text{softmax}(s_nW + b)
\]

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

- Can be applied to text classification-like tasks
Using RNNs: Acceptor Architecture

\[
\begin{align*}
\hat{y} & \quad \text{sn} \\
& \quad \ldots \\
& \quad s_3 \\
& \quad \ldots \\
& \quad s_2 \\
& \quad \ldots \\
& \quad s_1 \\
& \quad \ldots \\
& \quad s_0
\end{align*}
\]
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

Finite State Machines

Recurrent Neural Networks

Training RNNs

RNN Variants
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 $\theta$ of $R$ are shared:

$$\frac{\partial L}{\partial \theta} = \sum_{i=1}^{n} \frac{\partial L(\ldots R(x_i, s_{i-1}))}{\partial \theta}$$
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

Finite State Machines

Recurrent Neural Networks

Training RNNs

RNN Variants
Recent popularization of RNNs has been based on language modeling (Mikolov, 2012).

In particular RNNs allow for non-Markovian models:

\[ p(w_i|w_1, \ldots, w_{i-1}; \theta) = O(s_i) \]

Compare this to the feed-forward windowed approach:

\[ p(w_i|w_{i-n+1}, \ldots, w_{i-1}; \theta) = O(s_i) \]
RNN as Transducer

- Can reuse hidden state each time

\[
p(w_i|w_1, \ldots, w_{i-1}; \theta) = O(s_i) = O(R(s_{i-1}, x_i))
\]

\[
p(w_{i+1}|w_1, \ldots, w_i; \theta) = O(R(s_i, x_{i+1}))
\]
Transducers Formally

- Prediction next $\hat{y}_i$ as we go

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

  $$\hat{y}_i = O(s_i) = \text{softmax}(s_iW + b)$$

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

\[
\begin{align*}
\hat{y}_1 & \quad \hat{y}_2 & \quad \hat{y}_3 \\
\hat{s}_1 & \quad \hat{s}_2 & \quad \hat{s}_3 \\
\hat{x}_1 & \quad \hat{x}_2 & \quad \hat{x}_3
\end{align*}
\]
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)
Bidirectional RNNs

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

\[ w_1 w_2 [w_3 w_4 w_5 w_6 w_7 w_8] \]
Bidirectional Models

- For all \( i \in \{1, \ldots, n\} \)
  \[
    s_i^f = R^f(s_{i-1}, x_i)
  \]

- For all \( i \in \{1, \ldots, n\} \)
  \[
    s_i^b = R^b(s_{i+1}, x_i)
  \]

- For all \( i \in \{1, \ldots, n\} \)
  \[
    \hat{y}_i = O([s_i^b, s_i^f]) = [s_i^b, s_i^f]W + b
  \]
Bidirection Models

Many applications:

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