Continual Lifelong Meta-Learning & Artificial Curiosity

Jürgen Schmidhuber
The Swiss AI Lab IDSIA
Univ. Lugano & SUPSI
http://www.idsia.ch/~juergen

NNAISENSE
Jürgen Schmidhuber
You_again Shmidhoobuh

Continual Learning (Mark Ring, 1994-)
Meta-Learning (JS, 1987-)
Artificial Curiosity (JS, 1990-)
“True” Learning to Learn (L2L) is not just transfer learning!
Even a simple feedforward NN can transfer-learn to learn new images faster through pre-training on other image sets.

True L2L is not just about learning to adjust a few hyperparameters such as mutation rates in evolution strategies (e.g., Rechenberg & Schwefel, 1960s).
Radical L2L is about encoding the initial learning algorithm in a universal language (e.g., on an RNN), with primitives that allow to modify the code itself in arbitrary computable fashion. Then surround this self-referential, self-modifying code by a recursive framework that ensures that only “useful” self-modifications are executed or survive (Recursive Self-Improvement).
Lifelong R-Learning to Learn Learning Algorithms (1994)

Jürgen Schmidhuber
The Swiss AI Lab IDSIA
Univ. Lugano & SUPSI
http://www.idsia.ch/~juergen

NNAISENSE

1987: Diploma thesis on meta-learning how to learn how to learn & recursive self-improvement
### Internal State

<table>
<thead>
<tr>
<th>Addresses</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5321</td>
</tr>
<tr>
<td>1</td>
<td>-44</td>
</tr>
<tr>
<td>2</td>
<td>810</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
</tr>
<tr>
<td>4</td>
<td>-3322</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>-189</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>237</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
</tr>
</tbody>
</table>

### Instruction Pointer

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: ADD(a1,a2,a3)</td>
<td>0.001 0.0014 0.9 0.24 0.001 0.0014 0.9 0.9</td>
</tr>
<tr>
<td>1: MUL(a1,a2,a3)</td>
<td>0.001 0.0014 0.04 0.01 0.001 0.0014 0.04 0.04</td>
</tr>
<tr>
<td>2: SUB(a1,a2,a3)</td>
<td>0.99 0.0014 0.01 0.01 0.99 0.0014 0.01 0.01</td>
</tr>
<tr>
<td>3: JMLEQ(a1,a2,a3)</td>
<td>0.001 0.99 0.01 0.01 0.001 0.99 0.01 0.01</td>
</tr>
<tr>
<td>4: MOVEAGENT(a1,a2)</td>
<td>0.001 0.0014 0.01 0.7 0.001 0.0014 0.01 0.01</td>
</tr>
<tr>
<td>5: InvokeSSA()</td>
<td>0.004 0.0014 0.01 0.01 0.004 0.0014 0.01 0.01</td>
</tr>
<tr>
<td>6: INCPROB(a1,a2)</td>
<td>0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01</td>
</tr>
<tr>
<td>7: DECPROB(a1,a2)</td>
<td>0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01</td>
</tr>
</tbody>
</table>

### External Environment

Variables Probability Distributions

### Perceptions
### Internal State

<table>
<thead>
<tr>
<th>Addresses</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTENTS</td>
<td>5321</td>
<td>-44</td>
<td>810</td>
<td>-2</td>
<td>-3322</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>-189</td>
<td>2</td>
<td>237</td>
<td>6</td>
</tr>
</tbody>
</table>

### Instruction Pointer

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
</tr>
<tr>
<td>0.0014</td>
</tr>
<tr>
<td>0.9</td>
</tr>
<tr>
<td>0.24</td>
</tr>
<tr>
<td>0.001</td>
</tr>
<tr>
<td>0.0014</td>
</tr>
<tr>
<td>0.9</td>
</tr>
<tr>
<td>0.9</td>
</tr>
</tbody>
</table>

### Perceptions

- External Environment

### Variable Probability Distributions

- ADD(a1, a2, a3)
- MUL(a1, a2, a3)
- SUB(a1, a2, a3)
- JMLEQ(a1, a2, a3)
- MOVEAGENT(a1, a2)
- InvokeSSA()
- INCROB(a1, a2)
- DECROB(a1, a2)
INTERNAL STATE

<table>
<thead>
<tr>
<th>ADDRESSES</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5321</td>
</tr>
<tr>
<td>1</td>
<td>-44</td>
</tr>
<tr>
<td>2</td>
<td>810</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
</tr>
<tr>
<td>4</td>
<td>-3322</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>-189</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>237</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
</tr>
</tbody>
</table>

INSTRUCTION POINTER

PARAMETERS

PERCEPTIONS

EXTERNAL ENVIRONMENT

VARIABLE PROBABILITY DISTRIBUTIONS

0 = ADD(a1, a2, a3)  
1 = MUL(a1, a2, a3)  
2 = SUB(a1, a2, a3)  
3 = JMPLEQ(a1, a2, a3)  
4 = MOVEAGENT(a1, a2)  
5 = InvokeSSA()  
6 = INCRBO(a1, a2)  
7 = DCRBO(a1, a2)
### Table: Addresses and Contents

<table>
<thead>
<tr>
<th>Addresses</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = ADD(a1, a2, a3)</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.9</td>
<td>0.24</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.9</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = MUL(a1, a2, a3)</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.04</td>
<td>0.01</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.04</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 = SUB(a1, a2, a3)</td>
<td>0.99</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.01</td>
<td>0.99</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 = JMLEQ(a1, a2, a3)</td>
<td>0.001</td>
<td>0.99</td>
<td>0.01</td>
<td>0.01</td>
<td>0.001</td>
<td>0.99</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 = MOVEAGENT(a1, a2)</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.7</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 = InvokeSSA()</td>
<td>0.004</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.4</td>
<td>0.004</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 = INCPROB(a1, a2)</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.01</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 = DECPROB(a1, a2)</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.01</td>
<td>0.001</td>
<td>0.0014</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram: Internal State Diagram

- **Instruction Pointer:**
- **Perceptions:**
- **External Environment:**
- **Self-Modification:**
- **Variable Probability Distributions:**
Success-story algorithm (SSA) for self-modifying code (since 1994)

R(t): Reward until time t. Stack of past check points \(v_1v_2v_3\ldots\) with self-mods in between. SSA undoes selfmods after \(v_i\) that are not followed by long-term reward acceleration up until t (now):

\[
\frac{R(t)}{t} < \frac{[R(t)-R(v_1)]}{(t-v_1)} < \frac{[R(t)-R(v_2)]}{(t-v_2)} < \ldots
\]
1997: Lifelong meta-RL with self-modifying policies and success-story algorithm: 2 agents, 2 doors, 2 keys. 1st southeast wins 5, the other 3. Through recursive self-modifications only: from 300,000 steps per trial down to 5,000.
Universal Search: run all programs until one of them finds and verifies a solution, where a program of $k$ bits gets $2^{-k}$ of total search time.

Leonid Levin
1973

Fastest solver for given problem class, save for constant factor $O(n^3) 10^{100} = O(n^3)$.

Time-optimal incremental search and algorithmic transfer learning in program space

Branches of search tree are program prefixes

Node-oriented backtracking restores partially solved task sets & modified memory components on error or when $\sum t > PT$
61 primitive instructions operating on stack-like and other internal data structures. For example:

```plaintext
callfun(fn), jumpif(val,address), quote(), unquote(), boost_probability(n,val) ....
```

Programs are integer sequences; data and code look the same; makes functional programming easy.
Towers of Hanoi: incremental solutions

- $+1\text{ms}, \ n=1$: \textit{(movdisk)}
- 1 day, $n=1,2$: \textit{(c4 c3 cpn c4 by2 c3 by2 exec)}
- 3 days, $n=1,2,3$: \textit{(c3 dec boostq defnp c4 calltp c3 c5 calltp endnp)}
- 4 days: $n=4, n=5, \ldots, n=30$: \textit{by same double-recursive program}
- Profits from 30 earlier context-free language tasks ($1^n 2^n$): \textit{transfer learning}
- 93,994,568,009 prefixes tested
- 345,450,362,522 instructions
- 678,634,413,962 time steps
- longest single run: 33 billion steps (5% of total time)! Much deeper than recent memory-based “deep learners” …
- top stack size for restoring storage: $< 20,000$
What the found **Towers of Hanoi** solver does:

- `(c3 dec boostq defnp c4 calltp c3 c5 calltp endnp)`
- **Prefix** increases P of double-recursive procedure:
  Hanoi(Source,Aux,Dest,n): IF n=0 exit; ELSE BEGIN
  Hanoi(Source,Dest,Aux,n-1); move top disk from Aux to Dest;
  Hanoi(Aux,Source,Dest,n-1); END
- **Prefix boosts** instructions of previously frozen program, which happens to be a previously learned solver of a context-free language \(1^n 2^n\). This rewrites search procedure itself: **Benefits of metalearning!**
- **Prefix probability** 0.003; **suffix probability** \(3 \times 10^{-8}\); total probability \(9 \times 10^{-11}\)
- **Suffix probability without prefix execution**: \(4 \times 10^{-14}\)
- That is, Hanoi does profit from \(1^n 2^n\) experience and incremental learning (OOPS excels at algorithmic transfer learning): speedup factor 1000
Gödel Machine (2003): agent-controlling program that speaks about itself, ready to rewrite itself in arbitrary fashion once it has found a proof that the rewrite is useful, given a user-defined utility function. Theoretically optimal self-improver!
Initialize Gödel Machine by Marcus Hutter’s asymptotically fastest method for all well-defined problems.

Given \( f: X \rightarrow Y \) and \( x \in X \), search proofs to find program \( q \) that provably computes \( f(z) \) for all \( z \in X \) within time bound \( t_q(z) \); spend most time on \( f(x) \)-computing \( q \) with best current bound

\[ n^3 + 10^{1000} = n^3 + O(1) \]

As fast as fastest \( f \)-computer, save for factor \( 1 + \varepsilon \) and \( f \)-specific const. independent of \( x \)!
Separation of Storage and Control (Zuse 1936) for NNs: End-to-End Differentiable Neural Stack Machines (Das, Giles, Mike Mozer, 1992), NTM & DNC (Graves et al 2014-16) & Memory Nets (Weston et al 2014)

Neural stack machine of 1992-1993

Figure 1: The demon model.

Figure 3: A continuous stack. The symbols indicate the contents; the height of a stack entry indicates its thickness, also given by the number to the right. The top composite symbol on the stack is a combination of the items forming a total thickness of 1.0; the next composite symbol is a combination of the items making up the next 1.0 units of thickness.

An RNN, but no LSTM yet. In 2001, however, Sepp Hochreiter taught a meta-LSTM to learn a learning algorithm for quadratic functions that was faster than backprop.
1993: More elegant Hebb-inspired addressing to go from (#hidden) to (#hidden)^2 temporal variables: gradient-based RNN learns to control internal end-to-end differentiable spotlights of attention for fast differentiable memory rewrites – again fast weights

Schmidhuber, ICANN 1993: Reducing the ratio between learning complexity and number of time-varying variables in fully recurrent nets.

Similar NIPS 2016 paper by Ba et al. See I. Schlag at NIPS Metalearning Symposium 2017!
New fast weight addressing scheme: Imanol Schlag @ NIPS Meta-learning Workshop 2017
2005:
Reinforcement-Learning or Evolving RNNs with Fast Weights

Robot learns to balance 1 or 2 poles through 3D joint

Gomez & Schmidhuber:
Co-evolving recurrent neurons learn deep memory POMDPs.
GECCO 2005

http://www.idsia.ch/~juergen/evolution.html


A bit like universal AIXI, but with feasible local search
IJNS 1991: R-Learning of Visual Attention on 100,000 times slower computers

http://people.idsia.ch/~juergen/attentive.html

Fig. 1. A typical visual scene. The diameters of the receptive fields of the retina's input units are indicated by circles.

Fig. 2. An artificial fovea provides inputs for a control network which is able to move the fovea around. A model network is trained to predict the next input from the current input and the current controller action.
1991: current goal=extra fixed input
2018: all of this is coming back!

Fig. 5. One controller for various targets specified by an additional constant input: Examples of fovea trajectories leading from various start positions to different targets. The first target is near the left corner of the triangle. The second target is near the lower corner.
RoboCup World Champion 2004, Fastest League, 5m/s

Lookahead expectation & planning with neural networks (Schmidhuber, IEEE INNS 1990): successfully used for RoboCup by Alexander Gloye-Förster (went to IDSIA)

http://www.idsia.ch/~juergen/learningrobots.html

Alex @ IDSIA, led FU Berlin’s RoboCup World Champion Team 2004
Train agent inside of its own hallucinated dream generated by its world model, and transfer policy back into actual environment.

Made possible by David Ha (Google)
On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning RNN-based Controllers (RNNAIs) and Recurrent Neural World Models

http://arxiv.org/abs/1511.09249
How to motivate the controller to improve the world model?
1990s: UNSUPERVISED NEURAL NETS FIGHT EACH OTHER IN A MINIMAX GAME. EACH NET MINIMIZES THE VALUE FUNCTION MAXIMIZED BY THE OTHER TO LEARN A MODEL OF THE PROBABILITY DISTRIBUTION ON GIVEN DATA OR TO GENERATE EXPERIMENTS YIELDING INTRINSIC REWARD FOR CURIOSITY.
1990: Active Unsupervised Minimax for RL

Adversarial Reinforcement Learning (RL) for agents with Artificial Curiosity (1990): A reward-maximising neural control network C learns to generate action sequences or experiments in an environment. It gets intrinsic reward in proportion to the prediction errors of a separate neural network called the world model M. M learns to predict future inputs, given past inputs and actions. Again, in the absence of external reward, C is maximising exactly the same value function that M is minimising. This motivates C to invent and generate experiments that lead to "novel" situations where M does not yet know how to predict well [plan1, int1].


More than 40 follow-up papers on artificial curiosity:
http://people.idsia.ch/~juergen/interest.html
http://people.idsia.ch/~juergen/creativity.html
1991: Predictability Minimization (PM): 2 unsupervised nets fight minimax game to model given data distribution

Encoder maximizes objective minimized by predictor. Saddle point = ideal factorial code: $P(\text{pattern}) = P(c_1)P(c_2)\ldots P(c_n)$
1996: PM applied to images: learns orientation-sensitive bar detectors, on-center-off-surround detectors, etc
PM v GAN: latent space v original data space

<table>
<thead>
<tr>
<th>DATA</th>
<th>PM</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINIMAX TRAINED ENCODER</td>
<td>Standard decoder (often omitted)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA</th>
<th>GAN</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard encoder (InfoGAN)</td>
<td>MINIMAX TRAINED DECODER</td>
</tr>
</tbody>
</table>
1997-2002: More Sophisticated Unsupervised Minimax for RL:
What’s interesting? Exploring the predictable

Two dueling, reward-maximizing modules (both general computers) called *left brain* and *right brain* collectively design an *experiment*: a (probabilistic) program that defines how to execute an action sequence in the environment, and how to compute the final experimental outcome through an instruction sequence implementing a computable function (e.g., a binary yes/no classification) of the observation sequence triggered by the experiment. Both brains can predict experimental outcomes before they are known. If their predictions or hypotheses differ, after having generated and executed the experiment, the surprised loser pays an *intrinsic reward* to the winner in a *zero sum game*. Each brain is maximising the value function minimised by the other. This may also accelerate the intake of external reward [int5-7].
1997-2002: artificial curiosity through active unsupervised minimax accelerates real reward

Figure 8: Experiment 2a: LEFT’s (top) and RIGHT’s first 100 (of 576) probability distributions after simulation 1. Grey scales indicate probability magnitudes (white = close to 0, black = close to 1). The probability mass of many (but not all) columns is concentrated in a single value. Both brains are almost identical due to SSAandCopy PLAs. Their stacks are quite different though.


More on artificial curiosity: [http://people.idsia.ch/~juergen/interest.html](http://people.idsia.ch/~juergen/interest.html)

[http://people.idsia.ch/~juergen/creativity.html](http://people.idsia.ch/~juergen/creativity.html)
Maximize Future Fun(\(\text{Data X, O(t)}\)) -
\[ \frac{\partial \text{CompResources}(X, O(t))}{\partial t} \]

My formal theory of fun & novelty & surprise & attention & creativity & curiosity & art & science & humor

http://www.idsia.ch/~juergen/creativity.html
PowerPlay not only solves but also continually invents problems at the borderline between what's known and unknown - training an increasingly general problem solver by continually searching for the simplest still unsolvable problem.
AAAII 2013 BEST STUDENT VIDEO AWARD
w. M Stollenga, K Frank, J Leitner, L Pape, A Foerster, J Koutnik
DRAWBACKS
OF CURIOSITY
neural networks-based
artificial intelligence

THE DAWN OF AI