Recurrent Neural Networks
**Introduction**

- Traditional feedforward network assume that all inputs and outputs are independent of each other.
- Counterexample – language/speech modeling
  - Predicting the next word in a sentence depends on the entire sequence of words before the current word.
  - Example: “The man who wore a wig on his head went inside”
    - Who went inside? Man or wig?

A recurrent neural network and the unfolding in time of the computation involved in its forward computation

Notations

• $x_t$ = input at time step t.
  • E.g., One-hot vector corresponding to word of a sentence.

• $s_t$ = hidden state at time step t (“memory” of the network).
  Calculated based on the previous hidden state and the input at
  the current step: $s_t = f(Ux_t + Ws_{t-1})$.
  • $f$ = activation function (tanh, ReLU, etc).
  • $s_{-1}$, required to calculate first hidden state, typically initialized to all
    zeroes.

• $o_t$ = output at step t.
  • E.g., if we wanted to predict the next word in a sentence it would be
    a vector of probabilities across our vocabulary. $o_t = \text{softmax}(Vs_t)$.

Introduction

• Recurrent, because the same task is performed for every element of a sequence
• Also viewed as having a “memory”
  • Will be useful in understanding LSTM networks, a type of RNN
• Unlike a traditional deep network, RNN shares same parameters (U, V, W above) across all steps.
  • Greatly reduces the total number of parameters we need to learn.

• **Language Modeling and Generating Text**
  • Given a sequence of words we want to predict the probability of each word given the previous words
  • Can also be looked at as a generative model = fun!
  • Typical input is a sequence of words; output = sequence of predicted words
    • During training, $o_t = x_{t+1}$

• **More**
  • Recurrent neural network based language model
  • Extensions of Recurrent neural network based language model
  • Generating Text with Recurrent Neural Networks

Example Applications

• More fun ones: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

• **Machine Translation**

  • Sequence of words in source language -> Sequence of words in target language

  ![Diagram of RNN for Machine Translation]

• Machine Translation
  • Sequence of words in source language -> Sequence of words in target language

• More
  • A Recursive Recurrent Neural Network for Statistical Machine Translation
  • Sequence to Sequence Learning with Neural Networks
  • Joint Language and Translation Modeling with Recurrent Neural Networks

• Speech Recognition
  • Given an input sequence of acoustic signals from a sound wave, predict a sequence of phonetic segments together with their probabilities

• More
  • Towards End-to-End Speech Recognition with Recurrent Neural Networks

Example Applications

• **Image Caption Generation**
  • Generate descriptions for unlabeled images
  • Even aligns generated words with features found in images

• **More**

Example Applications

• Image Caption Generation

Training RNNs

• Recommended method: Backprop Through Time (BPTT)

• Limitations of BPTT
  • Vanishing Gradients
  • Exploding Gradients

• How to overcome
  • LSTMs (1997)
  • Recent Variants including GRUs (2014)
Backprop Through Time (BPTT)

• Recall:

\[ s_t = \tanh(Ux_t + Ws_{t-1}) \]
\[ \hat{y}_t = \text{softmax}(Vs_t) \]

• Loss function: e.g. Cross-entropy loss.

\[ E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \]
\[ E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) \]
\[ = - \sum_t y_t \log \hat{y}_t \]

• Slight change in notation: \( o \) replaced by \( \hat{y} \)

\( y_t \) is ground truth

Backprop Through Time (BPTT)

- Unrolled RNN

Goal

- Calculate error gradients w.r.t. U, V and W
- Learn weights using SGD

Just like we sum up errors, we also sum up gradients at each time step for one training example

\[
\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}
\]

Backprop Through Time (BPTT)

\[
\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V} \\
= \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial V} \\
= (\hat{y}_3 - y_3) \otimes s_3
\]

Recall:

Where \( z_3 = V s_3 \) and \( \otimes \) is outer product

What do you think about gradient of $E_3$ w.r.t. $W$?

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial W}$$

Is that complete?

$$s_3 = \tanh(U x_t + W s_2)$$

$=>$ Chain rule needs to be applied again

We would have:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

Similar to backprop, you can define:

$$\delta^{(3)}_2 = \frac{\partial E_3}{\partial z_2} = \frac{\partial E_3}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial z_2}$$

with  

$$z_2 = Ux_2 + Ws_1$$

• Do you see any problem?
  • Sequences (sentences) can be quite long, perhaps 20 words or more - need to back-propagate through many layers!
  • Vanishing gradient problem

Vanishing Gradient Problem

\[
\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}
\]

can be rewritten as:

\[
\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left( \prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}
\]

• Happens that the 2-norm of the Jacobian matrix is upper-bounded by 1
  • Pascanu et al, On the difficulty of training recurrent neural networks, ICML 2013

Vanishing Gradient Problem

\[ \frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left( \prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W} \]

- For sigmoid activations -> gradient is upper-bounded by 1
- What does this tell you?
- Gradients will vanish over time, and long-range dependencies will only worsen learning

Vanishing Gradient Problem

\[
\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left( \prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}
\]

- What if weights are high?
- Could lead to **exploding gradient problem**
- Why isn’t this much of a problem?
  - Will show up easily as NaN
  - Clipping gradients works!
- What about vanishing gradients?
  - Solutions exist – we will see in next few slides!

• **Training RNNs with backpropagation** (Hinton)
• **A toy example of training an RNN** (Hinton)
• **Why is it difficult to train an RNN** (Hinton)
Training RNNs

• Recommended method: Backprop Through Time (BPTT)

• Limitations of BPTT
  • Vanishing Gradients
  • Exploding Gradients

• How to overcome
  • LSTMs (1997)
  • Recent Variants including GRUs (2014)
Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM)

Input gate: scales input to cell (write)
Output gate: Scales output from cell (read)
Forget gate: Scales old cell value (reset)

- Learns model and context at the same time – very powerful!
- Graph is no more sequential
LSTM: Implementation

\[ i = \sigma(x_t U^i + s_{t-1} W^i) \]
\[ f = \sigma(x_t U^f + s_{t-1} W^f) \]
\[ o = \sigma(x_t U^o + s_{t-1} W^o) \]
\[ g = \tanh(x_t U^g + s_{t-1} W^g) \]
\[ c_t = c_{t-1} \circ f + g \circ i \]
\[ s_t = \tanh(c_t) \circ o \]

- Bias ignored
- Note the activation functions used. Why?

What can you tell about $C_t$, if $f$ is always 1?

- What happens if you fix input gate to all 1s, the forget gate to all 0s, output gate to all 1s?
- *Almost* the standard RNN. Why almost?
- Tanh added here

$$
i = \sigma(x_t U^i + s_{t-1} W^i) \\
f = \sigma(x_t U^f + s_{t-1} W^f) \\
o = \sigma(x_t U^o + s_{t-1} W^o) \\
g = \tanh(x_t U^g + s_{t-1} W^g) \\
c_t = c_{t-1} \circ f + g \circ i \\
s_t = \tanh(c_t) \circ o$$

Training LSTMs

• BPTT
  • For more details, please see Alex Graves’ book on RNN (Sec 4.6, pg 36-38): http://www.cs.toronto.edu/~graves/preprint.pdf
LSTMs History

• 1997 - RTRL + BPTT (No forget gate)

• 1999 – Introduced forget gate

• 2000 – Peephole connections

• 2005 – Vanilla LSTM (as we know today) – Used BPTT completely
  • Graves, Alex and Schmidhuber, Jurgen. Framewise ″ phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 18(5–6):602–610, July 2005
How do LSTMs avoid the vanishing gradient problem?

- Let’s take a look at:

  \[ i = \sigma(x_t U^i + s_{t-1} W^i) \]
  \[ f = \sigma(x_t U^f + s_{t-1} W^f) \]
  \[ o = \sigma(x_t U^o + s_{t-1} W^o) \]
  \[ g = \tanh(x_t U^g + s_{t-1} W^g) \]
  \[ c_t = c_{t-1} \odot f + g \odot i \]
  \[ s_t = \tanh(c_t) \odot o \]

- Cell state at t-1 will receive:

\[ \frac{\partial}{\partial c_{t-1}} := \frac{\partial}{\partial c_{t-1}} + f \odot \frac{\partial L}{\partial c_t} \]

- What does this tell you?
  - Depends only on forget gate! What are its values?
LSTM variants

Legend
- unweighted connection
- weighted connection
- connection with time-lag
- branching point
- multiplication
- sum over all inputs
- gate activation function (always sigmoid)
- input activation function (usually tanh)
- output activation function (usually tanh)
LSTM variants

• LSTM with peephole connections

\[
\begin{align*}
    f_t &= \sigma (W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \\
    i_t &= \sigma (W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\
    o_t &= \sigma (W_o \cdot [C_t, h_{t-1}, x_t] + b_o)
\end{align*}
\]

• Coupled forget and input gates

\[
C_t = f_t \ast C_{t-1} + (1 - f_t) \ast \tilde{C}_t
\]

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
What happens if reset gate is set to all 1s and update gate to all 0s?

GRUs vs LSTMs: Summary

• A GRU has two gates, an LSTM has three gates. What does this tell you?

• In GRUs
  • No internal memory \((c_t)\) different from the exposed hidden state.
  • No output gate as in LSTMs.

• The input and forget gates of LSTMs are coupled by an update gate in GRUs, and the reset gate (GRUs) is applied directly to the previous hidden state.

• GRUs: No nonlinearity when computing the output.

(Read http://colah.github.io/posts/2015-08-Understanding-LSTMs/ for explanations with illustrations)

LSTM Cell in Torch

```plaintext
local function make_lstm_step(opt, input, prev_h, prev_c)
    local function new_input_sum()
        local x_to_h = nn.Linear(opt.rnn_size, opt.rnn_size)
        local h_to_h = nn.Linear(opt.rnn_size, opt.rnn_size)
        return nn.CAddTable(){{ x_to_h(input), h_to_h(prev_h)}}
    end
    local in_gate = nn.Sigmoid()(new_input_sum())
    local forget_gate = nn.Sigmoid()(new_input_sum())
    local cell_gate = nn.Tanh()(new_input_sum())
    local next_c = nn.CAddTable(){{
        nn.CMulTable()({{forget_gate, prev_c}},
        nn.CMulTable()({{in_gate, cell_gate}})}
    local out_gate = nn.Sigmoid()(new_input_sum())
    local next_h = nn.CMulTable()({{out_gate, nn.Tanh()(next_c)}}
    return next_h, next_c
end
```
LSTMs for Sequence-to-Sequence Prediction

Target sequence

Source: Ilya Sutskever et al
LSTMs for Sequence-to-Sequence Prediction

Source: Ilya Sutskever
Bidirectional RNNs

- Bidirectional RNNs

- Deep Bidirectional RNNs

Learning to Execute

• Zaremba and Sutskever, 2015

Input:

```
j=8584
for x in range(8):
j+=920
b=(1500+j)
print((b+7567))
```

Target: 25011.

Input:

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else 5308)
```

Target: 12184.

Figure 1: Example programs on which we train the LSTM. The output of each program is a single integer. A “dot” symbol indicates the end of the integer, which has to be predicted by the LSTM.
Which is Real?

from his travels it might have been
from his travels it might have been
from his travels it might have been
from his travels it might have been
from his travels it might have been

http://www.cs.toronto.edu/~graves/handwriting.html
• Loosely based on visual attention mechanism in humans
• Being able to focus on a certain region of an image with “high resolution” while perceiving the surrounding image in “low resolution”, and then adjusting the focal point over time.

• How to learn long-range dependencies?
• Hacks
  • Provide the input in reverse order
  • Provide the same input twice
• How can we do better?
  • Attention

• Each decoded output word now depends on a weighted combination of input states
  
  • Bahdanau et al, ICLR 2015
  
  • Very costly though – how to do better?


Figure 1: A) Glimpse Sensor: Given the coordinates of the glimpse and an input image, the sensor extracts a retina-like representation $\rho(x_t, l_{t-1})$ centered at $l_{t-1}$ that contains multiple resolution patches. B) Glimpse Network: Given the location $(l_{t-1})$ and input image $(x_t)$, uses the glimpse sensor to extract retina representation $\rho(x_t, l_{t-1})$. The retina representation and glimpse location is then mapped into a hidden space using independent linear layers parameterized by $\theta_g^0$ and $\theta_g^1$ respectively using rectified units followed by another linear layer $\theta_g^2$ to combine the information from both components. The glimpse network $f_g(\cdot; \{\theta_g^0, \theta_g^1, \theta_g^2\})$ defines a trainable bandwidth limited sensor for the attention network producing the glimpse representation $g_t$. C) Model Architecture: Overall, the model is an RNN. The core network of the model $f_h(\cdot; \theta_h)$ takes the glimpse representation $g_t$ as input and combining with the internal representation at previous time step $h_{t-1}$, produces the new internal state of the model $h_t$. The location network $f_l(\cdot; \theta_l)$ and the action network $f_a(\cdot; \theta_a)$ use the internal state $h_t$ of the model to produce the next location to attend to $l_t$ and the action/classification $a_t$ respectively. This basic RNN iteration is repeated for a variable number of steps.
Show, Attend and Tell (Xu et al, ICML 2015)

1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

(b) A person is standing on a beach with a surfboard.
Neural Turing Machines (Graves et al., arXiv 2014)