Space-Time Algebra:
A Model for Neocortical Computation

J E Smith

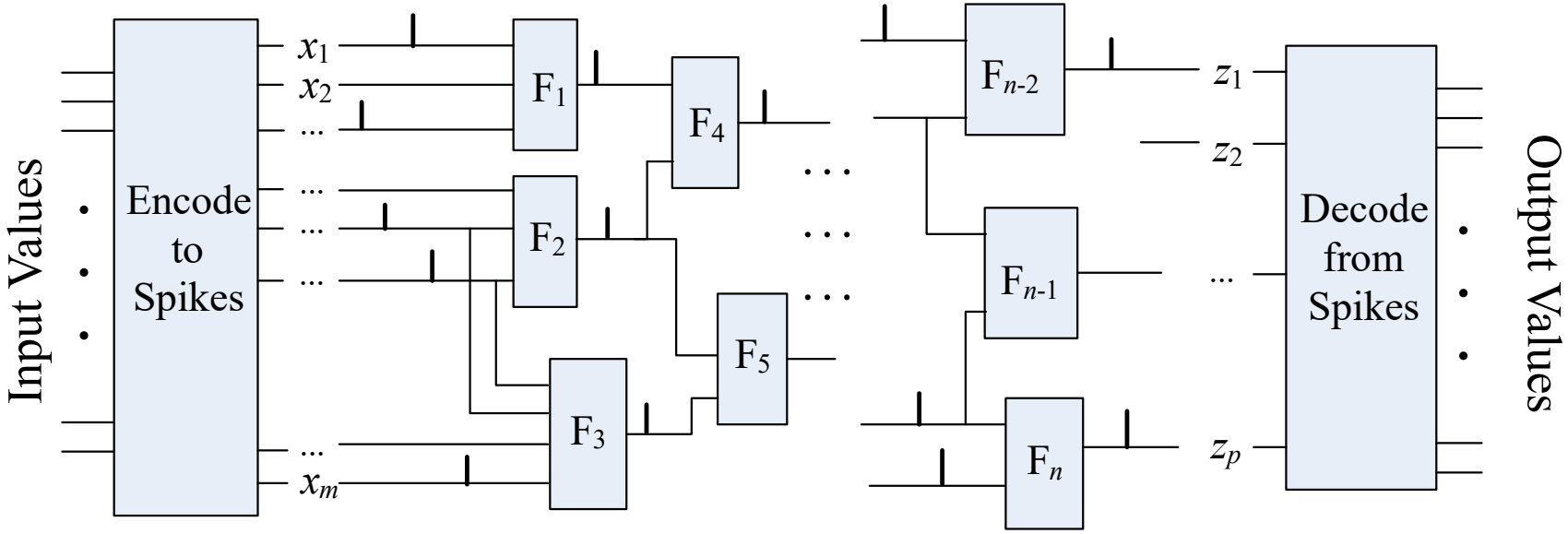
June 2018

Missoula, MT

jes.at.ece.wisc.edu@gmail.com

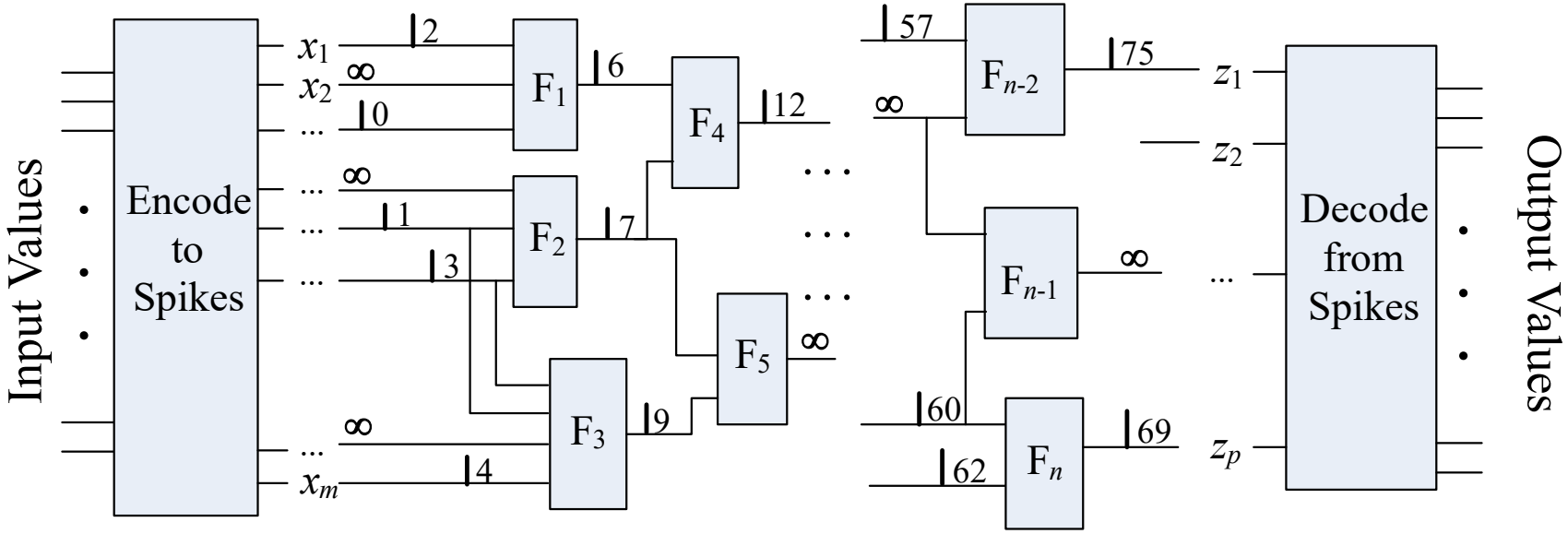
A Design Challenge

- Consider designing networks that compute functions as follows:
  - Input values: encoded as patterns of transient events (e.g. “voltage spikes”).
    Spike times encode values
  - Computation: a wave of spikes passes from inputs to outputs
    Feedforward flow
    At most one spike per line
  - Output values: output spike pattern is decoded into a useful form



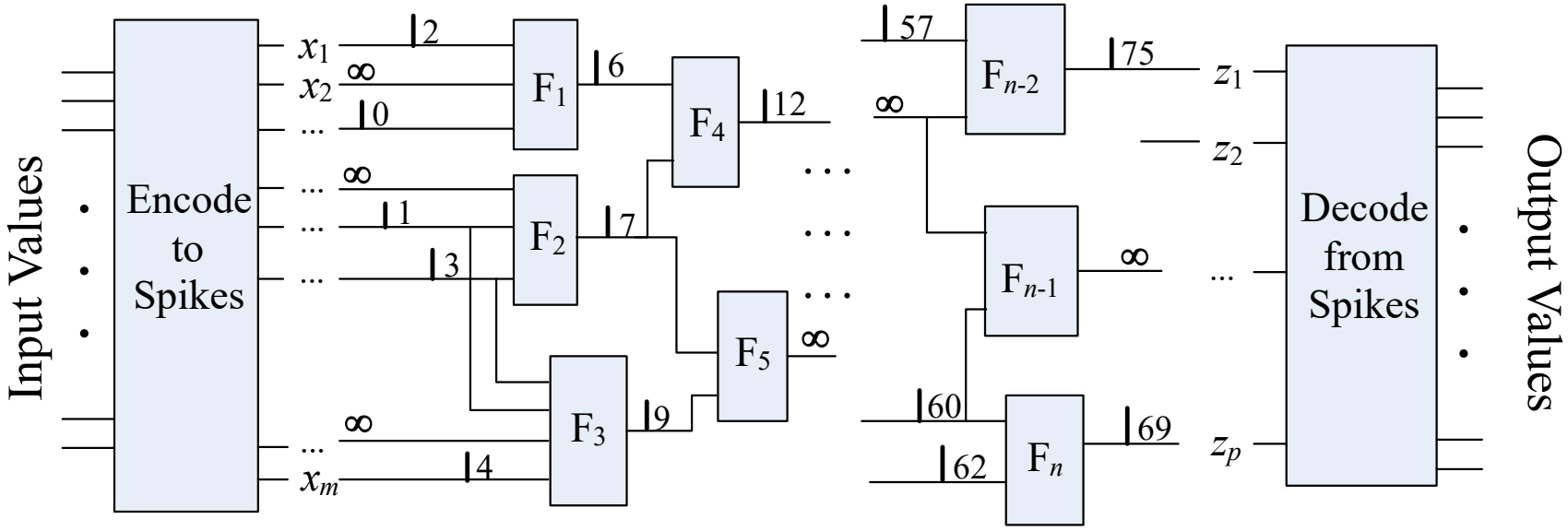
Asynchronous, Local Computation

- Assign values according to spike times
  - Non-spike is denoted as "∞"
- Each functional block operates asynchronously and locally:
  - Begins computation when its first input spike arrives
  - Operates on input values relative to first spike time
  - Eventually produces an output spike or no spike ("∞")



Why Would Anyone Want To Do This?

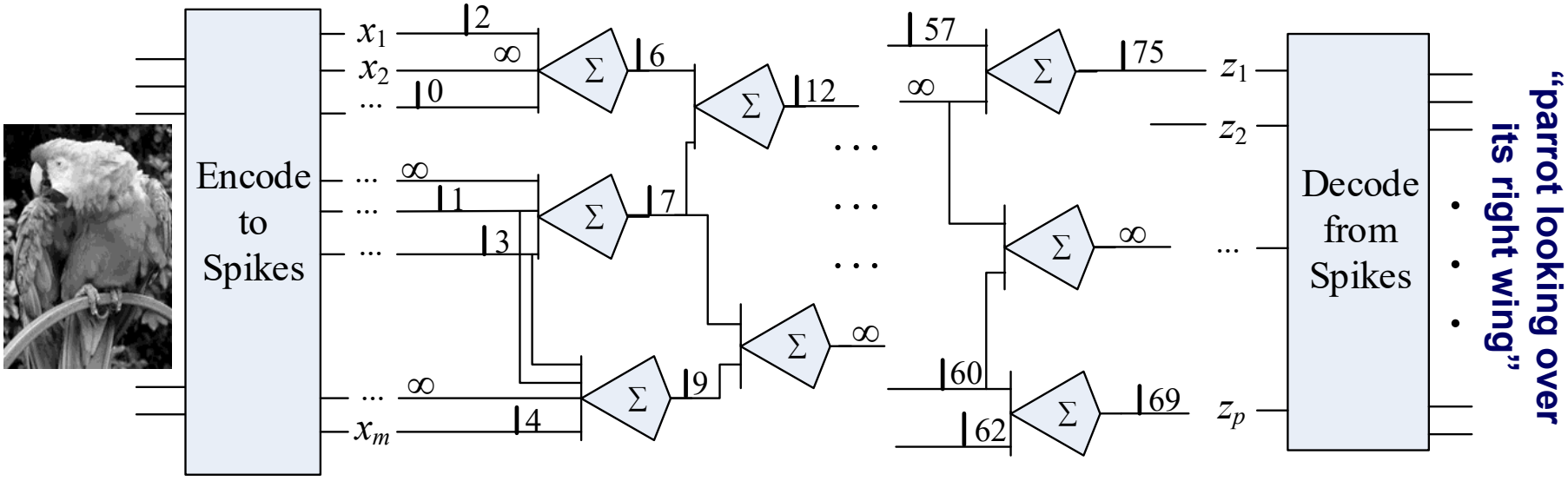
- Conceptually simple, but this really seems like a difficult way to design a computing device, even if we assume ideal delays.



Why Would Anyone Want To Do This?

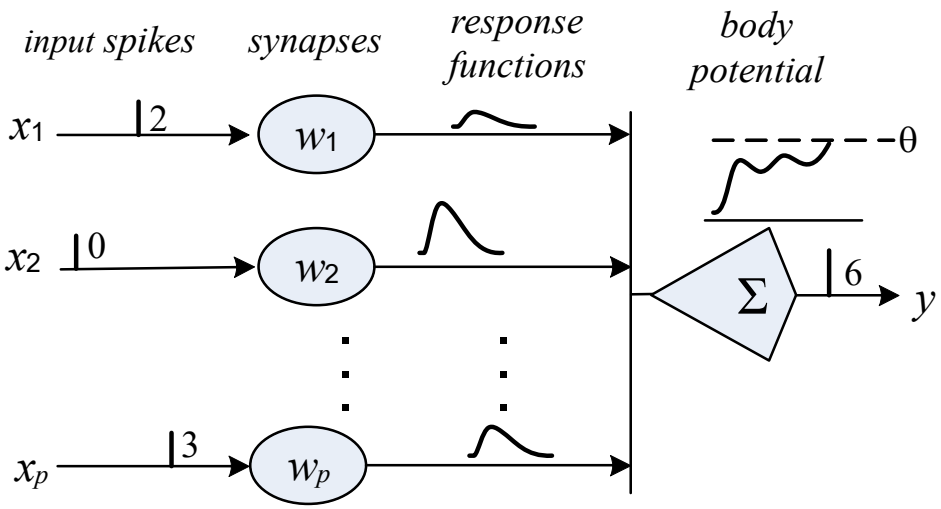
- Call the functional blocks “neurons”, and we have an important class of *biologically plausible* spiking neural networks

  Potentially, perform brain-like functions in a brain-like way

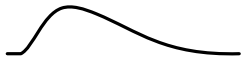


Spiking Neuron Model

- Basic Spike Response Model (SRM0) -- Kistler, Gerstner, & van Hemmen (1997)



response for high weight



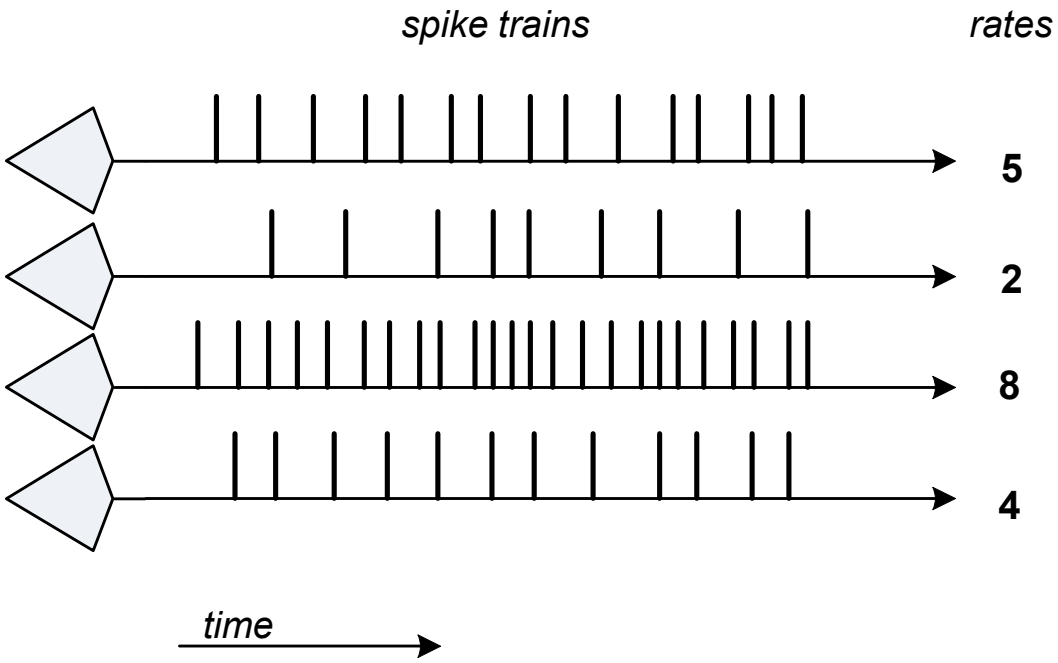
response for lower weight

- A volley of spikes is applied at inputs
- At each input's synapse, a spike generates a pre-defined response function
  - Synaptic weight (w_i) determines amplitude: larger weight ⇒ higher amplitude
  - *Excitatory* response shown; *Inhibitory* response has opposite polarity
- Responses are summed linearly at neuron body
- Fire output spike if/when potential exceeds threshold value (θ)

Spiking Neural Networks

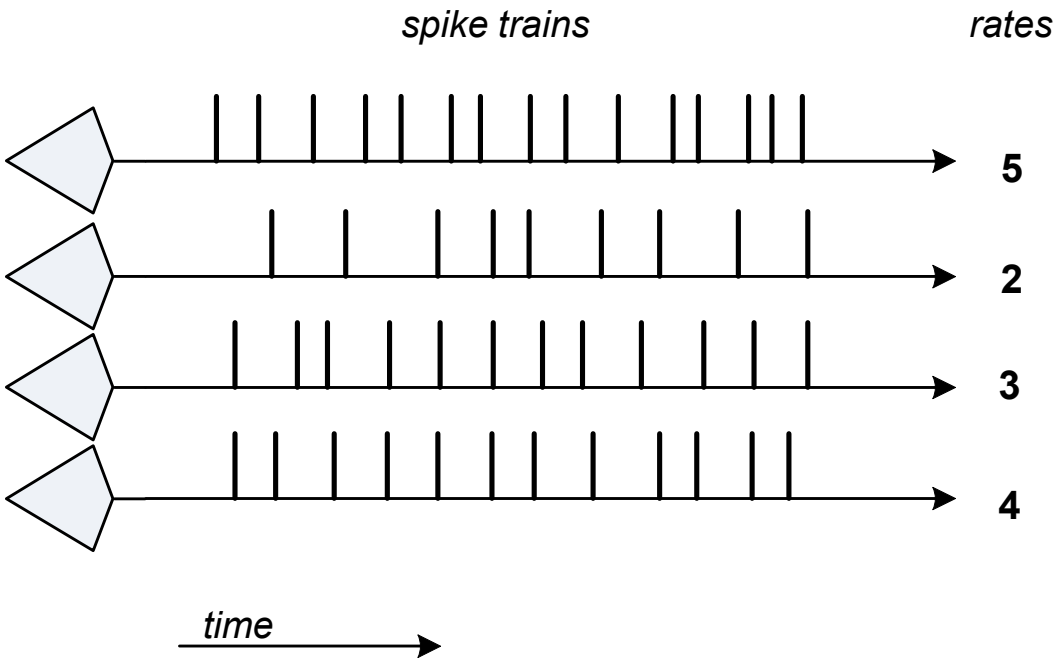
Communicating with Spikes: Rate Coding

- Rate Coding: The name just about says it all
  - The first thing a typical computer engineer would think of doing
- A defining feature*: Changing the value on a given line does not affect values on other lines*



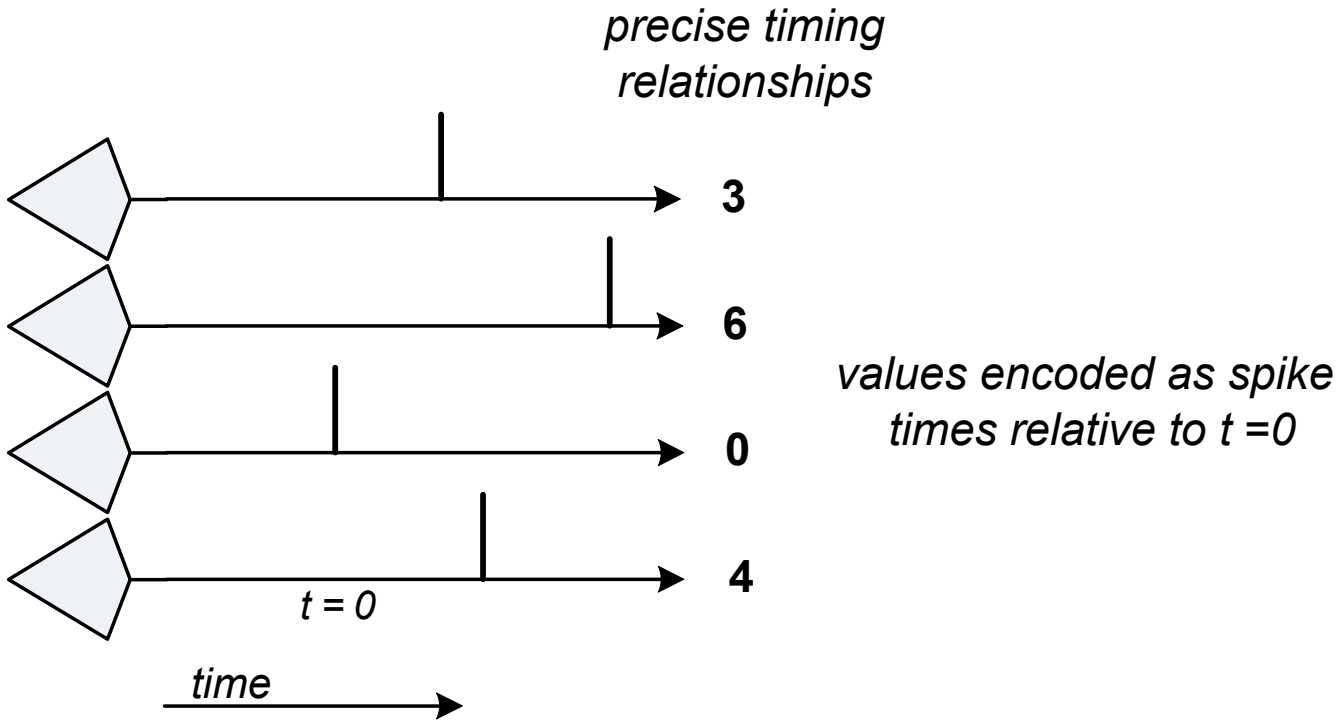
Communicating with Spikes: Rate Coding

- Rate Coding: The name just about says it all
  - The first thing a typical computer engineer would think of doing
- A defining feature*: Changing the value on a given line does not affect values on other lines*



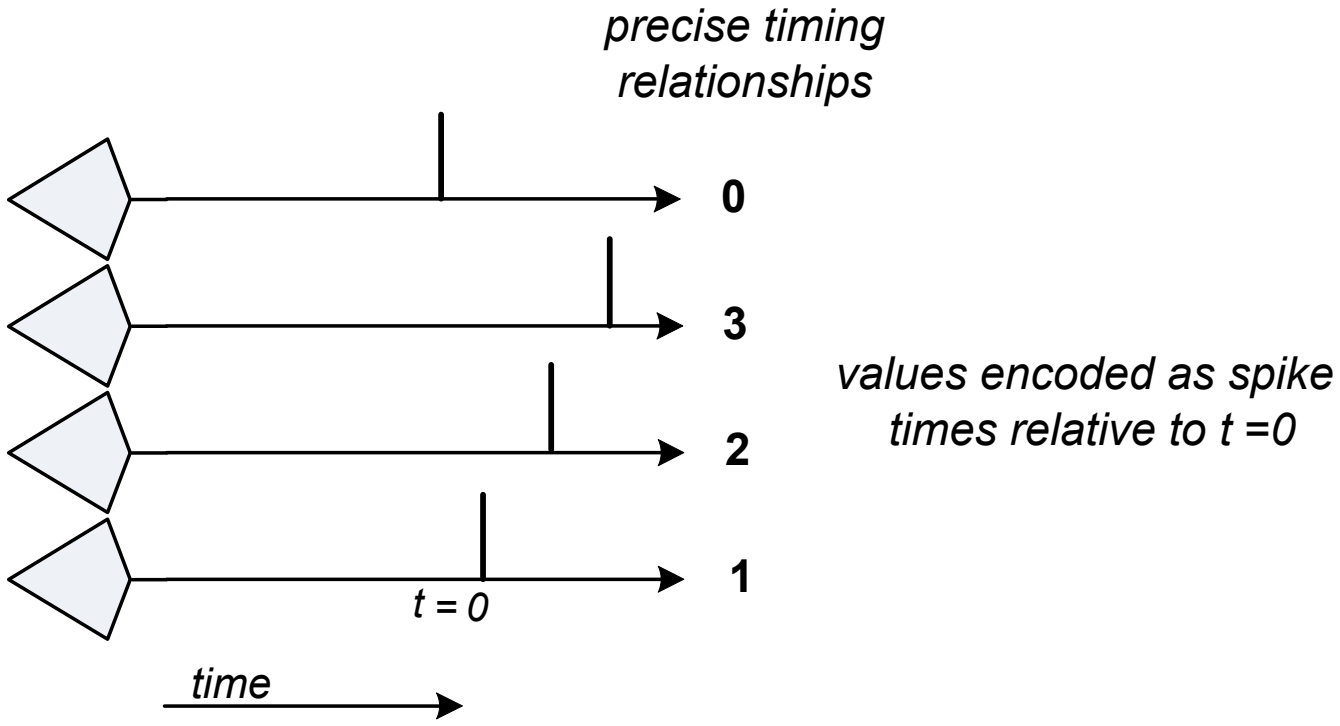
Temporal Coding

- Use relative timing relationships across multiple parallel lines to encode information
- A defining feature: *Changing the value on a given line may affect values on any/all the other lines*



Temporal Coding

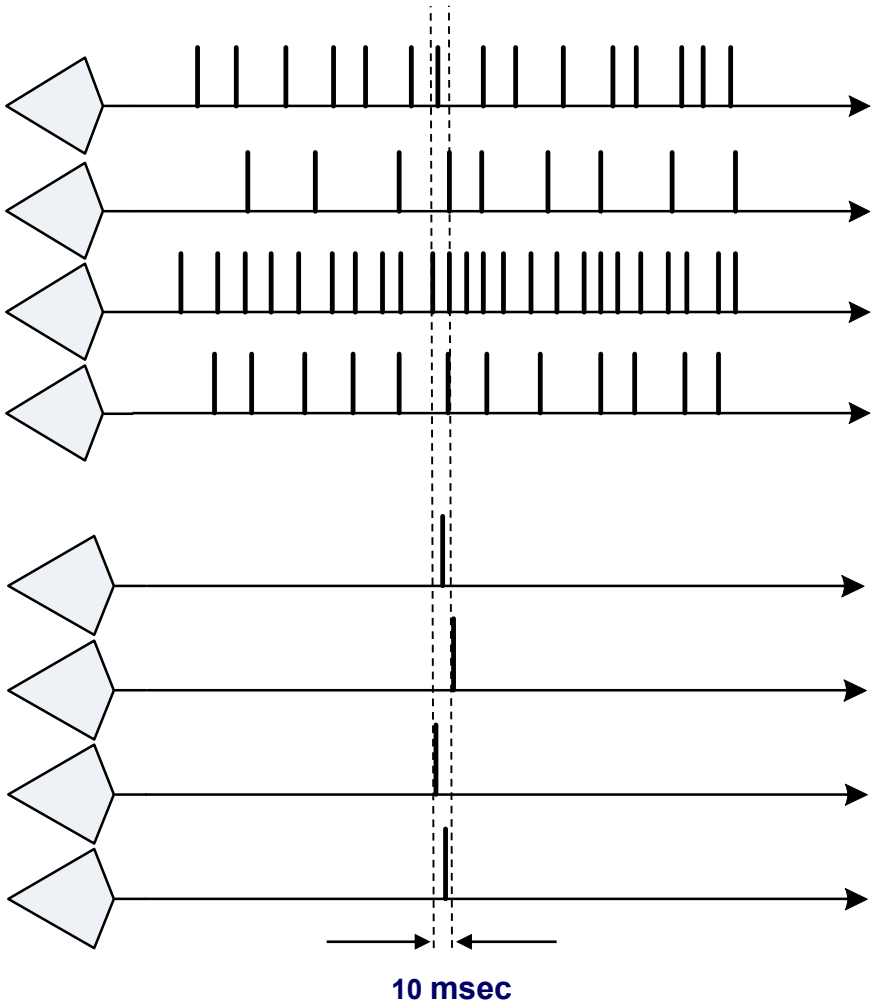
- Use relative timing relationships across multiple parallel spikes to encode information
- A defining feature: *Changing the value on a given line may affect values on any/all the other lines*



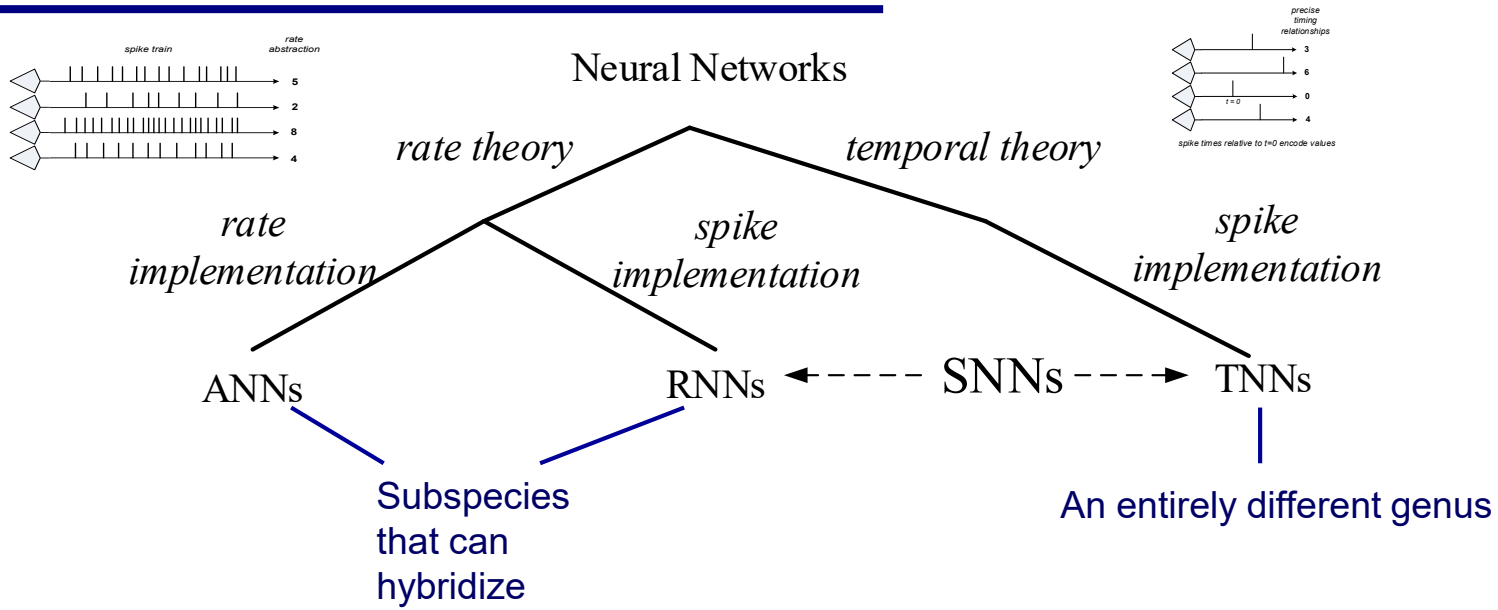
Plot on Same (Plausible) Time Scale

Temporal method is

- An order of magnitude faster
- An order of magnitude more efficient (#spikes)



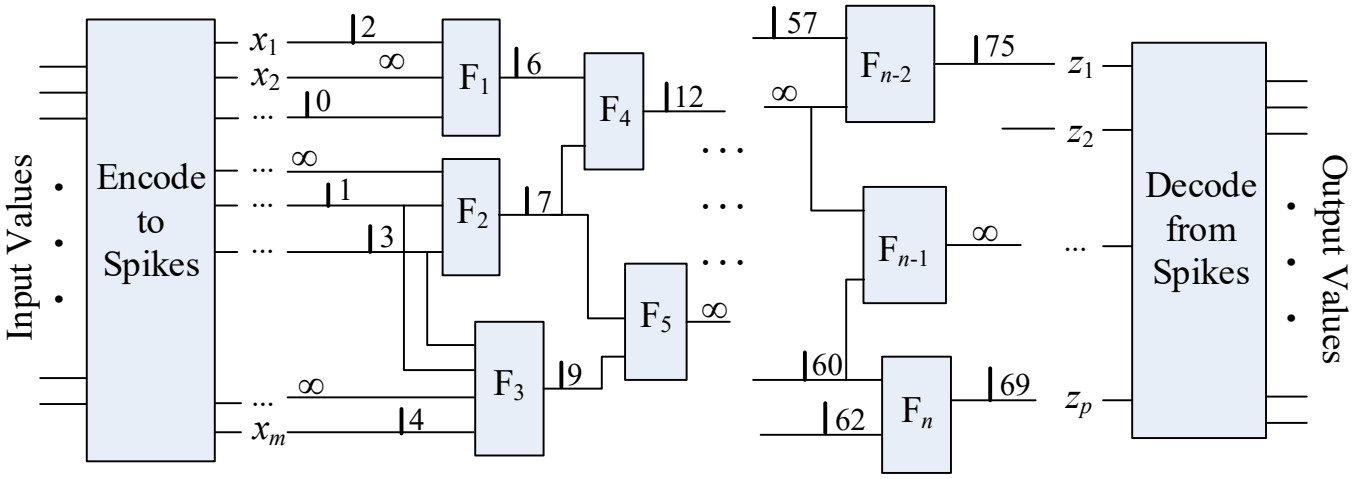
Neural Network Taxonomy



RNNs and TNNs are two very different models, both with SNN implementations, and they should not be conflated

Space-Time Algebra

Space-Time Computing Network (informal)

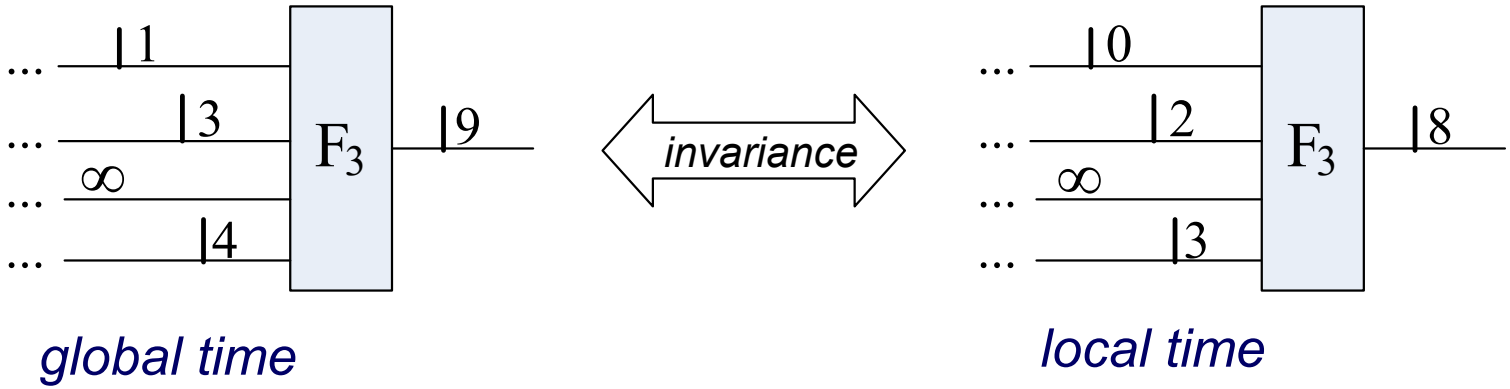


A *Space-Time Computing Network* is a feedforward composition of functions, F_i, where:

1) Each F_i is a ***total function*** w/ ***finite implementation***
2) Each F_i ***operates asynchronously***
   Begins computing when the first input spike arrives
3) Each F_i is ***causal***
   The output spike time is independent of later input spike times
4) Each F_i is ***invariant***
   If all the input spikes are delayed by some constant amount then the output spike is delayed by the same constant amount

Temporal Invariance

- A key feature for supporting localized, asynchronous operation
- *Normalize* a volley by subtracting first spike time from all the spike times
  - A normalized volley has at least one spike at time = 0
  - Each functional block observes only normalized volleys



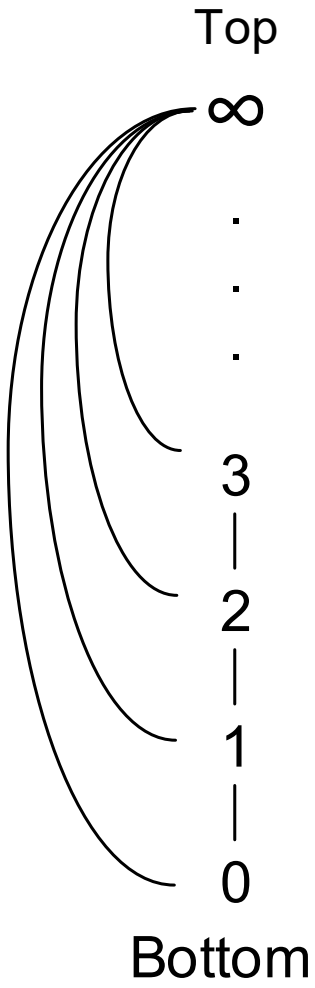
- A functional block interprets inputs and outputs according to its own local time frame

  ⇒ The network designer works with small non-negative integers

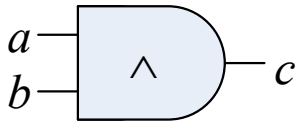
Space-Time Algebra

Bounded Distributive Lattice

- 0, 1, 2,…, ∞
- Interpretation: points in time
- *not complemented*

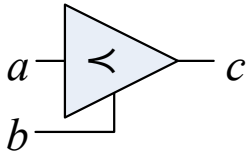


Primitive S-T Operations



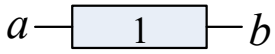
min: if $a < b$ then $c = a$
else $c = b$

"atomic excitation"



lt: if $a < b$ then $c = a$
else $c = \infty$

"atomic inhibition"

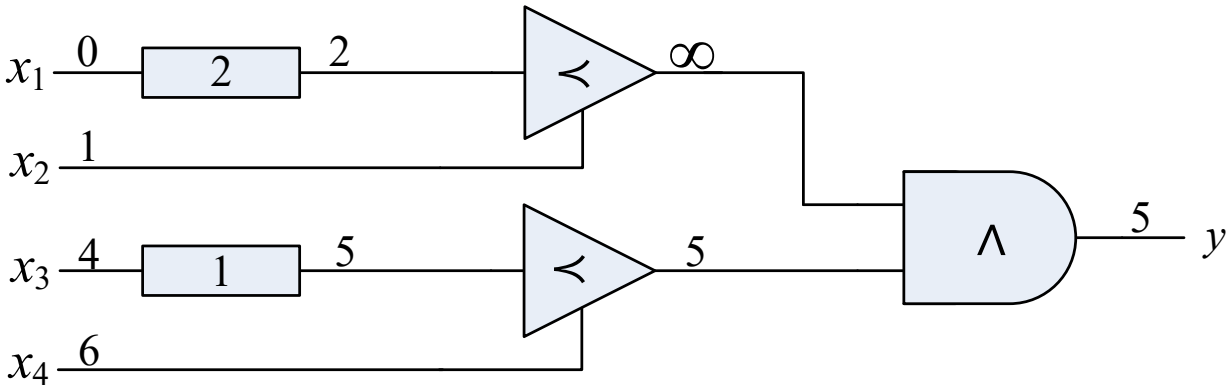


inc: $b = a + 1$

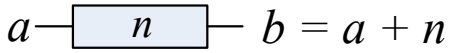
"atomic delay"

Space-Time Networks

- *Theorem*: Any feedforward composition of *s-t* functions is an *s-t* function
  - ⇒ Build networks by composing *s-t* primitives
  - Example:



note: shorthand for n increments in series:



Elementary Functions

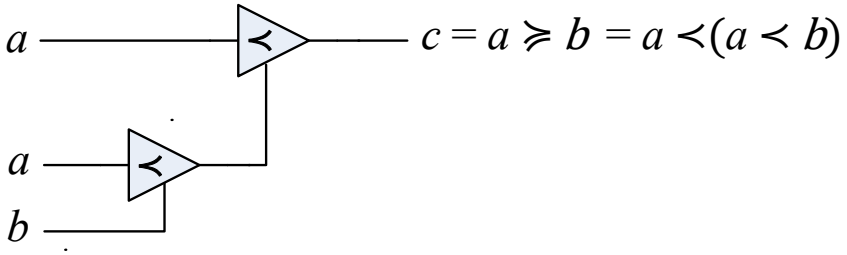
- *Increment* and *identity* are the only one-input functions
- Table of all two-input *s-t* functions
  - Many are non-commutative

function	name	symbol
if $a < b$ then a; else b	min	$\wedge$
if $a > b$ then a; else b	max	$\vee$
if $a < b$ then a; else ∞	less than	$\prec$
if $a \leq b$ then a; else ∞	less or equal	$\preccurlyeq$
if $a > b$ then a; else ∞	greater than	$\succ$
if $a \geq b$ then a; else ∞	greater or equal	$\succcurlyeq$
if $a = b$ then a; else ∞	equal	$\equiv$
if $a \neq b$ then a; else ∞	not equal	$\not\equiv$
if $a < b$ then a else if $b < a$ then b; else ∞	exclusive min	x$\wedge$
if $a > b$ then a else if $b > a$ then b; else ∞	exclusive max	x$\vee$

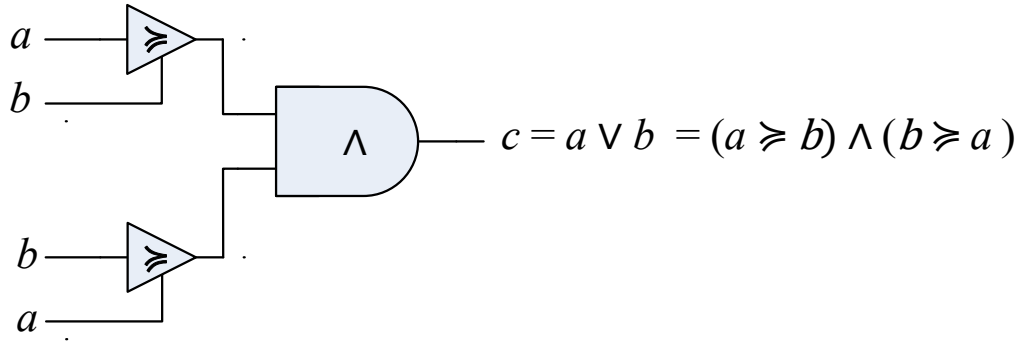
Implementing Some Elementary Functions

greater or equal, using *lt*

if $a \geq b$ then $c = a$
else $c = \infty$

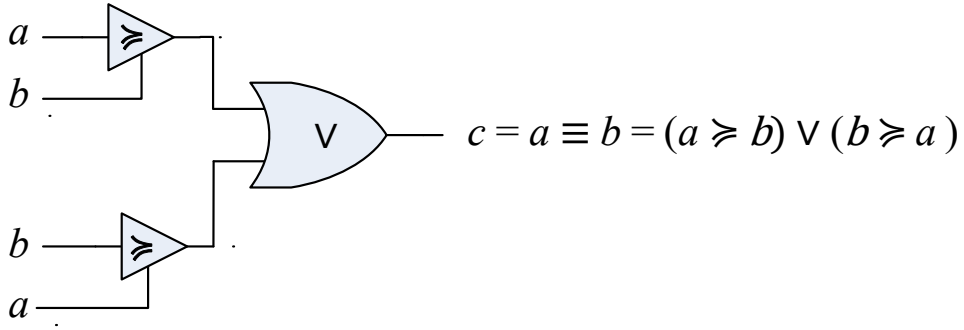


max using *ge* and *min*



equals using *ge* and *max*

if $a = b$ then $c = a$
else $c = \infty$

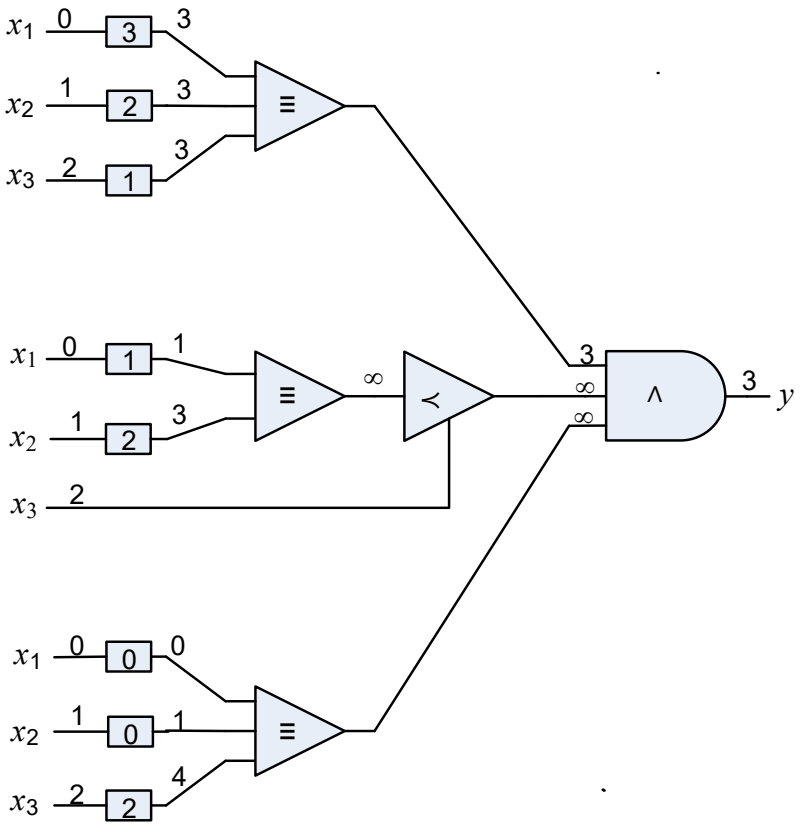


ack: Cristóbal Camarero

Completeness

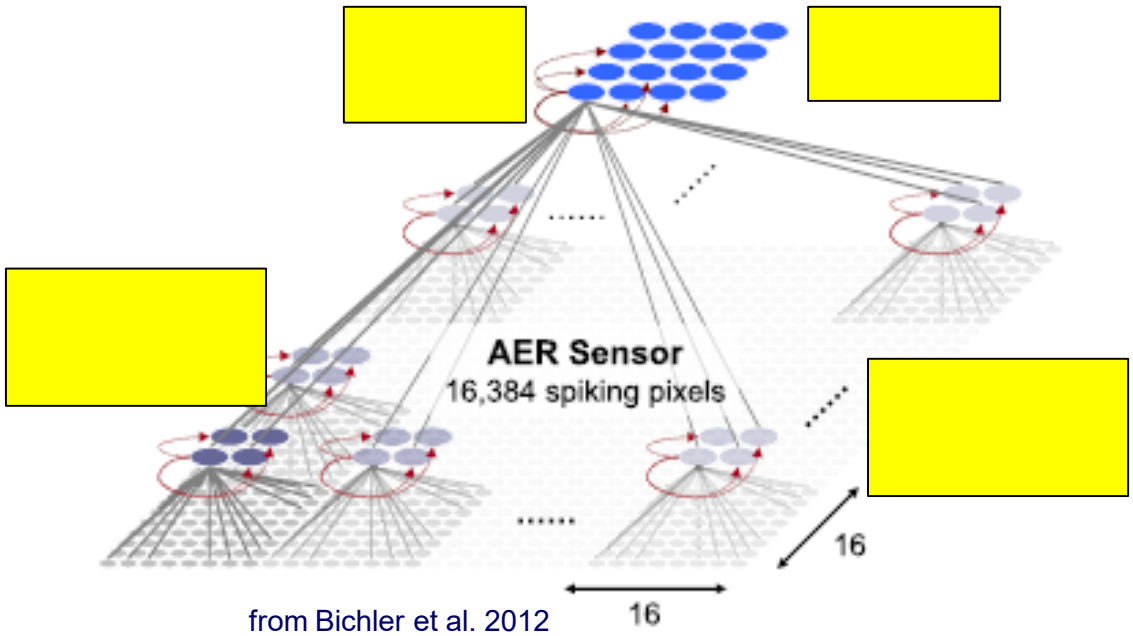
- *Theorem*: *min*, *inc*, and *lt* are functionally complete for the set of *s-t* functions
- *Proof:* by construction of normalized function table
  - *Equals* implemented w/ *min* and *lt*
  - *Example:* table with 3 entries

x_1	x_2	x_3	y
0	1	2	3
1	0	∞	2
2	2	0	2



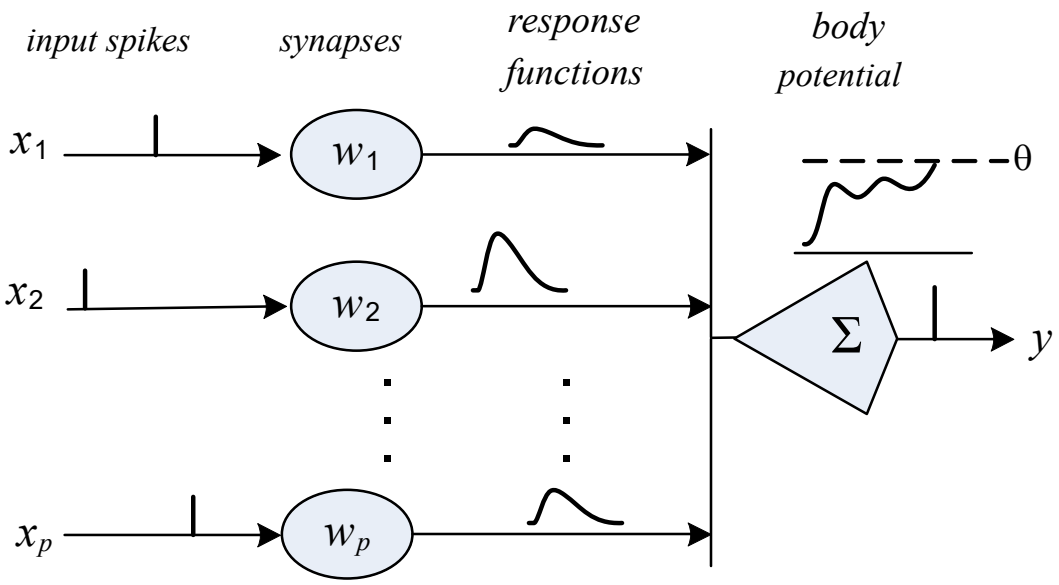
Implement TNNs With S-T Primitives

- *Corollary*: All TNNs can be implemented with *min*, *inc*, and *lt*
  - Theorem/Corollary say we can do it… so let's construct basic TNN elements
  - Focus on excitatory neurons and lateral (winner-take-all) inhibition



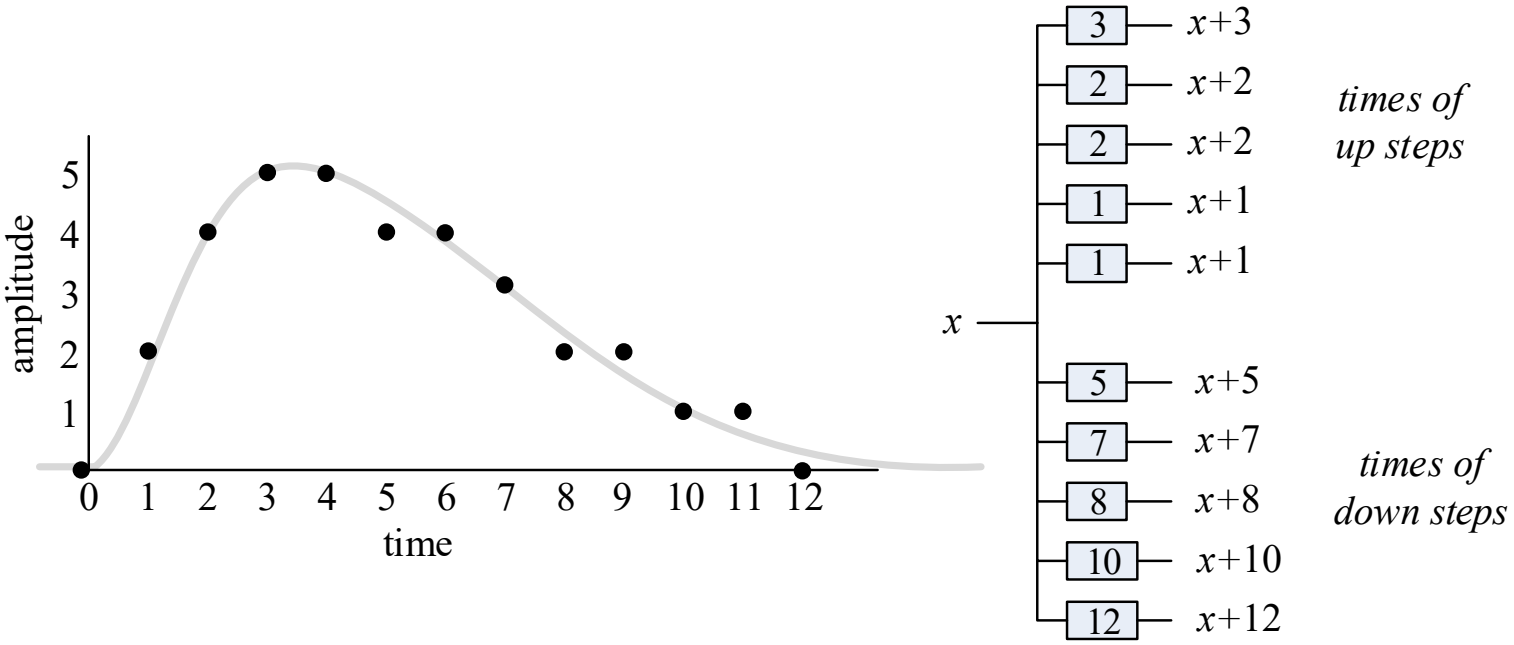
from Bichler et al. 2012

Quick Review: SRM0 Neuron Model



SRM0 Neurons

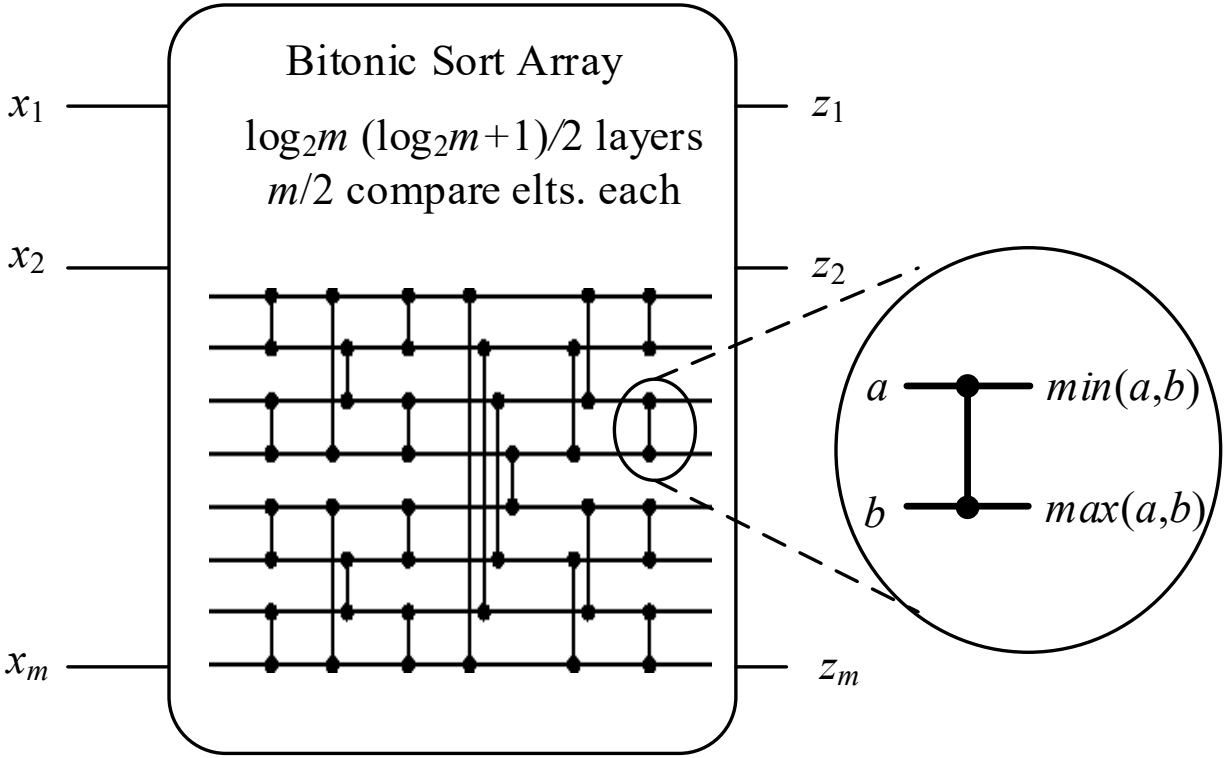
- Response Functions
  - Can be specified as times of *up* and *down* steps



This not a toy example – it is realistically low resolution

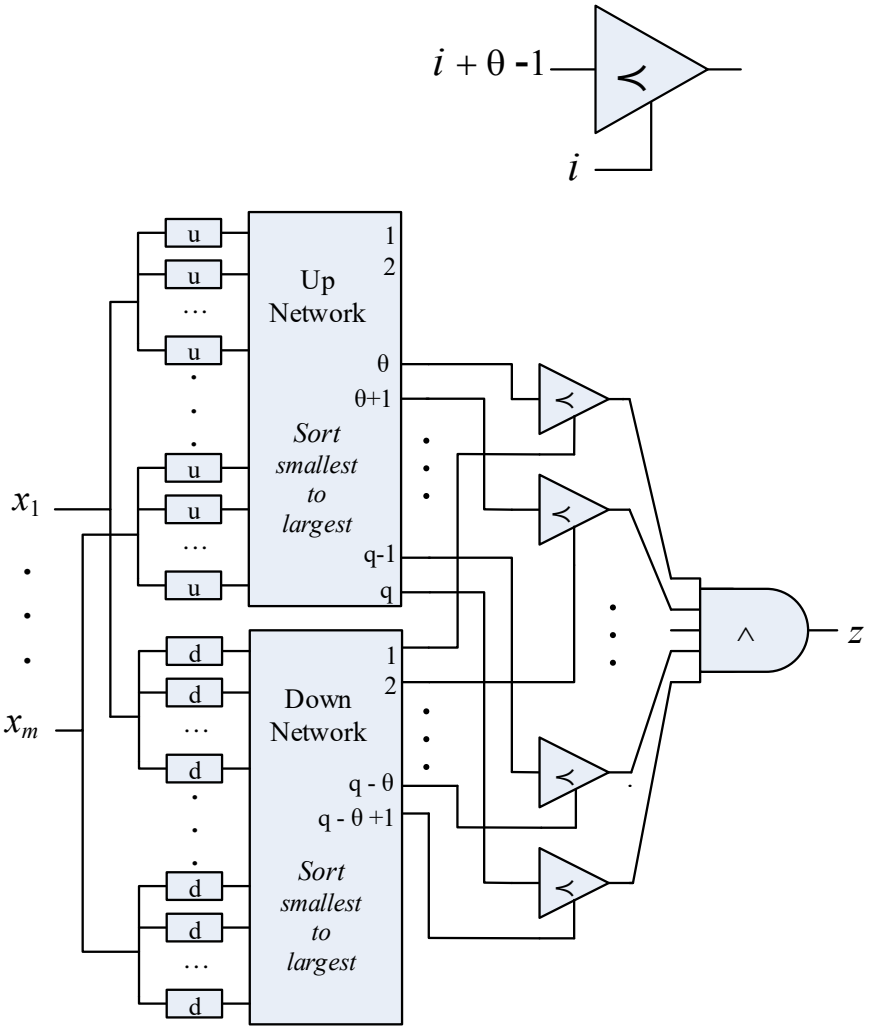
Bitonic Sort (Batcher 1968)

- A key component of the SRM0 construction
- A Bitonic Sorting Network is a composition of *max*/*min* elements
- *max/min* are *s-t* functions, so *Sort* is an *s-t* function



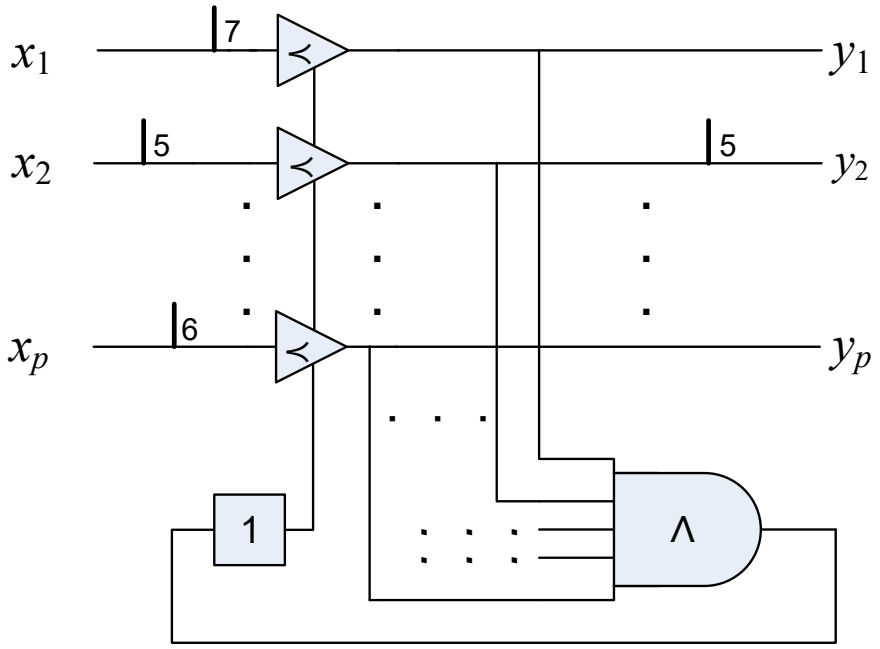
Complete SRM0 neuron

- Up Network: sort times of all *up* steps
- Down Network: sort times of all *down* steps
- $\prec$ & *min* tree determines the first time *no. of up steps* $\geq$ *no. of down steps* + θ
  - This is the first time the threshold is reached
  - Note: *error in paper*; $q - \theta+1$, *not* $q - (\theta+1)$
- Supports *all possible* finite response functions
  - I.e., all response functions that can be physically implemented
  - Includes inhibitory (negative) response functions



Lateral Inhibition

- Inhibitory neurons modeled *en masse*
- Winner-Take-All (WTA)
  - Only the first spike in a volley is allowed to pass from inputs to outputs
  - If there is a tie for first, all first spikes pass

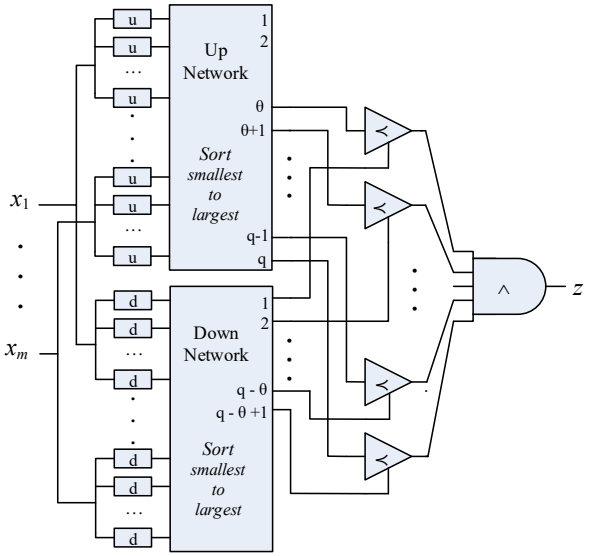


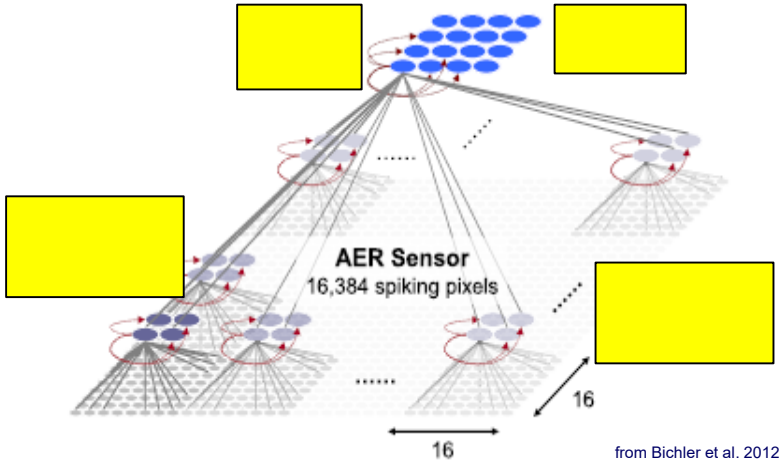
Aside: this is an example of physical feedback, but no functional feedback

TNN Summary

- We have implemented the primary TNN components as s-t functions

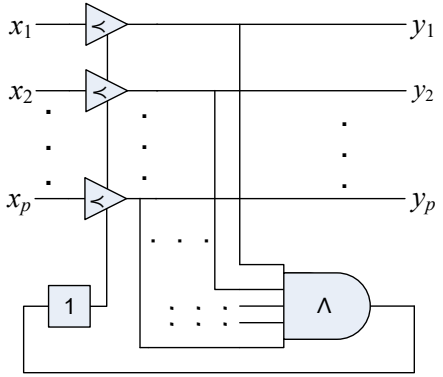
Neurons





from Bichler et al. 2012

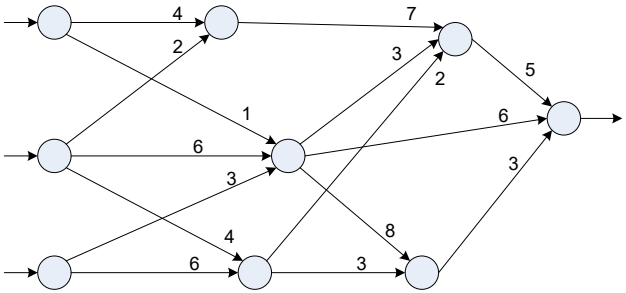
Lateral Inhibition

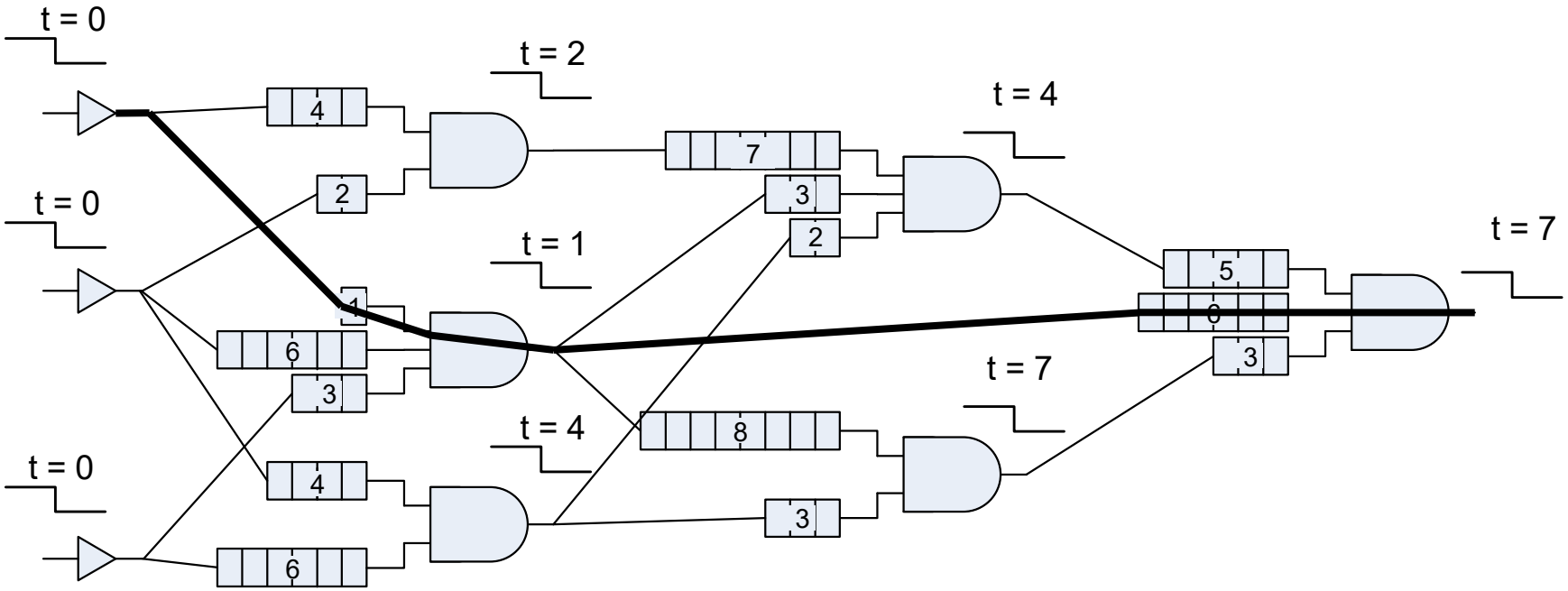


Race Logic: A Direct Implementation

Race Logic: Shortest Path

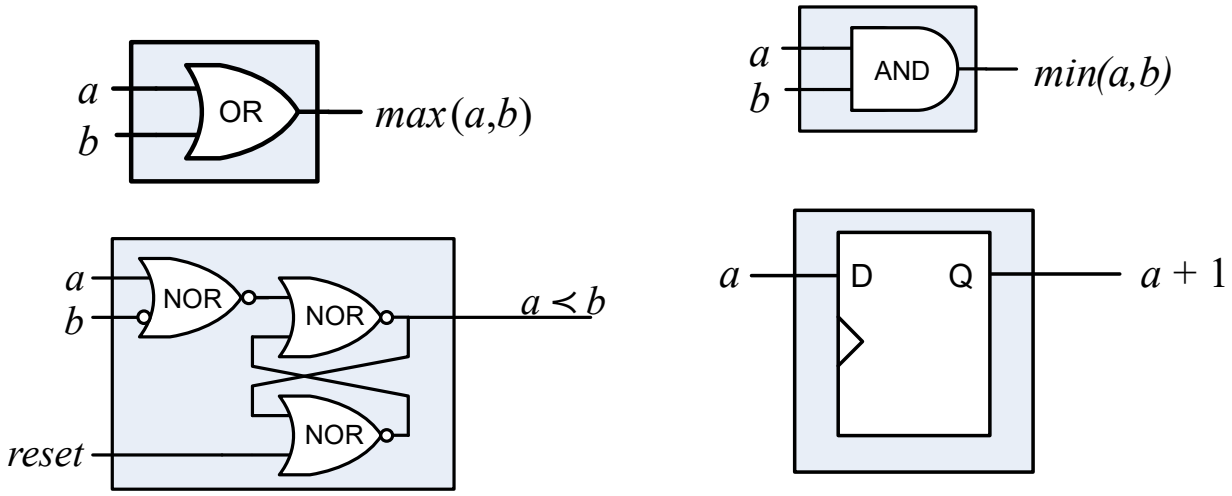
- By Madhavan, Sherwood, Strukov (2015)
- Find shortest path in a weighted directed graph
  - *Weights* implemented as shift register delays
  - *Min* functions implemented as AND gates





Generalized Race Logic

- *S-T* primitives implemented directly with conventional digital circuits
  - Signal via 1 → 0 transitions rather than spikes



⇒ We can implement SRM0 neurons and WTA inhibition with off-the-shelf CMOS

⇒Very fast and efficient TNNs

Construct TNN architectures in the neuroscience domain
Implement directly with off-the-shelf CMOS

Concluding Remarks

The Temporal Resource

The *flow of time* can be used effectively as a *communication* and *computation resource.*

- The *flow of time* has huge engineering advantages
  - It is free – time flows whether we want it to or not
  - It requires no space
  - It consumes no energy
- Yet, we (humans) try to eliminate the effects of time when constructing computer systems
  - Assume worst case delays (w/ synchronizing clocks) & delay-independent asynchronous circuits
  - *This may be the best choice for conventional computing problems and technologies*
- How about natural evolution?
  - Tackles completely different set of computing problems
  - With a completely different technology

The Box: The way we (humans) perform computation

- We try to eliminate temporal effects when implementing functions
  - *s-t* uses the uniform flow of time as a key resource
- We use *add* and *mult* as primitives for almost all mathematical models
  - Neither *add* nor *mult* is an *s-t* function
- We prefer high resolution (precision) data representations
  - *s-t* practical only for very low-res direct implementations
- We strive for complete functional completeness
  - *s-t* primitives complete *only* for *s-t* functions
  - There is no inversion, complementation, or negation
- We strive for algorithmic, top-down design
  - TNNs are based on *implied functions*
  - Bottom-up, empirically-driven design methodology

A Remarkable Conjecture

- Addition and Multiplication are not space-time functions
  - They fail invariance:

$$(a + 1) + (b + 1) \neq (a + b) + 1$$

$$(a + 1) \cdot (b + 1) \neq (a \cdot b) + 1$$

- Conventional machine learning models depend on addition and multiplication as primitives
  - Vector inner product: (inputs · weights)

IF the brain's cognitive functions *depend* on *space-time* operations

Then: conventional machine learning, on its current trajectory, will never implement the brain's cognitive functions

s-t algebra is more than a different set of primitives –
it describes a different set of functions

Acknowledgements

Raquel Smith, Mikko Lipasti, Mark Hill, Margaret Martonosi, Cristobal Camarero, Mario Nemirovsky, Mikko Lipasti, Tim Sherwood, Advait Madhavan, Shlomo Weiss, Ido Guy, Ravi Nair, Joel Emer, Quinn Jacobson, Abhishek Bhattacharjee