CENG 501
Deep Learning
Week 14

Sinan Kalkan
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:

\[ L = \sum_t L_t \]

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

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

\[
= \sum_{t=1..n} \frac{\partial L}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W^{hy}}
\]
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:
\[ L = \sum_t L_t \]

The Vanilla RNN Model

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

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

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

In general:
\[
\mathbf{h}_t = \tanh(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1})
\]
\[
\hat{\mathbf{y}}_t = \text{softmax}(W^{hy} \cdot \mathbf{h}_t)
\]
\[
L_t = \text{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t)
\]

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

Previously on CENG501!

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

\[
= \frac{\partial L}{\partial \mathbf{h}_n} \frac{\partial \mathbf{h}_n}{\partial W^{xh}} + \frac{\partial L}{\partial \mathbf{h}_{n-1}} \frac{\partial \mathbf{h}_{n-1}}{\partial W^{xh}} + \ldots + \frac{\partial L}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial W^{xh}}
\]

\[
\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}
\]

(calculated before)
Exploding and vanishing gradients problem

Solution 1: Gradient clipping for exploding gradients:

\[
\hat{g} \leftarrow \frac{\partial \mathcal{L}}{\partial \theta} \\
\text{if } \|g\| \geq \text{threshold} \text{ then} \\
\quad \hat{g} \leftarrow \frac{\text{threshold}}{\|g\|} g \\
\text{end if}
\]

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

\[
\Omega = \sum_k \Omega_k = \sum_k \left( \frac{\| \frac{\partial \mathcal{L}}{\partial x_{k+1}} \|}{\| \frac{\partial \mathcal{L}}{\partial x_k} \|} - 1 \right)^2
\]
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 $h_t$:
  $$c_t = f \odot c_{t-1} + i \odot g$$
  $$h_t = o \odot \tanh(c_t)$$
  where $\odot$ is the element-wise product of vectors

Alternative formulation:

- Input gate: $i_t = g(W_{xi} x_t + W_{hi} h_{t-1} + b_i)$
- Forget gate: $f_t = g(W_{xf} x_t + W_{hf} h_{t-1} + b_f)$
- Output gate: $o_t = g(W_{xo} x_t + W_{ho} h_{t-1} + b_o)$

Eqs: Karpathy
A simple scenario

- Alphabet: h, e, l, o
- Text to train to predict: “hello”
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!
Training

Previously on CENG501!

“straw hat”

training example

before:

\[ h_0 = \max(0, Wx_0 \ast x_0) \]

now:

\[ h_0 = \max(0, Wx_0 \ast x_0 + Wih \ast v) \]
Neural Machine Translation

Cho: From Sequence Modeling to Translation

$f = (La, croissance, économique, s’est, ralentie, ces, dernières, années, .)

e = (Economic, growth, has, slowed, down, in, recent, years, .)
Today

• Recurrent Neural Networks
• Attention & Transformers
Administrative Issues

• Quiz on ODTUclass
• Programming Assignment 2 (PA2):
  • Deadline: 26 May – 30 May
• Programming Assignment 3 (PA3):
  • 2 June – 16 June
• Final exam:
  • 12 June Monday at 17:00.
  • BMB-1 & BMB-2
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

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

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, \([; ; \cdot]\) 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 representation is shown in 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:

\[ W^{\text{out}} = Y^{\text{target}} X^T \left( XX^T + \beta I \right)^{-1}, \tag{9} \]

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

Overfitting (regularization):

\[ W^{\text{out}} = \arg \min_{W^{\text{out}}} \frac{1}{N_Y} \sum_{i=1}^{N_Y} \left( \sum_{n=1}^{T} (y_i(n) - y_i^{\text{target}}(n))^2 + \beta \left\| W_i^{\text{out}} \right\|^2 \right), \]
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
An Empirical Exploration of Recurrent Network Architectures

Rafal Jozefowicz
Google Inc.
RAFAJK@GOOGLE.COM

Wojciech Zaremba
New York University, Facebook
WOJZAREMBA@GMAIL.COM

Ilya Sutskever
Google Inc.
ILYASI@GOOGLE.COM
VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

Andrej Karpathy* Justus Johnson† I J Fei-Fei
Department of Computer Science, Stanford University
{karpathy,jcjohnn,feifeili}@cs.stanford.edu
Attention
Attention

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:

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

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 \( h_j \):

\[ c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \]

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

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

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: $a_j = \text{softmax}(e_j)$
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:

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

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

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

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

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>
</table>
| Content-base attention      | \[
\text{score}(s_t, h_i) = \text{cosine}[s_t, h_i]\]

  |                                                                                      | Graves2014        |
| Additive(*)                 | \[
\text{score}(s_t, h_i) = v_a^\top \text{tanh}(W_a[s_t; h_i])\]

  |                                                                                      | Bahdanau2015      |
| Location-Base               | \[
\alpha_{t,i} = \text{softmax}(W_s s_t)\]

  Note: This simplifies the softmax alignment to only depend on the target position.  |
| General                     | \[
\text{score}(s_t, h_i) = s_t^\top W_a h_i\]

  where \(W_a\) is a trainable weight matrix in the attention layer.  |
| Dot-Product                 | \[
\text{score}(s_t, h_i) = s_t^\top h_i\]

  |                                                                                      | Luong2015         |
| Scaled Dot-Product(\^)      | \[
\text{score}(s_t, h_i) = \frac{s_t^\top h_i}{\sqrt{n}}\]

  Note: very similar to the dot-product attention except for a scaling factor; where \(n\) is the dimension of the source hidden state.  |

(*) 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.

# Attention Types

<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}(\frac{QK^T}{\sqrt{d_k}})V
**Input**

**Thinking**

**Machines**

**Embedding**

\(X_1\)  \(X_2\)

**Queries**

\(q_1\)  \(q_2\)

**Keys**

\(k_1\)  \(k_2\)

**Values**

\(v_1\)  \(v_2\)

\(W^Q\)  \(W^K\)  \(W^V\)

https://jalammar.github.io/illustrated-transformer/
https://jalammar.github.io/illustrated-transformer/
Input
Embedding
Queries
Keys
Values
Score
Divide by 8 ( $\sqrt{d_k}$ )
Softmax

Thinking

Machines

$x_1$ $x_2$

$q_1$ $q_2$

$k_1$ $k_2$

$v_1$ $v_2$

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

14

12

0.88

0.12

https://jalammar.github.io/illustrated-transformer/
Input

Embedding

Queries

Keys

Values

Score

Divide by 8 (\(\sqrt{d_k}\))

Softmax

Softmax

X

Value

Sum

Thinking

Machines

\(x_1\)

\(x_2\)

\(q_1\)

\(q_2\)

\(k_1\)

\(k_2\)

\(v_1\)

\(v_2\)

\[q_1 \cdot k_1 = 112\]

\[q_1 \cdot k_2 = 96\]

\[14\]

\[12\]

\[0.88\]

\[0.12\]

\(z_1\)

\(z_2\)

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://e2eml.school/transformers.html
• https://jalammar.github.io/illustrated-transformer/
Attention: BERT

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-1

- 12 layer decoder-only transformer
- Unsupervised pretraining
  - BookCorpus dataset
- Supervised finetuning
  - Textual alignment
  - QA & commonsense reasoning
  - Semantic similarity
  - Classification
Abstract

Natural language processing tasks, such as question answering, machine translation, reading comprehension, and summarization, are typically approached with supervised learning on task-specific datasets. We demonstrate that language models begin to learn these tasks without any explicit supervision when trained on a new dataset of millions of webpages called WebText. When conditioned on a document plus questions, the answers generated by the language model reach 55 F1 on the CoQA dataset - matching or exceeding the performance of 3 out of 4 baseline systems without using the 127,000+ training examples. The capacity of the language model is essential to the success of zero-shot task transfer and increasing it improves performance in a log-linear fashion across tasks. Our largest model, GPT-2, is a 15B parameter Transformer that achieves state-of-the-art results on 7 out of 8 tested language modeling datasets in a zero-shot setting but still underfits WebText. Samples from the model reflect these improvements and contain coherent paragraphs of text. These findings suggest a promising path towards building language processing systems which learn to perform tasks from their naturally occurring demonstrations.
GPT-3

- 175B parameters!

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/1qtPa1c...

---

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 the 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 sea-side, and found the whole place twittering like a starling-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 all the story of your life, in language so exceedingly small and pointed that even you will be glad you can't understand it.

---

This post is one of the best GPT-3 evaluations I've seen. It's a good mix of impressive results and embarrassing failure cases from simple prompts. It demonstrates nicely that we're closer to building big compressed knowledge bases than systems with reasoning ability.

---

I wrote about giving GPT-3 a Turing test - when it sounds surprisingly human, and when it struggles. lacker.io/2020/07/06/...

Q: What is your favorite animal?
A: My favorite animal is a dog.

Q: Why?
A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?
A: Two reasons that a dog might be in a bad mood are if

Q: How many eyes does a giraffe have?
A: A giraffe has two eyes.

---

GPT-3 often performs like a clever student who hasn't done their reading trying to bullshit their way through an exam. Some well-known facts, some half-truths, and some straight lies, strung together in what first looks like a smooth narrative.
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.

“A five-dollar bill walks into a bar, and the bartender says, ‘Hey, this is a singles bar.’” Or: “A neutron walks into a bar and orders a drink—and asks what he owes. The bartender says, ‘For you, no charge.’” And so on.

“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
Instruct-GPT & GPT3.5 (a.k.a., ChatGPT)

Training language models to follow instructions with human feedback

Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.
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
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
- $\delta : (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ 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.
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

Figure 1: Neural Turing Machine Architecture. During each update cycle, the controller network receives inputs from an external environment and emits outputs in response. It also reads to and writes from a memory matrix via a set of parallel read and write heads. The dashed line indicates the division between the NTM circuit and the outside world.

Neural Turing Machines

Alex Graves  gravesa@google.com
Greg Wayne  gregwayne@google.com
Ivo Danihelka  danihelka@google.com

Google DeepMind, London, UK
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 $e_t \in [0,1]^M$
  \[
  \hat{M}_t(i) \leftarrow M_{t-1}(i)(1 - w_t(i)e_t)
  \]
  $1$: vector of ones. Multiplication here is pointwise.

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

- Content-based addressing

- Location-based addressing
  - In a sense, use variable “names” to access content

Figure 1: Neural Turing Machine Architecture. During each update cycle, the controller network receives inputs from an external environment and emits outputs in response. It also reads to and writes from a memory matrix via a set of parallel read and write heads. The dashed line indicates the division between the NTM circuit and the outside world.

Figure 2: Flow Diagram of the Addressing Mechanism. The key vector, $k_t$, and key strength, $\beta_t$, are used to perform content-based addressing of the memory matrix, $M_t$. The resulting content-based weighting is interpolated with the weighting from the previous time step based on the value of the interpolation gate, $g_t$. The shift weighting, $s_t$, determines whether and by how much the weighting is rotated. Finally, depending on $\gamma_t$, the weighting is sharpened and used for memory access.
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”:
    $$w_t^\xi (i) \leftarrow \exp(\beta_t K[k_t, M_t(i)]) \div \sum_j \exp(\beta_t K[k_t, M_t(j)])$$
  • $\beta_t$: 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 x 20</td>
<td>10^{-4}</td>
<td>17,162</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>1</td>
<td>100</td>
<td>128 x 20</td>
<td>10^{-4}</td>
<td>16,712</td>
</tr>
<tr>
<td>Associative</td>
<td>4</td>
<td>256</td>
<td>128 x 20</td>
<td>10^{-4}</td>
<td>146,845</td>
</tr>
<tr>
<td>N-Grams</td>
<td>1</td>
<td>100</td>
<td>128 x 20</td>
<td>3 x 10^{-5}</td>
<td>14,656</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>8</td>
<td>512</td>
<td>128 x 20</td>
<td>3 x 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>100</td>
<td>128 x 20</td>
<td>10^{-4}</td>
<td>67,561</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>1</td>
<td>100</td>
<td>128 x 20</td>
<td>10^{-4}</td>
<td>66,111</td>
</tr>
<tr>
<td>Associative</td>
<td>1</td>
<td>100</td>
<td>128 x 20</td>
<td>10^{-4}</td>
<td>70,330</td>
</tr>
<tr>
<td>N-Grams</td>
<td>1</td>
<td>100</td>
<td>128 x 20</td>
<td>3 x 10^{-5}</td>
<td>61,749</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>5</td>
<td>2 x 100</td>
<td>128 x 20</td>
<td>3 x 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 x 256</td>
<td>3 x 10^{-5}</td>
<td>1,352,909</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>3 x 512</td>
<td>3 x 10^{-5}</td>
<td>5,312,007</td>
</tr>
<tr>
<td>Associative</td>
<td>3 x 256</td>
<td>10^{-4}</td>
<td>1,344,518</td>
</tr>
<tr>
<td>N-Grams</td>
<td>3 x 128</td>
<td>10^{-4}</td>
<td>331,905</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>3 x 128</td>
<td>3 x 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/attempts
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, iylasu}@google.com
Neural Programmer

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
qvl@google.com

Ilya Sutskever
Google Brain
ilyas@google.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.
Memory Networks


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
{jase, 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
ViT: Vision Transformers

Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).
Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Ze Liu†‡, Yutong Lin†‡, Yue Cao*, Han Hu‡‡, Yixuan Wei†
Zheng Zhang, Stephen Lin, Baining Guo
Microsoft Research Asia
{v-zeilui, v-yutlin, yuecao, hanhu, v-yixwe, zhez, stevelin, bainguo}@microsoft.com

Figure 2. An illustration of the \textit{shifted window} approach for computing self-attention in the proposed Swin Transformer architecture. In layer $l$ (left), a regular window partitioning scheme is adopted, and self-attention is computed within each window. In the next layer $l+1$ (right), the window partitioning is shifted, resulting in new windows. The self-attention computation in the new windows crosses the boundaries of the previous windows in layer $l$, providing connections among them.
# Swin Transformer V2: Scaling Up Capacity and Resolution

Ze Liu*, Han Hu††, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning
Yue Cao, Zheng Zhang, Li Dong, Furu Wei, Baining Guo

Microsoft Research Asia

{v-zeliul, hanhu, t-yutonglin, t-zhuyao, t-zhixie, t-yixuanwei, v-jianing}@microsoft.com

{yuecao, zhe, lidongl, fuwei, baingguo}@microsoft.com

## Table

<table>
<thead>
<tr>
<th>Method</th>
<th>param</th>
<th>pre-train images</th>
<th>pre-train length (#it)</th>
<th>pre-train im size</th>
<th>fine-tune time</th>
<th>fine-tune im size</th>
<th>ImageNet-1K-V1 top-1 acc</th>
<th>ImageNet-1K-V2 top-1 acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwinV1-B</td>
<td>88M</td>
<td>IN-22K-14M</td>
<td>1.3B</td>
<td>224^2</td>
<td>&lt;30^t</td>
<td>384^2</td>
<td>86.4</td>
<td>76.58</td>
</tr>
<tr>
<td>SwinV1-L</td>
<td>197M</td>
<td>IN-22K-14M</td>
<td>1.3B</td>
<td>224^2</td>
<td>&lt;10^t</td>
<td>384^2</td>
<td>87.3</td>
<td>77.46</td>
</tr>
<tr>
<td>VIT-G [65]</td>
<td>1.8B</td>
<td>JFT-3B</td>
<td>164B</td>
<td>224^2</td>
<td>&gt;30k</td>
<td>518^2</td>
<td>90.45</td>
<td>83.33</td>
</tr>
<tr>
<td>V-MoE [44]</td>
<td>14.7B*</td>
<td>JFT-3B</td>
<td>-</td>
<td>224^2</td>
<td>16.8k</td>
<td>518^2</td>
<td>90.35</td>
<td>-</td>
</tr>
<tr>
<td>CoAtNet-7 [11]</td>
<td>2.44B</td>
<td>JFT-3B</td>
<td>-</td>
<td>224^2</td>
<td>20.1k</td>
<td>512^2</td>
<td>90.88</td>
<td>-</td>
</tr>
<tr>
<td>SwinV2-B</td>
<td>88M</td>
<td>IN-22K-14M</td>
<td>1.3B</td>
<td>192^2</td>
<td>&lt;30^t</td>
<td>384^2</td>
<td>87.1</td>
<td>78.08</td>
</tr>
<tr>
<td>SwinV2-L</td>
<td>197M</td>
<td>IN-22K-14M</td>
<td>1.3B</td>
<td>192^2</td>
<td>&lt;20^t</td>
<td>384^2</td>
<td>87.7</td>
<td>78.31</td>
</tr>
<tr>
<td>SwinV2-G</td>
<td>3.0B</td>
<td>IN-22K-ext-70M</td>
<td>3.5B</td>
<td>192^2</td>
<td>&lt;0.5k^t</td>
<td>640^2</td>
<td>90.17</td>
<td>84.00</td>
</tr>
</tbody>
</table>

## Diagram

![Diagram of Swin Transformer V1 and V2](image-url)

### V1
- **Softmax**
- **Parameterized RPB**

### V2
- **MLP**
- **Layer Norm**
- **Attention**
- **cosine(q, k)/τ**
- **Log-CPB**

## Notes
- CopyPaste [17]
- SwinV1-L [35]
- YOLOR [53]
- CBNet [32]
- DyHead [10]
- SoftTeacher [60]

## Metrics

<table>
<thead>
<tr>
<th>Method</th>
<th>train I(W) size</th>
<th>test I(W) size</th>
<th>mini-val (AP)</th>
<th>test-dev (AP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>box mask</td>
<td>box mask</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CopyPaste [17]</td>
<td>1280(9)</td>
<td>1280(9)</td>
<td>57.0</td>
<td>57.3</td>
</tr>
<tr>
<td>SwinV1-L [35]</td>
<td>800(7)</td>
<td>ms(7)</td>
<td>58.0</td>
<td>58.7</td>
</tr>
<tr>
<td>YOLOR [53]</td>
<td>1280(9)</td>
<td>1280(9)</td>
<td>59.6</td>
<td>60.1</td>
</tr>
<tr>
<td>CBNet [32]</td>
<td>1400(7)</td>
<td>ms(7)</td>
<td>60.3</td>
<td>60.6</td>
</tr>
<tr>
<td>DyHead [10]</td>
<td>1200(9)</td>
<td>ms(-)</td>
<td>60.7</td>
<td>61.3</td>
</tr>
<tr>
<td>SwinV2-L (HTC++)</td>
<td>1100(12)</td>
<td>ms(12)</td>
<td>58.8</td>
<td>60.8</td>
</tr>
<tr>
<td>SwinV2-G (HTC++)</td>
<td>1536(32)</td>
<td>ms(48)</td>
<td>61.7</td>
<td>63.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>53.1</td>
<td>53.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>61.9</td>
<td>62.5</td>
</tr>
</tbody>
</table>

* denotes the best performance.
† denotes the top-4 performance.
\( \times \) denotes the top-8 performance.

This analysis reveals that Swin Transformer V2 significantly outperforms its predecessor in terms of both capacity and resolution, making it a promising model for future research and applications.
SimMIM: A Simple Framework for Masked Image Modeling

Zhenda Xie*  Zheng Zhang*  Yue Cao*
Yutong Lin  Jianmin Bao  Zhuliang Yao  Qi Dai  Han Hu*

Microsoft Research Asia

{t-zhxie, zhez, yuecao, t-yutonglin, jianmin.bao, t-zhuyao, qi.dai, hanhu}@microsoft.com

Figure 1. An illustration of our simple framework for masked language modeling, named SimMIM. It predicts raw pixel values of the randomly masked patches by a lightweight one-layer head, and performs learning using a simple $\ell_1$ loss.

Figure 2. Illustration of masking area generated by different masking strategies using a same mask ratio of 0.6: square masking [36], block-wise masking [1] apply on 16-sized patches, and our simple random masking strategy on different patch sizes (e.g., 4, 8, 16 and 32).
A ConvNet for the 2020s

Zhuang Liu$^{1,2}$, Hanzi Mao$^1$ Chao-Yuan Wu$^1$ Christoph Feichtenhofer$^1$ Trevor Darrell$^2$ Saining Xie$^{1,\dagger}$

$^1$Facebook AI Research (FAIR) $^2$UC Berkeley

Code: https://github.com/facebookresearch/ConvNeXt

<table>
<thead>
<tr>
<th>Model</th>
<th>ImageNet-1K Acc (%)</th>
<th>ImageNet-22K Pre-trained Acc (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50/200</td>
<td>78.6</td>
<td>81.5</td>
</tr>
<tr>
<td>ResNetXt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inverted Bottleneck</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large Kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Micro Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConvNeXt-T/B</td>
<td>81.8</td>
<td>81.3</td>
</tr>
</tbody>
</table>

Figure 1. ImageNet-1K classification results for • ConvNets and ◆ vision Transformers. Each bubble’s area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take $224^2/384^2$ images respectively. ResNet and ViT results were obtained with improved training procedures over the original papers. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.
Self-supervised learning
- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don’t know and predict that.
Fig. 3. Illustration of self-supervised learning by rotating the entire input images. The model learns to predict which rotation is applied. (Image source: Gidaris et al. 2018)

Fig. 4. Illustration of self-supervised learning by predicting the relative position of two random patches. (Image source: Doersch et al., 2015)

Fig. 8. Illustration of context encoder. (Image source: Pathak, et al., 2016)

Siamese Networks

Contrastive Loss
(Chopra et al., 2005)

$y = 1$ for “similar” pairs:

$$
\mathcal{L}_{cont}(x_i, x_j, \theta) = 1[y_i = y_j]||f_\theta(x_i) - f_\theta(x_j)||^2_2 + 1[y_i \neq y_j] \max(0, \ e - ||f_\theta(x_i) - f_\theta(x_j)||^2_2)^2
$$


https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html#triplet-loss
Triplet Loss (Schroff et al., 2015)

\[
\mathcal{L}_{\text{triplet}}(x, x^+, x^-) = \sum_{x \in \mathcal{X}} \max \left( 0, \|f(x) - f(x^+)^2\|_2 - \|f(x) - f(x^-)^2\|_2 + \epsilon \right)
\]

Fig. 1. Illustration of triplet loss given one positive and one negative per anchor. (Image source: Schroff et al. 2015)

https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html#triplet-loss

https://omoindrot.github.io/triplet-loss
Lifted Structure Loss (Song et al., 2015)

Let $D_{ij} = \|f(x_i) - f(x_j)\|_2$, a structured loss function is defined as

$$\mathcal{L}_{\text{struct}} = \frac{1}{2|P|} \sum_{(i,j) \in P} \max(0, \mathcal{L}^{(ij)}_{\text{struct}})^2$$

where $\mathcal{L}^{(ij)}_{\text{struct}} = D_{ij} + \max \left( \max_{(i,k) \in \mathcal{N}} e - D_{ik}, \max_{(j,k) \in \mathcal{N}} e - D_{jk} \right)$

Fig. 2. Illustration compares contrastive loss, triplet loss and lifted structured loss. Red and blue edges connect similar and dissimilar sample pairs respectively. (Image source: Song et al. 2015)

https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html
N-pair Loss (Sohn 2016)

\[
\mathcal{L}_{N\text{-pair}}(x, x^+, \{x_i^-\}_{i=1}^{N-1}) = \log \left( 1 + \sum_{i=1}^{N-1} \exp(f(x)^T f(x_i^-) - f(x)^T f(x^+)) \right) \\
= -\log \frac{\exp(f(x)^T f(x^+))}{\exp(f(x)^T f(x^+)) + \sum_{i=1}^{N-1} \exp(f(x)^T f(x_i^-))}
\]

https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html
Simple Contrastive Learning (Chen et al., 2020)

3) The contrastive learning loss is defined using cosine similarity \( \text{sim}(\cdot, \cdot) \). Note that the loss operates on an extra projection layer of the representation \( g(\cdot) \) rather than on the representation space directly. But only the representation \( \mathbf{h} \) is used for downstream tasks.

\[
L_{\text{SimCLR}}^{(i,j)} = - \log \frac{\exp(\text{sim}(z_i, z_j) / \tau)}{\sum_{k=1}^{2N} \delta_{\{k \neq i\}} \exp(\text{sim}(z_i, z_k) / \tau)}
\]

where \( \delta_{\{k \neq i\}} \) is an indicator function: 1 if \( k \neq i \) 0 otherwise.

SimCLR needs a large batch size to incorporate enough negative samples to achieve good performance.

---

https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html
Momentum Contrast (MoCo – He et al., 2019)

\[ L_{\text{MoCo}} = - \log \frac{\exp(q \cdot k^+/\tau)}{\sum_{i=1}^{N} \exp(q \cdot k_i/\tau)} \]

\[ \theta_k \leftarrow m\theta_k + (1 - m)\theta_q \]

Fig. 12. Illustration of how Momentum Contrast (MoCo) learns visual representations. (Image source: He et al, 2019)

https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html
Bootstrap Your Own Latent (BYOL – Grill et al., 2020)

• Does not use negative samples

Given an image \( x \), the BYOL loss is constructed as follows:

- Create two augmented views: \( v = t(x) \); \( v' = t'(x) \) with augmentations sampled \( t \sim T, t' \sim T' \);
- Then they are encoded into representations, \( y_\theta = f_\theta(v) \); \( y'_\theta = f_\theta(v') \);
- Then they are projected into latent variables, \( z_\theta = g_\theta(y_\theta) \); \( z'_\theta = g_\theta(y'_\theta) \);
- The online network outputs a prediction \( q_\theta(z_\theta) \);
- Both \( q_\theta(z_\theta) \) and \( z' \) are L2-normalized, giving us \( \hat{q}_\theta(z_\theta) = q_\theta(z_\theta) / \| q_\theta(z_\theta) \| \) and \( \hat{z}' = z' / \| z' \| \);
- The loss \( L^{\text{BYOL}}_\theta \) is MSE between L2-normalized prediction \( \hat{q}_\theta(z) \) and \( \hat{z}' \);
- The other symmetric loss \( L^{\text{BYOL}}_{\theta'} \) can be generated by switching \( v' \) and \( v \); that is, feeding \( v' \) to online network and \( v \) to target network.
- The final loss is \( L^{\text{BYOL}}_{\theta} + L^{\text{BYOL}}_{\theta'} \) and only parameters \( \theta \) are optimized.

Fig. 10. The model architecture of BYOL. After training, we only care about \( f_\theta \) for producing representation, \( y = f_\theta(x) \), and everything else is discarded. sg means stop gradient. (Image source: Grill, et al 2020)

\[ \xi \leftarrow \tau \xi + (1 - \tau) \theta. \]

https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html
Simple Siamese Representation Learning (SimSiam – Chen et al., 2020)

• “BYOL without momentum encoder”

---

**Algorithm 1 SimSiam Pseudocode, PyTorch-like**

```python
# f: backbone + projection mlp
# h: prediction mlp

for x in loader:  # load a minibatch x with n samples
    x1, x2 = aug(x), aug(x)  # random augmentation
    z1, z2 = f(x1), f(x2)  # projections, n-by-d
    p1, p2 = h(z1), h(z2)  # predictions, n-by-d

    L = D(p1, z2)/2 + D(p2, z1)/2  # loss

    L.backward()  # back-propagate
    update(f, h)  # SGD update

def D(p, z):  # negative cosine similarity
    z = z.detach()  # stop gradient

    p = normalize(p, dim=1)  # 12-normalize
    z = normalize(z, dim=1)  # 12-normalize

    return -(p*z).sum(dim=1).mean()
```

---

https://github.com/facebookresearch/simsiam
Resources on SSL

• The rise of SSL, by Y. Lecun: https://www.youtube.com/watch?v=05wUrb5Ej8Q&t=21252s

Generative models
Generative Models: Outline

- Generative Adversarial Networks (GANs)
- Variational Autoencoders (VAEs)
- Energy-based Models
- Autoregressive Models
- Reversible Models
Generative Adversarial Networks
Generative Adversarial Networks (GANs)

• Originally proposed by Ian Goodfellow in 2014
• It all started in a pub 😊
Generative Adversarial Networks (GANs)

We have two networks:

- **Generator (G):** Generates a fake image given a noise (embedding) vector \(z\)
- **Discriminator (D):** Discriminates whether an image is fake or real.

http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them
Generative Adversarial Networks (GANs)

- With two competing networks, we solve the following minimax game:
  \[
  \min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_Z(z)} [\log (1 - D(G(z)))]
  \]

\(D(x)\): Probability that \(x\) is real (came from data).

\(\log (1 - D(G(z)))\) is minimized by \(G\).
Figure 1: Generative adversarial nets are trained by simultaneously updating the discriminative distribution \(D\), blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) \(p_{\text{data}}\) from those of the generative distribution \(p_{\text{g}}\) (G) (green, solid line). The lower horizontal line is the domain from which \(z\) is sampled, in this case uniformly. The horizontal line above is part of the domain of \(x\). The upward arrows show how the mapping \(x = G(z)\) imposes the non-uniform distribution \(p_{\text{g}}\) on transformed samples. \(G\) contracts in regions of high density and expands in regions of low density of \(p_{\text{g}}\). (a) Consider an adversarial pair near convergence: \(p_{\text{g}}\) is similar to \(p_{\text{data}}\) and \(D\) is a partially accurate classifier. (b) In the inner loop of the algorithm \(D\) is trained to discriminate samples from data, converging to \(D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{g}}(x)}\). (c) After an update to \(G\), gradient of \(D\) has guided \(G(z)\) to flow to regions that are more likely to be classified as data. (d) After several steps of training, if \(G\) and \(D\) have enough capacity, they will reach a point at which both cannot improve because \(p_{\text{g}} = p_{\text{data}}\). The discriminator is unable to differentiate between the two distributions, i.e. \(D(x) = \frac{1}{2}\).
Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used k = 1, the least expensive option, in our experiments.

for number of training iterations do
  for k steps do
    • Sample minibatch of m noise samples \{z^{(1)}, \ldots, z^{(m)}\} from noise prior \(p_g(z)\).
    • Sample minibatch of m examples \{x^{(1)}, \ldots, x^{(m)}\} from data generating distribution \(p_{data}(x)\).
    • Update the discriminator by ascending its stochastic gradient:
      \[
      \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].
      \]
  end for
  • Sample minibatch of m noise samples \{z^{(1)}, \ldots, z^{(m)}\} from noise prior \(p_g(z)\).
  • Update the generator by descending its stochastic gradient:
    \[
    \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right).
    \]
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and “deconvolutional” generator)

Fig: Goodfellow et al., 2014.
Mode collapse in GANs

• Problem:
  • The generator network maps the different z (embedding/noise) values into similar images.
Mode collapse in GANs

• Solutions:
  • Changing the training procedure (use batch discrimination instead of individual discrimination)
  • Experience replay (show old fake images again and again)
  • Use a different loss (+ enforce diversity)
  • ...

• Other tips and tricks:
Deep Convolutional GAN

- GAN with convolutional layers
- More stable

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.
Conditional GANs

http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them
Text to image with GANs

This flower has small, round violet petals with a dark purple center.

\[ \hat{x} := G(z, \varphi(t)) \]

This flower has small, round violet petals with a dark purple center.

\[ D(\hat{x}, \varphi(t)) \]

Generator Network

Discriminator Network

(this small bird has a pink breast and crown, and black primaries and secondaries)

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee, 2016.
Cycle GAN

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Jun-Yan Zhu* 
Taesung Park* 
Phillip Isola 
Alexei A. Efros
Berkeley AI Research (BAIR) laboratory, UC Berkeley

Figure 1: Given any two unordered image collections $X$ and $Y$, our algorithm learns to automatically "translate" an image from one into the other and vice versa: (left) 1,074 Monet paintings and 5,753 landscape photos from Flickr; (center) 1,177 zebras and 939 horses from ImageNet; (right) 1,273 summer and 854 winter Yosemite photos from Flickr. Example application (bottom): using a collection of paintings of a famous artist, learn to render a user's photograph into their style.

https://junyanz.github.io/CycleGAN/
Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators $D_Y$ and $D_X$. $D_Y$ encourages $G$ to translate $X$ into outputs indistinguishable from domain $Y$, and vice versa for $D_X$, $F$, and $X$. To further regularize the mappings, we introduce two “cycle consistency losses” that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F),$$

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))].$$

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1].$$
Cycle GAN

https://junyanz.github.io/CycleGAN/
Example

https://www.digitaltrends.com/cool-tech/nvidia-ai-winter-summer-car/
GAN -- state of the art

https://github.com/NVlabs/stylegan2
The zoo of GANs

• https://deephunt.in/the-gan-zoo-79597dc8c347
Variational-AE

Fig: http://kvfrans.com/variational-autoencoders-explained/
Variational Inference

• Why need VI?
  • Intractability

1. **Intractability**: the case where the integral of the marginal likelihood $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)\,dz$ is intractable (so we cannot evaluate or differentiate the marginal likelihood), where the true posterior density $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$ is intractable (so the EM algorithm cannot be used), and where the required integrals for any reasonable mean-field VB algorithm are also intractable. These intractabilities are quite common and appear in cases of moderately complicated likelihood functions $p_\theta(x|z)$, e.g. a neural network with a nonlinear hidden layer.

2. **A large dataset**: we have so much data that batch optimization is too costly; we would like to make parameter updates using small minibatches or even single datapoints. Sampling-based solutions, e.g. Monte Carlo EM, would in general be too slow, since it involves a typically expensive sampling loop per datapoint.
• $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$: The dataset consisting of $N$ i.i.d. samples of continuous or discrete variable $\mathbf{x}$.

• We assume that the data is generated by some random process, involving unobserved continuous random variable $\mathbf{z}$.
  1. A random $\mathbf{z}^{(i)}$ is generated from some prior distribution $p_{\theta^*}(\mathbf{z})$.
  2. A value $\mathbf{x}^{(i)}$ is generated from a conditional distribution $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$.

• **Condition:** $p_{\theta^*}(\mathbf{z})$ and $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$ are differentiable ($\theta^*$ are parameters).

• **Limitation:** $\mathbf{z}^{(i)}$ and parameters $\theta^*$ (true parameters) are unknown.

• **Problem:** $p_{\theta}(\mathbf{x})$ and $p_{\theta}(\mathbf{z} \mid \mathbf{x}) = p_{\theta}(\mathbf{x} \mid \mathbf{z})p_{\theta}(\mathbf{z})/p_{\theta}(\mathbf{x})$ are intractable.
Solution:

- Replace intractable true posterior $p_\theta(z \mid x)$ with a recognition model $q_\phi(z \mid x)$.
- $q_\phi(z \mid x)$: probabilistic encoder. Produces a probability distribution over $z$ given $x$.
- $p_\theta(x \mid z)$: probabilistic decoder. Produces a probability distribution over $x$ given $z$.

Fig: http://kvfrans.com/variational-autoencoders-explained/
A practical issue

• Problematic with very high variance and impractical
• Solution: Reparameterize $\tilde{z} \sim q_\phi(z|x)$ using a differentiable transformation $g_\phi(\epsilon, x)$ with an auxiliary noise variable $\epsilon$:
  \[
  \tilde{z} = g_\phi(\epsilon, x) \quad \text{with} \quad \epsilon \sim p(\epsilon)
  \]

Fig: http://kvfrans.com/variational-autoencoders-explained/
Training

• How can we know $q_\phi(z|x)$ approximates $p(z|x)$ well?

$$KL(q_\phi(z|x) \parallel p(z|x)) = E_q[\log q_\phi(z|x)] - E_q[\log p(x,z)] + \log p(x)$$

• Goal: find parameters that minimize this divergence.
• However, this is impossible to compute because of $p(x)$
• Let us re-write the equation:

$$\log p(x) = -E_q[\log q_\phi(z|x)] + E_q[\log p(x,z)] + KL(q_\phi(z|x) \parallel p(z|x))$$

Evidence Lower Bound (ELBO)

• KL divergence is always greater than or equal to zero
• This means that minimizing KL divergence is equivalent to maximizing the ELBO term (note that $p(x)$ is constant given the dataset)
• ELBO can be re-written as follows for a single data point:

\[
ELBO_i(\phi) = E_{q_\phi(z|x_i)}[\log p(x_i|z)] - KL(q_\phi(z|x_i) \parallel p(z))
\]

• ELBO is the negative of the loss function:

\[
ELBO_i(\phi) = -L_i(\theta, \phi)
\]

\[
= - \left( E_{q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - KL(q_\phi(z|x_i) \parallel p(z)) \right)
\]

(reconstruction loss) (regularizer)
Resources

• https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
• http://kvfrans.com/variational-autoencoders-explained/
• https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf
Other Methods: Boltzmann Machines

- By Hinton & Sejnowski (1985)
- Boltzmann machines can be seen as the **stochastic counterpart of Hopfield nets**
- In fact, they have the same energy definition:
  \[ E = - \sum_i \sum_{j<i} w_{ij} s_i s_j + \sum_i \theta_i s_i \]
- However, we have hidden neurons now
  - The availability of hidden variables \( \Rightarrow \) bigger class of distributions that can be modeled \( \Rightarrow \) in principle, we can model distributions of arbitrary complexity
- Moreover, it is stochastic
  \[ p_{i=1} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)} \]
Other Methods: Autoregressive models

Figure 1: **Left:** A visualization of the PixelCNN that maps a neighborhood of pixels to prediction for the next pixel. To generate pixel $x_i$ the model can only condition on the previously generated pixels $x_1, \ldots x_{i-1}$. **Middle:** an example matrix that is used to mask the 5x5 filters to make sure the model cannot read pixels below (or strictly to the right) of the current pixel to make its predictions. **Right:** Top: PixelCNNs have a *blind spot* in the receptive field that cannot be used to make predictions. Bottom: Two convolutional stacks (blue and purple) allow to capture the whole receptive field.
Deep reinforcement learning
The agent receives reward $r_t$ for its actions.
More formally

• An agent’s behavior is defined by a policy, $\pi$:
  \[ \pi : S \rightarrow \mathcal{P}(\mathcal{A}) \]

  $S$: The space of states.

  $\mathcal{A}$: The space of actions.

• The "return" from a state is usually:
  \[ R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i) \]

  $r(s_i, a_i)$: the reward for action $a_i$ in state $s_i$.

  $\gamma$: discount factor.

• Goal: Learn a policy that maximizes the expected return from the starting position:
  \[ \mathbb{E}_{r_i,s_i\sim E,a_i\sim \pi} [R_1] \]

http://www.cs.ubc.ca/~murphyk/Bayes/pomdp.html
More formally

• We can define an expected return for taking action \( a_t \) at state \( s_t \):

\[
Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i\geq t}, s_{i\geq t}\sim E, a_{i\geq t}\sim \pi} [R_t | s_t, a_t]
\]

• This can be rewritten as (called the Bellman equation):

\[
Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}\sim E} [r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1}\sim \pi} [Q^\pi(s_{t+1}, a_{t+1})]]
\]
Reinforcement Learning in/with Deep Networks

• Two general approaches:
  • Value gradients
  • Policy gradients
Q values of actions are predicted at the output.

Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map $\phi$, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

 LETTER

Human-level control through deep reinforcement learning


2015

doi:10.1038/nature14236
network. We refer to a neural network function approximator with weights $\theta$ as a Q-network. A Q-network can be trained by adjusting the parameters $\theta_i$ at iteration $i$ to reduce the mean-squared error in the Bellman equation, where the optimal target values $r + \gamma \max_{a'} Q^*(s', a')$ are substituted with approximate target values $y = r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$, using parameters $\theta_i^-$ from some previous iteration. This leads to a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration $i$,

$$L_i(\theta_i) = \mathbb{E}_{s,a,r} \left[ (\mathbb{E}_q [y|s,a] - Q(s,a; \theta_i))^2 \right]$$

---

**LETTER**

doi:10.1038/nature14236

**Human-level control through deep reinforcement learning**

Volodymyr Mnih*, Koray Kavukcuoglu*, David Silver*, Andrei A. Rusu1, Joel Veness1, Marc G. Bellemare1, Alex Graves4, Martin Riedmiller1, Andreas K. Fidjeland1, Georg Ostrovski1, Stig Petersen1, Charles Beattie1, Amir Sadik1, Ioannis Antonoglou1, Helen King1, Daan Wierstra1, Shane Legg1 & Demis Hassabis2

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1$, $M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1, T$ do
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
Policy gradients

[Diagram of a neural network with raw pixels input and a hidden layer with a probability of moving UP output]

http://karpathy.github.io/2016/05/31/rl/
Policy gradients

http://karpathy.github.io/2016/05/31/rl/