Learning Long-Term Dependencies with Gradient Descent is Difficult

- IEEE Trans. on Neural Networks 1994

  Yoshua Bengio, Patrice Simard, Paolo Frasconi

- Presented by:

  Matt Grimes , Ayse Naz Erkan

Recurrent Neural Networks - Definition

- A neural network with feedback. Provides
  - Memory
  - Context-sensitive processing of streams.

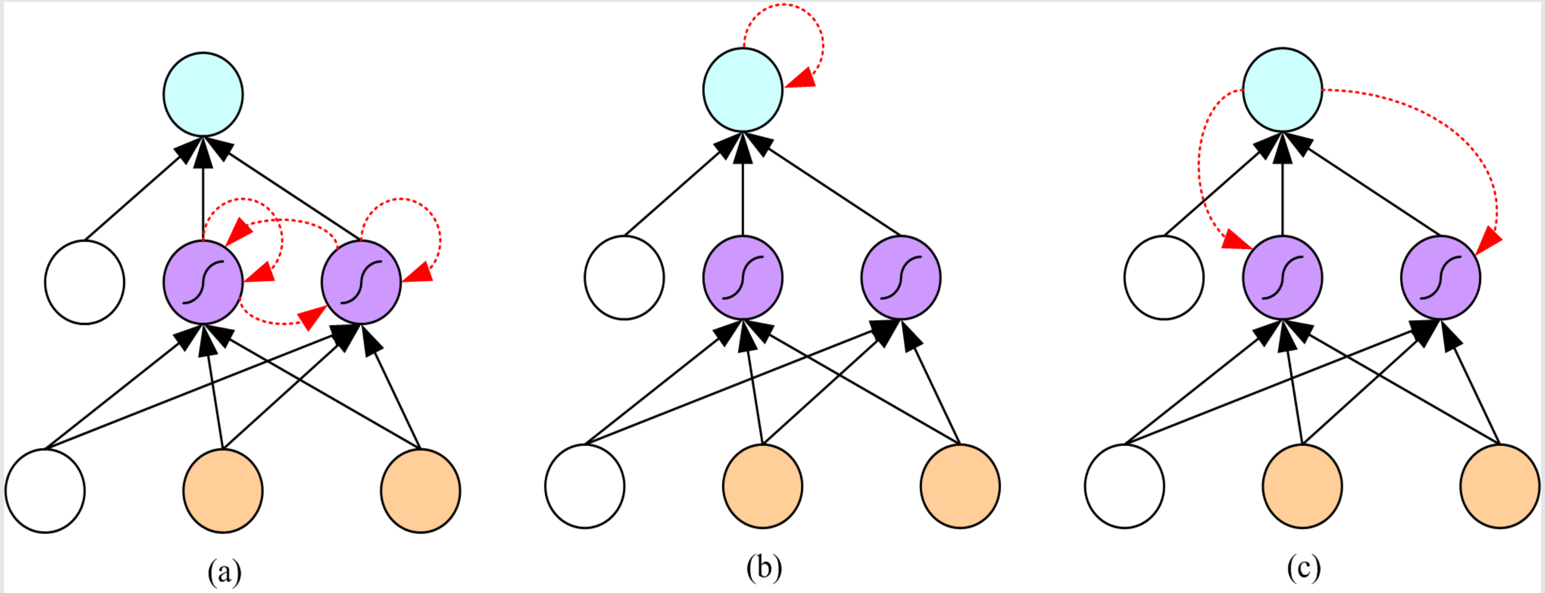


Illustration from *Introduction to machine learning*, E. Alpaydin 2005

Training algorithms for recurrent networks

Back-propagation through time (Rumelhart, Hinton,Williams,86)

Applies backprop to recurrent networks by unfolding them through N timesteps and treating them like static networks (stores the activations of units while going forward in time.)

Others, forward propagation algorithms

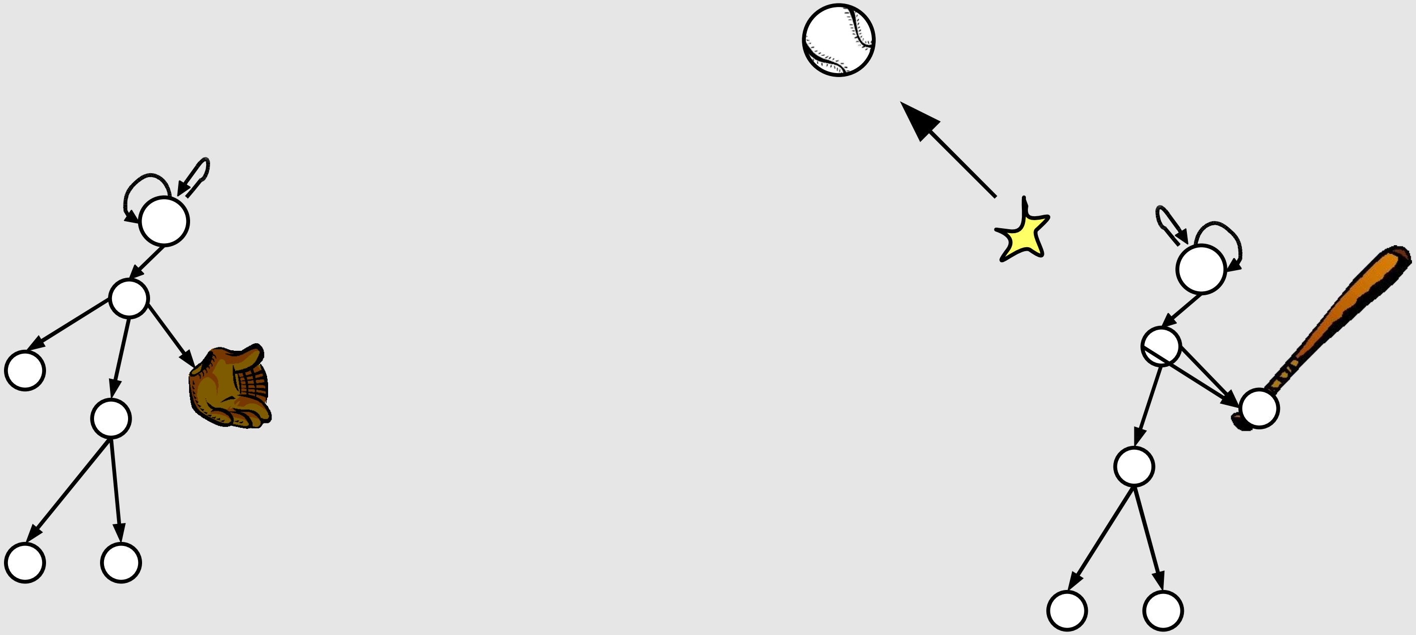
Computationally more expensive but local in time, can be applied for online learning.

A variant for constrained recurrent networks
(Gori, Bengio, Mori, 89) (Mozer, 89)

Local in time, computational requirement proportional to number of weights. However, has limited storage capabilities with general sequences.

From the dynamical systems perspective:

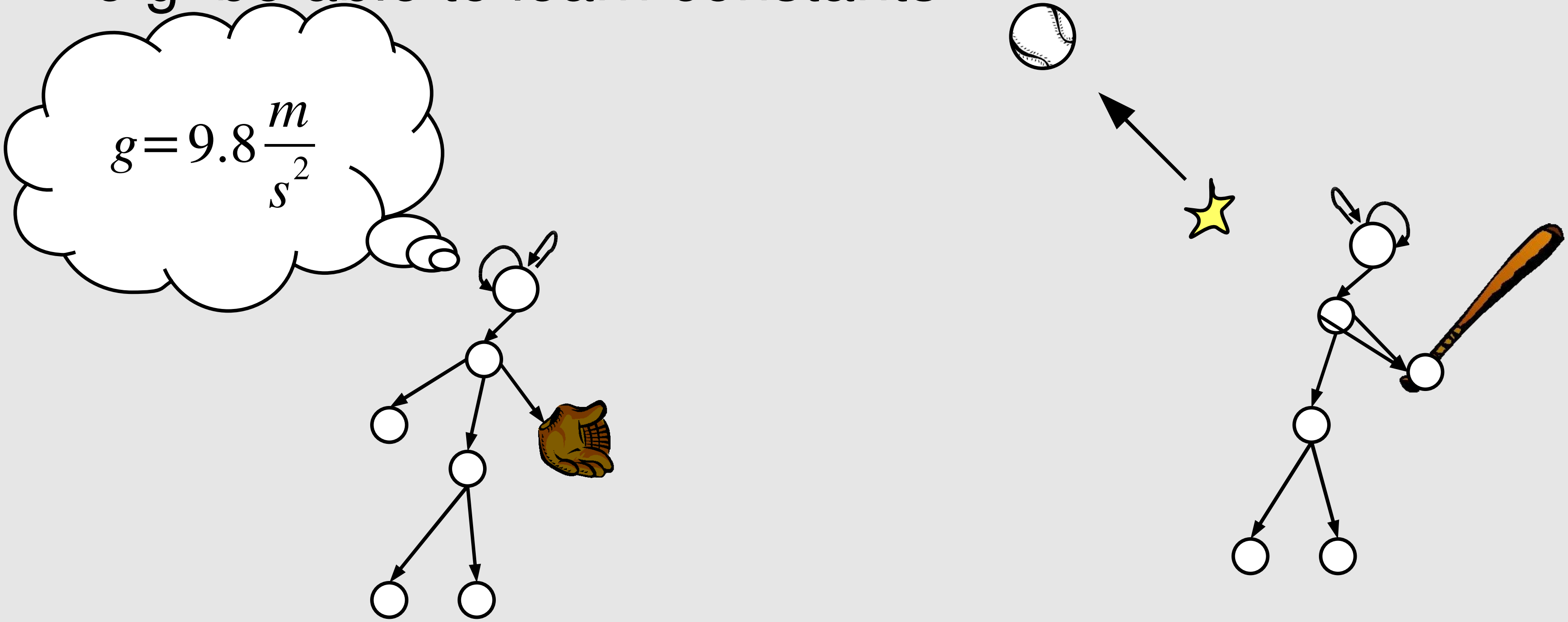
Three basic requirements ***to be able to learn relevant state information***



Feature wishlist

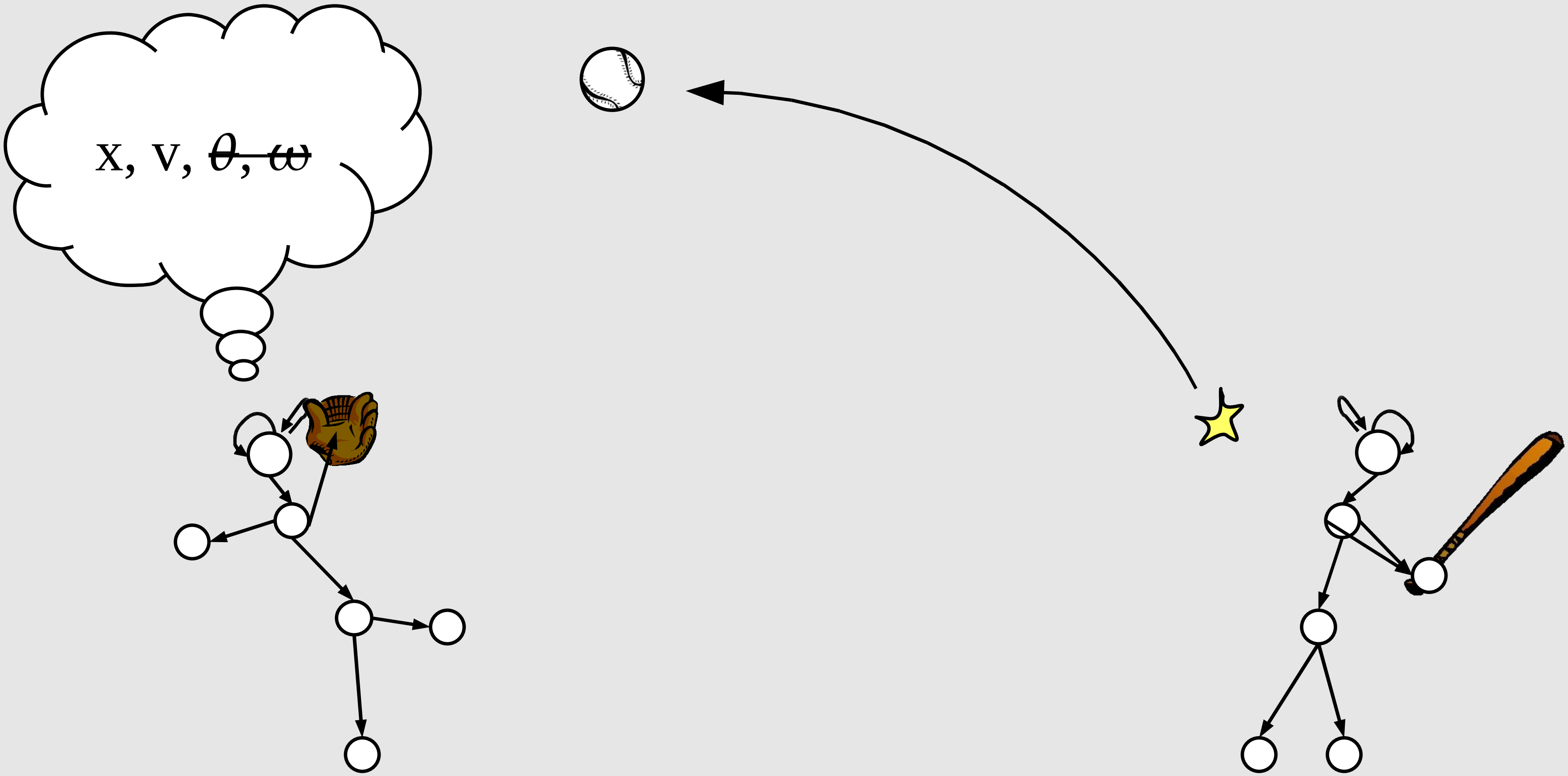
I. Be able to store information for an **arbitrary duration**

- e.g. be able to learn constants



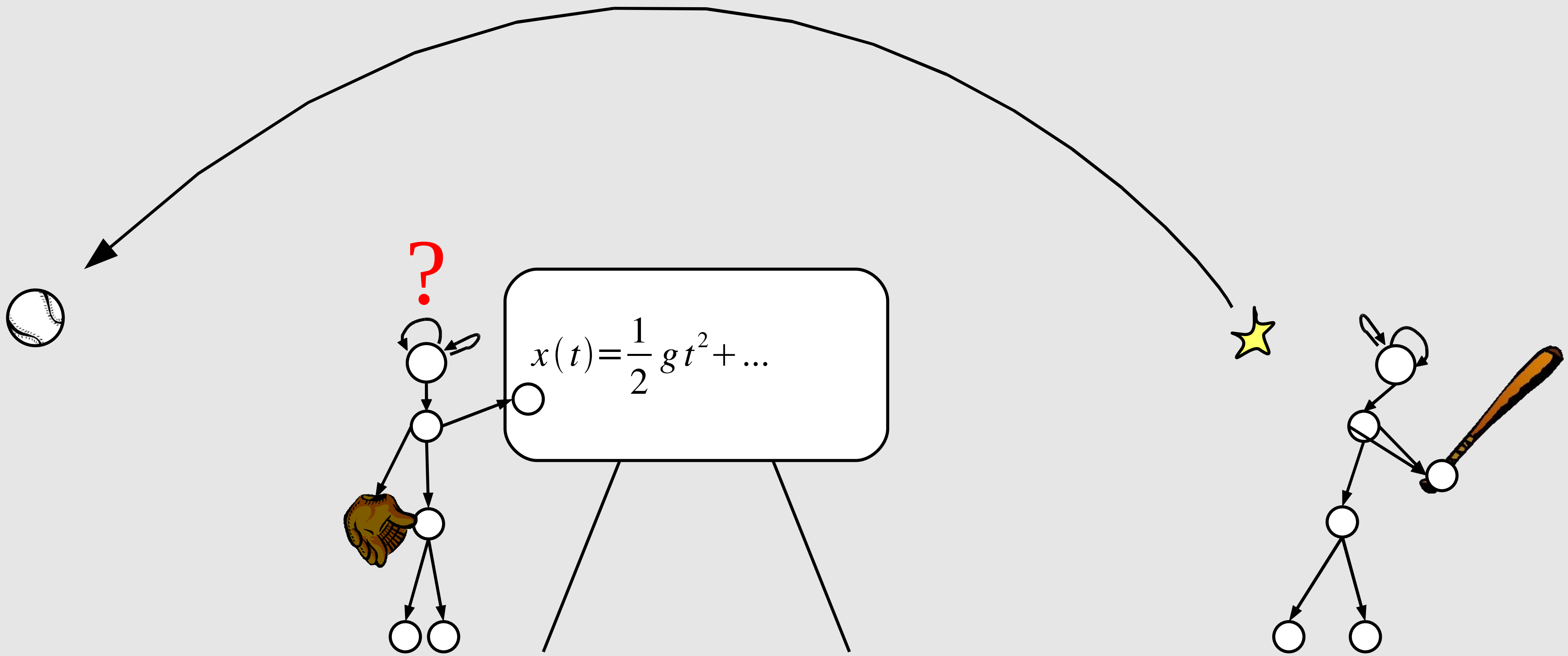
Feature wishlist

II. Be resistant to noise, irrelevant input



Feature wishlist

III. System parameters should be trainable in a reasonable amount of time

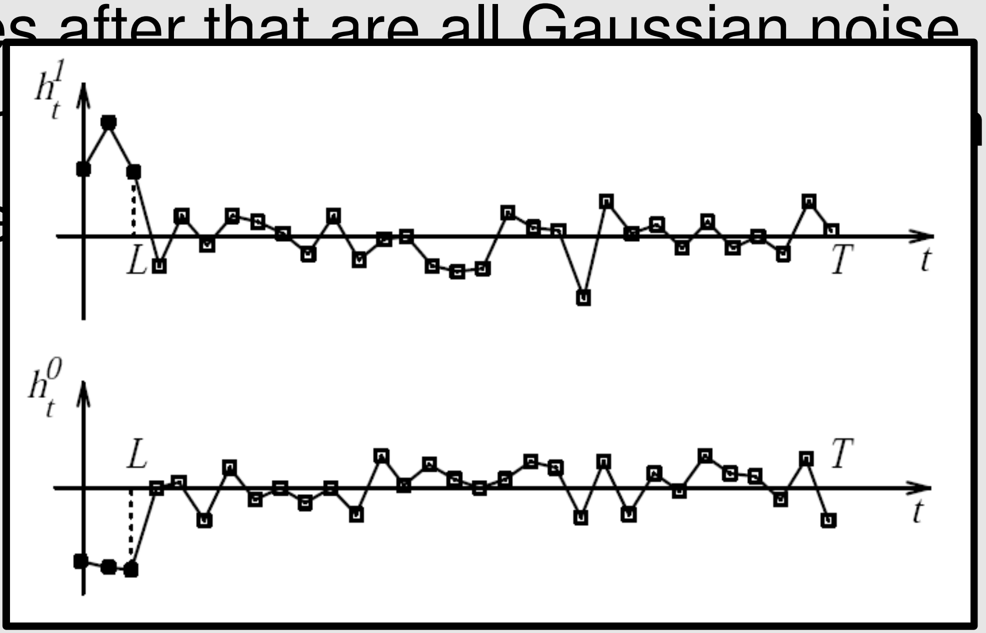


Problem

- “Storage” in recurrent neural nets via gradient-based backprop is problematic due to the trade-off between robust storage and efficient learning.

Min task problem - A simple case

- Classify a sequence based on first L values.
- Decision is given at the end. Therefore, the system must remember its state for an arbitrary amount of time. (1st requirement)
- The values after that are all Gaussian noise, so irrelevant for determ ce. (2nd requireme

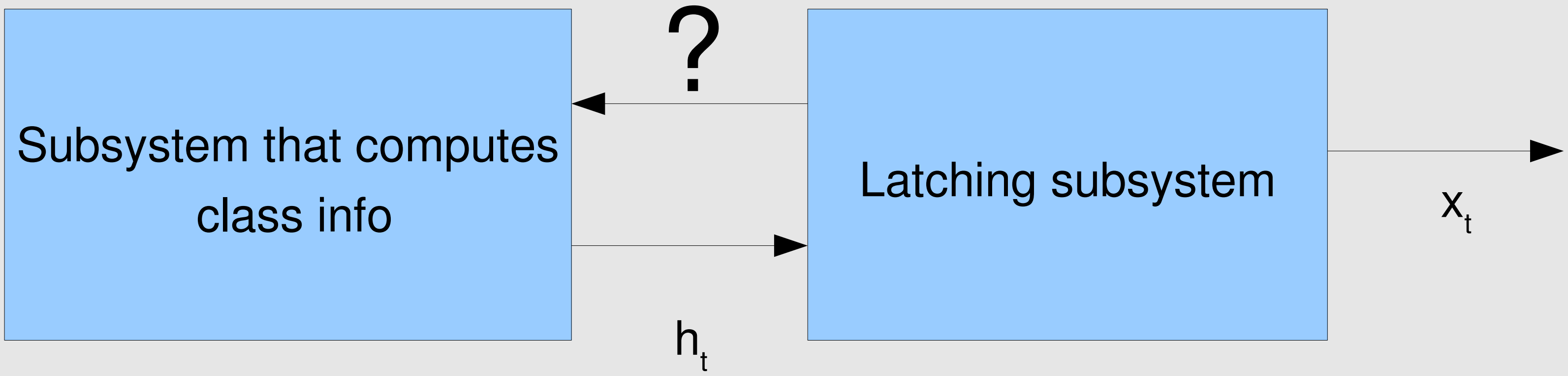


Min task problem

We are interested in the learning capabilities of the latching problem. The big question is : Can latching subsystem propagate error information back to the subsystem that feeds it?

The ability to learn $h_0,...,h_L$ is the measure of effectiveness of gradient info.

(3rd requirement)



h_t is the result of the computation that extracts class information.

A simple recurrent network candidate solution

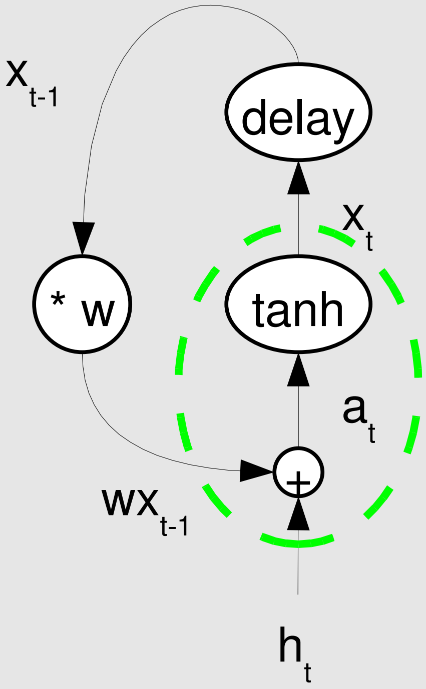
- Input:

  h_t

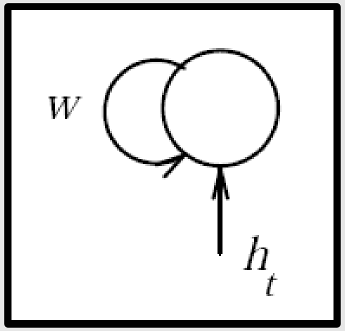
- Output:

  $x_t = \tanh(a_t)$

- Parameters:
  - Trainable input h_t
  - w_o



In the paper:



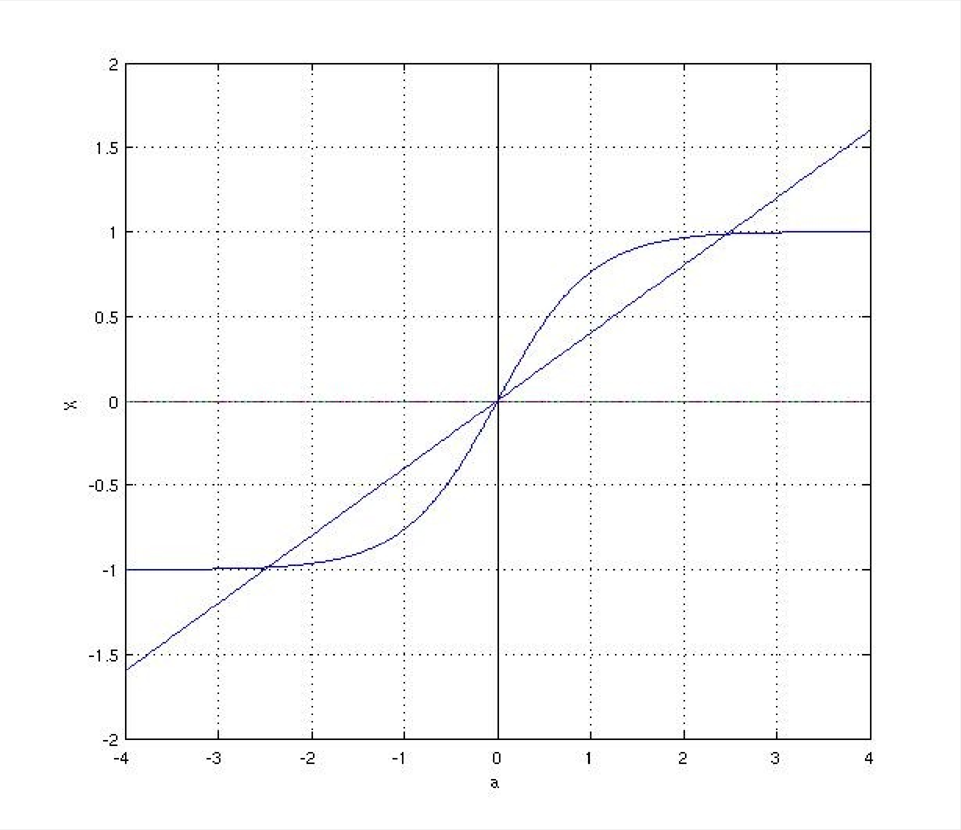
Simple recurrent network candidate solution

- An attractor is an invariant point $a_t = a_{t-1}$
- For the autonomous case:

$$a_t = w \tanh(a_{t-1})$$

- Attractors at nonzero solutions to "$a_t = a_{t-1}$"

$$\frac{a_t}{w} = \tanh(a_{t-1})$$



Simple recurrent network candidate solution

Non-autonomous case (input $h_t \neq 0$):

- Escape threshold h*:
  - As long as lh_tl < h*, neuron won't escape attractor. the activation of your neuron does not change.
- If lh_tl > h*, then the system may be pushed to another attractor.

Experimental Results for the minimal task problem

Convergence density

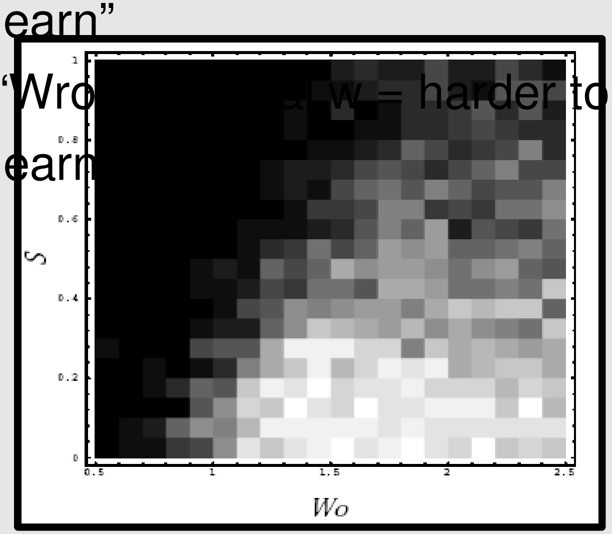
w_0 (initial weight) vs

s (noise variance)

L=3, T=20

“Bigger noise (s) = harder to learn”

“Wro[illegible]al w = harder to learn[illegible]

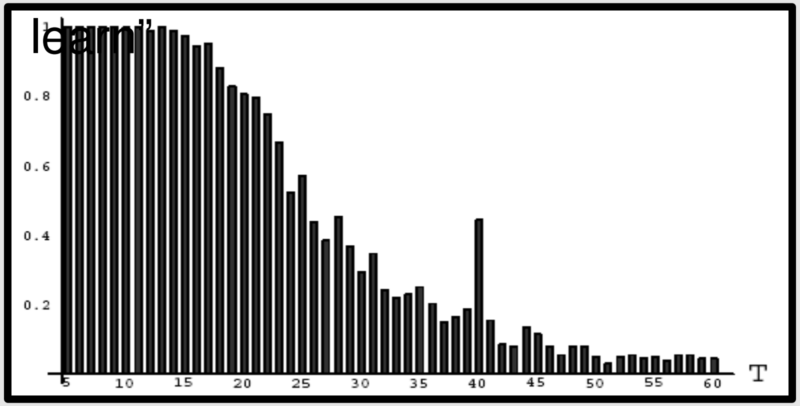


Frequency of training convergence vs T

S=0.2

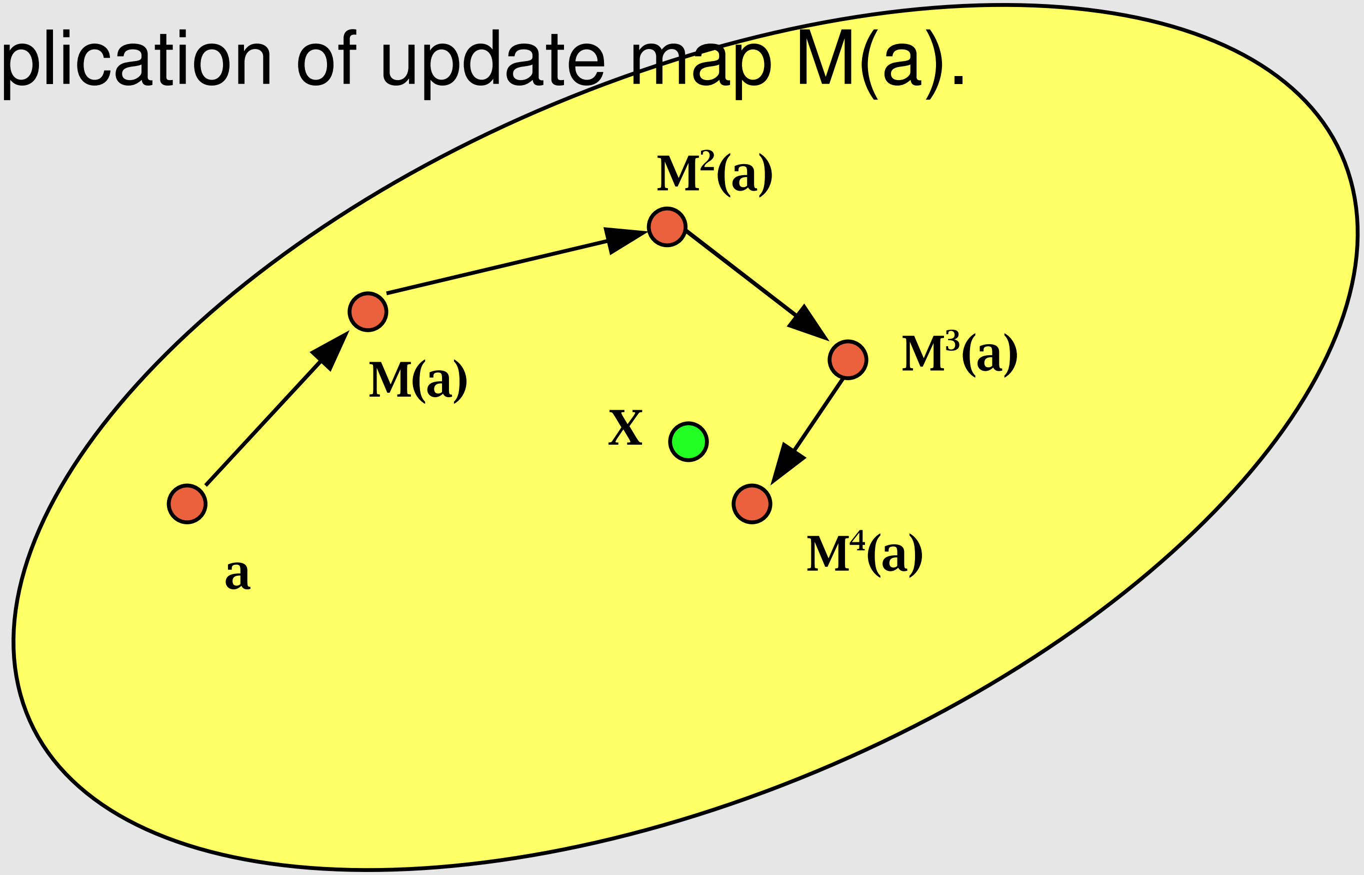
w=1.25

“Longer noise sequence = harder to learn”



Discrete dynamical systems

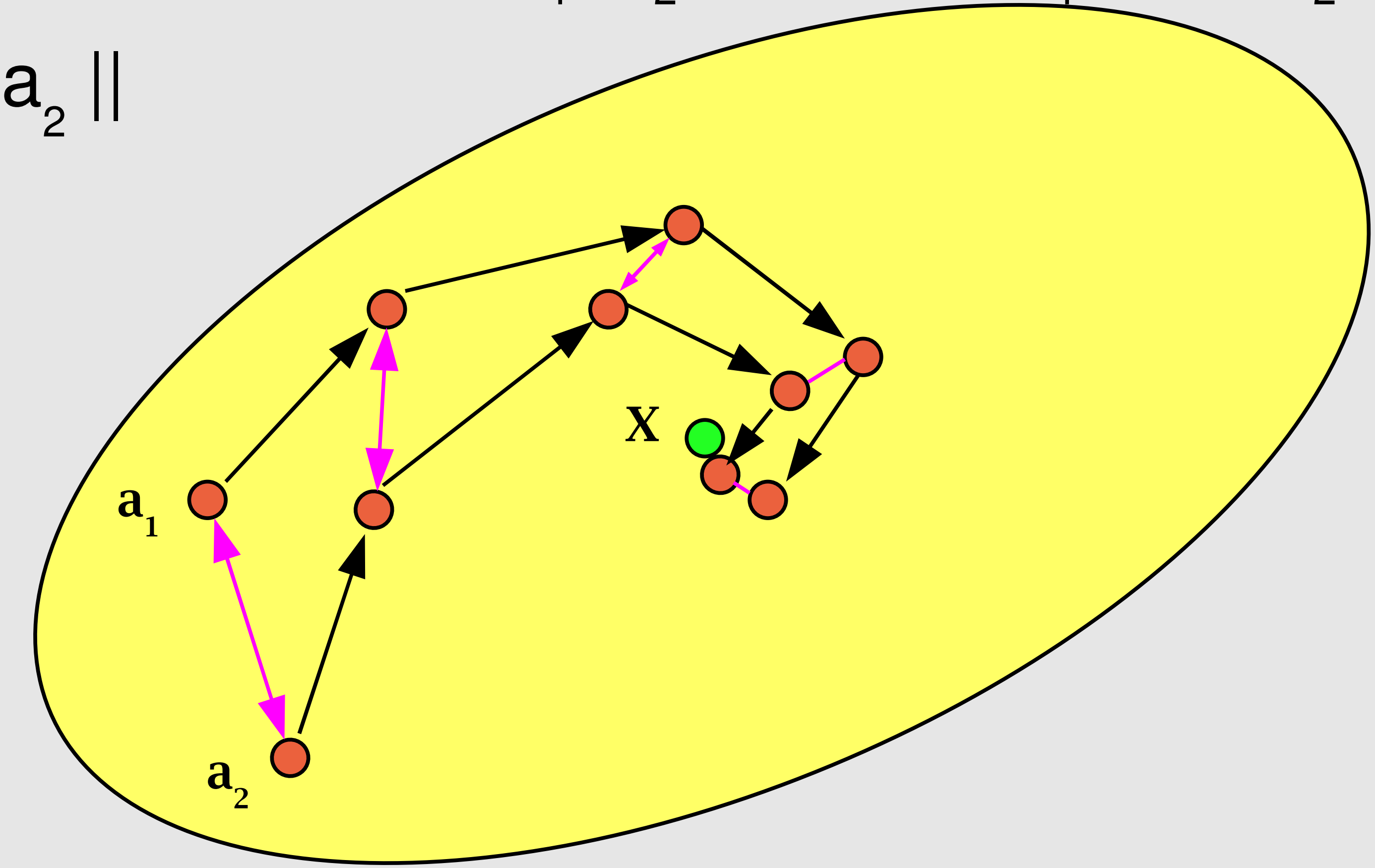
- The **basin of attraction** of an **attractor** X
  - A set of points a that converge to X under repeated application of update map M(a).



Discrete dynamical systems

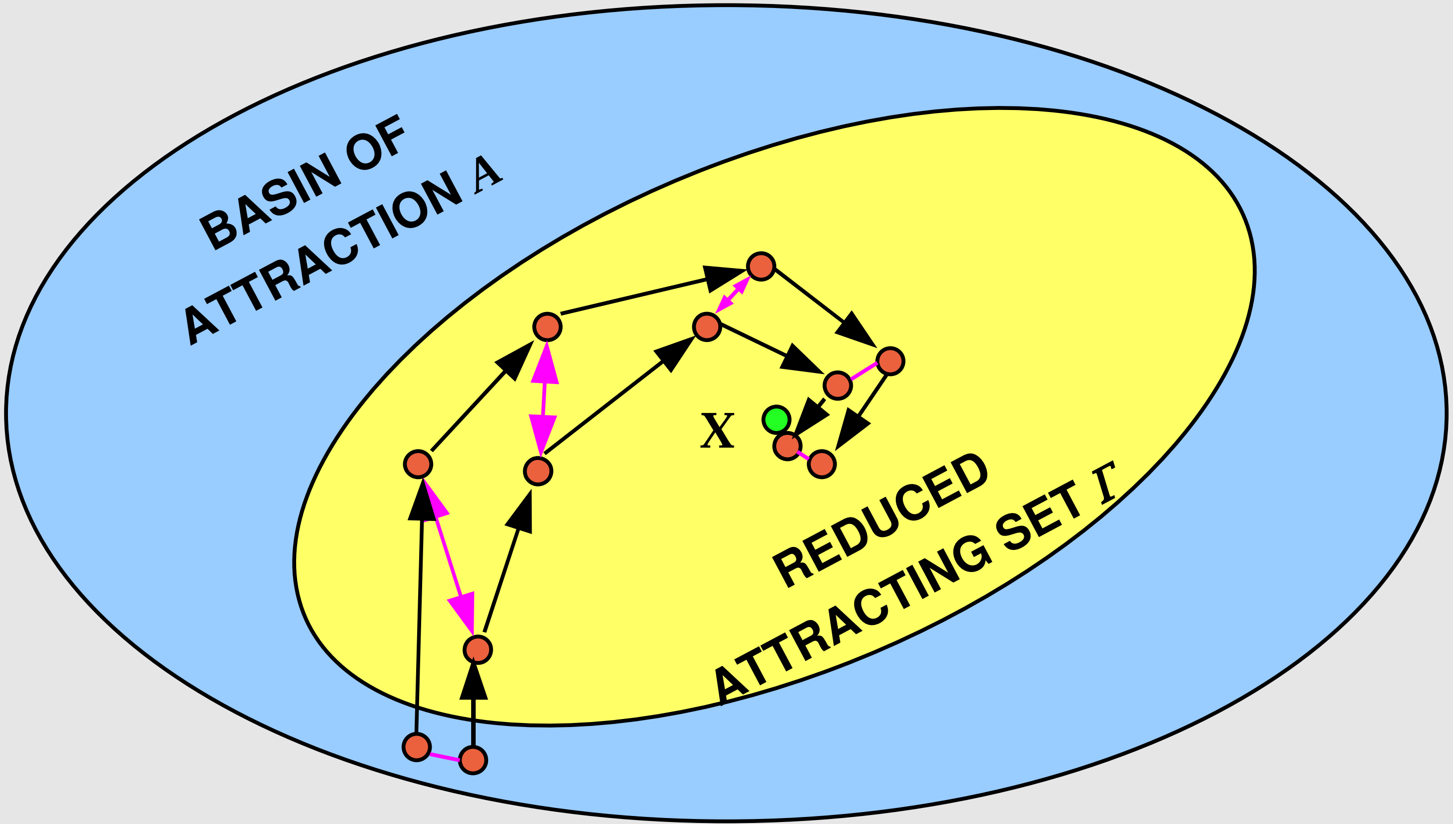
- Map M is **contracting** on a set A if:

For any two points a_1, a_2 in A, $|| M(a_1) - M(a_2) || < || a_1 - a_2 ||$



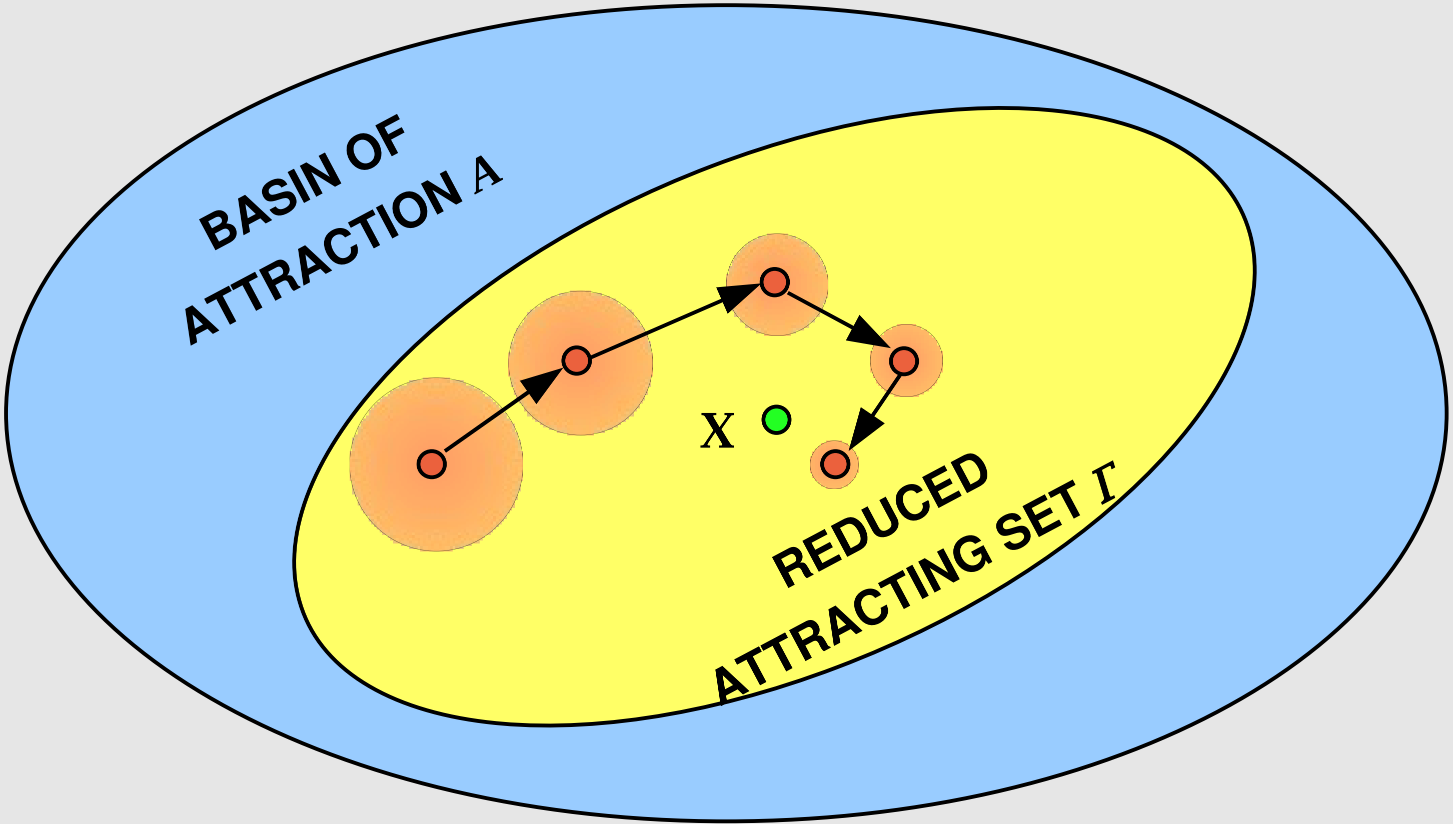
Discrete dynamical systems

- X is a **hyperbolic attractor** that contracts on its **reduced attracting set** Γ if:
  - For all a in Γ, all eigenvalues of $M^{t'}(a) < 1$



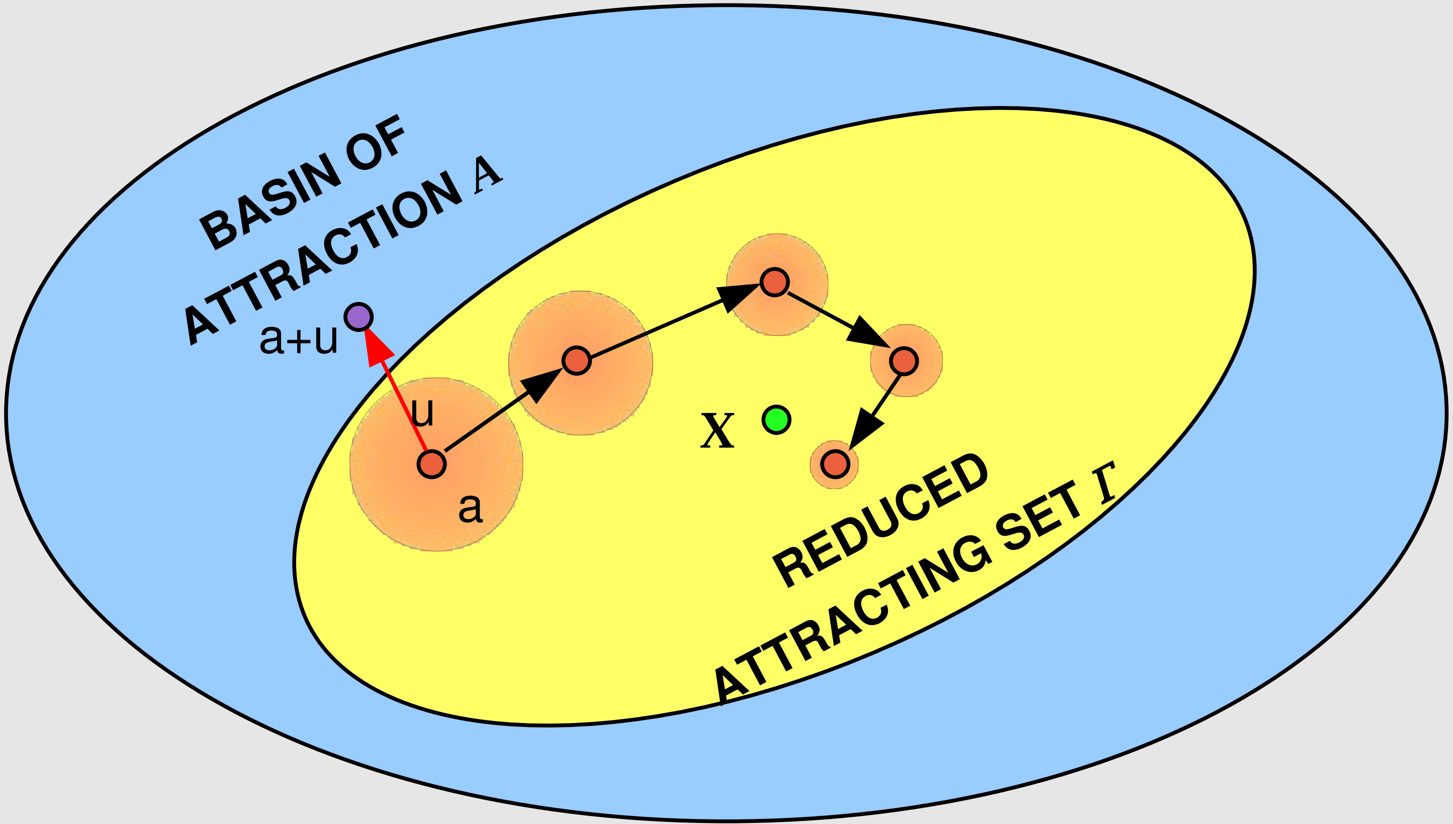
Discrete dynamical systems

- The sphere of uncertainty of a point, if contained in Γ, will shrink at each timestep under no input.



Discrete dynamical systems

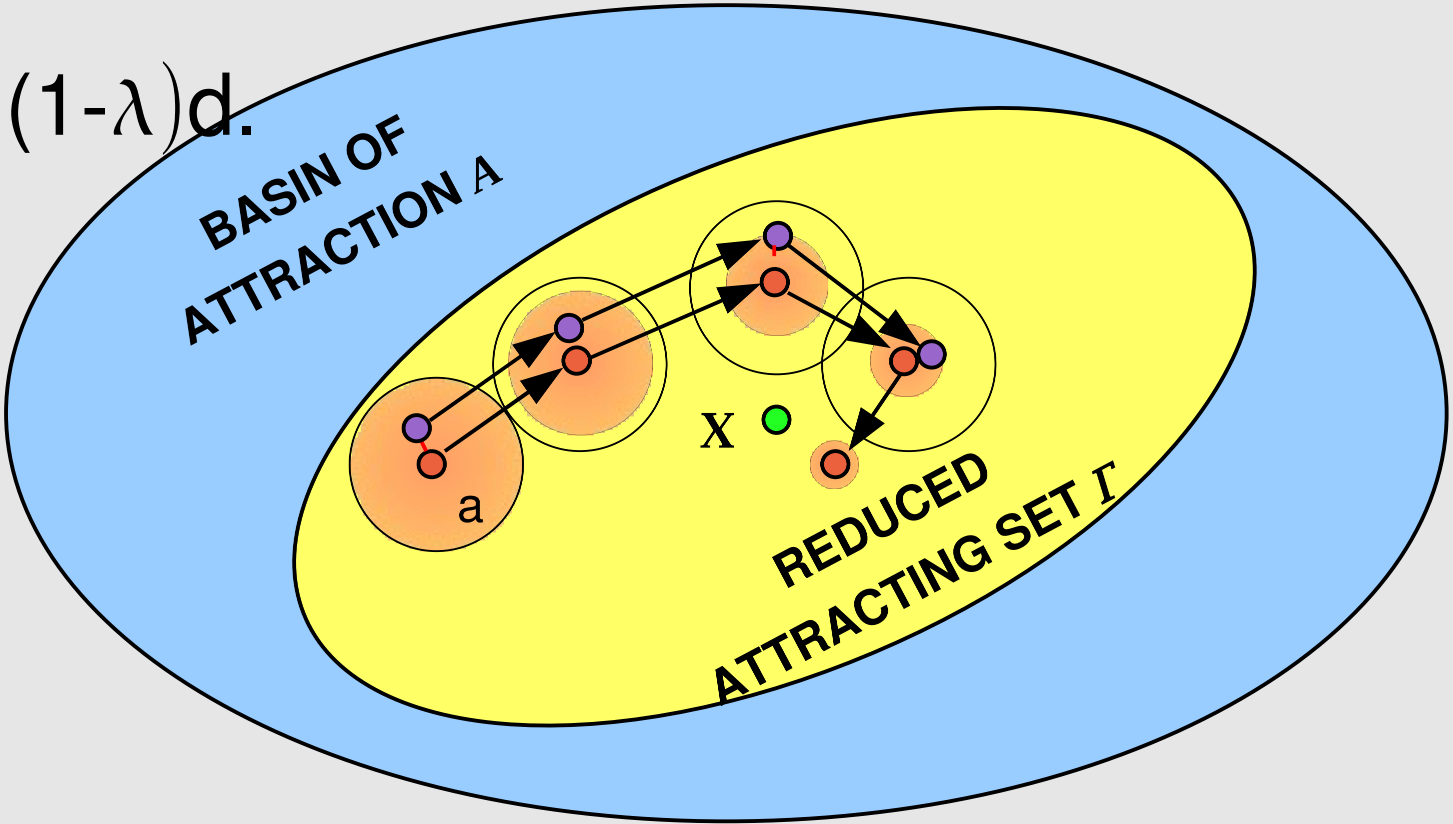
- A system in state $\mathbf{a}$ is **robustly latched** to X if no input $\mathbf{u}$ will bump it out of Γ.



Discrete dynamical systems

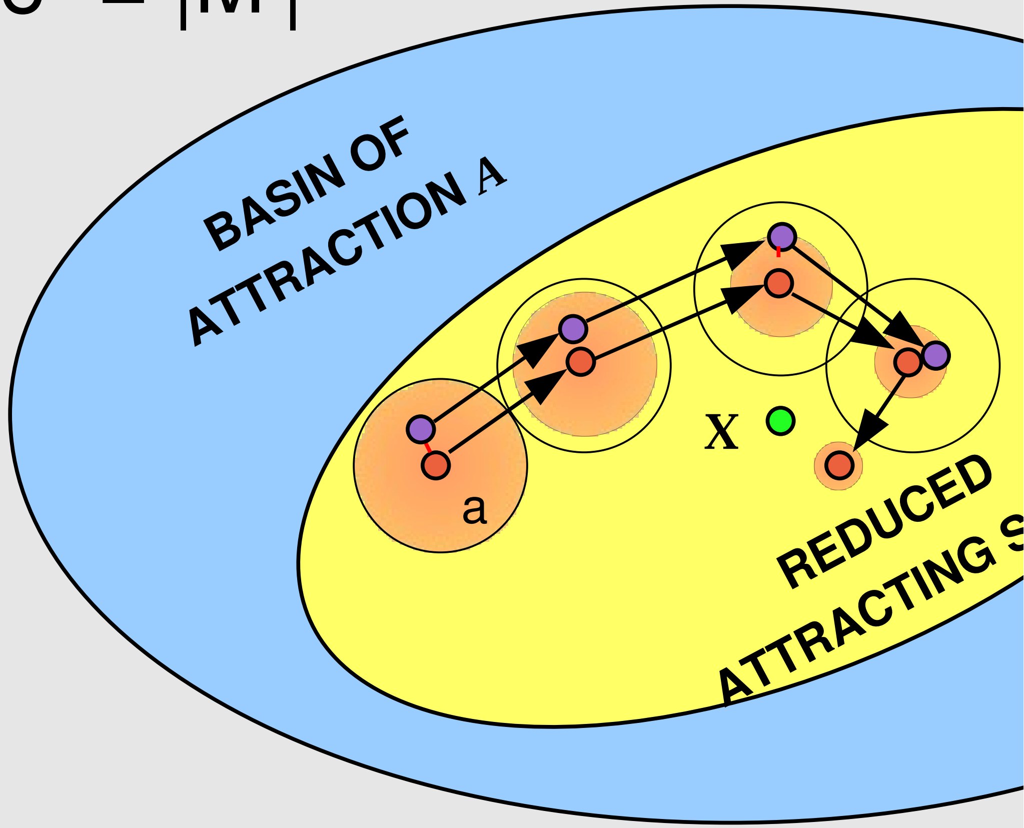
- Let λ be the maximum “shrinkage rate”.
- $M^t(a+u)$ will remain within a radius d of $M^t(a)$ as long as

$$||u_t|| < (1-\lambda)d.$$



Discrete dynamical systems

- “Minimum shrinkage rate” = |M'|
- Within Γ, |M'| < 1



$$\left|\frac{da_t}{da_{t-1}}\right| = \left|M'(a_{t-1})\right| < 1$$

$$\left|\frac{da_t}{da_0}\right| = \left|\prod_{\tau=0}^{t-1} M'(a_\tau)\right| \to 0$$

$$\left|\frac{da_t}{da_0}\right|$$

vanishes as t

Effect of the gradient

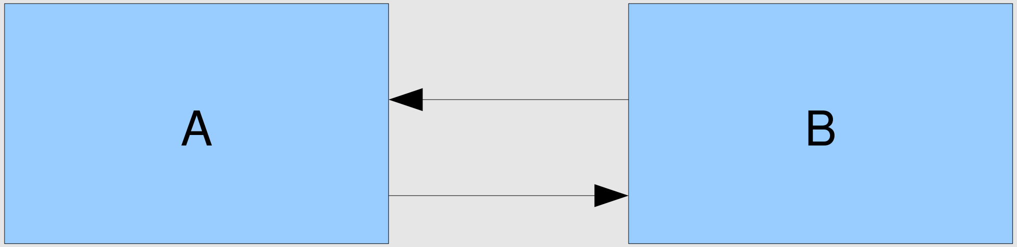
The derivative of the cost C at time t with respect to the parameters of the system is

$$\frac{\partial C_t}{\partial W} = \sum_{\tau \leqslant t} \frac{\partial C_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W} = \sum_{\tau \leqslant t} \frac{\partial C_t}{\partial a_t} \frac{\partial a_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W}$$

From the previous analysis, we know for terms with $\tau \lll t$ $\left| \frac{\partial C_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W} \right| \rightarrow 0$

This means that even though there is a change in the state variable that allows the system to change the basin of attractor, this would not be reflected in the gradient info.

Effect of gradient

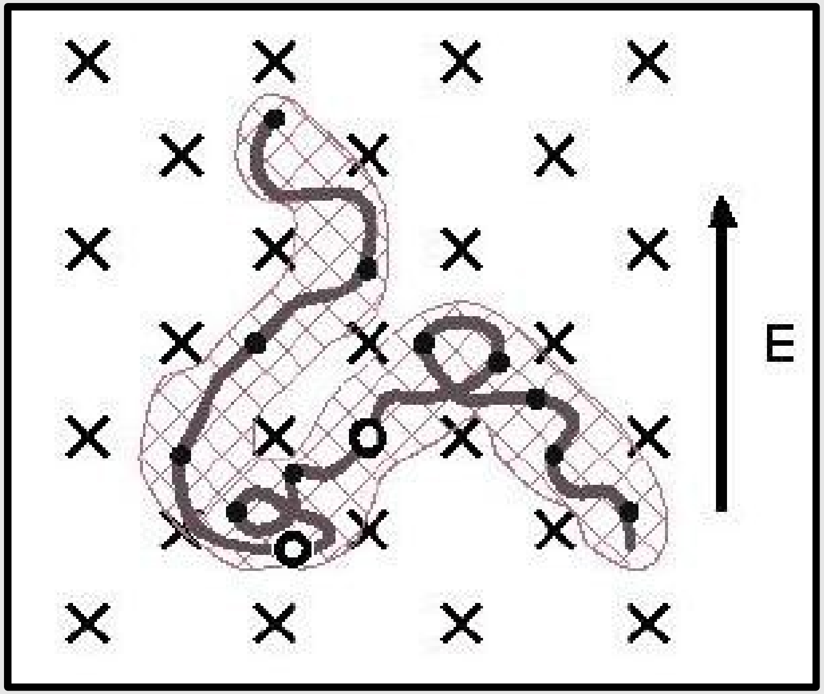


If B has not been trained enough for latching then the gradient of B's output with respect to A's early output will be zero. A's early outputs can't influence B's error at time T, B does not remember them.

On the other hand, if B has latched, then the terms in the gradient that refer to the earlier states -the times when real input came, in other words what really matters for training the classifier- have vanished according to the theorem. So training A is very difficult.

Alternatives to gradient descent

- Stochastic search
  - Simulated annealing
  - Multi-grid random search
- Curvature-weighted gradient descent
  - Quasi-newton
  - Time-weighted QN
- Discrete error



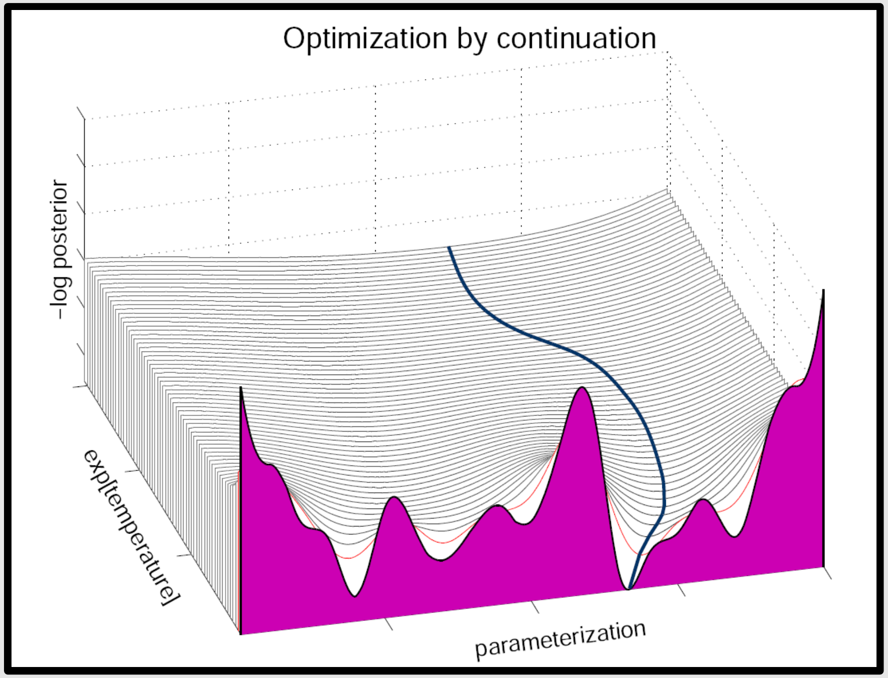
Simulated Annealing

- Metropolis criterion
  - Accept uphill steps with probability:

$$P_a = \exp\left(-\frac{\Delta C}{T}\right)$$

- Dynamic selection hyperrectangle
  - Update dimensions to keep acceptance rate

There are many annealing approaches:



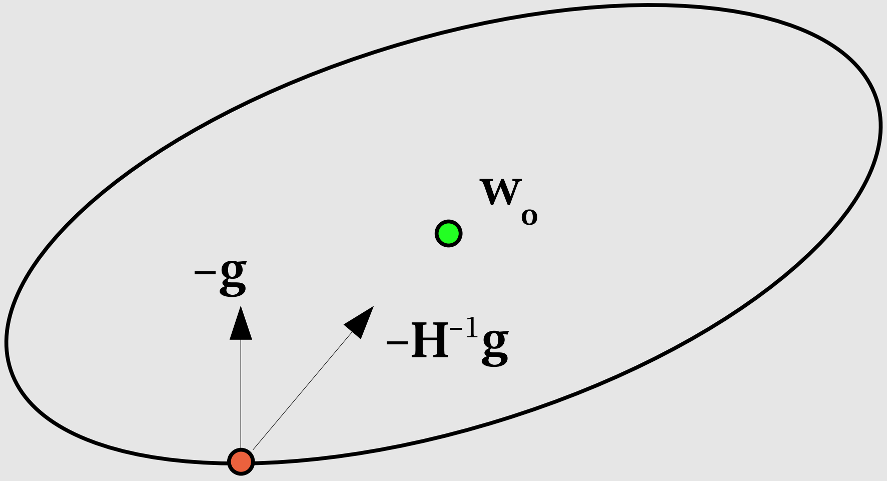
A deterministic annealing example from:
Matthew Brand and Aaron Hertzmann,

Multi-grid Random Search

- The problem isn't local minima, it's long plateaus (small gradients)
- Multigrid random search == gradient descent without the uphills.

Quasi-newton methods

- Newton update
  - Uses Hessian H for more accurate update.
  - Costly, $O(NW^2)$ (N = # examples)
  - Could point to maximum if $||H|| < 0$



Quasi-Newton methods

- An alternative (Becker, LeCun 1989):
- Only use diagonal elements of Hessian
  - H^{-1} in O(NW) time
  - Separate equations for each weight

$$\Delta \boldsymbol{w} = -\boldsymbol{H}^{-1} \frac{\partial C(p)}{\partial w_i}$$

$$\Delta w_i = \frac{-1}{\left| \frac{\partial^2 C(p)}{\partial w_i^2} \right|} \times \frac{\partial C(p)}{\partial w_i}$$

Quasi-Newton methods

Some tweaks:

- η = learning rate
- μ = insurance against zero curvature

$$\Delta w_i = -\frac{\eta}{\left|\frac{\partial^2 C(p)}{\partial w_i^2}\right| + \mu} \times \frac{\partial C(p)}{\partial w_i}$$

“When curvature is low, take bigger jumps”

Quasi-Newton methods

- Apply to our recurrent net by:
  - Unfolding recurrent net into deep static net:
  - Treating the unfolded weights as different variables.
  - Keep them close to each other via some soft constraint.

$$\Delta w_i = -\sum_t \frac{\eta}{\left|\frac{\partial^2 C(p)}{\partial w_{it}^2}\right| + \mu} \times \frac{\partial C(p)}{\partial w_{it}}$$

Quasi-Newton methods

- Compare with backprop/gradient descent:

$$\Delta w_i = -\sum_t \frac{\eta}{\left|\frac{\partial^2 C(p)}{\partial w_{it}^2}\right| + \mu} \times \frac{\partial C(p)}{\partial w_{it}}$$

$$\Delta w_i = -\eta \sum_t \frac{\partial C(p)}{\partial w_{it}}$$

Discrete error propagation

- We're shoehorning a continuous solution (gradient descent) into a binary problem (latching):

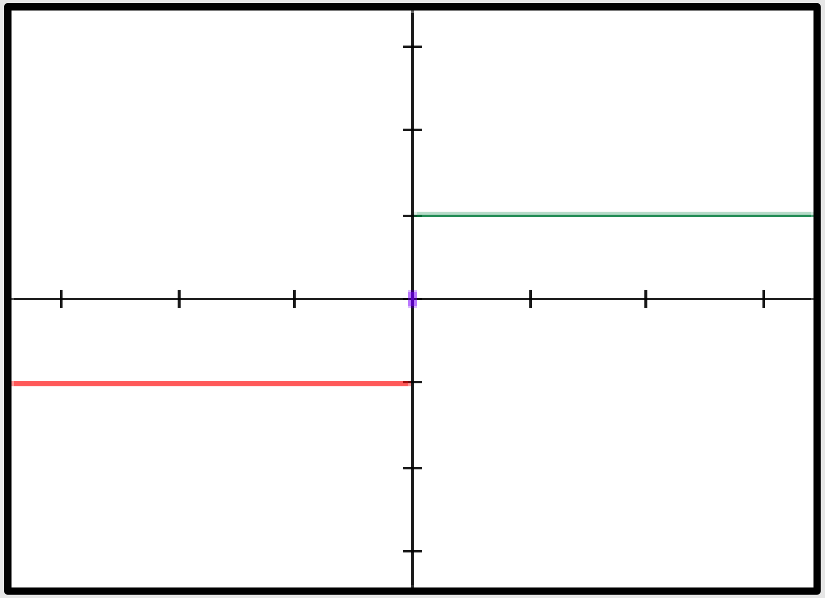
$$\overset{\textit{continuous}}{C(y)} \Leftrightarrow \overset{\textit{discrete}}{y(x)} \Leftrightarrow x$$

- How can we back-propagate error signals across discrete functions?

Discrete error propagation

- Let y = sign(x)

$$\Delta x = \begin{cases} -x & if \quad |\Delta y| = 2 \\ 0 & if \quad \Delta y = 0 \end{cases}$$



- Backpropagated $\Delta \tilde{y}$:

$$\Delta \tilde{y} = \eta \left(\frac{\partial C}{\partial y} \right)^{-1}$$

- Problem: $\Delta \tilde{y}$
  - not necessarily -2, 2,

Discrete error propagation

- Translate continuous gradient g= $\frac{\partial C}{\partial y}$ into discrete value Δy using a discrete stochastic function:

$$\Delta y(g) = \begin{cases} 2 & \textit{with probability} \quad \dfrac{g - MIN}{MAX - MIN} \\ -2 & \textit{with probability} \quad \dfrac{MAX - g}{MAX - MIN} \end{cases}$$

$$E(\Delta y(g)) = g$$

- Expectation quickly converges to g when

Results

3 problems:

- Latch problem
  - Classify noisy sequence where the first L values hover around a constant.
- 2-sequence problem
  - Latch problem with non-constant first L values.
- Parity problem
  - Is the sum of a sequence even or odd.

Results

- Simulated annealing
  - Good: best overall error.
  - Bad: requires ~$10^{1.8}$ times more iterations than other methods.
- Multigrid stochastic search
  - Good: comparable speed to gradient-based methods.
  - Bad: comparable susceptibility to local minima.

Results

- Standard backprop
  - Vanishing gradient problem, local minima.
- Time-weighted quasi-newton
  - Same problems as backprop, just less bad.
- Discrete error propagation:
  - Usually the fastest, error rate comparable to non-annealing methods.
  - Particularly good at parity problem: robust to local minima?

End

Questions?

Appendix: Derivation of Newton update

$$
\begin{aligned}
E(w) &= E(w_o) + \frac{1}{2}(w - w_o)^T H (w - w_o) \\
\frac{dE}{dw} &= H(w - w_o) \\
w_o &= w - H^{-1}\frac{dE}{dw}
\end{aligned}
$$