Character-level Text Modeling

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

- Modelling:
  \[ p(c_{n+1} \mid c_n, ..., c_1) \]

- In general, we just take the last $N$ characters:
  \[ p(c_{n+1} \mid c_n, ..., c_{n-(N-1)}) \]

- 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

# 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
Word-level Text Modeling

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

- Modelling:
  
  $$p(\omega_{n+1} | \omega_n, \ldots, \omega_1)$$

- In general, we just take the last $N$ words:
  
  $$p(\omega_{n+1} | \omega_n, \ldots, \omega_{n-(N-1)})$$

- Learn $p(\omega_{n+1} = 'Turkey' | 'Ankara is the capital of ')$ from data such that:
  
  $$p(\omega_{n+1} = 'Turkey' | 'Ankara is the capital of ') > p(\omega_{n+1} = 'UK' | 'Ankara is the capital of ')$$
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).
More examples

Male-Female

Verb tense

Country-Capital

Previously on CENG501!
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

Previously on CENG501!
Overview

Pre-trained word embedding is also used

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

Image: Karpathy
Training

Previously on CENG501!

```
before:
h0 = max(0, Wxh * x0)

now:
h0 = max(0, Wxh * x0 + Wih * v)
```
Neural Machine Translation

• Model

Each box is an LSTM or GRU cell.

Sutskever et al. 2014

Haitham Elmarakeby
Neural Machine Translation

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

Cho: From Sequence Modeling to Translation

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

• Recurrent Neural Networks (RNNs)
  • Attention

• Neural Turing Machines

<table>
<thead>
<tr>
<th>Week</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<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):
  • Deadline: 21 June

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

• Projects:
  • Deadline: 5 July.

17 June : Last day of classes
09-12 July : Bayram
12 July : Submitting grades
17 July : Incomplete grades
Attention
Attention

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho Yoshua Bengio
Université de Montréal

BLEU: Bilingual Evaluation Understudy
https://cloud.google.com/translate/automl/docs/evaluate#bleu
In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i|y_1,\ldots,y_{i-1}, x) = g(y_{i-1}, s_i, c_i),$$

(4)

where $s_i$ is an RNN hidden state for time $i$, computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

It should be noted that unlike the existing encoder–decoder approach (see Eq. (2)), here the probability is conditioned on a distinct context vector $c_i$ for each target word $y_i$.

The context vector $c_i$ depends on a sequence of annotations $(h_1,\ldots,h_{T_x})$ to which an encoder maps the input sentence. Each annotation $h_i$ contains information about the whole input sequence with a strong focus on the parts surrounding the $i$-th word of the input sequence. We explain in detail how the annotations are computed in the next section.

The context vector $c_i$ is, then, computed as a weighted sum of these annotations $h_j$:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j,$$

(5)

The weight $\alpha_{ij}$ of each annotation $h_j$ is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

(6)

where

$$e_{ij} = a(s_{i-1}, h_j)$$

is an alignment model which scores how well the inputs around position $j$ and the output at position $i$ match. The score is based on the RNN hidden state $s_{i-1}$ (just before emitting $y_i$, Eq. (4)) and the $j$-th annotation $h_j$ of the input sentence.

We parametrize the alignment model $a$ as a feedforward neural network which is jointly trained with all the other components of the proposed system. Note that unlike in traditional machine translation,
Attention mechanism: A two-layer neural network.

Input: $z_i$ and $h_j$

Output: $e_j$, a scalar for the importance of word $j$.

The scores of words are normalized: $a_j = \text{softmax}(e_j)$
Attention

What does Attention in Neural Machine Translation Pay Attention to?

Hamidreza Ghader and Christof Monz
Informatics Institute, University of Amsterdam, The Netherlands
h.ghader, c.monz@uva.nl

2017
Attention Types

• Let’s rewrite Bahdanau et al.’s attention model:

\[ c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \] ; Context vector for output \( y_t \)

\[ \alpha_{t,i} = \text{align}(y_t, x_i) \] ; How well two words \( y_t \) and \( x_i \) are aligned.

\[ = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))} \] ; Softmax of some predefined alignment score.

\[ \text{score}(s_t, h_i) = v_a^\top \text{tanh}(W_a[s_t; h_i]) \]

where both \( v_a \) and \( W_a \) are weight matrices to be learned in the alignment model.

# Attention Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Alignment score function</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-base attention</td>
<td>score(s_t, h_i) = \text{cosine}[s_t, h_i]</td>
<td>Graves2014</td>
</tr>
<tr>
<td>Additive(*)</td>
<td>score(s_t, h_i) = v^T_a \text{tanh}(W_a[s_t; h_i])</td>
<td>Bahdanau2015</td>
</tr>
<tr>
<td>Location-Base</td>
<td>(\alpha_{ti} = \text{softmax}(W_a s_t))</td>
<td>Luong2015</td>
</tr>
<tr>
<td></td>
<td>Note: This simplifies the softmax alignment to only depend on the target position.</td>
<td></td>
</tr>
<tr>
<td>General</td>
<td>score(s_t, h_i) = s^T_t W_a h_i</td>
<td>Luong2015</td>
</tr>
<tr>
<td></td>
<td>where (W_a) is a trainable weight matrix in the attention layer.</td>
<td></td>
</tr>
<tr>
<td>Dot-Product</td>
<td>score(s_t, h_i) = s^T_t h_i</td>
<td>Luong2015</td>
</tr>
<tr>
<td>Scaled Dot-Product(^)</td>
<td>score(s_t, h_i) = \frac{s^T_t h_i}{\sqrt{n}}</td>
<td>Vaswani2017</td>
</tr>
<tr>
<td></td>
<td>Note: very similar to the dot-product attention except for a scaling factor; where (n) is the dimension of the source hidden state.</td>
<td></td>
</tr>
</tbody>
</table>

(*) Referred to as “concat” in Luong, et al., 2015 and as “additive attention” in Vaswani, et al., 2017.
(^) It adds a scaling factor \(1/\sqrt{n}\), motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention(&amp;)</td>
<td>Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.</td>
<td>Cheng2016</td>
</tr>
<tr>
<td>Global/Soft</td>
<td>Attending to the entire input state space.</td>
<td>Xu2015</td>
</tr>
<tr>
<td>Local/Hard</td>
<td>Attending to the part of input state space; i.e. a patch of the input image.</td>
<td>Xu2015; Luong2015</td>
</tr>
</tbody>
</table>
Self-attention

Fig. 6. The current word is in red and the size of the blue shade indicates the activation level. (Image source: Cheng et al., 2016)

Soft/hard attention

Fig. 7. “A woman is throwing a frisbee in a park.” (Image source: Fig. 6(b) in Xu et al. 2015)

Global/local attention

Fig. 8. Global vs local attention (Image source: Fig 2 & 3 in Luong, et al., 2015)

Attention: Transformer

• Vanilla self attention:

\[ e_i' = \sum_j \frac{\exp(e_j^T e_i)}{\sum_m \exp(e_m^T e_i)} e_j \]

• Scaled-dot product attention:

\[ e_i' = \sum_j \frac{\exp(k(e_j^T q(e_i)))}{\sum_m \exp(k(e_m^T q(e_i)))} v(e_j) \]

Attention\((Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\)
Input

Embedding

Queries

Keys

Values

Score

Thinking

Machines

$x_1$

$q_1$

$k_1$

$v_1$

$q_1 \cdot k_1 = 112$

$x_2$

$q_2$

$k_2$

$v_2$

$q_1 \cdot k_2 = 96$

https://jalammar.github.io/illustrated-transformer/
https://jalammar.github.io/illustrated-transformer/
Fig. 17. The full model architecture of the transformer. (Image source: Fig 1 & 2 in Vaswani, et al., 2017.)

Tutorial on transformers

• https://e2e.ml/school/transformers.html
• https://jalammar.github.io/illustrated-transformer/
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different downstream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).
GPT-3

- 175B parameters!
- Similar to BERT, a transformer-based model pretrained with masked language tasks.

Figure 1.1: **Language model meta-learning.** During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term “in-context learning” to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

Figure 1.2: **Larger models make increasingly efficient use of in-context information.** We show in-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description (see Sec. 3.9.2). The steeper “in-context learning curves” for large models demonstrate improved ability to learn a task from contextual information. We see qualitatively similar behavior across a wide range of tasks.
Another attempt at a longer piece. An imaginary Jerome K. Jerome writes about Twitter. All I seeded was the title, the author’s name and the first "It", the rest is done by #gpt3

Here is the full-length version as a PDF: drive.google.com/file/d/IqtPa1c...

The importance of being on twitter
by Jerome K. Jerome
London, Summer 1897

It is a curious fact that the last remaining form of social life in which people of London are still interested is Twitter. I was struck with this curious fact when I went on one of my periodical holidays to the seaside, and found the whole place twittering like a starving cage. I called it an anomaly, and it is.

I spoke to the sexton, whose cottage, like all sexton's cottages, is full of antiquities and interesting relics of former centuries. I said to him, "My dear sexton, what does all this twittering mean?" And he replied, "Why, sir, of course it means Twitter." "Ah!" I said, "I know about that. But what is Twitter?"

"It is a system of short and pithy sentences strung together in groups, for the purpose of conveying useful information to the initiated, and entertainment and the exercise of wits to the initiated, and entertainment and the exercise of wits to the rest of us."

"Very interesting," I said, "Has it a name?"
"It has," he said, "It is called Twitter."
"Yes," I said, "I know that, but what is it?"
"It is a system of information," he said.
"Oh, yes," I replied, "But what is it?"

"Why, sir," he said, "you can go up to any of the gentlemen you see twittering in the street, and say to him, 'You are a fool,' or 'Your wife is an adulteress,' or 'You have stolen that hat,' and if he is a member of the initiated he will answer you in the same form and tell you that you are a liar, or that your eyes resemble the eyes of a duck, or that you have stepped out of your part in the last charade you acted in, or that you were for a short time a statistician in a Government Office, and he will go on to tell you the whole story of your life, in language so exceedingly small and pointed that even you will be glad you can't understand it.

that's all
Two Natural-Language AI Algorithms Walk Into A Bar...

...And reveal some persistently bigoted tendencies of GPT-3

Abubakar Abid, an electrical engineer researching artificial intelligence at Stanford University, got curious. He has access to GPT-3, the massive natural language model developed by the California-based lab OpenAI, and when he tried giving it a variation on the joke—“Two Muslims walk into”—the results were decidedly not funny. GPT-3 allows one to write text as a prompt, and then see how it expands on or finishes the thought. The output can be eerily human...and sometimes just eerie. Sixty-six out of 100 times, the AI responded to “two Muslims walk into a...” with words suggesting violence or terrorism.

“Two Muslims walked into a...gay bar in Seattle and started shooting at will, killing five people.” Or: “...a synagogue with axes and a bomb.” Or: “...a Texas cartoon contest and opened fire.”

“At best it would be incoherent,” said Abid, “but at worst it would output very stereotypical, very violent completions.”
• Environmental & financial costs
  • Require vast data
    • Not necessarily diverse
    • Includes bias
    • Accountability/liability
• Stochastic Parrots
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.
A Practical Guide to Applying Echo State Networks

Mantas Lukoševičius

\[ \tilde{x}(n) = \tanh \left( W^{\text{in}}[1; u(n)] + W x(n - 1) \right), \]
\[ x(n) = (1 - \alpha)x(n - 1) + \alpha \tilde{x}(n), \]

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, \([::] \) 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 \( \tilde{x}(n) \equiv x(n) \).

![Echo State Network Diagram](image1)

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

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 \( W^{fb} \) from \( y(n - 1) \) to \( \tilde{x}(n) \) in (2). A graphical
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:

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

(9)

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.
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
Final remarks on RNNs
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.
VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

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

*equal contribution
†corresponding author
Neural Turing Machines
Why need other mechanisms?

• We mentioned before that RNNs are Turing Complete, right?
• The issues are:
  • The vanishing/exploding gradients (LSTM and other tricks address these issues)
  • However, # of parameters increase in LSTMs with the number of layers
  • Despite its advantages, LSTMs still fail to generalize to sequences longer than the training sequences
  • The answer to addressing bigger networks with less parameters is a better abstraction of the computational components, e.g., in a form similar to Turing machines

Weston et al., 2015
Turing Machine

Wikipedia:

Following Hopcroft and Ullman (1979, p. 148), a (one-tape) Turing machine can be formally defined as a 7-tuple $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ where

- $Q$ is a finite, non-empty set of states
- $\Gamma$ is a finite, non-empty set of tape alphabet symbols
- $b \in \Gamma$ is the blank symbol (the only symbol allowed to occur on the tape infinitely often at any step during the computation)
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of input symbols
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a partial function called the transition function, where $L$ is left shift, $R$ is right shift. (A relatively uncommon variant allows “no shift”, say N, as a third element of the latter set.) If $\delta$ is not defined on the current state and the current tape symbol, then the machine halts.\(^{[21]}\)
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of final or accepting states. The initial tape contents is said to be accepted by $M$ if it eventually halts in a state from $F$.

Anything that operates according to these specifications is a Turing machine.

Fig: Ucoluk & Kalkan, 2012
Neural Turing Machines

• If we make every component differentiable, we can train such a complex machine

• Accessing only a part of the network is problematic
  • Unlike a computer (TM), we need a differentiable access mechanism
Neural Turing Machines: Reading

• Let memory $\mathbf{M}$ be an $N \times M$ matrix
  • $N$: the number of “rows”
  • $M$: the size of each row (vector)
• Let $\mathbf{M}_t$ be the memory state at time $t$
• $w_t$: a vector of weightings over $N$ locations emitted by the read head at time $t$. Since the weights are normalized:
  $$\sum_i w_t(i) = 1, \quad 0 \leq w_t(i) \leq 1, \forall i$$
• $\mathbf{r}_t$: the read vector of length $M$:
  $$\mathbf{r}_t \leftarrow \sum_i w_t(i)\mathbf{M}_t(i).$$
• which is differentiable, and therefore, trainable.
Neural Turing Machines: Writing

• Writing = erasing content + adding new content
  • Inspired from LSTM’s forgetting and addition gates.

• Erasing: Multiply with an erase vector $\mathbf{e}_t \in [0,1]^M$
  \[ \tilde{\mathbf{M}}_t(i) \leftarrow \mathbf{M}_{t-1}(i)[1 - w_t(i)\mathbf{e}_t] \]
  $\mathbf{1}$: vector of ones. Multiplication here is pointwise.

• Adding: Add an add vector $\mathbf{a}_t \in [0,1]^M$:
  \[ \mathbf{M}_t(i) \leftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\mathbf{a}_t \]
Neural Turing Machines: Addressing

• Content-based addressing

• Location-based addressing
  • In a sense, use variable “names” to access content
Neural Turing Machines: Content-based Addressing

• Each head (reading or writing head) produces an $M$ length key vector $k_t$
  • $k_t$ is compared to each vector $M_t(i)$ using a similarity measure $K[.,.]$, e.g., cosine similarity:
    \[ K[u, v] = \frac{u \cdot v}{||u|| \cdot ||v||} \]

• From these similarity measures, we obtain a vector of “addressing”:
    \[ w^c_t(i) \leftarrow \frac{\exp(\beta_t K[k_t, M_t(i)])}{\sum_j \exp(\beta_t K[k_t, M_t(j)])} \]

• $\beta_t$: amplifies or attenuates the precision of the focus
Neural Turing Machines: Location-based Addressing

• Important for e.g. iteration over memory locations, or jumping to an arbitrary memory location

• First: Interpolation between addressing schemes using “interpolation gate” $g_t$:
  \[
  w_t^g \leftarrow g_t w_t^c + (1 - g_t) w_{t-1}
  \]
  - If $g_t = 1$: weight from content-addressable component is used
  - If $g_t = 0$: weight from previous step is used

• Second: rotationally shift weight to achieve location-based addressing using convolution:
  \[
  \hat{w}_t(i) \leftarrow \sum_{j=0}^{N-1} w_t^g(j) s_t(i - j)
  \]
  - $s_t$: shift amount. Three elements for how “much” to shift left, right or keep as it is.
  - It needs to be “sharp”. To keep it sharp, each head emits a scalar $\gamma^t \geq 1$:
  \[
  w_t(i) \leftarrow \frac{\hat{w}_t(i)\gamma^t}{\sum_j \hat{w}_t(j)\gamma^t}
  \]
Neural Turing Machines: Controller Network

• Free parameters
  • The size of the memory
  • Number of read-write heads
  • Range of allowed rotation shifts
  • Type of the neural network for controller

• Alternatives:
  • A recurrent network such as LSTM with its own memory
    • These memory units might be considered like “registers” on the CPU
  • A feed-forward network
    • Can use the memory to achieve recurrence
    • More transparent
Neural Turing Machines: Training

- Binary targets
  - Logistic sigmoid output layers
  - Cross-entropy loss
- Other schemes possible
- Tasks:
  - Copy from input to output
  - Repeat Copy: Make n copies of the input
  - Associative recall: Present a part of a sequence to recall the remaining part
  - N-gram: Learn distribution of 6-grams and make predictions for the next bit based on this distribution
  - Priority sort: Associate a priority as part of each vector and as the target place the sequence according to the priority

<table>
<thead>
<tr>
<th>Task</th>
<th>#Heads</th>
<th>Controller Size</th>
<th>Memory Size</th>
<th>Learning Rate</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>1</td>
<td>100</td>
<td>128 × 20</td>
<td>10⁻⁴</td>
<td>17,162</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>1</td>
<td>100</td>
<td>128 × 20</td>
<td>10⁻⁴</td>
<td>16,712</td>
</tr>
<tr>
<td>Associative</td>
<td>4</td>
<td>256</td>
<td>128 × 20</td>
<td>10⁻⁴</td>
<td>146,845</td>
</tr>
<tr>
<td>N-Grams</td>
<td>1</td>
<td>100</td>
<td>128 × 20</td>
<td>3 × 10⁻⁵</td>
<td>14,656</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>8</td>
<td>512</td>
<td>128 × 20</td>
<td>3 × 10⁻⁵</td>
<td>508,305</td>
</tr>
</tbody>
</table>

Table 1: NTM with Feedforward Controller Experimental Settings

<table>
<thead>
<tr>
<th>Task</th>
<th>#Heads</th>
<th>Controller Size</th>
<th>Memory Size</th>
<th>Learning Rate</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>1</td>
<td>3 × 256</td>
<td>128 × 20</td>
<td>3 × 10⁻⁵</td>
<td>67,561</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>1</td>
<td>3 × 256</td>
<td>128 × 20</td>
<td>3 × 10⁻⁵</td>
<td>67,561</td>
</tr>
<tr>
<td>Associative</td>
<td>1</td>
<td>3 × 256</td>
<td>128 × 20</td>
<td>3 × 10⁻⁵</td>
<td>70,330</td>
</tr>
<tr>
<td>N-Grams</td>
<td>1</td>
<td>3 × 256</td>
<td>128 × 20</td>
<td>3 × 10⁻⁵</td>
<td>61,749</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>5</td>
<td>2 × 512</td>
<td>128 × 20</td>
<td>3 × 10⁻⁵</td>
<td>269,038</td>
</tr>
</tbody>
</table>

Table 2: NTM with LSTM Controller Experimental Settings

<table>
<thead>
<tr>
<th>Task</th>
<th>Network Size</th>
<th>Learning Rate</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>3 × 256</td>
<td>3 × 10⁻⁵</td>
<td>1,352,909</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>3 × 256</td>
<td>3 × 10⁻⁵</td>
<td>5,312,007</td>
</tr>
<tr>
<td>Associative</td>
<td>3 × 256</td>
<td>3 × 10⁻⁵</td>
<td>3,444,518</td>
</tr>
<tr>
<td>N-Grams</td>
<td>3 × 128</td>
<td>3 × 10⁻⁵</td>
<td>331,905</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>3 × 128</td>
<td>3 × 10⁻⁵</td>
<td>384,424</td>
</tr>
</tbody>
</table>

Table 3: LSTM Network Experimental Settings
Neural Turing Machines: Training

Figure 3: Copy Learning Curves.

Figure 7: Repeat Copy Learning Curves.

Figure 10: Associative Recall Learning Curves for NTM and LSTM.

Figure 18: Priority Sort Learning Curves.
Other variants/attempt
Figure 2: One timestep of the NRAM architecture with \( R = 4 \) registers. The LSTM controller gets the „binarized” values \( r_1, r_2, \ldots \) stored in the registers as inputs and outputs the description of the circuit in the grey box and the probability of finishing the execution in the current timestep (See Sec. 3.3 for more detail). The weights of the solid thin connections are outputted by the controller. The weights of the solid thick connections are trainable parameters of the model. Some of the modules (i.e. READ and WRITE) may interact with the memory tape (dashed connections).

Published as a conference paper at ICLR 2016

NEURAL RANDOM-ACCESS MACHINES

Karol Kurach* & Marcin Andrychowicz* & Ilya Sutskever
Google
{kkurach, marcina, iylas}@google.com
Neural Programmer

```
Timestep t

Controller

Input

Soft Selection

Arithmetic and logic operations

Apply

Data

Memory

Output
```

Published as a conference paper at ICLR 2016

NEURAL PROGRAMMER: INDUCING LATENT PROGRAMS WITH GRADIENT DESCENT

Arvind Neelakantan*
University of Massachusetts Amherst
arvind@cs.umass.edu

Quoc V. Le
Google Brain
qvi@google.com

Ilya Sutskever
Google Brain
illys@clarklab.com
Figure 3: Examples of our model on Convex hulls (left), Delaunay (center) and TSP (right), trained on \( m \) points, and tested on \( n \) points. A failure of the LSTM sequence-to-sequence model for Convex hulls is shown in (a). Note that the baselines cannot be applied to a different length from training.
Bilbo travelled to the cave. Gollum dropped the ring there. Bilbo took the ring.
Bilbo went back to the Shire. Bilbo left the ring there. Frodo got the ring.
Frodo journeyed to Mount-Doom. Frodo dropped the ring there. Sauron died.
Frodo went back to the Shire. Bilbo travelled to the Grey-havens. The End.
Where is the ring? A: Mount-Doom
Where is Bilbo now? A: Grey-havens
Where is Frodo now? A: Shire

Published as a conference paper at ICLR 2015

MEMORY NETWORKS

Jason Weston, Sumit Chopra & Antoine Bordes
Facebook AI Research
770 Broadway
New York, USA
{jas, spchopra, abordes}@fb.com
Inferring and Executing Programs for Visual Reasoning

Justin Johnson¹  Bharath Hariharan²  Laurens van der Maaten²
Judy Hoffman¹  Li Fei-Fei¹  C. Lawrence Zitnick²  Ross Girshick²

¹Stanford University  ²Facebook AI Research

2017

Figure 2. System overview. The program generator is a sequence-to-sequence model which inputs the question as a sequence of words and outputs a program as a sequence of functions, where the sequence is interpreted as a prefix traversal of the program’s abstract syntax tree. The execution engine executes the program on the image by assembling a neural module network [2] mirroring the structure of the predicted program.
More studies

• Differentiable Neural Machines
  • https://deepmind.com/blog/differentiable-neural-computers/

• Universal Turing Machine