Computer Scientists Prove Why Bigger Neural Networks Do Better

Two researchers show that for neural networks to be able to remember better, they need far more parameters than previously thought.

Object detection

Regions of Interest (RoIs): Regions that are likely to contain objects

Previously on CENG501!
Object detection

Object category (car, person, ..)
Object position & size (x, y, w, h)

Classification Loss $\mathcal{L}_c$
Localisation Loss $\mathcal{L}_r$

Total Loss: $\mathcal{L} = \mathcal{L}_c + w_r \mathcal{L}_r$

Object detection

Previously on CENG501!

Two-stage Approaches:
Fast R-CNN, Faster R-CNN, Libra R-CNN

Region Proposal Network

Feature Extraction Network

Regions of Interest

Classification Network

Regression Network

Detections

Fixed set of region-of-interests (anchors)

One-stage Approaches:
SSD, YOLO, RetinaNet

Previously on CENG501!
Example detectors

• R-CNN
• Fast R-CNN
• Faster R-CNN
• Mask R-CNN
• YOLO
• SSD
• RetinaNet
• FCOS
Takes 19 layer VGG as the base (no FC layers)

Max pooling is replaced by avg pooling since it produced more appealing results
Overall loss:

$$\mathcal{L}_{total}(\tilde{p}, \tilde{a}, \tilde{x}) = \alpha \mathcal{L}_{content}(\tilde{p}, \tilde{x}) + \beta \mathcal{L}_{style}(\tilde{a}, \tilde{x})$$
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
Unfolding

Feed-forward networks

Recurrent networks

time →
Backpropagation through Vanilla RNN

In general:
\[ h_t = \tanh(W^{xh} \cdot x_t + W^{hh} \cdot h_{t-1}) \]
\[ \hat{y}_t = \text{softmax}(W^{hy} \cdot h_t) \]
\[ L_t = CE(\hat{y}_t, y_t) \]

In total:
\[ L = \sum_t L_t \]

\[ \frac{\partial L}{\partial W^{hy}} =? \]

\[ = \frac{\partial L}{\partial \hat{y}_n} \frac{\partial \hat{y}_n}{\partial W^{hy}} + \frac{\partial L}{\partial \hat{y}_{n-1}} \frac{\partial \hat{y}_{n-1}}{\partial W^{hy}} + \ldots + \frac{\partial L}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial W^{hy}} \]

\[ = \sum_{t=1..n} \frac{\partial L}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W^{hy}} \]
Backpropagation through Vanilla RNN

In general:
\[ h_t = \tanh(W^{xh} \cdot x_t + W^{hh} \cdot h_{t-1}) \]
\[ \hat{y}_t = \text{softmax}(W^{hy} \cdot h_t) \]
\[ L_t = CE(\hat{y}_t, y_t) \]

In total:
\[ L = \sum_t L_t \]

Previously on CENG501!
Backpropagation through Vanilla RNN

In general:

\[ h_t = \tanh(W^{xh} \cdot x_t + W^{hh} \cdot h_{t-1}) \]

\[ \hat{y}_t = \text{softmax}(W^{hy} \cdot h_t) \]

\[ L_t = CE(\hat{y}_t, y_t) \]

In total:

\[ L = \sum_t L_t \]

\[
\frac{\partial L}{\partial W^{xh}} = ?
\]

\[
= \frac{\partial L}{\partial h_n} \frac{\partial h_n}{\partial W^{xh}} + \frac{\partial L}{\partial h_{n-1}} \frac{\partial h_{n-1}}{\partial W^{xh}} + \cdots + \frac{\partial L}{\partial h_1} \frac{\partial h_1}{\partial W^{xh}}
\]

\[
\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} + \frac{\partial L}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t}
\]

(calculated before)

Previously on CENG501!
Exploding and vanishing gradients problem

Solution 1: Gradient clipping for exploding gradients:

```
Algorithm 1 Pseudo-code for norm clipping

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

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

\[
\Omega = \sum_k \Omega_k = \sum_k \left( \left| \frac{\partial \mathcal{E}}{\partial x_{k+1}} \frac{\partial x_{k+1}}{\partial x_k} \right| - 1 \right)^2
\]
Exploding and vanishing gradients problem

• Solution 2:
  • Use methods like LSTM
LSTM in detail

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

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

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

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

Eqs: Karpathy

Image: C. Olah

Alternative formulation:

\[
\begin{align*}
i_t & = g(W_z i x_t + W_h i h_{t-1} + b_i) \\
f_t & = g(W_z f x_t + W_h f h_{t-1} + b_f) \\
o_t & = g(W_z o x_t + W_h o h_{t-1} + b_o)
\end{align*}
\]
LSTM Variants #3: Gated Recurrent Units

• Changes:
  • No explicit memory; memory = hidden output
  • \( Z = \text{memorize new and forget old} \)

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

Figure 1: Activations — c for LSTM and h for GRU — for networks trained on $a^n b^n$ and $a^n b^n c^n$. The LSTM has clearly learned to use an explicit counting mechanism, in contrast with the GRU.
Today

• Recurrent Neural Networks (RNNs)
  • Text/language modeling
  • Image captioning
  • Machine translation

<table>
<thead>
<tr>
<th>Week</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 13 (3 June)</td>
<td>RNNs</td>
</tr>
<tr>
<td>Week 14 (10 June)</td>
<td>RNNs + Attention</td>
</tr>
<tr>
<td>Week 15 (17 June)</td>
<td>Recent Studies/Approaches</td>
</tr>
</tbody>
</table>
Administrative Issues

• Programming Assignment 2 (PA2):
  • Deadline: 5 June

• Programming Assignment 3 (PA3):
  • Tentative Deadline: 19 June

• Final exam:
  • Tentative Deadline: 23-26 June.

• Projects:
  • Tentative Deadline: 5 July.

From: Pashacoffee.com
Example: Character-level Text Modeling
Character-level Text Modeling

• Problem definition: Find $c_{n+1}$ given $c_1, c_2, \ldots, c_n$.

• Modelling:

$$p(c_{n+1} \mid c_n, \ldots, c_1)$$

• In general, we just take the last $N$ characters:

$$p(c_{n+1} \mid c_n, \ldots, c_{n-(N-1)})$$

• Learn $p(c_{n+1} = 'a' \mid 'Ankar')$ from data such that

$$p(c_{n+1} = 'a' \mid 'Ankar') > p(c_{n+1} = 'o' \mid 'Ankar')$$
A simple scenario

- Alphabet: h, e, l, o
- Text to train to predict: “hello”
Sampling: Greedy

• Greedy sampling: Take the most likely word at each step

```python
from numpy import array
from numpy import argmax

# greedy decoder
def greedy_decoder(data):
    # Index for largest probability each row
    return [argmax(s) for s in data]

# define a sequence of 10 words over a vocab of 5 words
data = [[0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5]]

data = array(data)

# decode sequence
result = greedy_decoder(data)
print(result)
```

Running the example outputs a sequence of integers that could then be mapped back to words in the vocabulary.

```
[4, 0, 4, 0, 4, 0, 4, 0, 4, 0]
```

Code: https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/
Sampling: Beam Search

• What happens if we want $k$ most likely sequences instead of one?
• Beam search: Consider $k$ most likely words at each step, and expand search.

Figure: http://mttalks.ufal.ms.mff.cuni.cz/index.php

Figure: https://geekyisawesome.blogspot.com.tr/2016/10/using-beam-search-to-generate-most.html
Sampling: **Beam Search**

- Beam search: Consider $k$ most likely words at each step, and expand search.
  (take log for numerical stability; take $-\log()$ for minimizing the score)

```python
from math import log
from numpy import array, argmax

# beam search
def beam_search_decoder(data, k):
    sequences = [[[], 0.0]]
    # walk over each step in sequence
    for row in data:
        all_candidates = list()
        # expand each current candidate
        for i in range(len(sequences)):
            seq, score = sequences[i]
            for j in range(len(row)):
                candidate = [seq + [j], score + log(row[j])]
                all_candidates.append(candidate)
        # order all candidates by score
        ordered = sorted(all_candidates, key=lambda tup:tup[1])
        # select k best
        sequences = ordered[:k]
    return sequences
```

```python
# define a sequence of 10 words over a vocab of 5 words
data = [[0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1]]

# decode sequence
result = beam_search_decoder(data, 3)

# print result
for seq in result:
    print(seq)
```

Code: https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/
More on beam search

• Beam search is applied during inference.
• With modifications on the training procedure, it is possible to use it during training as well.

Sequence-to-Sequence Learning as Beam-Search Optimization

Sam Wiseman and Alexander M. Rush
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA, USA
{swiseman,srush}@seas.harvard.edu

https://arxiv.org/abs/1606.02960
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.

Slide: Hinton
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?

ymmartamattomyydellansakaan
Sample predictions (when trained on the works of Shakespeare):

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

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’ capitalide was 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 Imminences]], 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]], to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Bajoard]], 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/7754800786d17551963a09.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 traped of aid exile.]]
Sample predictions
(when trained on Latex documents):

• Using multi-layer LSTM
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. (6\textsuperscript{th} 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.
Example: WORD-level Text Modeling
Word-level Text Modeling

• Problem definition: Find $\omega_{n+1}$ given $\omega_1, \omega_2, ..., \omega_n$.

• Modelling:

$$p(\omega_{n+1} \mid \omega_n, ..., \omega_1)$$

• In general, we just take the last $N$ words:

$$p(\omega_{n+1} \mid \omega_n, ..., \omega_{n-(N-1)})$$

• Learn $p(\omega_{n+1} = 'Turkey' \mid 'Ankara is the capital of ')$ from data such that:

$$p(\omega_{n+1} = 'Turkey' \mid 'Ankara is the capital of ') > p(\omega_{n+1} = 'UK' \mid 'Ankara is the capital of ')$$
A handicap

• The number of characters is low enough to handle without doing anything extra.
  • English has 26 characters.

• The situation is very different for words.
  • English has ~ 170,000 different words!

• This increases dimensionality and makes it difficult to capture “semantics”.

• Solution: Map words to a lower dimensional space, a.k.a. word embedding (word2vec).
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

More examples
More examples

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

http://deeplearning4j.org/word2vec
More examples

http://deeplearning4j.org/word2vec
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)

• 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
Note that the weight vector is a look-up table

https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1
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)

• Given a sentence:
  the quick brown fox jumped over the lazy dog
• For each word, take context to be
  (N-words to the left, N-words to the right)
• If $N = 1$ (context, word):
  ([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...
Two different ways to train

2. Using word to predict a target context (skip-gram)

<table>
<thead>
<tr>
<th>Window Size</th>
<th>Text</th>
<th>Skip-grams</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ The <strong>wide</strong> road shimmered ] in the hot sun.</td>
<td>wide, the wide, road shimmered, in</td>
</tr>
<tr>
<td>2</td>
<td>The [ <strong>wide</strong> road <strong>shimmered</strong> ] in the ] hot sun.</td>
<td>shimmered, wide shimmered, road shimmered, in shimmered, the</td>
</tr>
<tr>
<td></td>
<td>The wide road shimmered in [ the hot <strong>sun</strong> ] .</td>
<td>sun, the sun, hot</td>
</tr>
<tr>
<td></td>
<td>[ The <strong>wide</strong> road shimmered in ] the hot sun.</td>
<td>wide, the wide, road shimmered, in</td>
</tr>
<tr>
<td>3</td>
<td>[ The <strong>wide</strong> road <strong>shimmered</strong> in the hot ] sun.</td>
<td>shimmered, the shimmered, wide shimmered, road shimmered, in shimmered, the shimmered, hot</td>
</tr>
<tr>
<td></td>
<td>The wide road shimmered [ in the hot <strong>sun</strong> ] .</td>
<td>sun, in sun, the sun, hot</td>
</tr>
</tbody>
</table>

https://www.tensorflow.org/tutorials/text/word2vec
Some details

• CBOW is called continuous BOW since the context is regarded as a BOW and it is continuous.

• In both approaches, the networks are composed of linear units

• The output units are usually normalized with the softmax

• According to Mikolov:
  • “Skip-gram: works well with small amount of the training data, represents well even rare words or phrases.
  
  • CBO: several times faster to train than the skip-gram, slightly better accuracy for the frequent words”
Example: Image Captioning
Demo video

https://vimeo.com/146492001
Overview

Pre-trained word embedding is also used.

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

Image: Karpathy
one to one

one to many

many to one

many to many

many to many

http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Training

“straw hat”

before:

\[ h_0 = \max(0, W_{xh} \ast x_0) \]

now:

\[ h_0 = \max(0, W_{xh} \ast x_0 + W_{ih} \ast v) \]
test image
test image

sample! <END> token => finish.
Example: Neural Machine Translation
Neural Machine Translation

• Model

Each box is an LSTM or GRU cell.

Sutskever et al. 2014

Haitham Elmarakeby
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Neural Machine Translation

Cho: From Sequence Modeling to Translation
Neural Machine Translation

- Model- encoder
Neural Machine Translation

• Model- *decoder*

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

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

Given
(i) the “summary” \((h)\) of the input sequence,
(ii) the previous output / word \((f_{t-1})\)
(iii) the previous state \((z_{t-1})\)

the hidden state of the decoder is:
\[
z_t = RNN(z_{t-1}, f_{t-1}, h)
\]

Then, we can find the most likely next word:
\[
P(f_t | f_{t-1}, f_{t-2}, ..., h) = p(f_t | z_t, f_{t-1}, h)
\]
Encoder-decoder

• Jointly trained to maximize

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

This can be done with CNNs
Check the following tutorial

• http://smerity.com/articles/2016/google_nmt_arch.html