Structured Neural Networks (I)

CS 690N, Spring 2018
Advanced Natural Language Processing
http://people.cs.umass.edu/~brenocon/anlp2018/

Brendan O’Connor
College of Information and Computer Sciences
University of Massachusetts Amherst
Structured neural networks?

- How to deal with arbitrary length inputs?
  - Documents, sentences, long-distance history
- Build structure directly into network architectures
  - Convolutional
  - Recurrent
- Dynamic autodiff frameworks make training easy(-ish) (PyTorch, DyNet)
“Averaging” network

- Continuous Bag-of-Words

\[
\text{CBOW}(f_1, \ldots, f_k) = \frac{1}{k} \sum_{i=1}^{k} v(f_i)
\]

- Use averaged representation for e.g. softmax classifier
- Example: FastText doc classifier (Joulin et al. 2016)
  - Pre-trained word embeddings
  - Bag-of-words, Bag-of-ngrams
  - Hashing (ngram embeddings randomly shared)
  - Hierarchical softmax speed trick
- With >100k sentiment labeled training docs, performs better than explicit feature logistic regression
Convolutional NN

- Sentence representation independent of sentence length:
  - Sliding window of concatenated word embeddings
  - Feedforward transform then elementwise max across positions
- Final sentence representation could be used in various ways: e.g. classification (Kim 2014). Use joint training.
- Only learns local dependencies (like n-grams)

**Figure 4:** 1d convolution+pooling over the sentence "the quick brown fox jumped over the lazy dog". This is a narrow convolution (no padding is added to the sentence) with a window size of 3. Each word is translated to a 2-dim embedding vector (not shown). The embedding vectors are then concatenated, resulting in 6-dim window representations. Each of the seven windows is transferred through a 6 ⇥ 3 filter (linear transformation followed by element-wise tanh), resulting in seven 3-dimensional filtered representations. Then, a max-pooling operation is applied, taking the max over each dimension, resulting in a final 3-dimensional pooled vector.

The resulting vector $c$ is a representation of the sentence in which each dimension reflects the most salient information with respect to some prediction task. $c$ is then fed into a downstream network layers, perhaps in parallel to other vectors, culminating in an output layer which is used for prediction. The training procedure of the network calculates the loss with respect to the prediction task, and the error gradients are propagated all the way back through the pooling and convolution layers, as well as the embedding layers.

Besides being useful for prediction, a by-product of the training procedure is a set of parameters $W$, $B$ and embeddings $v()$ that can be used in a convolution and pooling architecture to encode arbitrary length.
Some authors treat the output at position to pass, while those corresponding to near-zero values are blocked by using a sigmoid function. Indices in then added to another vector. The values of range \([0, 1]\) that is multiplied component-wise with another vector.

Memory cell should be forgotten. Concretely, a gate problem. The main idea behind the LSTM is to introduce as part of the state representation architecture (Hochreiter & Schmidhuber, 1997) was designed to solve the vanishing gradients problem. The Long Short-Term Memory (LSTM) is designed to capture long-range dependencies. The S-RNN to capture long-range dependencies. The S-RNN is hard to train effectively because of the vanishing gradients problem (Pascanu et al., 2012). Error signals (gradients) in later steps in the sequence diminish quickly in back-propagation process, and do not reach earlier input signals, making it hard for training is to set the parameters of for the task we are tying to solve.

In spite of its simplicity, the Simple RNN provides strong results for sequence tagging. That is, the state at position that is multiplied component-wise with another vector. The simplest RNN formulation, known as an Elman Network or Simple-RNN (S-RNN), was proposed by Elman (1990) and explored for use in language modeling by Mikolov (2012). Simple RNNs for language modeling, see the PhD thesis by Mikolov (2012).

Recurrent Unit

- Simple (“vanilla”) RNN (Elman 1990)

\[
\begin{align*}
  s_i &= R_{SRNN}(s_{i-1}, x_i) = g(x_i W^x + s_{i-1} W^s + b) \\
  y_i &= O_{SRNN}(s_i) = s_i
\end{align*}
\]

\(s_i, y_i \in \mathbb{R}^{d_s}, x_i \in \mathbb{R}^{d_x}, W^x \in \mathbb{R}^{d_x \times d_s}, W^s \in \mathbb{R}^{d_s \times d_s}, b \in \mathbb{R}^{d_s}\)

- Other local models: LSTM and GRU

Figure 6: Graphical representation of an RNN (unrolled).
RNN Uses

- **Acceptor**

![Figure 7: Acceptor RNN Training Graph.](image_url)

- **Transducer**

![Figure 8: Transducer RNN Training Graph.](image_url)
RNN Uses

- Encoder-decoder

RNN Uses

Goldberg history. The power of the ability to condition on arbitrarily long histories is demonstrated in generative character-level RNN models, in which a text is generated character by character, each character conditioning on the previous ones (Sutskever, Martens, & Hinton, 2011). The generated texts show sensitivity to properties that are not captured by n-gram language models, including line lengths and nested parenthesis balancing. For a good demonstration and analysis of the properties of RNN-based character level language models, see the work of Karpathy, Johnson, and Li (2015).

10.2.4 Encoder-Decoder

Finally, an important special case of the encoder scenario is the Encoder-Decoder framework (Cho, van Merrienboer, Bahdanau, & Bengio, 2014a; Sutskever et al., 2014). The RNN is used to encode the sequence into a vector representation \( y_n \), and this vector representation is then used as auxiliary input to another RNN that is used as a decoder. For example, in a machine-translation setup the first RNN encodes the source sentence into a vector representation \( y_n \), and then this state vector is fed into a separate (decoder) RNN that is trained to predict (using a transducer-like language modeling objective) the words of the target language sentence based on the previously predicted words as well as \( y_n \). The supervision happens only for the decoder RNN, but the gradients are propagated all the way back to the encoder RNN (see Figure 9).

Figure 9: Encoder-Decoder RNN Training Graph.
Language Modelling: Review

Language models aim to represent the history of observed text \((w_1, \ldots, w_{t-1})\) succinctly in order to predict the next word \((w_t)\):

- With count based n-gram LMs we approximate the history with just the previous \(n\) words.
- Neural n-gram LMs embed the same fixed n-gram history in a continues space and thus capture correlations between histories.
- With Recurrent Neural Network LMs we drop the fixed n-gram history and compress the entire history in a fixed length vector, enabling long range correlations to be captured.
Capturing Long Range Dependencies

If an RNN Language Model is to outperform an n-gram model it must discover and represent long range dependencies:

\[ p(\text{sandcastle} \mid \text{Alice went to the beach. There she built a}) \]

While a simple RNN LM can represent such dependencies in theory, can it learn them?
RNNs: Exploding and Vanishing Gradients

Consider the path of partial derivatives linking a change in cost$ _4$ to changes in $h_1$:

\[ h_n = g(V[x_n; h_{n-1}] + c) \]
\[ \hat{p}_n = \text{softmax}(Wh_n + b) \]

\[
\frac{\partial \text{cost}_4}{\partial h_1} = \frac{\partial \text{cost}_4}{\partial \hat{p}_4} \frac{\partial \hat{p}_4}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}
\]
RNNs: Exploding and Vanishing Gradients

Consider the path of partial derivatives linking a change in $\text{cost}_N$ to changes in $h_1$:

$$h_n = g(V[x_n; h_{n-1}] + c)$$

$$\hat{p}_n = \text{softmax}(Wh_n + b)$$

$$\frac{\partial \text{cost}_N}{\partial h_1} = \frac{\partial \text{cost}_N}{\partial \hat{p}_N} \frac{\partial \hat{p}_N}{\partial h_N} \left( \prod_{n \in \{N, \ldots, 2\}} \frac{\partial h_n}{\partial h_{n-1}} \right)$$
RNNs: Exploding and Vanishing Gradients

Consider the path of partial derivatives linking a change in cost \( N \) to changes in \( h_1 \):

\[
h_n = g(V[x_n; h_{n-1}] + c), \quad \frac{\partial \text{cost}_N}{\partial h_1} = \frac{\partial \text{cost}_N}{\partial \hat{p}_N} \frac{\partial \hat{p}_N}{\partial h_N} \left( \prod_{n \in \{N,\ldots,2\}} \frac{\partial h_n}{\partial h_{n-1}} \right)
\]
RNNs: Exploding and Vanishing Gradients

Consider the path of partial derivatives linking a change in $\text{cost}_N$ to changes in $h_1$:

$$h_n = g(V_x x_n + V_h h_{n-1} + c), \quad \frac{\partial \text{cost}_N}{\partial h_1} = \frac{\partial \text{cost}_N}{\partial \hat{p}_N} \frac{\partial \hat{p}_N}{\partial h_N} \left( \prod_{n \in \{N,...,2\}} \frac{\partial h_n}{\partial z_n} \frac{\partial z_n}{\partial h_{n-1}} \right)$$
Consider the path of partial derivatives linking a change in $\text{cost}_N$ to changes in $h_1$:

$$h_n = g(V_x x_n + V_h h_{n-1} + c),$$

$$\frac{\partial \text{cost}_N}{\partial h_1} = \frac{\partial \text{cost}_N}{\partial \hat{p}_N} \frac{\partial \hat{p}_N}{\partial h_n} \left( \prod_{n \in \{N, \ldots, 2\}} \frac{\partial h_n}{\partial z_n} \frac{\partial z_n}{\partial h_{n-1}} \right)$$

$$\frac{\partial h_n}{\partial z_n} = \text{diag} \left( g'(z_n) \right)$$

$$\frac{\partial z_n}{\partial h_{n-1}} = V_h$$
RNNs: Exploding and Vanishing Gradients

Consider the path of partial derivatives linking a change in cost\(_N\) to changes in \(h_1\):

\[
h_n = g\left(V_x x_n + V_h h_{n-1} + c\right), \quad \frac{\partial \text{cost}_N}{\partial h_1} = \frac{\partial \text{cost}_N}{\partial \hat{p}_N} \frac{\partial \hat{p}_N}{\partial h_N} \left(\prod_{n \in \{N,\ldots,2\}} \frac{\partial h_n}{\partial z_n} \frac{\partial z_n}{\partial h_{n-1}}\right)
\]

\[
\frac{\partial h_n}{\partial z_n} = \text{diag} \left(g'(z_n)\right) \quad \frac{\partial z_n}{\partial h_{n-1}} = V_h
\]

\[
\frac{\partial h_n}{\partial h_{n-1}} = \frac{\partial h_n}{\partial z_n} \frac{\partial z_n}{\partial h_{n-1}} = \text{diag} \left(g'(z_n)\right) V_h
\]
The core of the recurrent product is the repeated multiplication of $V_h$. If the largest eigenvalue of $V_h$ is:

- 1, then gradient will propagate,
- $> 1$, the product will grow exponentially (explode),
- $< 1$, the product shrinks exponentially (vanishes).
**LSTM** (Long short-term memory)

- **Goals:**
  1. Be able to “remember” for longer distances
  2. Stable backpropagation during training
  3. Augment individual timesteps with a number of specialized vectors and gating functions (Simpler alternative: GRU. But LSTM is most standard.)

- **Main state**
  - $c$: Memory cell
  - $h$: Hidden state

- **Update system**
  - $g$: proposed new values
  - $f$, $i$, $o$: Forget, Input, Output gates control acceptance of $g$ into new state

\[
\begin{align*}
    c_j &= c_{j-1} \odot f + g \odot i \\
    h_j &= \tanh(c_j) \odot o \\
    i &= \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \\
    f &= \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \\
    o &= \sigma(x_j W^{xo} + h_{j-1} W^{ho}) \\
    g &= \tanh(x_j W^{xg} + h_{j-1} W^{hg})
\end{align*}
\]

Christopher Olah: Understanding LSTM Networks
[colah.github.io/posts/2015-08-Understanding-LSTMs/]
memory component ("cell")

\[ c_{j-1} \quad c_j \]

\[ h_{j-1} \quad h_j \quad \text{hidden state} \]

\[ x_j \quad \text{input} \]

- Main Information
- Gating Function
Mathematically, the LSTM architecture is defined as:

$$s_j = R_{lstm}(s_{j-1}, x_j) = [c_j; h_j]$$

$$c_j = c_j^{t-1} \cdot f + g_i \cdot h_j$$

$$h_j = \text{tanh}(c_j)$$

$$i = \sigma(x_j W_{xi} + h_{j-1} W_{hi})$$

$$f = \sigma(x_j W_{xf} + h_{j-1} W_{hf})$$

$$o = \sigma(x_j W_{xo} + h_{j-1} W_{ho})$$

$$g = \text{tanh}(x_j W_{xg} + h_{j-1} W_{hg})$$

$$y_j = O_{lstm}(s_j) = h_j$$

The symbol \( \cdot \) is used to denote component-wise product. The state at time \( j \) is composed of two vectors, \( c_j \) and \( h_j \), where \( c_j \) is the memory component and \( h_j \) is the hidden state component. There are three gates, \( i \), \( f \) and \( o \), controlling for input, forget and output. The gate values are computed based on linear combinations of the current input \( x_j \) and the previous state \( h_{j-1} \), passed through a sigmoid activation function. An update candidate \( g \) is computed as a linear combination of \( x_j \) and \( h_{j-1} \), passed through a tanh activation function. The memory \( c_j \) is then updated: the forget gate controls how much of the previous memory to keep (\( c_j^{t-1} \cdot f \)), and the input gate controls how much of the proposed update to keep (\( g_i \cdot h_j \)). Finally, the value of \( h_j \) (which is also the output \( y_j \)) is determined based on the content of the memory \( c_j \), passed through a tanh non-linearity and controlled by the output gate. The gating mechanisms allow for gradients related to the memory part to stay high across very long time ranges.

For further discussion on the LSTM architecture see the PhD thesis by Alex Graves (2008), as well as the online-post by Olah (2015b). For an analysis of the behavior of an LSTM when used as a character-level language model, see the work of Karpathy et al. (2015).

For further explanation of the motivation behind the gating mechanism in the LSTM (and the GRU) and its relation to solving the vanishing gradient problem in recurrent neural networks, see Sections 4.2 and 4.3 in the detailed course notes of Cho (2015).

LSTMs are currently the most successful type of RNN architecture, and they are responsible for many state-of-the-art sequence modeling results. The main competitor of the LSTM-RNN is the GRU, to be discussed next.
Mathematically, the LSTM architecture is defined as:

\[
\begin{align*}
    s_j &= R_{lstm}(s_{j-1}, x_j) = [c_j; h_j] \\
    c_j &= c_{j-1} f + g_i h_j = \tanh(c_j) \\
    o_i &= (x_j W_{xi} + h_{j-1} W_{hi}) \\
    f &= (x_j W_{xf} + h_{j-1} W_{hf}) \\
    o &= (x_j W_{xo} + h_{j-1} W_{ho}) \\
    g &= \tanh(x_j W_{xg} + h_{j-1} W_{hg}) \\
    y_j &= O_{lstm}(s_j) = h_j
\end{align*}
\]

The symbol \( \cdot \) is used to denote component-wise product. The state at time \( j \) is composed of two vectors, \( c_j \) and \( h_j \), where \( c_j \) is the memory component and \( h_j \) is the hidden state component. There are three gates, \( i \), \( f \) and \( o \), controlling for input, forget and output. The gate values are computed based on linear combinations of the current input \( x_j \) and the previous state \( h_{j-1} \), passed through a sigmoid activation function. An update candidate \( g \) is computed as a linear combination of \( x_j \) and \( h_{j-1} \), passed through a tanh activation function. The memory \( c_j \) is then updated: the forget gate controls how much of the previous memory to keep (\( c_{j-1} f \)), and the input gate controls how much of the proposed update to keep (\( g_i \)). Finally, the value of \( h_j \) (which is also the output \( y_j \)) is determined based on the content of the memory \( c_j \), passed through a tanh non-linearity and controlled by the output gate. The gating mechanisms allow for gradients related to the memory part to stay high across very long time ranges.

For further discussion on the LSTM architecture see the PhD thesis by Alex Graves (2008), as well as the online-post by Olah (2015b). For an analysis of the behavior of an LSTM when used as a character-level language model, see the work of Karpathy et al. (2015).

For further explanation of the motivation behind the gating mechanism in the LSTM (and the GRU) and its relation to solving the vanishing gradient problem in recurrent neural networks, see Sections 4.2 and 4.3 in the detailed course notes of Cho (2015).
There are many variants on the LSTM architecture presented here. For example, forget gates were not
residual networks, see Sections 4.2 and 4.3 in the detailed course notes of Cho (2015).

For further discussion on the LSTM architecture see the PhD thesis by Alex Graves
(2008), as well as the online-post by Olah (2015b). For an analysis of the behavior of an
LSTM-RNN is the GRU, to be discussed next.

LSTMs are currently the most successful type of RNN architecture, and they are re-

doing. There are three gates, which act on the content of the memory
to keep (and the GRU) and its relation to solving the vanishing gradient problem in recurrent neural
networks, see Sections 4.2 and 4.3 in the detailed course notes of Cho (2015).

The symbol \(\phi\) is the memory component (“cell”)

\[
\begin{align*}
g &= \sigma(x_j W^{xg} + \text{h}_{j-1} W^{hg}) \\
\text{input} &= (x_j W^{xi} + \text{h}_{j-1} W^{hi}) \\
\text{forget} &= \sigma(x_j W^{xf} + \text{h}_{j-1} W^{hf}) \\
\text{output} &= \sigma(x_j W^{xo} + \text{h}_{j-1} W^{ho}) \\
\end{align*}
\]

For further explanation of the motivation behind the gating mechanism in the LSTM

\[
\text{memory component ("cell")}
\]

\[
\begin{align*}
c_j &= \text{memory component} \\
h_j &= \text{hidden state}
\end{align*}
\]
There are many variants on the LSTM architecture presented here. For example, forget gates were not output gate. The gating mechanisms allow for gradients related to the memory part of the original proposal by Hochreiter and Schmidhuber (1997), but are shown to be an important part of the LSTM architecture. The main information flow is as follows:

\[ c_j = c_{j-1} \odot f + g \odot i \]

- \( c_j \): Memory component ("cell")
- \( c_{j-1} \): Previous memory
- \( f \): Forget gate
- \( i \): Input gate
- \( g \): Output gate
- \( o \): Proposed state
- \( x_j \): Input vector
- \( h_{j-1} \): Previous hidden state
- \( h_j \): Current hidden state

\[ g = \tanh(x_j W^{xg} + h_{j-1} W^{hg}) \]

Gates are computed as follows:

\[ \begin{align*}
i &= \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \\
f &= \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \\
o &= \sigma(x_j W^{xo} + h_{j-1} W^{ho})
\end{align*} \]

Mathematically, the LSTM architecture is defined as:

\[ \begin{align*}
c_j &= c_{j-1} \odot f + g \odot i \\
h_j &= \sigma(c_j) \odot o
\end{align*} \]
There are many variants on the LSTM architecture presented here. For example, forget gates were not LSTM when used as a character-level language model, see the work of Karpathy et al. and comprehensive empirical comparison of various LSTM architectures see the work of Gre Koutník, Steunebrink, and Schmidhuber (2015). Other variants include peephole connections and gate-tying. For an overview of the architecture, there are three gates, \( f, i, o \), to keep the memory part of the original proposal by Hochreiter and Schmidhuber (1997), but are shown to be an important part of the architecture. Other variants include peephole connections and gate-tying. For an overview of the LSTM architecture, see the PhD thesis by Alex Graves (2015).

Mathematically, the LSTM architecture is defined as:

\[
\begin{align*}
    c_j &= c_{j-1} \odot f + g \odot i \\
    h_j &= \tanh(c_j) \odot o \\
    g &= \tanh(x_j W^g + h_{j-1} W^{hg}) \\
    i &= \sigma(x_j W^i + h_{j-1} W^{hi}) \\
    f &= \sigma(x_j W^f + h_{j-1} W^{hf}) \\
    o &= \sigma(x_j W^o + h_{j-1} W^{ho})
\end{align*}
\]
• Note many LSTM variants (peephole or not; $c_i+use \,(1-f)\ or\ not...$)
  [diagram: Gers and Schmidhuber 2001]
• LSTMs have a poor reputation for understandability... yet do something right...
  usually just used as a black-box
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

First, the devishin it son?

MONTANO:
'Tis true as full Squellen the rest me, my passacre. and nothink
my fairs,' done to vision of actious to thy to love, brings gods!

THUR:
Will comfited our flight offend make thy love;
Brother is oats at on thes:'--why, cross and so
her shouldestruck at one their hearina in all go to lives of
Costag,
To his he tyrant of you our the fill we hath trouble an over me?

KING JOHN:
Great though I gain; for talk to mine and to the Christ: a right
him out

http://karpathy.github.io/2015/05/21/rnn-effectiveness/
http://nbviewer.jupyter.org/gist/yoavg/d76121dfde2618422139

Tuesday, February 13, 18
Structure awareness

Cell sensitive to position in line:
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pushed forward into boats and into the ice-covered water and did not,
surrender.

Cell that turns on inside quotes:
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."

Cell that robustly activates inside if statements:
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig)
    {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(*(current->notifier)(current->notifier_data))) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}

A large portion of cells are not easily interpretable. Here is a typical example:
/* Unpack a filter field's string representation from user-space
   * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
       * defines the longest valid length.
       *
       http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- LSTMs used as a generic, sequence-aware model within language modeling, translation generation, classification and tagging

- Various LSTM-analyzing-text visualizations
  - [http://karpathy.github.io/2015/05/21/rnn-effectiveness/](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)
  - [http://lstm.seas.harvard.edu/](http://lstm.seas.harvard.edu/)

- Question: can they learn interactions we know are in natural language?
  - Thursday: Linzen et al.