Modern approaches to AGI: what is the shared core problem?

Prof. Alexey Potapov
ITMO University
2018

Innovation & Research Symposium @ Paris

Artificial General Intelligence

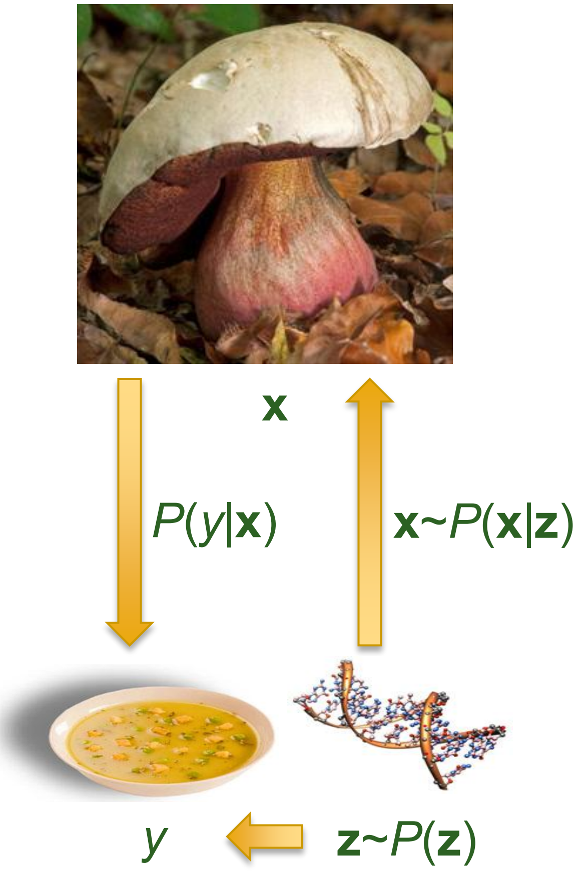
(General) intelligence is an agent's ability to efficiently achieve goals in a wide range of environments with insufficient knowledge and resources

Pei Wang, Ben Goertzel, Marcus Hutter, etc.

Subfields and Approaches

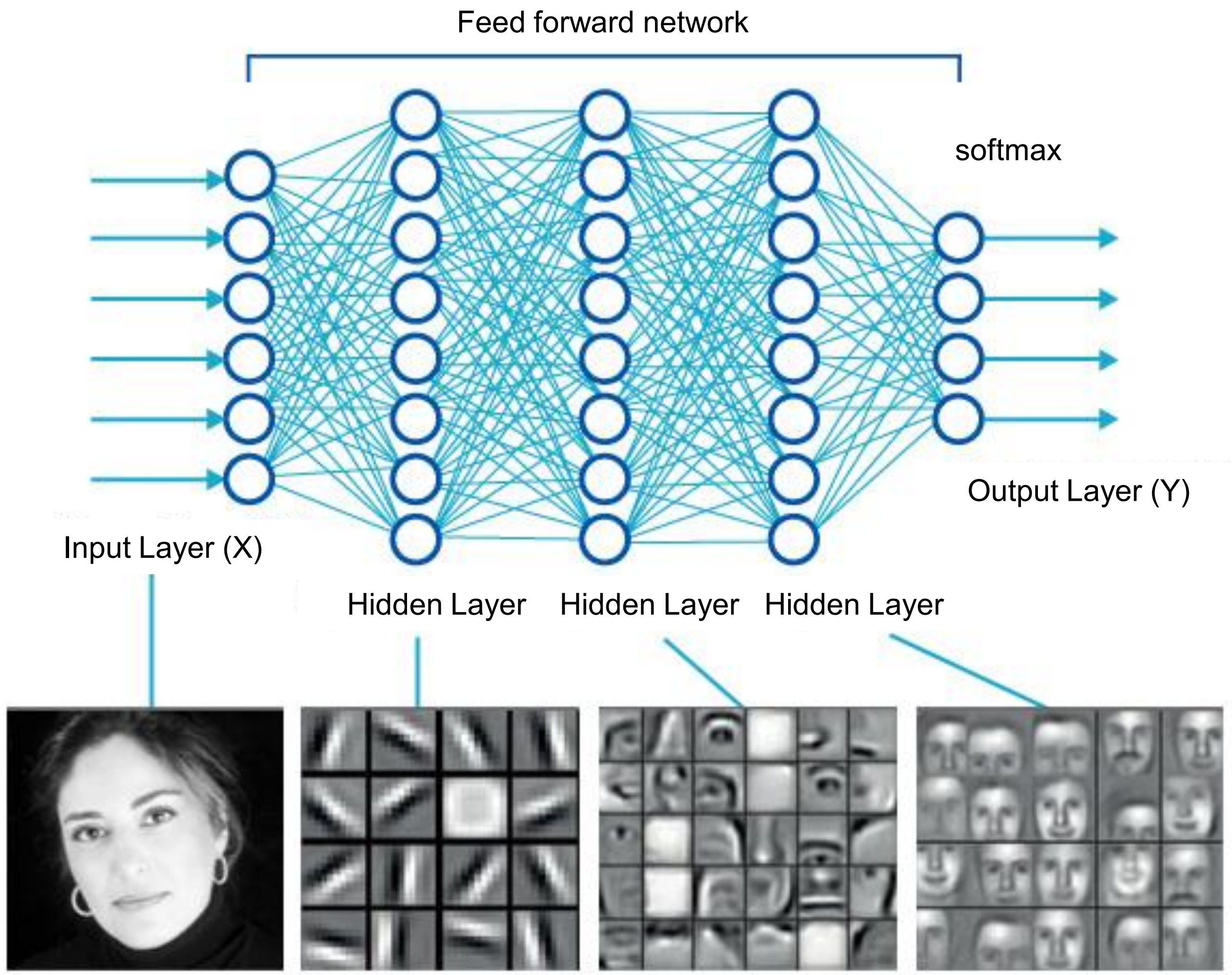
- Deep Learning
- Cognitive Architectures
- Probabilistic Models
- Universal Algorithmic Intelligence
- Reinforcement Learning

Discriminative and Generative Models



$\mathbf{x}$
$P(y|\mathbf{x})$
$\mathbf{x} \sim P(\mathbf{x}|\mathbf{z})$
y
$\mathbf{z} \sim P(\mathbf{z})$

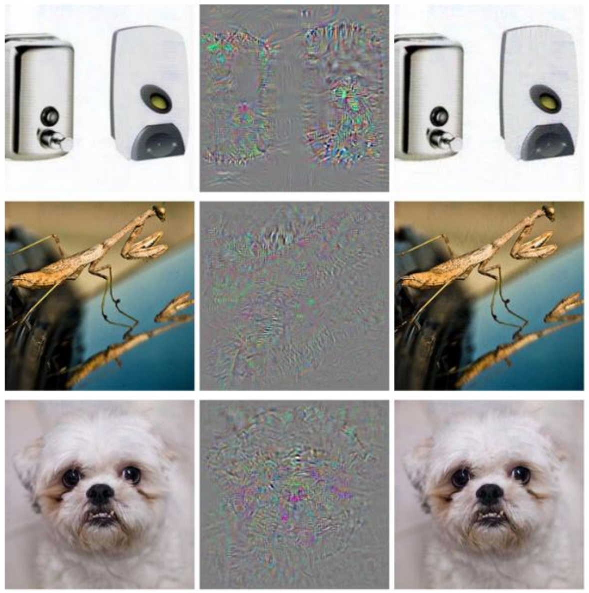
FFNNs as Discriminative Models

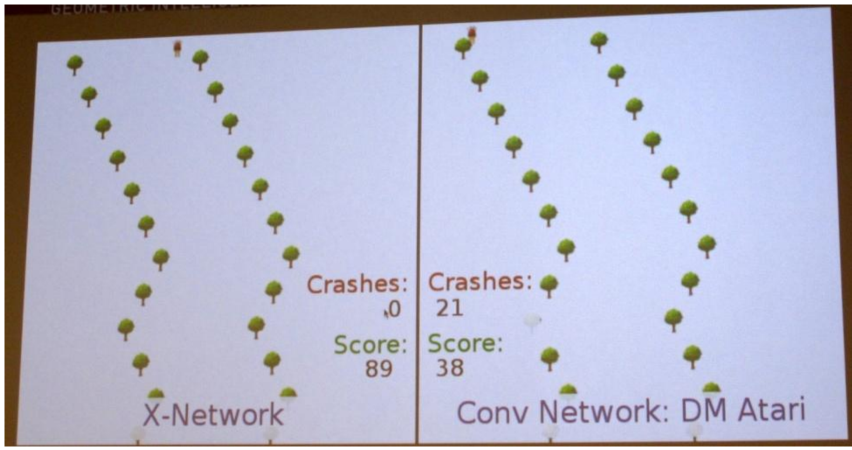


Direct approximation of $P(y|\mathbf{x})$

Critique of Deep Learning

- Weak generalization
  - Require large training sets; no one-shot learning
  - Cannot learn invariants
  - Vulnerable to Adversarial examples
  - Difficulties with transfer and unsupervised learning
- From AGI perspective
  - Encode higher-order statistics, but not causal, logical, spatio-temporal relations
  - Bad in high-level reasoning and planning, etc.

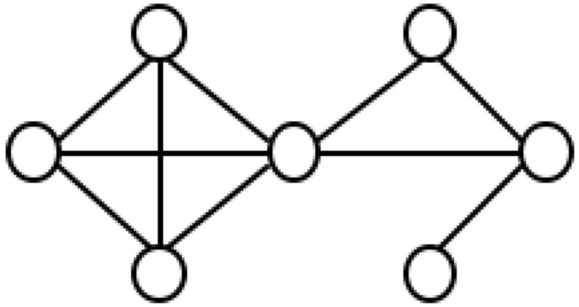




Images from: Szegedy, C. et al. Intriguing properties of neural networks. arXiv 1312.6199 (2013).
Gary Marcus. Keynote @ AGI-16

Generative Models: Graphical Models

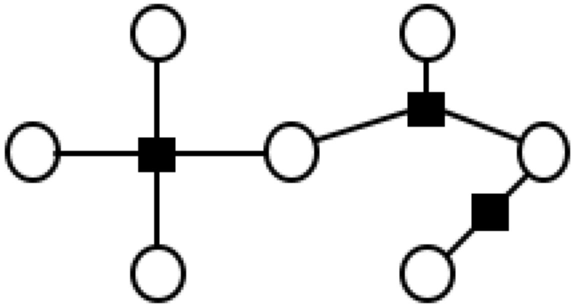
- Curse of dimensionality
- Chain-rule decomposition
- Conditional independence



$$P(x_{1:N}) = P(x_1)P(x_2|x_1)\ldots P(x_N|x_{1:N-1}) = \prod_{s=1}^{N} p(x_s|\mathbf{x}_{\pi(s)})$$

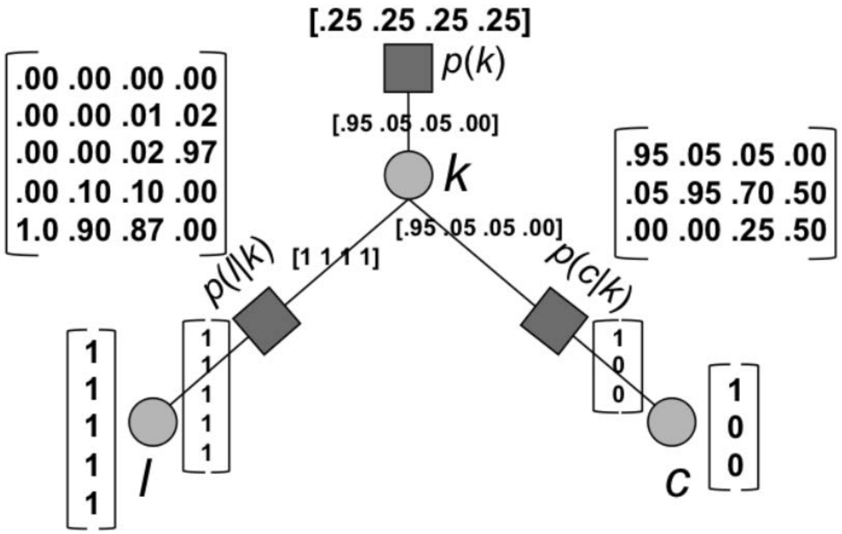
- Bayesian networks
- Markov networks
- Factor graphs
- Plate models, etc.

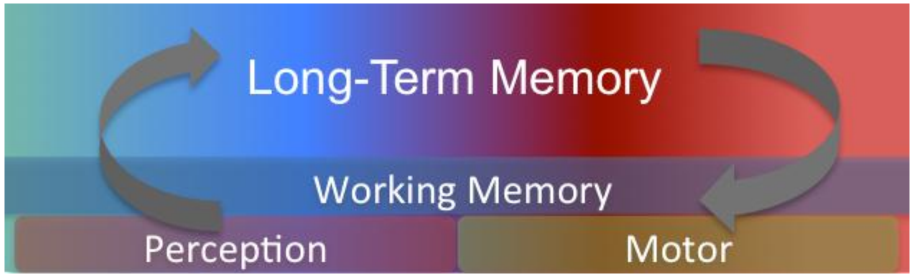
$$P(x_1,\ldots,x_N) = \frac{1}{Z}\prod_{C\in\mathrm{C}}\psi_C(\mathbf{x}_C)$$



Graphical CA Hypothesis: Sigma

- Graphical Architecture Hypothesis
    - Four desiderata:
        - grand unification
        - generic cognition
        - functional elegance
        - sufficient efficiency
- Deconstruction of all cognitive functions with the use of factor-graphs as a general cognitive firmware
- Reasoning: Message Passing
- Learning: Gradient Descent





Rosenbloom, P., Demski, A. & Ustun, V. (2017). The Sigma Cognitive Architecture and System: Towards Functionally Elegant Grand Unification. Journal of Artificial General Intelligence, 7(1), pp. 1-103.

Generative Models

- Produce x from z
- In opposite direction?

$$P(\mathbf{z}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}{P(\mathbf{x})} = \frac{P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}{\sum_{\mathbf{z}} P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}$$

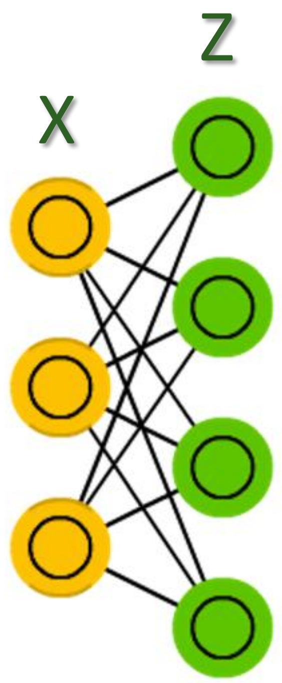
- Difficulties with marginalization

- RBM: simplest non-trivial probabilistic graphical model

$$P(\mathbf{x}) = \sum_{\mathbf{z}} P(\mathbf{x},\mathbf{z}) = \frac{1}{Z}\sum_{\mathbf{z}} e^{\mathbf{bx}+\mathbf{cz}+\mathbf{x}^T W\mathbf{z}}$$

$$P(z_i = 1|\mathbf{x}) = \sigma\left(\sum_j w_{ij}x_j + c_i\right)$$

- Difficulties with training



Probabilistic Models: Discriminative and Generative

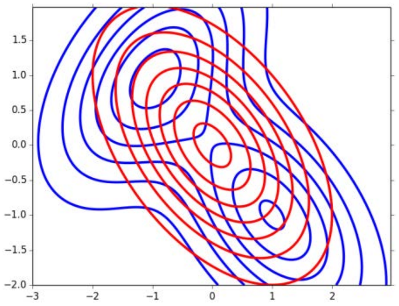
	Discriminative	Generative
Mapping	A mapping from the task to its solution space $P(y \mid \mathbf{x})$	A mapping from the solution space to data $\mathbf{z} \sim P(\mathbf{z})$, $\mathbf{x} \sim P(\mathbf{x} \mid \mathbf{z})$
Pros	• Efficient • Less assumptions of data distribution	• Flexible • Un/semi-supervised learning
Cons	• One-way inference only • Only supervised learning	• Additional assumptions of data distribution • Computationally inefficient inference

Probabilistic Models: Variational Bayes

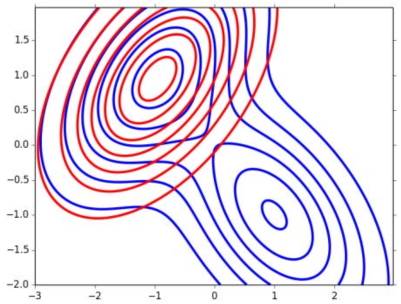
- Posteriors are difficult to compute

$$P(\mathbf{z}_i \mid \mathbf{x}_i,\theta) = \frac{P(\mathbf{x}_i,\mathbf{z}_i \mid \theta)}{P(\mathbf{x}_i \mid \theta)} = \frac{P(\mathbf{x}_i,\mathbf{z}_i \mid \theta)}{\int P(\mathbf{x}_i,\mathbf{z} \mid \theta)d\mathbf{z}}$$

$$\begin{tikzcd} P(z) & Z \arrow[d, bend right, "P_\theta(x|z)"'] \\ & X \arrow[u, bend right, "Q_\varphi(z|x)"'] \end{tikzcd}$$



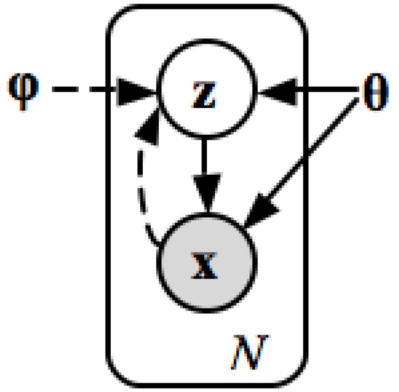
- Let’s approximate them with some easily computable $Q(\mathbf{z}|\mathbf{x}_i,\varphi)$
- Criterion: Kullback-Leibler divergence / variational lower bound for the marginal likelihood (evidence)



$$\log P(\mathbf{x}_i|\theta) = D_{KL}\big(Q(\mathbf{z}|\mathbf{x}_i,\varphi)\|P(\mathbf{z}|\mathbf{x}_i,\theta)\big) + L(\theta,\varphi|\mathbf{x}_i)$$

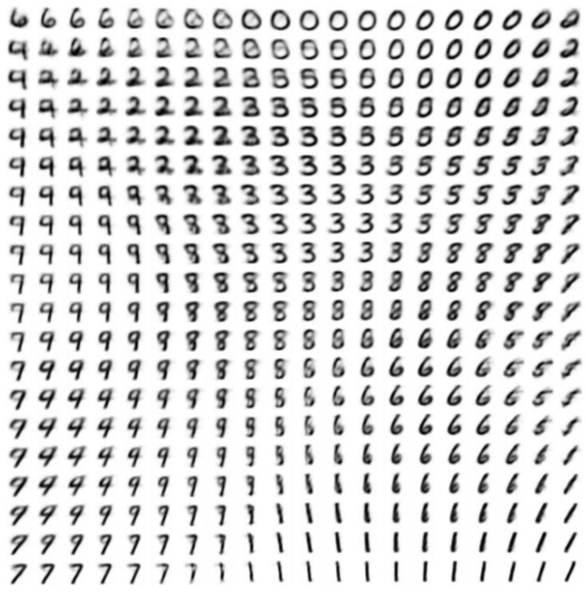
$$D_{KL}(Q\|P) = -\int Q(\mathbf{z}|\mathbf{x}_i,\varphi)\left[\log\frac{P(\mathbf{z}|\mathbf{x}_i,\theta)}{Q(\mathbf{z}|\mathbf{x}_i,\varphi)}\right]d\mathbf{z} \qquad L(\theta,\varphi|\mathbf{x}_i) = \int Q(\mathbf{z}|\mathbf{x}_i,\varphi)\left[\log\frac{P(\mathbf{x}_i,\mathbf{z}|\theta)}{Q(\mathbf{z}|\mathbf{x}_i,\varphi)}\right]d\mathbf{z}$$

Probabilistic Models in Deep Learning: Variational Autoencoders



$$\log P(\mathbf{x}_i|\theta) = D_{KL}\Big(Q(\mathbf{z}|\mathbf{x}_i,\varphi)\Big\|P(\mathbf{z}|\mathbf{x}_i,\theta)\Big) + L(\theta,\varphi|\mathbf{x}_i)$$

- Let’s learn the generative model and its variational approximation simultaneously
- Let’s represent P and Q as DNNs
- Add some heuristics



https://arxiv.org/abs/1312.6114

Probabilistic Models in Deep Learning: Generative Adversarial Networks

$$\min_{\theta} \max_{\varphi} V(\theta,\varphi) = E_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x}|\varphi)] + E_{\mathbf{z}\sim P_{\text{model}}(\mathbf{z})}[\log(1 - D(G(\mathbf{x}|\theta)|\varphi))]$$

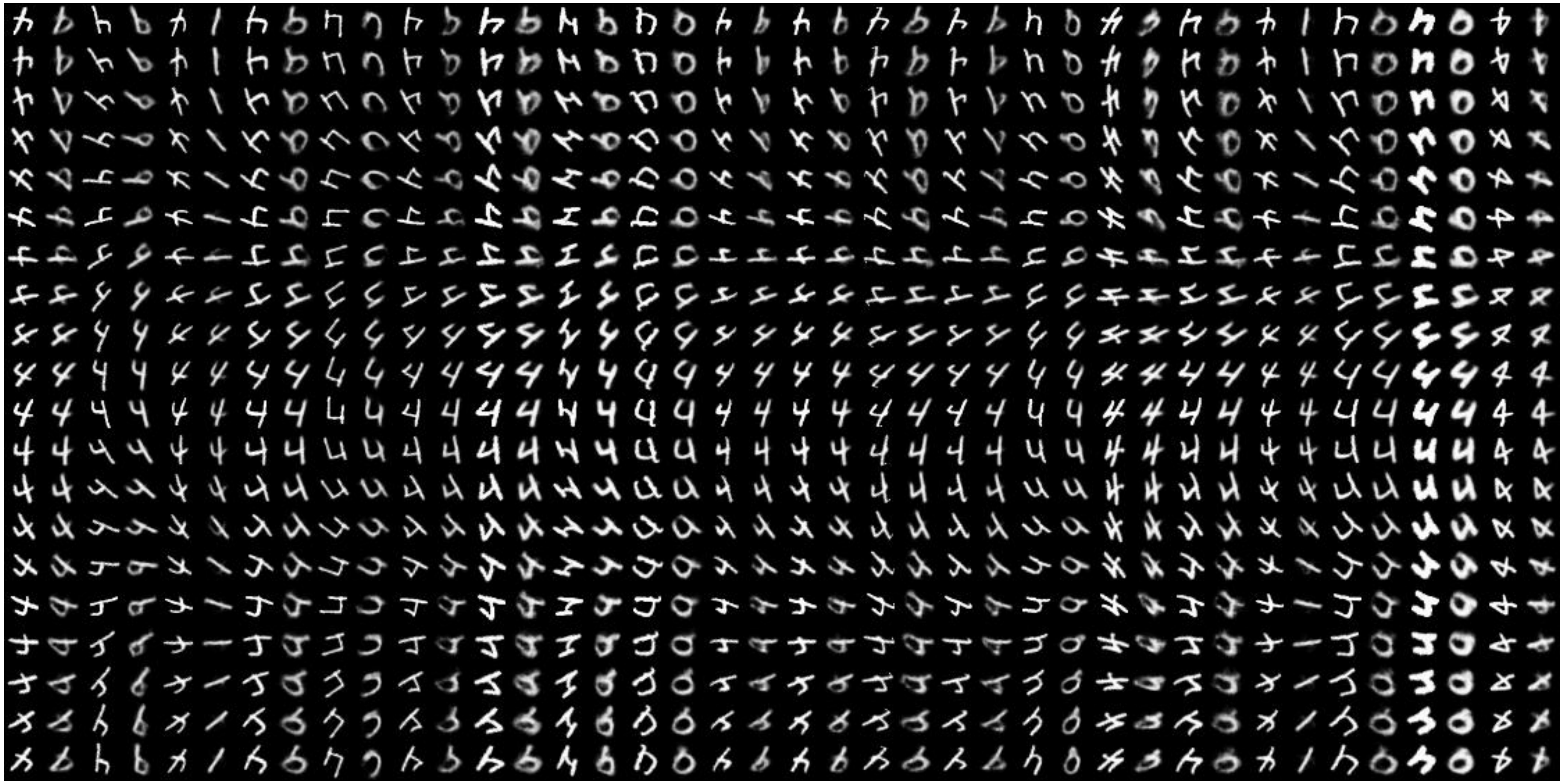
- Does not construct variational approximation of a posterior distribution
- Directly estimates the quality of a generative marginal distribution
- No sampling, just gradient descent

- DCGAN, WGAN, LSGAN, … BiGAN, InfoGAN, Bayesian GAN

https://arxiv.org/abs/1406.2661

Is it enough?

- Example: Adversarial Autoencoders
- Training sets: all digits except 4 were rotated by all angle



- No generalization of rotation

Universal Algorithmic Intelligence: Solomonoff Induction

- Universal priors $P(\mu) = 2^{-l(\mu)}$

μ – programs (binary strings) for Universal Turing Machine

- Marginal probability $M_U(x) = \sum_{\mu:U(\mu)=x*} 2^{-l(\mu)}$

- Prediction $M_U(y|x) = M_U(xy)/M_U(x)$

Convergence!

$$E_Q\left[\sum_{i=1}^{n}\left(Q(x_{i+1}=1\mid x_{1:i}) - P_U(x_{i+1}=1\mid x_{1:i})\right)^2\right] \le \frac{\ln 2}{2} K_U(Q)$$

- Optimal prediction for any (computable) data source
- No "no free lunch theorem"!

Universality of the algorithmic space

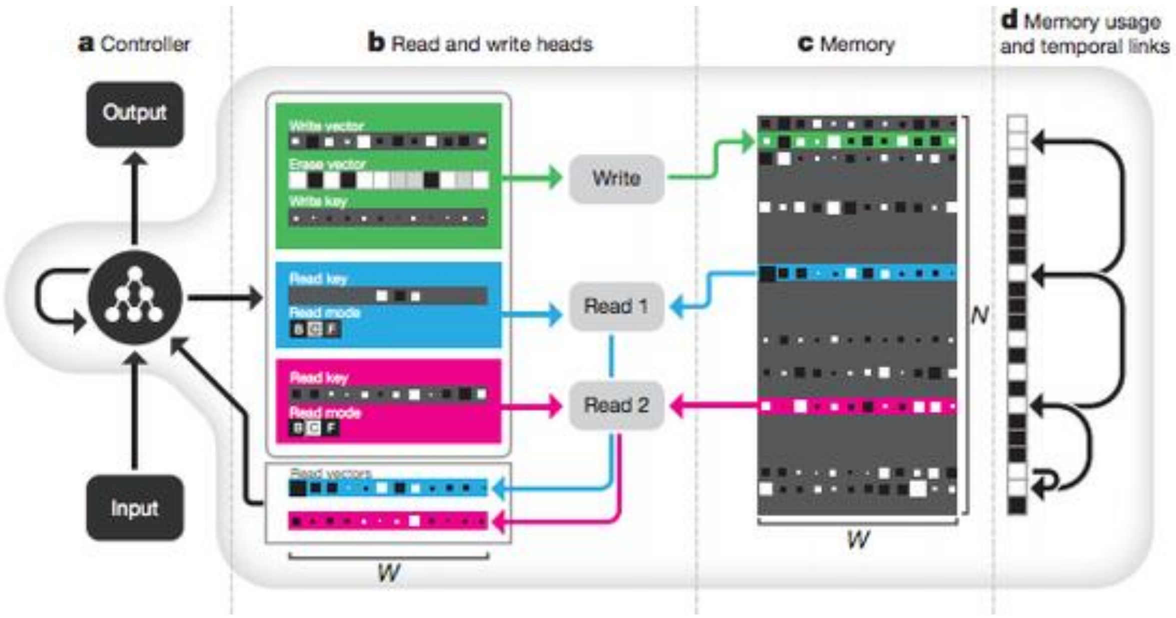
3.1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 5923078164 0628620899 8628034825 3421170679 8214808651 3282306647 0938446095 5058223172 5359408128 4811174502 8410270193 8521105559 6446229489 5493038196 4428810975 6659334461 2847564823 3786783165 2712019091 4564856692 3460348610 4543266482 1339360726 0249141273 7245870066 0631558817 4881520920 9628292540 9171536436 7892590360 0113305305 4882046652 1384146951 9415116094 3305727036 5759591953 0921861173 8193261179 3105118548 0744623799 6274956735 1885752724 8912279381 8301194912 9833673362 4406566430 8602139494 6395224737 1907021798 6094370277 0539217176 2931767523 8467481846 7669405132 0005681271 4526356082 7785771342 7577896091 7363717872 1468440901 2249534301 4654958537 1050792279 6892589235 4201995611 2129021960 8640344181 5981362977 4771309960 5187072113 4999999 ………

```
int a=10000,b,c=8400,d,e,f[8401],g;
main() {for(;b-c;)f[b++]=a/5;
for(;d=0,g=c*2;c-=14, printf("%.4d",e+d/a),e=d%a)
for(b=c;d+=f[b]*a,f[b]=d%--g,d/=g--,--b;d*=b);}
```

By D.T. Winter

Is DL that bad?

- RNN instead of finite state machine
- External memory with soft addressing
- End-to-end differentiable algorithms

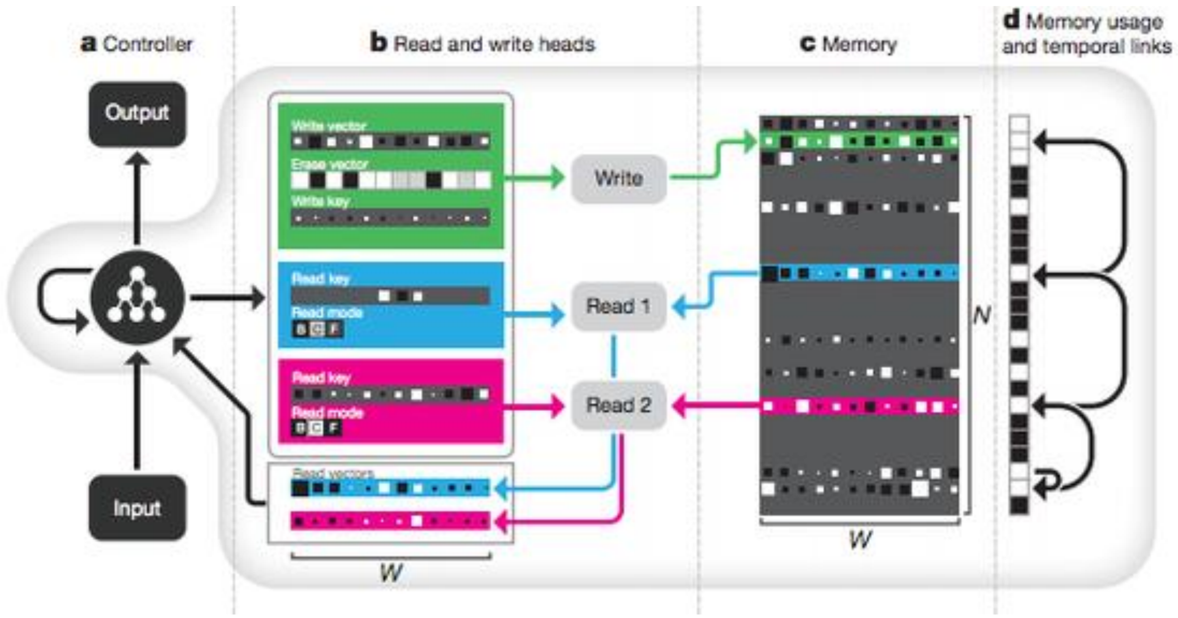


- Neural differentiable computer, Neural GPU, Neural programmer-interpreter, Differentiable Forth interpreter, etc.
- Memory augmented NNs, including deep RL

Image from: https://deepmind.com/blog/differentiable-neural-computers/

Is DL that bad?

- RNN instead of finite state machine
- External memory with soft addressing
- End-to-end differentiable algorithms

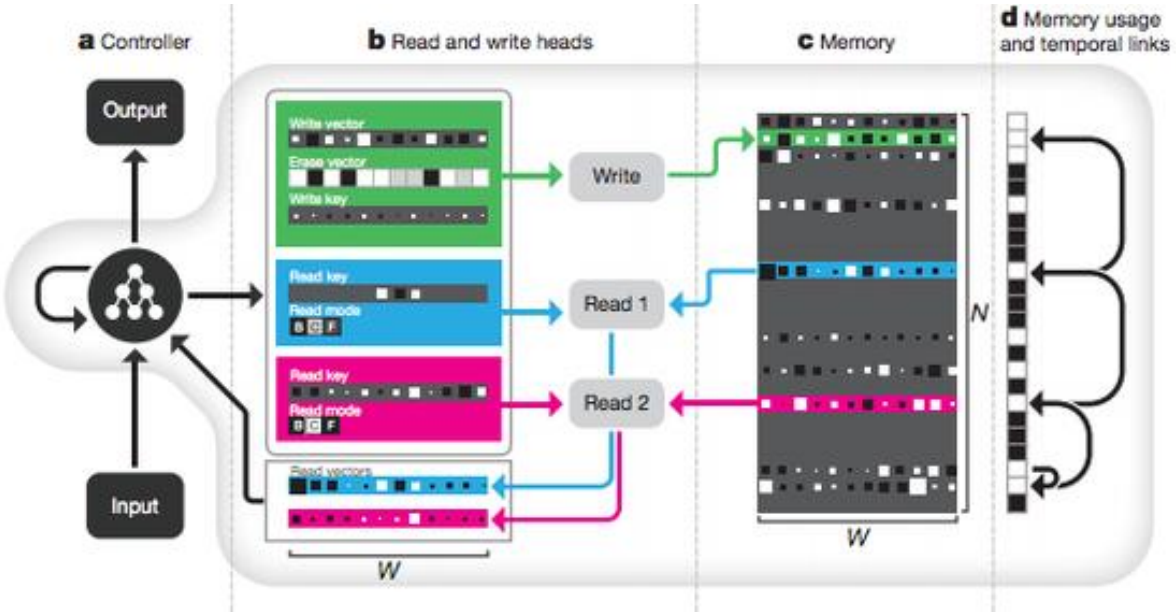


- Neural differentiable computer, Neural GPU, Neural programmer-interpreter, Differentiable Forth interpreter, etc.
- Memory augmented NNs, including deep RL
- Apparent trend towards universal induction within DL

Image from: https://deepmind.com/blog/differentiable-neural-computers/

Is DL that bad?

- RNN instead of finite state machine
- External memory with soft addressing
- End-to-end differentiable algorithms

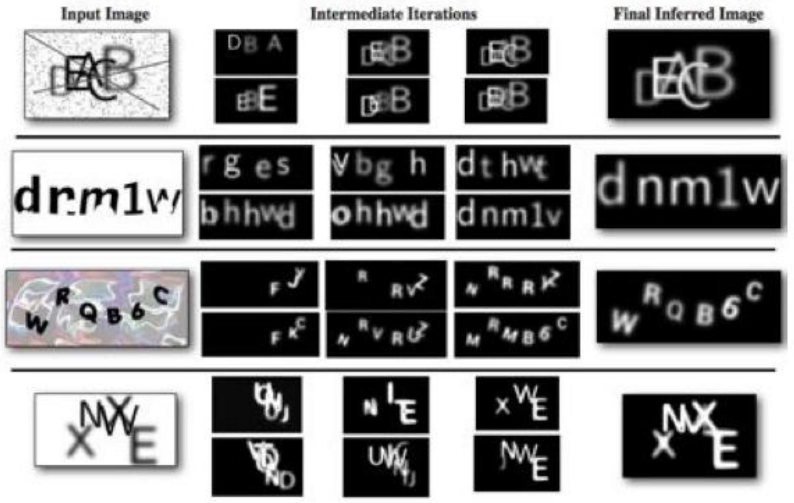


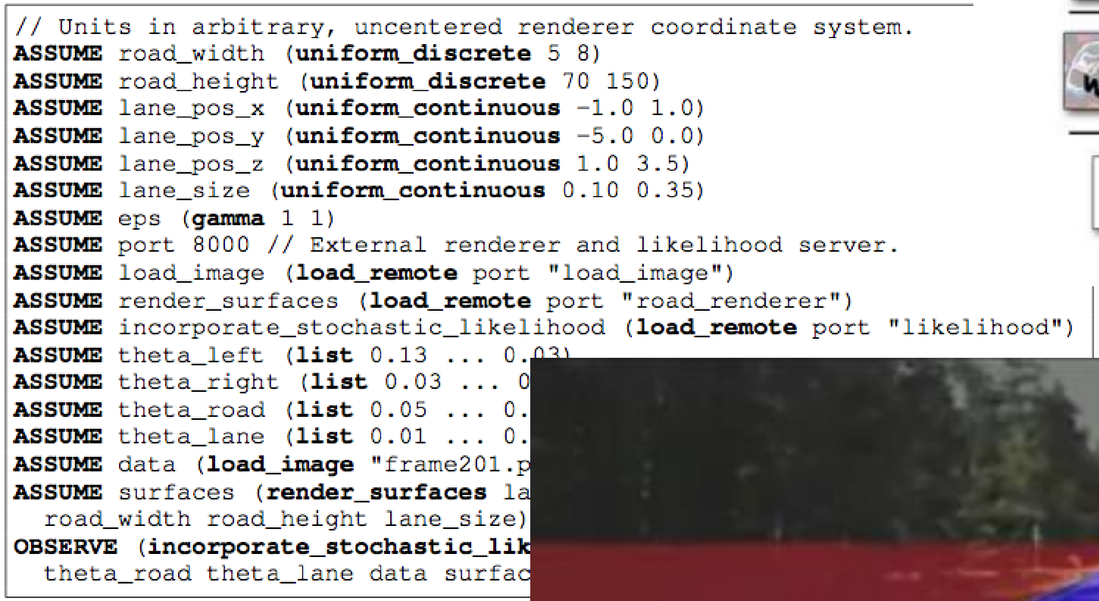
- Neural differentiable computer, Neural GPU, Neural programmer-interpreter, Differentiable Forth interpreter, etc.
- Memory augmented NNs, including deep RL
- Apparent trend towards universal induction within DL
- But gradient descent is not enough to learn algorithms

Image from: https://deepmind.com/blog/differentiable-neural-computers/

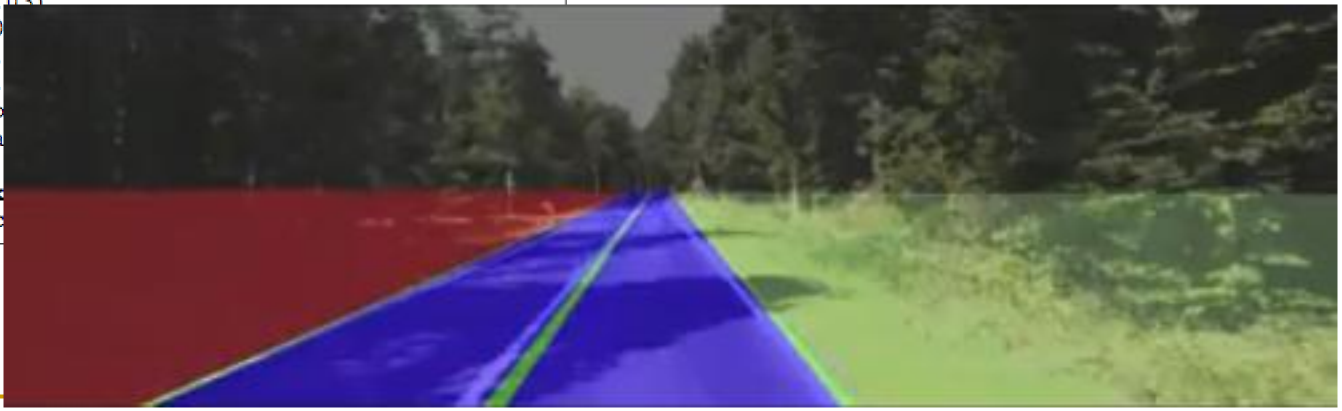
What’s about probabilistic models?

- Graphical models in computer vision, knowledge representations, etc.
- Probabilistic programming
- Probabilistic models of cognition





```
// Units in arbitrary, uncentered renderer coordinate system.
ASSUME road_width (uniform_discrete 5 8)
ASSUME road_height (uniform_discrete 70 150)
ASSUME lane_pos_x (uniform_continuous -1.0 1.0)
ASSUME lane_pos_y (uniform_continuous -5.0 0.0)
ASSUME lane_pos_z (uniform_continuous 1.0 3.5)
ASSUME lane_size (uniform_continuous 0.10 0.35)
ASSUME eps (gamma 1 1)
ASSUME port 8000 // External renderer and likelihood server.
ASSUME load_image (load_remote port "load_image")
ASSUME render_surfaces (load_remote port "road_renderer")
ASSUME incorporate_stochastic_likelihood (load_remote port "likelihood")
ASSUME theta_left (list 0.13 ... 0.03)
ASSUME theta_right (list 0.03 ... 0
ASSUME theta_road (list 0.05 ... 0.
ASSUME theta_lane (list 0.01 ... 0.
ASSUME data (load_image "frame201.p
ASSUME surfaces (render_surfaces la
  road_width road_height lane_size)
OBSERVE (incorporate_stochastic_lik
  theta_road theta_lane data surfac
```



Images from: Mansinghka, V., Kulkarni, T., Perov, Y., Tenenbaum, J.: Approximate Bayesian Image Interpretation using Generative Probabilistic Graphics Programs. Advances in NIPS, arXiv:1307.0060 [cs.AI] (2013).

Probabilistic Programming: Knowledge Representation

```
var generate = function() {
    var worksInHospital = flip(0.01)
    var smokes = flip(0.2)
    var lungCancer = flip(0.01) || (smokes && flip(0.02))
    var TB = flip(0.005) || (worksInHospital && flip(0.01))
    var cold = flip(0.2) || (worksInHospital && flip(0.25))
    var stomachFlu = flip(0.1)
    var other = flip(0.1)
    var cough = ((cold && flip(0.5)) || (lungCancer && flip(0.3)) ||
                (TB && flip(0.7)) || (other && flip(0.01)))
    var fever = ((cold && flip(0.3)) || (stomachFlu && flip(0.5)) ||
                (TB && flip(0.2)) || (other && flip(0.01)))
    var chestPain = ((lungCancer && flip(0.4)) ||
                    (TB && flip(0.5)) || (other && flip(0.01)))
    var shortnessOfBreath = ((lungCancer && flip(0.4)) ||
                            (TB && flip(0.5)) || (other && flip(0.01)))
    condition(cough && chestPain && shortnessOfBreath)
    return {lungCancer: lungCancer, TB: TB}
}
```

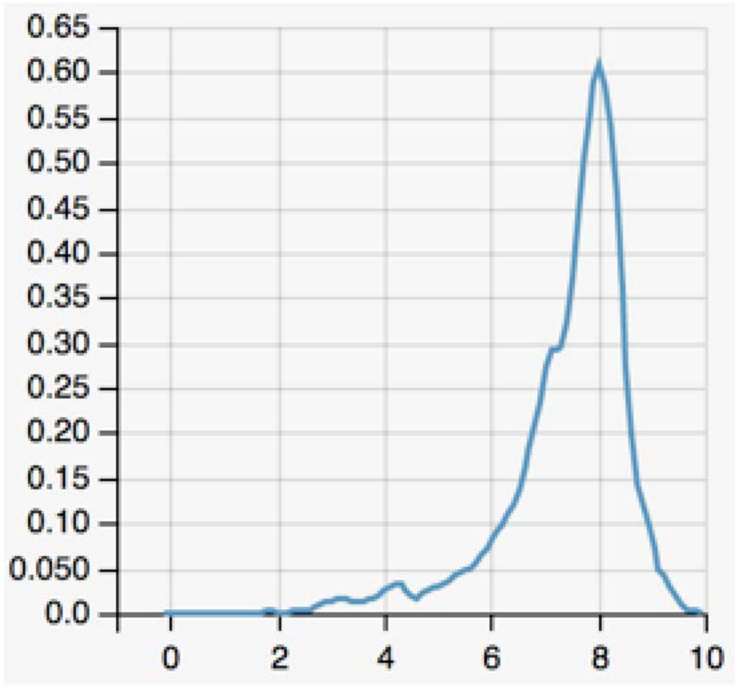
Probabilistic Programming: Reasoning and Problem Solving

```
var task = [10, 8, -8, -12, 15, 3]
var target = 1
var generate = function() {
   var subset = repeat(task.length, flip)
   var sum = reduce(function(x, acc)
                 { return acc + (x[1] ? x[0] : 0) },
                 0, zip(task, subset))
   condition(sum == target)
   return subset
}
Infer({method:          "rejection",          samples:          100,
      model: generate})
```

```
→ {"probs":[0.48,0.52],
  "support":[[true,true,true,true,false,true],
             [true,false,false,true,false,true]]}
```

Probabilistic Programming: Learning

- Arbitrary models including but not limited to graphical models
- Model selection, structure learning
- Bayesian Occam Razor for free



```
var xs = [0   , 1   , 2   , 3   ]
var ys = [0.01, 0.99, 4.02, 5.97]
var linreg = function() {
   var a = gaussian(0, 1)
   var b = gaussian(0, 1)
   var sigma = gamma(1, 1)
   var f = function(x) { return a * x + b }
   var check = function(x, y)
                   { observe(Gaussian({mu: f(x), sigma: sigma}), y) }
   map2(check, xs, ys)
   return f(4)
}
Infer({method: 'MCMC', samples: 10000, model: linreg})
```

Probabilistic Programming: Neural Bayes

```
import edward as ed
from edward.models import Normal
def neural_network(X):
  h = tf.tanh(tf.matmul(X, W_0) + b_0)
  h = tf.tanh(tf.matmul(h, W_1) + b_1)
  h = tf.matmul(h, W_2) + b_2
  return tf.reshape(h, [-1])
W_0 = Normal(loc=tf.zeros([D, 10]), scale=tf.ones([D, 10]))
...
b_2 = Normal(loc=tf.zeros(1), scale=tf.ones(1))
X = tf.placeholder(tf.float32, [N, D])
y = Normal(loc=neural_network(X), scale=0.1 * tf.ones(N))
qW_0 = Normal(loc=tf.Variable(tf.random_normal([D, 10])),
  scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, 10]))))
...
inference = ed.KLqp({W_0: qW_0, b_0: qb_0, W_1: qW_1, b_1: qb_1,
                    W_2: qW_2, b_2: qb_2},
                   data={X: X_train, y: y_train})
```

Learning Probabilistic Programs

```
(lambda (stack-id)
  (* 2.0 (* (*
      (* -1.0 (safe-uc 0.0 2.0))
      (safe-uc (safe-uc 4.0
        (+ (safe-log 2.0) -1.0))
        (* (safe-div 2.0
          -55.61617747203855)
          (if (< (safe-uc
            (safe-uc
            27.396810474207317
             (safe-uc -1.0 2.0)) 2.0) 2.0)
            4.0 -1.0)))) -1.0)))
```

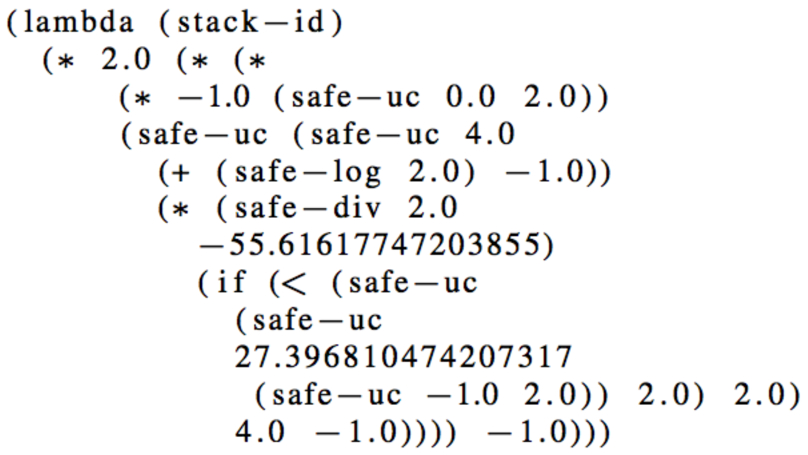
```
lambda (par stack-id) (* (begin (define sym0 0.0)
   (exp (safe-uc -1.0 (safe-sqrt (safe-uc
   (safe-div (safe-uc 0.0 (safe-uc 0.0 3.14159))
     par) (+ 1.0 (safe-uc (begin (define sym2
     (lambda (var1 var2 stack-id) (dec var2)))
     (sym2 (safe-uc -2.0 (* (safe-uc 0.0 (begin
     (define sym4 (safe-uc sym0 (* (+ (begin
     (define sym5 (lambda (var1 var2 stack-id)
     (safe-div (+ (safe-log (dec 0.0)) -1.0) var1)))
     (sym5 (exp par) 1.0 0)) 1.0) 1.0))) (if (< (safe-uc
     par sym4) 1.0) sym0 (safe-uc 0.0 -1.0)))) sym0))
     (safe-div sym0 (exp 1.0)) 0)) 0.0))))))) par))
```

- Higher-order PPLs allow for learning probabilistic programs from data by means of probabilistic programs (while learning of graphical models cannot be expressed in terms of graphical models)

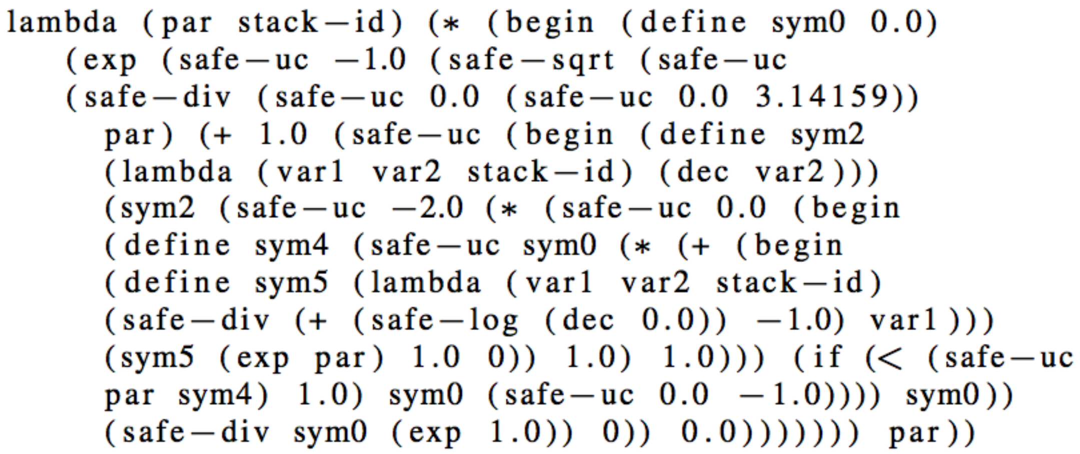
- Probabilistic Programming implements a form of universal induction

https://arxiv.org/pdf/1407.2646v1.pdf

Learning Probabilistic Programs



```
(lambda (stack-id)
  (* 2.0 (* (*
      (* -1.0 (safe-uc 0.0 2.0))
      (safe-uc (safe-uc 4.0
        (+ (safe-log 2.0) -1.0))
        (* (safe-div 2.0
          -55.61617747203855)
          (if (< (safe-uc
            (safe-uc
            27.396810474207317
             (safe-uc -1.0 2.0)) 2.0) 2.0)
            4.0 -1.0)))) -1.0)))
```



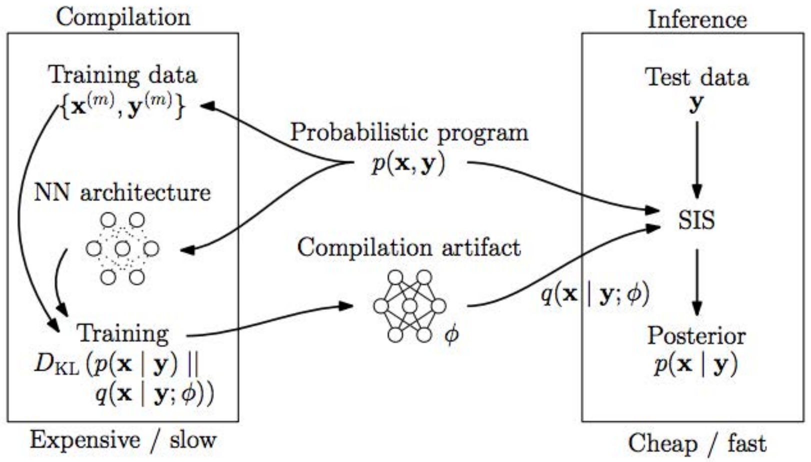
```
lambda (par stack-id) (* (begin (define sym0 0.0)
   (exp (safe-uc -1.0 (safe-sqrt (safe-uc
   (safe-div (safe-uc 0.0 (safe-uc 0.0 3.14159))
     par) (+ 1.0 (safe-uc (begin (define sym2
     (lambda (var1 var2 stack-id) (dec var2)))
     (sym2 (safe-uc -2.0 (* (safe-uc 0.0 (begin
     (define sym4 (safe-uc sym0 (* (+ (begin
     (define sym5 (lambda (var1 var2 stack-id)
     (safe-div (+ (safe-log (dec 0.0)) -1.0) var1)))
     (sym5 (exp par) 1.0 0)) 1.0) 1.0))) (if (< (safe-uc
     par sym4) 1.0) sym0 (safe-uc 0.0 -1.0)))) sym0))
     (safe-div sym0 (exp 1.0)) 0)) 0.0))))))) par))
```

- Higher-order PPLs allow for learning probabilistic programs from data by means of probabilistic programs

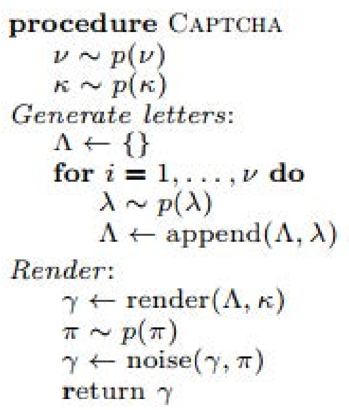
(while learning of graphical models cannot be expressed in terms of graphical models)

- Probabilistic Programming implements a form of universal induction
- MCMC inference is not scalable enough

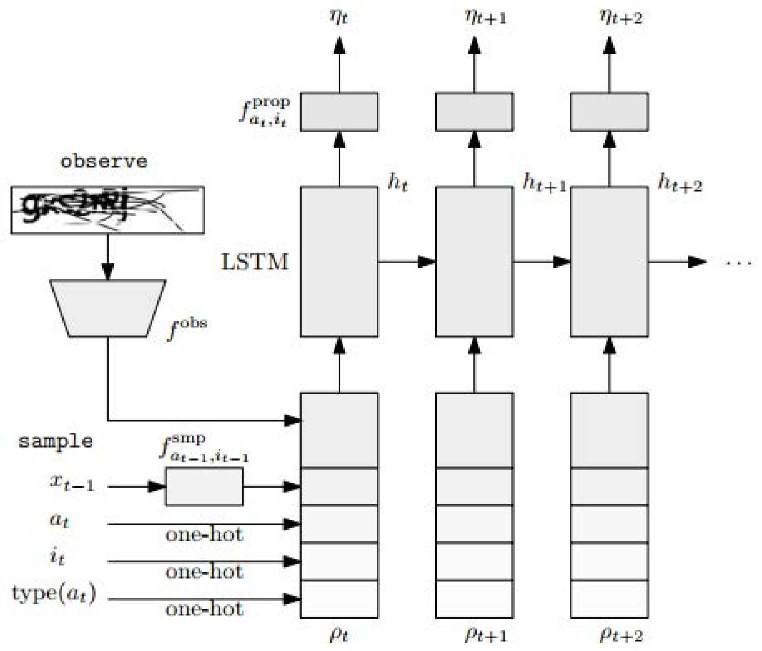
Deep Amortized Probabilistic Inference


Compilation
Training data $\{\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\}$
NN architecture
Training $D_{\mathrm{KL}}(p(\mathbf{x} \mid \mathbf{y}) \,||\, q(\mathbf{x} \mid \mathbf{y}; \phi))$
Expensive / slow
Probabilistic program $p(\mathbf{x}, \mathbf{y})$
Compilation artifact
ϕ
$q(\mathbf{x} \mid \mathbf{y}; \phi)$
Inference
Test data $\mathbf{y}$
SIS
Posterior $p(\mathbf{x} \mid \mathbf{y})$
Cheap / fast

- A generative model specified as a probabilistic program is ‘compiled’ into a discriminative model specified as a neural networks



```
procedure CAPTCHA
    $\nu \sim p(\nu)$
    $\kappa \sim p(\kappa)$
  Generate letters:
    $\Lambda \leftarrow \{\}$
    for $i = 1, \dots, \nu$ do
        $\lambda \sim p(\lambda)$
        $\Lambda \leftarrow \mathrm{append}(\Lambda, \lambda)$
  Render:
    $\gamma \leftarrow \mathrm{render}(\Lambda, \kappa)$
    $\pi \sim p(\pi)$
    $\gamma \leftarrow \mathrm{noise}(\gamma, \pi)$
    return $\gamma$
```



- Generative model is not learned

Gap between universal and pragmatic methods

- Universal methods
  - can work in arbitrary computable environment
  - incomputable or computationally infeasible
  - approximations are either inefficient or not universal
- Practical methods
  - work in non-toy environments
  - set of environments is highly restricted

=> Bridging this gap is necessary

More Efficient Universal Induction

- Choice of the reference machine
  - Only ‘exponentially small’ number of models can be inferred given limited computational resources
- Incremental learning
- Genetic Programming
- Incremental Self-Improvement
  - HSearch: instead of enumerating all programs, enumerate all proofs, so only programs are executed which are provably solve the problem with provably bounded computational time
  - Gödel machine: searcher+solver – searches for proof techniques which output proofs about useful self-rewrites including rewriting both solver and searcher itself. Completely self-referential
- Still impractical

Metacomputations in Universal Intelligence

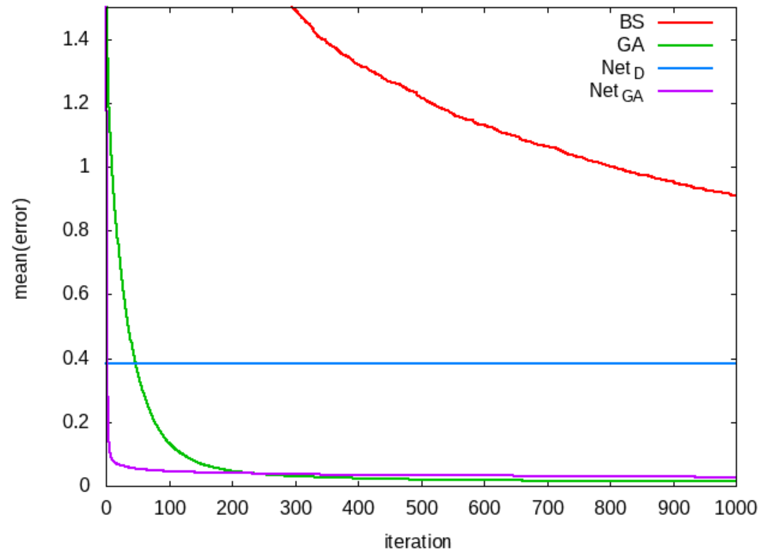
- Program specialization = construction of its efficient projection on one of its parameters
    - E.g. specialized interpreter w.r.t. program = compiled program (Futamura-Turchin projections)
    - Specialized specializer w.r.t. interpreter = compiler
- Specialized MCMC w.r.t. generative model = discriminative model
- Specialized universal induction w.r.t. Turing-incomplete reference machine = narrow machine learning method

Khudobakhshov V. Metacomputations and program-based knowledge representation // AGI-13
Potapov A. Rodionov S. Making universal induction efficient by specialization // AGI-14

Metacomputations in Universal Intelligence

- Still not enough ➔ Partial specialization

- Discriminative models are not always possible
- But we can do much better than blind or metaheuristic search
- E.g. genetic algorithms with data-guided trainable crossover

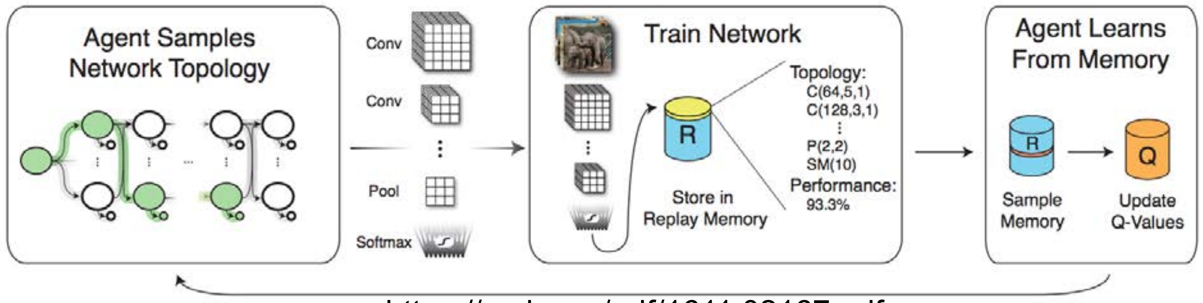


- Task: $f(\mathbf{z}|\mathbf{A},\mathbf{b}) = |\mathbf{A}\mathbf{z}-\mathbf{b}|^2$ $\mathbf{z}^* = \mathbf{A}^{-1}\mathbf{b}$
- Net_D – FFNN, which learns to produce z* from A and b
- NetGA – DNN, which produces next candidate z from A, b, z’, z’’
- GA – Traditional Genetic Algorithms
- BS – Brute force search

Potapov A. Rodionov S. Genetic Algorithms with DNN-Based Trainable Crossover as an Example of Partial Specialization of General Search // Proc. Artificial General Intelligence, AGI’17. P. 101-111.

Meta-learning with DNNs as an example

- A neural network that embeds its own meta-levels
- Learning to learn using gradient descent
- Learning to learn by gradient descent by gradient descent
- Learning to reinforcement learn
- RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning
- Meta-Learning with Memory-Augmented Neural Networks
- Designing Neural Network Architectures using Reinforcement Learning
- …



https://arxiv.org/pdf/1611.02167.pdf

One approach to AGI

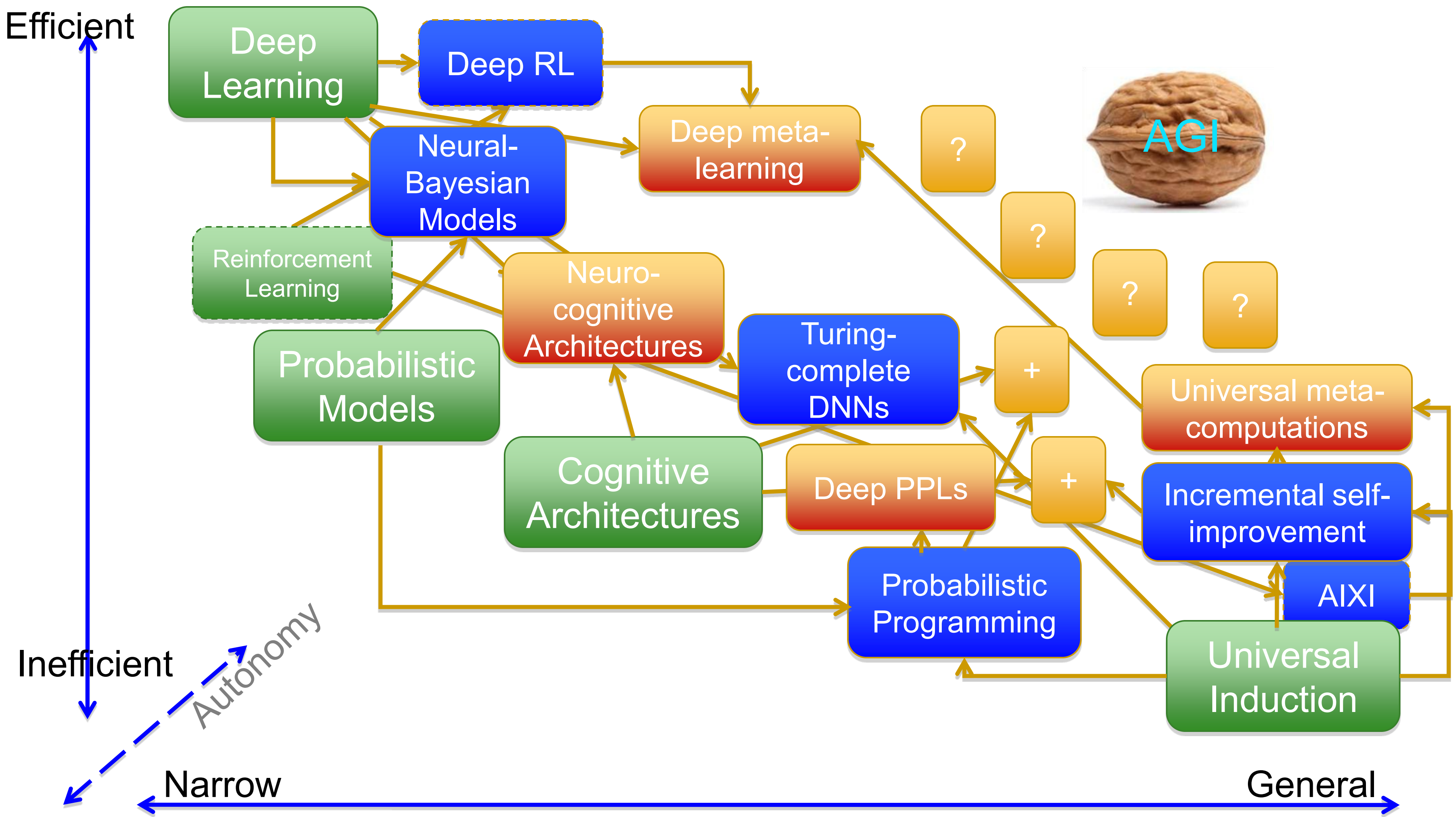
- Extended probabilistic programming language:
  - Probabilistic programs as generative models (basic)
  - Representation of discriminative models (available)
  - Self-referential interpreter with controllable inference

  → A cognitive architecture with knowledge management do deal with learnt domain-dependent specialized models
- OpenCog
  - Cognitive architecture with Turing-complete knowledge representation

  →OpenCoggy probabilistic programming with inference meta-learning extended with deep learning models

https://wiki.opencog.org/w/OpenCoggy_Probabilistic_Programming
https://blog.opencog.org/2017/10/14/inference-meta-learning-part-i/
https://github.com/opencog/semantic-vision/wiki/About-the-SynerGAN-architecture

Approaching AGI



Thank you for attention!

Contact: potapov@aideus.com