CENG 783

Special topics in Deep Learning

Week 11 – 12-13
Recurrent Neural Networks

Sinan Kalkan
Sequence Labeling/Modeling: Motivation
Why do we need them?

A. Graves, “Supervised Sequence Labelling with Recurrent Neural Networks”, 2012.
Different types of sequence learning/recognition problems

• Sequence Classification
  • A sequence to a label
  • E.g., recognizing a single spoken word
  • Length of the sequence is fixed
  • Why RNNs then? Because sequential modeling provides robustness against translations and distortions.

• Segment Classification
  • Segments in a sequence correspond to labels

• Temporal Classification
  • General case: sequence (input) to sequence (label) modeling.
  • No clue about where input or label starts.

A. Graves, “Supervised Sequence Labelling with Recurrent Neural Networks”, 2012.
Recurrent Neural Networks
Recurrent Neural Networks (RNNs)

- RNNs are very powerful because:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.
- More formally, RNNs are Turing complete.

Adapted from Hinton
Recurrent Neural Nets

- Temporal pattern recognition
  - OUTPUT
  - CONTEXT
  - INPUT
  - Example: speech recognition
  - Example: event recognition
  - Example: natural language understanding

- Sequence generation
  - OUTPUT
  - CONTEXT
  - PLAN
  - Example: speech production
  - Example: motor control
  - Example: planning and acting

- Pattern completion / constraint satisfaction
  - GANG
  - AGE
  - EDUCATION
  - OCCUPATION
  - INSTANCE
  - NAME
Some examples

Jordan Networks

Elman Networks

“context” neurons

Figs: David Kriesel
Challenge

- Back propagation is designed for feedforward nets
- What would it mean to back propagate through a recurrent network?
  - error signal would have to travel back in time
Unfolding

Feed-forward networks

Recurrent networks

Unfolding
Unfolding (another example)

Figure: Michael Mozer
How an RNN works

Learned representation of sequence.

projections (activities x weights)

activities (vectors of values)

the cat sat on the mat

hidden to hidden input to hidden

Alec Radford
You can stack them too

Alec Radford
Unfolding implications

- Entails duplication of weights => weight sharing
- Sharing weights means their gradients will be accumulated over time and reflected on the weights
- Unfolded network has the same dynamics of the RNN for a fixed number of time steps!
Back-propagation Through Time
Simple Recurrent Network (SRN)
(Elman Network)
Reminder: Backpropagation with weight constraints

It is easy to modify the backprop algorithm to incorporate linear constraints between the weights. We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.

– So if the weights started off satisfying the constraints, they will continue to satisfy them.

To constrain: \( w_1 = w_2 \)

we need: \( \Delta w_1 = \Delta w_2 \)

compute: \( \frac{\partial E}{\partial w_1} \) and \( \frac{\partial E}{\partial w_2} \)

use \( \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \) for \( w_1 \) and \( w_2 \)
Backpropagation through time

We can think of the recurrent net as a layered, feed-forward net with shared weights and then train the feed-forward net with weight constraints.

We can also think of this training algorithm in the time domain:

- The forward pass builds up a stack of the activities of all the units at each time step.
- The backward pass peels activities off the stack to compute the error derivatives at each time step.
- After the backward pass we add together the derivatives at all the different times for each weight.
Cross-entropy loss:

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

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

Accumulate errors over time (treat the whole sequence as a training example):

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

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

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

\[
\frac{\partial C_t}{\partial \mathbf{W}} = \sum_{t'=1}^{t} \frac{\partial C_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t'}} \frac{\partial h_{t'}}{\partial \mathbf{W}}, \quad \text{where} \quad \frac{\partial h_t}{\partial h_{t'}} = \prod_{k=t'+1}^{t} \frac{\partial h_k}{\partial h_{k-1}}
\]
An irritating extra issue

We need to specify the initial activity state of all the hidden and output units.

We could just fix these initial states to have some default value like 0.5.

But it is better to treat the initial states as learned parameters.

We learn them in the same way as we learn the weights.

- Start off with an initial random guess for the initial states.
- At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state.
- Adjust the initial states by following the negative gradient.
A good toy problem for a recurrent network

We can train a feedforward net to do binary addition, but there are obvious regularities that it cannot capture efficiently.

- We must decide in advance the maximum number of digits in each number.
- The processing applied to the beginning of a long number does not generalize to the end of the long number because it uses different weights.

As a result, feedforward nets do not generalize well on the binary addition task.
The algorithm for binary addition

This is a finite state automaton. It decides what transition to make by looking at the next column. It prints after making the transition. It moves from right to left over the two input numbers.
A recurrent net for binary addition

The network has two input units and one output unit.
It is given two input digits at each time step.
The desired output at each time step is the output for the column that was provided as input two time steps ago.

– It takes one time step to update the hidden units based on the two input digits.
– It takes another time step for the hidden units to cause the output.
Sum of three numbers

Addition and subtraction in the same network

Generalized network: Any length, any operation

https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-sequence-learning/
The problem of exploding or vanishing gradients

What happens to the magnitude of the gradients as we backpropagate through many layers?
– If the weights are small, the gradients shrink exponentially.
– If the weights are big the gradients grow exponentially.

Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.

In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish.
– We can avoid this by initializing the weights very carefully.

Even with good initial weights, it's very hard to detect that the current target output depends on an input from many time-steps ago.
– So RNNs have difficulty dealing with long-range dependencies.
Real-Time Recurrent Learning (RTRL)

- Propagate error forward in time

\[ s_k(t + 1) = \sum_{l \in U} w_{kl} y_l(t) + \sum_{l \in I} w_{kl} x^\text{net}_l(t) = \sum_{l \in U \cup I} w_{kl} x_l(t). \]

\[ y_k(t + 1) = f_k(s_k(t + 1)), \]

\[ p_{ij}^k(t) = \frac{\partial y_k(t)}{\partial w_{ij}}. \]

\[ p_{ij}^k(t + 1) = f'_k(s_k(t + 1)) \left[ \sum_{l \in U} w_{kl} p_{ij}^l(t) + \delta_{ik} x_j(t) \right] \]

---

Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity

Ronald J. Williams  
College of Computer Science, Northeastern University

David Zipser  
Department of Cognitive Science, University of California, San Diego
BPTT vs RTRL

One complaint about BPTT is that it requires saving activity states of all units at all previous times.

**Space complexity of BPTT:** $O(N^2T)$
**Time complexity of BPTT:** $O(N^3T)$

Williams & Zipser have developed an alternative algorithm called real-time recurrent learning (RTRL) which does not require storing activity states.

**Space complexity of RTRL:** $O(N^3)$
**Time complexity of RTRL:** $O(N^4)$

Both RTRL and BPTT compute exact gradient of the error with respect to the weights. They have the same power and limitations.

For large $T$, RTRL may be more efficient. Not particularly useful in practice.
Problem With BPTT and RTRL

While BPTT is in principle capable of learning relationships among temporal events, in practice it is weak.

E.g., detecting contingencies spanning temporal gaps

\[ e_1 \ldots e_2 \]

\( e_2 \) dependent on \( e_1 \)

Input is a sequence of symbols: \( A, B, C, D, E, F, X, Y \)

Task is to predict next symbol in sequence

Sample sequences:

\[
\begin{align*}
\text{Gap} = 2: & \quad X \ A \ B \ X \\
\text{Gap} = 6: & \quad X \ A \ B \ C \ D \ E \ F \ X  \\
\text{Gap} = 4: & \quad Y \ A \ B \ Y \\
\text{Gap} = 8: & \quad Y \ A \ B \ C \ D \ E \ F \ Y
\end{align*}
\]

Learning two-sequence training set with a sequence-recog.
architecture and BPTT is not reliable for gaps of 4 or more

\[
\begin{array}{|c|c|}
\hline
\text{Gap} & \% \text{failures after 10 epochs} \\
\hline
2 & 0 \\
4 & 36 \\
6 & 92 \\
8 & 100 \\
\hline
\end{array}
\]

(Mozer, 1992)

Problem: BPTT ok at discovering structure that is local in time, but not good at handling structure at a more global scale (long temporal intervals, & involving high order statistics).

Slide: Michael Mozer
Exploding and vanishing gradients problem

• Solution 1: Gradient clipping for exploding gradients:

\[
\hat{g} \leftarrow \frac{\partial E}{\partial \theta}
\]

if \( \|\hat{g}\| \geq \text{threshold} \) then
\[\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g}\]
end if

• For vanishing gradients: Regularization term that penalizes changes in the magnitudes of back-propagated gradients

\[
\Omega = \sum_k \Omega_k = \sum_k \left( \left\| \frac{\partial E}{\partial \mathbf{x}_{k+1}} \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \right\| - 1 \right)^2
\]

---

On the difficulty of training recurrent neural networks

Razvan Pascanu
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

Tomas Mikolov
Speech@FIT, Brno University of Technology, Brno, Czech Republic

Yoshua Bengio
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

Figure 6. We plot the error surface of a single hidden unit recurrent network, highlighting the existence of high curvature walls. The solid lines depict standard trajectories that gradient descent might follow. Using dashed arrow the diagram shows what would happen if the gradients is rescaled to a fixed size when its norm is above a threshold.
Exploding and vanishing gradients problem

• Solution 2:
  • Use methods like LSTM
Long Short Term Memory (LSTM) Networks
RNN

- Basic block diagram
Key Problem

- Learning long-term dependencies is hard
Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps). They designed a memory cell using logistic and linear units with multiplicative interactions.

Information gets into the cell whenever its “write” gate is on.

The information stays in the cell so long as its “keep” gate is on.

Information can be read from the cell by turning on its “read” gate.
Meet LSTMs

• How about we explicitly encode memory?
LSTM in detail

- We first compute an activation vector, $a$:
  \[ a = W_x x_t + W_h h_{t-1} + b \]

- Split this into four vectors of the same size:
  \[ a_i, a_f, a_o, a_g \leftarrow a \]

- We then compute the values of the gates:
  \[ i = \sigma(a_i) \quad f = \sigma(a_f) \quad o = \sigma(a_o) \quad g = \tanh(a_g) \]
  where $\sigma$ is the sigmoid.

- The next cell state $c_t$ and the hidden state $h_t$:
  \[ c_t = f \odot c_{t-1} + i \odot g \quad h_t = o \odot \tanh(c_t) \]
  where $\odot$ is the element-wise product of vectors

Eqs: Karpathy
LSTMs Intuition: Memory

• Cell State / Memory

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
LSTMs Intuition: Forget Gate

• Should we continue to remember this “bit” of information or not?

\[ f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f) \]
LSTMs Intuition: Input Gate

- Should we update this “bit” of information or not?
  - If so, with what?

\[
i_t = \sigma \left( W_i \cdot [h_{t-1}, x_t] + b_i \right)
\]
\[
\tilde{C}_t = \tanh \left( W_C \cdot [h_{t-1}, x_t] + b_C \right)
\]
LSTMs Intuition: Memory Update

- Forget that + memorize this

\[
C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t
\]
LSTMs Intuition: Output Gate

• Should we output this “bit” of information to “deeper” layers?

\[ o_t = \sigma \left( W_o \left[ h_{t-1}, x_t \right] + b_o \right) \]

\[ h_t = o_t \times \text{tanh} \left( C_t \right) \]
LSTMs

• A pretty sophisticated cell

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
LSTM Variants #1: Peephole Connections

- Let gates see the cell state / memory

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

(C) Dhruv Batra

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
LSTM Variants #2: Coupled Gates

• Only memorize new if forgetting old

\[ C_t = f_t \times C_{t-1} + (1 - f_t) \times \tilde{C}_t \]
LSTM Variants #3: Gated Recurrent Units

- Changes:
  - No explicit memory; memory = hidden output
  - $Z = \text{memorize new and forget old}$

\[
\begin{align*}
    z_t &= \sigma (W_z \cdot [h_{t-1}, x_t]) \\
    r_t &= \sigma (W_r \cdot [h_{t-1}, x_t]) \\
    \tilde{h}_t &= \tanh (W \cdot [r_t \ast h_{t-1}, x_t]) \\
    h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]
References

• A very detailed explanation with nice figures

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Example: Character-level Text Modeling
Modeling text: Advantages of working with characters

The web is composed of character strings.

Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words (this turns out to be true, as we shall see).

Pre-processing text to get words is a big hassle

- What about morphemes (prefixes, suffixes etc)
- What about subtle effects like “sn” words?
- What about New York?
- What about Finnish

ymmartamattomyydeIansaikaan...
A sub-tree in the tree of all character strings

There are exponentially many nodes in the tree of all character strings of length N.

In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

If the nodes are implemented as hidden states in an RNN, different nodes can share structure because they use distributed representations.

The next hidden representation needs to depend on the conjunction of the current character and the current hidden representation.
An obvious recurrent neural net

It’s a lot easier to predict 86 characters than 100,000 words.
A simple scenario

- Alphabet: h, e, l, o
- Text to train to predict: “hello”
Sample predictions (when trained on the works of Shakespeare):

- 3-level RNN with 512 hidden nodes in each layer

PANDARUS:
Alas, I think he shall be come approached and the day
When little strain 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.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Sample predictions (when trained on Wikipedia):

- Using LSTM

Naturalism and decision for the majority of Arab countries' capitals were grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]
Sample predictions  
(when trained on Latex documents):

- Using multi-layer LSTM

For $\bigoplus_{m=1, \ldots, m}$ where $\mathcal{L}_{m*} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on $X$. $U$ is a closed immersion of $S$, then $\mathcal{F} \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \text{Spec}(\mathcal{O}) = \bigcup \mathcal{F} \to \mathcal{U} \to V.$$  

and the comparely in the fibre product covering we have to prove the lemma generated by $\prod \mathcal{U} \times V \to V$. Consider the maps $\mathcal{M}$ along the set of points $\text{Sch}_{\mathcal{M}}$ and $\mathcal{U} \to \mathcal{V}$ is the fibre category of $\mathcal{S}$ in $\mathcal{U}$ in Section ?? and the fact that any $\mathcal{U}$ affine, see Morphisms, Lemma ??, hence we obtain a scheme $\mathcal{S}$ and any open subset $\mathcal{W} \subset \mathcal{U}$ in $\text{Sh}(\mathcal{G})$ such that $\text{Spec}(\mathcal{R}) \to \mathcal{S}$ is smooth or an

$$\mathcal{U} = \bigcup \mathcal{W}_i \times S, \mathcal{U}_i$$

which has a nonzero morphism we may assume that $\mathcal{f}_i$ is of finite presentation over $\mathcal{S}$. We claim that $\mathcal{O}_{\mathcal{X}, \mathcal{r}}$ is a scheme where $x, x', s' \in S'$ such that $\mathcal{O}_{\mathcal{X}, \mathcal{r}} \to \mathcal{O}_{\mathcal{X}', \mathcal{r}}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathcal{M}_{1/(\mathcal{S}/\mathcal{S})}$ and we win.

To prove we see that $\mathcal{F}_n$ is covering of $\mathcal{X}'$, and $\mathcal{T}_n$ is an object of $\mathcal{F}_{\mathcal{X}/\mathcal{S}}$ for $i > 0$ and $\mathcal{F}_n$ exists and let $\mathcal{F}_n$ be a presheaf of $\mathcal{O}_{\mathcal{X}/\mathcal{S}}$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = \mathcal{U}/\mathcal{F}$ we have to show that

$$\overline{\mathcal{M}} = \mathcal{T} \times \text{Spec}(\mathcal{M})\mathcal{O}_{\mathcal{X}, \mathcal{r}} - \mathcal{T}_n$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = \bigcup \mathcal{F}_n \times \text{Spec}(\mathcal{M})\mathcal{O}_{\mathcal{X}, \mathcal{r}} - \mathcal{T}_n$$

and

$$\mathcal{V} = \bigcup \mathcal{F}_n \times \text{Spec}(\mathcal{M})\mathcal{O}_{\mathcal{X}, \mathcal{r}} - \mathcal{T}_n$$

is an open subset of $\mathcal{X}$. Thus $\mathcal{V}$ is affine. This is a continuous map of $\mathcal{X}$ is the inverse, the groupoid scheme $\mathcal{S}$.

Proof. See discussion of sheaves of sets.

The result for prove any open covering follows from the less of Example ??, it may replace $\mathcal{S}$ and $\mathcal{X}$ by $\text{Spec}(\mathcal{M})\mathcal{O}_{\mathcal{X}/\mathcal{S}}$ which gives an open subspace of $\mathcal{X}$ and $\delta_{\text{aff}}$ equal to $\delta_{\text{aff}}$, see Descent, Lemma ??, Namely, by Lemma ?? we see that $\mathcal{R}$ is geometrically regular over $\mathcal{S}$.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/
He was elected President during the Revolutionary War and forgave Opus Paul at Rome. The regime of his crew of England, is now Arab women's icons in and the demons that use something between the characters‘ sisters in lower coil trains were always operated on the line of the ephemeral street, respectively, the graphic or other facility for deformation of a given proportion of large segments at RTUS). The B every chord was a "strongly cold internal palette pour even the white blade.”
Some completions produced by the model

Sheila thrunges (most frequent)

People thrunge (most frequent next character is space)

Shiela, Thrungelini del Rey (first try)

The meaning of life is literary recognition. (6th try)

The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer).
What does it know?

It knows a huge number of words and a lot about proper names, dates, and numbers.

It is good at balancing quotes and brackets.

- It can count brackets: none, one, many

It knows a lot about syntax but its very hard to pin down exactly what form this knowledge has.

- Its syntactic knowledge is not modular.

It knows a lot of weak semantic associations

- E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable.
RNNs for predicting the next word

Tomas Mikolov and his collaborators have trained quite large RNNs on quite large training sets using BPTT.

- They do better than feed-forward neural nets.
- They do better than the best other models.
- They do even better when averaged with other models.

RNNs require much less training data to reach the same level of performance as other models.

RNNs improve faster than other methods as the dataset gets bigger.

- This is going to make them very hard to beat.
Word-level RNN for news title generation

https://github.com/larspars/word-rnn

Click-o-Tron

3D Video Brings Clean Energy To The Real Economy

David Beckham & Victoria Beckham's Talk Show Gets 2014 Golden Girls Love

The area is still in such big power, but I have some advice for those this ye...

White House Attacks The Obama Plan

Women learned from their families in a post-9/11 New York Times story that is...

Not The Same Thing: New Rule Can’t

http://clickotron.com/
Word Embedding
(word2vec)

Fig: http://www.languagejones.com/blog-1/2015/11/1/word-embedding
Why do we embed words?

- 1-of-n encoding is not suitable to learn from
  - It is sparse
  - Similar words have different representations
  - Compare the pixel-based representation of images: Similar images/objects have similar pixels

- Embedding words in a map allows
  - Encoding them with fixed-length vectors
  - “Similar” words having similar representations
  - Allows complex reasoning between words:
    - king - man + woman = queen

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>NEAREST TOKEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris - France + Italy</td>
<td>Rome</td>
</tr>
<tr>
<td>bigger - big + cold</td>
<td>colder</td>
</tr>
<tr>
<td>sushi - Japan + Germany</td>
<td>bratwurst</td>
</tr>
<tr>
<td>Cu - copper + gold</td>
<td>Au</td>
</tr>
<tr>
<td>Windows - Microsoft + Google</td>
<td>Android</td>
</tr>
<tr>
<td>Montréal Canadiens - Montreal + Toronto</td>
<td>Toronto Maple Leafs</td>
</tr>
</tbody>
</table>

Table 1: Mikolov et al. [3] showcase simple additive properties of their word embeddings.

More examples

http://deeplearning4j.org/word2vec
More examples

Male-Female

Verb tense

Country-Capital
More examples

• Geopolitics: *Iraq* - *Violence* = *Jordan*
• Distinction: *Human* - *Animal* = *Ethics*
• *President* - *Power* = *Prime Minister*
• *Library* - *Books* = *Hall*

http://deeplearning4j.org/word2vec
Using word embeddings

- E.g., for language modeling

- Given “I am eating …”, a language model can predict what can come next.
  - This both requires syntax and semantics (context)

- Before deep learning, n-gram (2-gram, 3-gram) models were state of the art.

Fig: https://devblogs.nvidia.com/parallelforall/understanding-natural-language-deep-neural-networks-using-torch/
Using word embeddings

Fig: https://devblogs.nvidia.com/parallelforall/understanding-natural-language-deep-neural-networks-using-torch/
Using word embeddings

Fig: https://devblogs.nvidia.com/parallelforall/understanding-natural-language-deep-neural-networks-using-torch/
word2vec

- “Similarity” to Sweden (cosine distance between their vector representations)

<table>
<thead>
<tr>
<th>Word</th>
<th>Cosine distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>norway</td>
<td>0.760124</td>
</tr>
<tr>
<td>denmark</td>
<td>0.715460</td>
</tr>
<tr>
<td>finland</td>
<td>0.620022</td>
</tr>
<tr>
<td>switzerland</td>
<td>0.588132</td>
</tr>
<tr>
<td>belgium</td>
<td>0.585835</td>
</tr>
<tr>
<td>netherlands</td>
<td>0.574631</td>
</tr>
<tr>
<td>iceland</td>
<td>0.562368</td>
</tr>
<tr>
<td>estonia</td>
<td>0.547621</td>
</tr>
<tr>
<td>slovenia</td>
<td>0.531408</td>
</tr>
</tbody>
</table>

http://deeplearning4j.org/word2vec
Two different ways to train

1. Using context to predict a target word (~ continuous bag-of-words)

2. Using word to predict a target context (skip-gram)
   - Produces more accurate results on large datasets
   - If the vector for a word cannot predict the context, the mapping to the vector space is adjusted
   - Since similar words should predict the same or similar contexts, their vector representations should end up being similar

http://deeplearning4j.org/word2vec
Two different ways to train

1. Using context to predict a target word (~ continuous bag-of-words)

https://www.tensorflow.org/versions/r0.7/tutorials/word2vec/index.html
Two different ways to train

2. Using word to predict a target context (skip-gram)
   - Produces more accurate results on large datasets

   - Given a sentence:
     
     the quick brown fox jumped over the lazy dog

   - For each word, take context to be

     \((N\text{-words to the left, } N\text{-words to the right})\)

   - If \(N = 1\) (context, word):

     \([\text{the, brown}], \text{quick}\), \([\text{quick, fox}], \text{brown}\), \([\text{brown, jumped}], \text{fox}\), ...
Now

- Example RNN applications:
  - Image captioning
  - Neural Machine Translation
- Echo-state Networks
- Time Delay Neural Networks
- Neural Turing Machines
Example: Image Captioning

Fig: https://github.com/karpathy/neuraltalk2
Demo video

https://vimeo.com/146492001
Overview

Pre-trained CNN (e.g., on imagenet)

Pre-trained word embedding is also used
Training

before:  \( h_0 = \max(0, Wxh \times x_0) \)

now:  \( h_0 = \max(0, Wxh \times x_0 + Wih \times v) \)
test image

sample!
sample! <END> token => finish.
Example: Neural Machine Translation
Check the following tutorial

- http://smerity.com/articles/2016/google_nmt_arch.html
Neural Machine Translation

- Model

Haitham Elmarakeby
Neural Machine Translation

- Model

Each box is an LSTM or GRU cell.

Sutskever et al. 2014

Haitham Elmarakeby
Neural Machine Translation

Model-encoder

Cho: From Sequence Modeling to Translation

$\mathbf{u}_i$

Word Sample

$p_i$

Word Probability

$\mathbf{Z}_i$

Recurrent State

$h_i$

Recurrent State

$\mathbf{S}_i$

Continuous-space Word Representation

$\mathbf{W}_i$

1-of-K coding

$f = (\text{La, croissance, économique, s'est, ralentie, ces, dernières, années, }.)$

$e = (\text{Economic, growth, has, slowed, down, in, recent, years. }.)$
Neural Machine Translation

- Model- encoder

Cho: From Sequence Modeling to Translation
Neural Machine Translation

- Model: decoder

\[ f = (\text{La, croissance, économique, s'est, ralentie, ces, dernières, années, .}) \]

\[ e = (\text{Economic, growth, has, slowed, down, in, recent, years, .}) \]

Cho: From Sequence Modeling to Translation

Haitham Elmarakeby
Decoder in more detail

Given

(i) the “summary” (c) of the input sequence,
(ii) the previous output / word (y(t − 1))
(iii) the previous state (h(t − 1))

the hidden state of the decoder is:

\[ h(t) = f(h(t - 1), y(t - 1), c) \]

Then, we can find the most likely next word:

\[ P(y(t) \mid y(t - 1), y(t - 2), \ldots, c) = g(h(t), y(t - 1), c) \]

\(f, g\): activation functions of our choice. For \( g\), we need a function that maps to probabilities, e.g., softmax.
Encoder-decoder

- Jointly trained to maximize

\[
\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(y_n \mid x_n),
\]
This can be done at char-level too

Echo State Networks

Reservoir Computing
Motivation

• “Schiller and Steil (2005) also showed that in traditional training methods for RNNs, where all weights (not only the output weights) are adapted, the dominant changes are in the output weights. In cognitive neuroscience, a related mechanism has been investigated by Peter F. Dominey in the context of modelling sequence processing in mammalian brains, especially speech recognition in humans (e.g., Dominey 1995, Dominey, Hoen and Inui 2006). Dominey was the first to explicitly state the principle of reading out target information from a randomly connected RNN. The basic idea also informed a model of temporal input discrimination in biological neural networks (Buonomano and Merzenich 1995).”

http://www.scholarpedia.org/article/Echo_state_network
Echo State Networks (ESN)

- Reservoir of a set of neurons
  - Randomly initialized and fixed
  - Run input sequence through the network and keep the activations of the reservoir neurons
  - Calculate the “readout” weights using linear regression.
- Has the benefits of recurrent connections/networks
- No problem of vanishing gradient

Li et al., 2015.
The reservoir

- Provides non-linear expansion
  - This provides a “kernel” trick.
- Acts as a memory
- Parameters:
  - $W_{in}$, $W$ and $\alpha$ (leaking rate).
- Global parameters:
  - Number of neurons: The more the better.
  - Sparsity: Connect a neuron to a fixed but small number of neurons.
  - Distribution of the non-zero elements: Uniform or Gaussian distribution. $W_{in}$ is denser than $W$.
  - Spectral radius of $W$: Maximum absolute eigenvalue of $W$, or the width of the distribution of its non-zero elements.
    - Should be less than 1. Otherwise, chaotic, periodic or multiple fixed-point behavior may be observed.
    - For problems with large memory requirements, it should be bigger than 1.
  - Scale of the input weights.
\[ \tilde{x}(n) = \tanh \left( W^{\text{in}} [1; u(n)] + W x(n - 1) \right), \quad (2) \]

\[ x(n) = (1 - \alpha) x(n - 1) + \alpha \tilde{x}(n), \quad (3) \]

where \( x(n) \in \mathbb{R}^{N_x} \) is a vector of reservoir neuron activations and \( \tilde{x}(n) \in \mathbb{R}^{N_x} \) is its update, all at time step \( n \), \( \tanh(\cdot) \) is applied element-wise, \([\cdot; \cdot]\) stands for a vertical vector (or matrix) concatenation, \( W^{\text{in}} \in \mathbb{R}^{N_x \times (1+N_u)} \) and \( W \in \mathbb{R}^{N_x \times N_x} \) are the input and recurrent weight matrices respectively, and \( \alpha \in (0, 1] \) is the leaking rate. Other sigmoid wrappers can be used besides the \( \tanh \), which however is the most common choice. The model is also sometimes used without the leaky integration, which is a special case of \( \alpha = 1 \) and thus \( \tilde{x}(n) \equiv x(n) \).

\[ y(n) = W^{\text{out}} [1; u(n); x(n)], \]

again stands for a vertical vector (or matrix) concatenation. An additional nonlinearity can be applied to \( y(n) \) in (4), as well as feedback connections \( W^{\text{fb}} \) from \( y(n-1) \) to \( \tilde{x}(n) \) in (2). A graphical

Fig. 1: An echo state network.
Training ESN

\[ Y^{\text{target}} = W^{\text{out}} X \]

Probably the most universal and stable solution to (8) in this context is ridge regression, also known as regression with Tikhonov regularization:

\[ W^{\text{out}} = Y^{\text{target}} X^T (XX^T + \beta I)^{-1}, \]

where \( \beta \) is a regularization coefficient explained in Section 4.2, and \( I \) is the identity matrix.

Overfitting (regularization):

\[ W^{\text{out}} = \arg \min_{W^{\text{out}}} \frac{1}{N_y} \sum_{i=1}^{N_y} \left( \sum_{n=1}^{T} (y_i(n) - y_i^{\text{target}}(n))^2 + \beta \|w_i^{\text{out}}\|^2 \right), \]
Beyond echo state networks

- **Good aspects of ESNs**
  Echo state networks can be trained very fast because they just fit a linear model.

- They demonstrate that it's very important to initialize weights sensibly.

- They can do impressive modeling of one-dimensional time-series.
  - but they cannot compete seriously for high-dimensional data.

- **Bad aspects of ESNs**
  They need many more hidden units for a given task than an RNN that learns the hidden→hidden weights.

- Ilya Sutskever (2012) has shown that if the weights are initialized using the ESN methods, RNNs can be trained very effectively.
  - He uses rmsprop with momentum.
Similar models

- Liquid State Machines (Maas et al., 2002)
  - A spiking version of Echo-state networks

- Extreme Learning Machines
  - Feed-forward network with a hidden layer.
  - Input-to-hidden weights are randomly initialized and never updated
Time Delay
Neural Networks
Skipped points
Skipping

• Stability
• Continuous-time recurrent networks
• Attractor networks

Stability of Discrete Time Recurrent Neural Networks and Nonlinear optimization problems

Dr. Nikita Barabanov, and Jayant Singh

Abstract We consider the method of Reduction of Dissipativity Domain to prove global Lyapunov stability of Discrete Time Recurrent Neural Networks. The standard and advanced criteria for Absolute Stability of these essentially nonlinear systems produce rather weak results. The method mentioned above is proved to be more powerful. It involves a multi-step procedure with maximization of special nonconvex functions over polytopes on every step. We derive conditions which guarantee an existence of at most one point of local maximum for such functions over every hyperplane. This nontrivial result is valid for wide range of neuron transfer functions.
An Empirical Exploration of Recurrent Network Architectures

Rafal Jozefowicz
Google Inc.

Wojciech Zaremba
New York University, Facebook

Ilya Sutskever
Google Inc.

RAFALJ GOOGLE.COM
WOJ.ZAREMBA GMAIL.COM
ILYASU GOOGLE.COM
VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

Andrej Karpathy* Justin Johnson* Li Fei-Fei
Department of Computer Science, Stanford University
{karpathy, jcjohns, feifeili}@cs.stanford.edu