Task (error-driven) Learning

- Last time we discussed **self organizing Hebbian learning**
  - Leverage correlations to grow detectors that correspond to things in the world (cats, professors…)
- Today we will discuss **task learning**
  - Task = producing a specific output pattern in response to an input pattern
  - e.g., reading; giving the correct answer to 3 + 3

Task Learning

- Task learning encompasses:
  - Giving an appropriate **response** to a **stimulus**
  - Arriving at an accurate **interpretation** of a **situation**
  - Generating a correct **expectation** of what will happen next
- in all of the above cases, there is a **correct answer**...

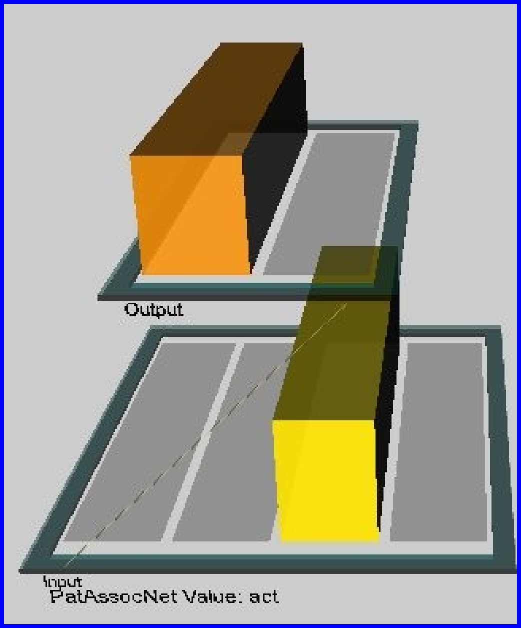
Overview

- How well can Hebbian rules support task learning?
- Not well enough! There are some input-output mappings that Hebb can not learn
- Error-correction learning and the delta rule
- Shortcomings of two-layer delta rule networks
- GeneRec: A biologically plausible error-driven learning rule for multilayer networks

Hebbian Task Learning

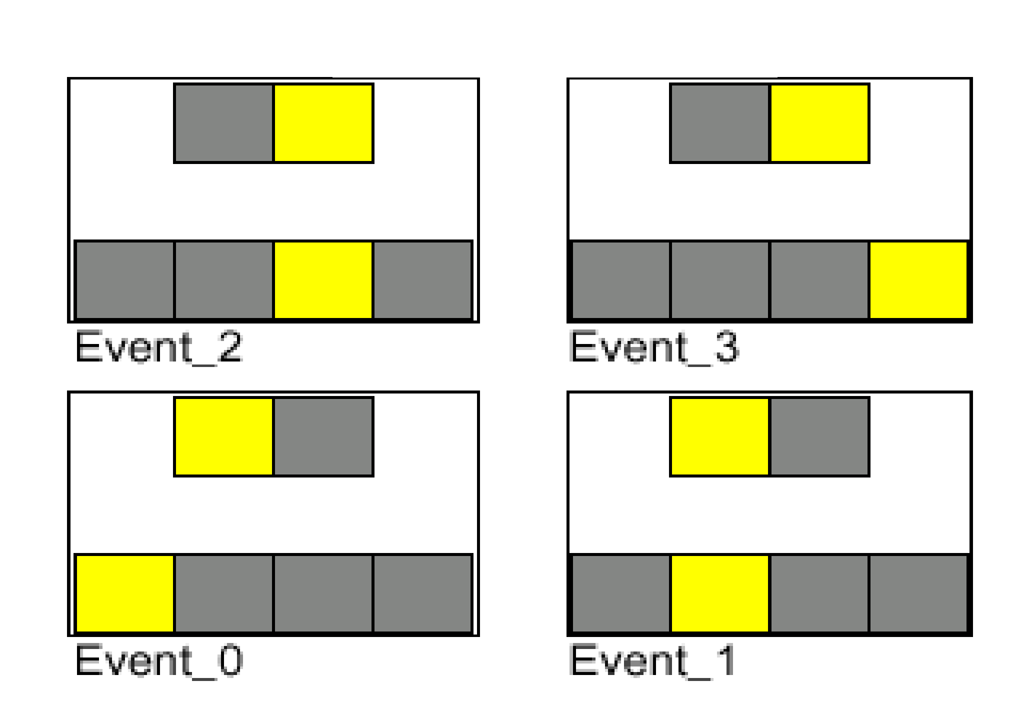
- If you want to learn an input-output association:

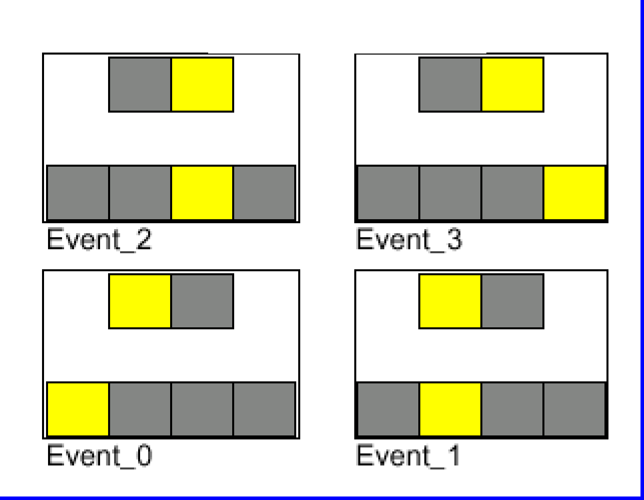
  - **clamp** the input pattern onto the input layer
  - clamp the output pattern onto the output layer
  - do Hebbian learning



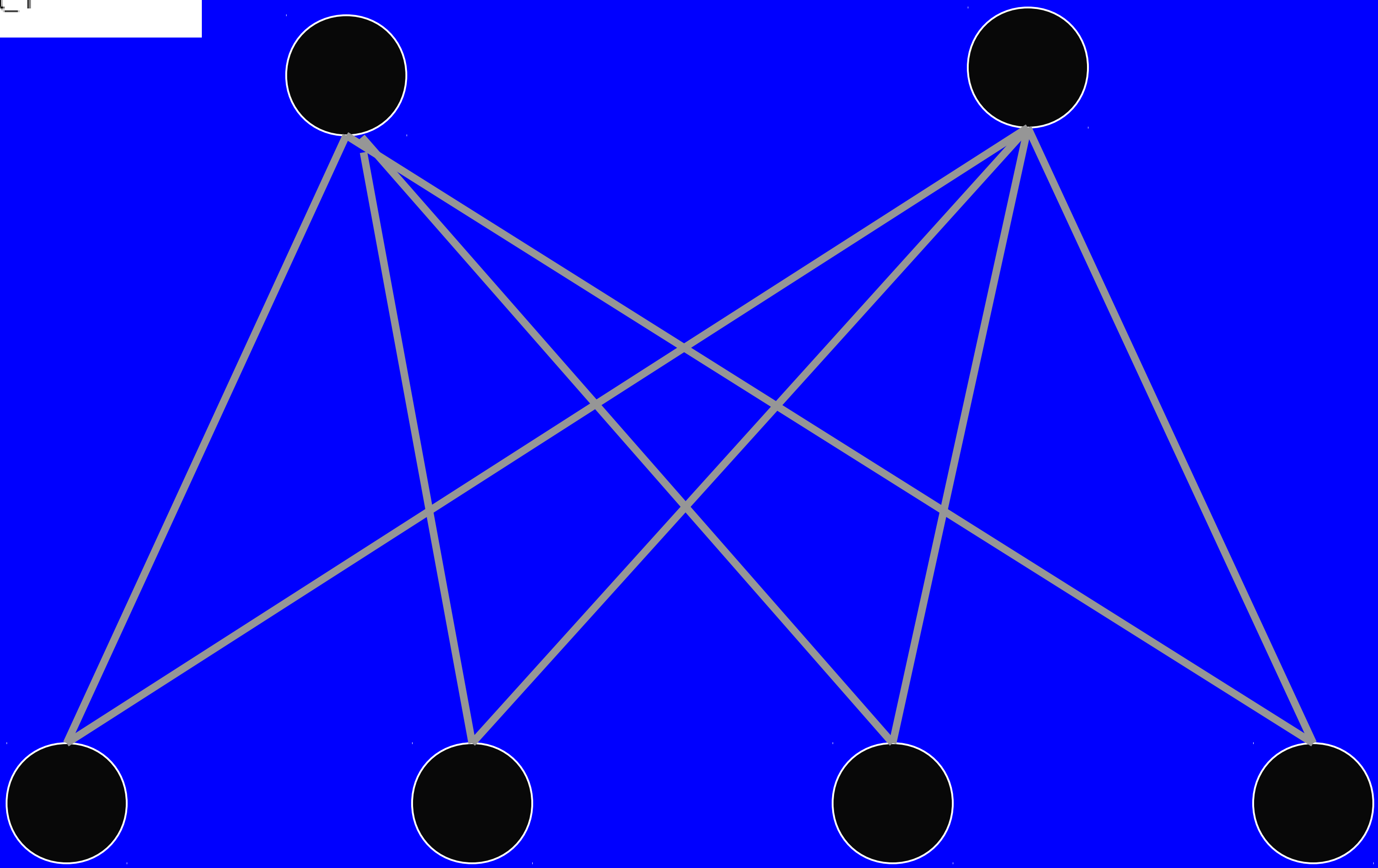
“Easy” Mapping

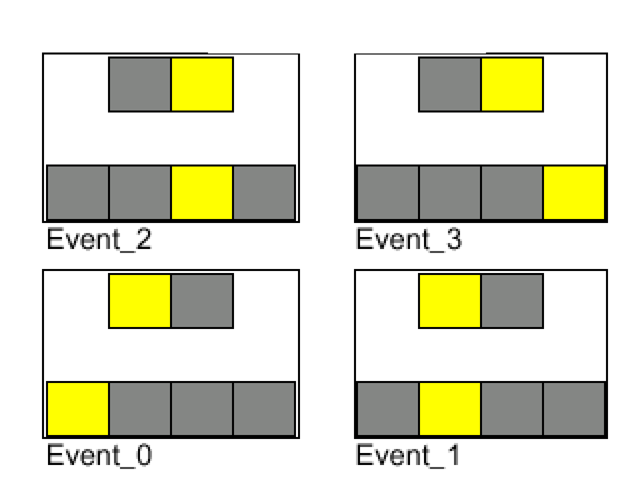
- no overlap between inputs



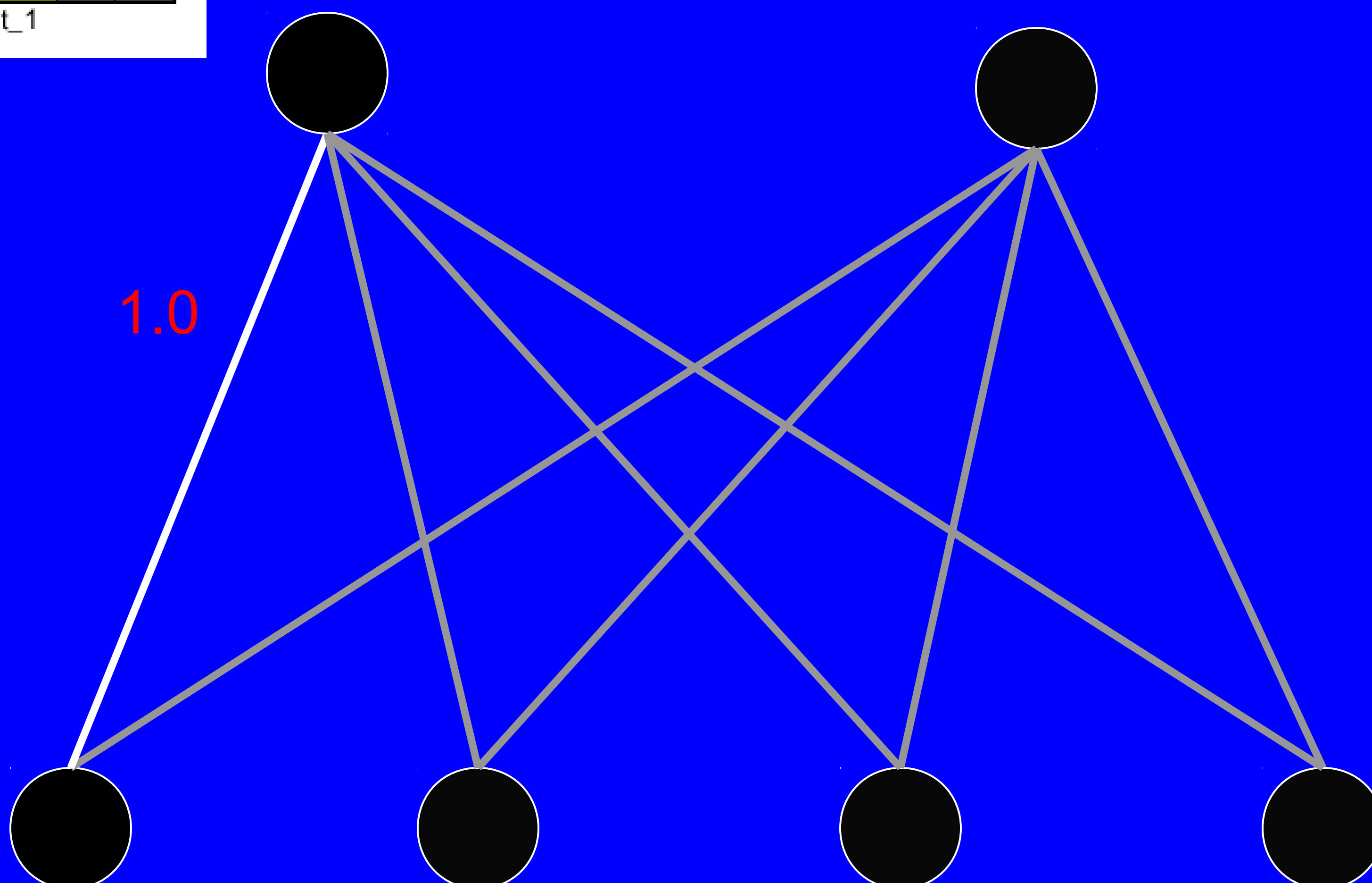


Hebbian learning:
weight ~ P(receiver active | sender active)

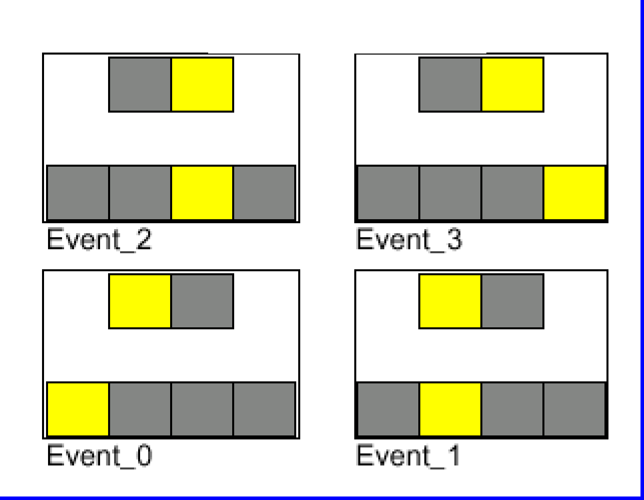


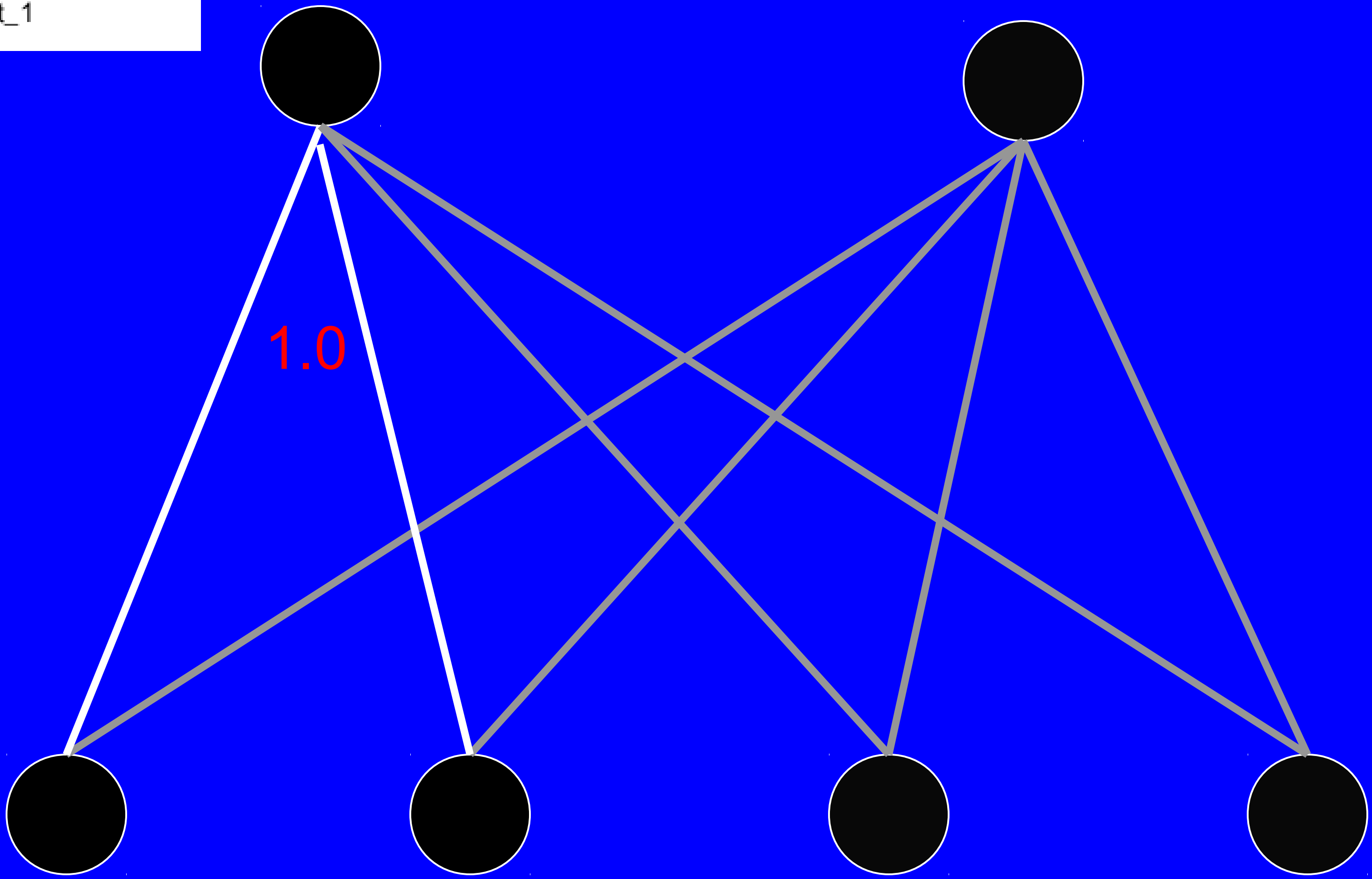


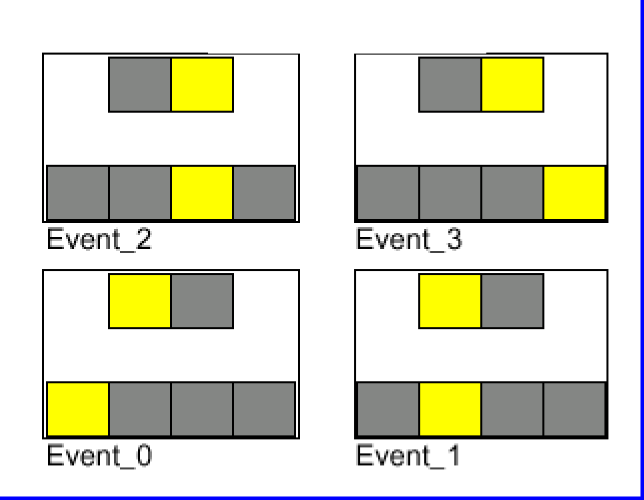
Hebbian learning:
weight ~ P(receiver active | sender active)



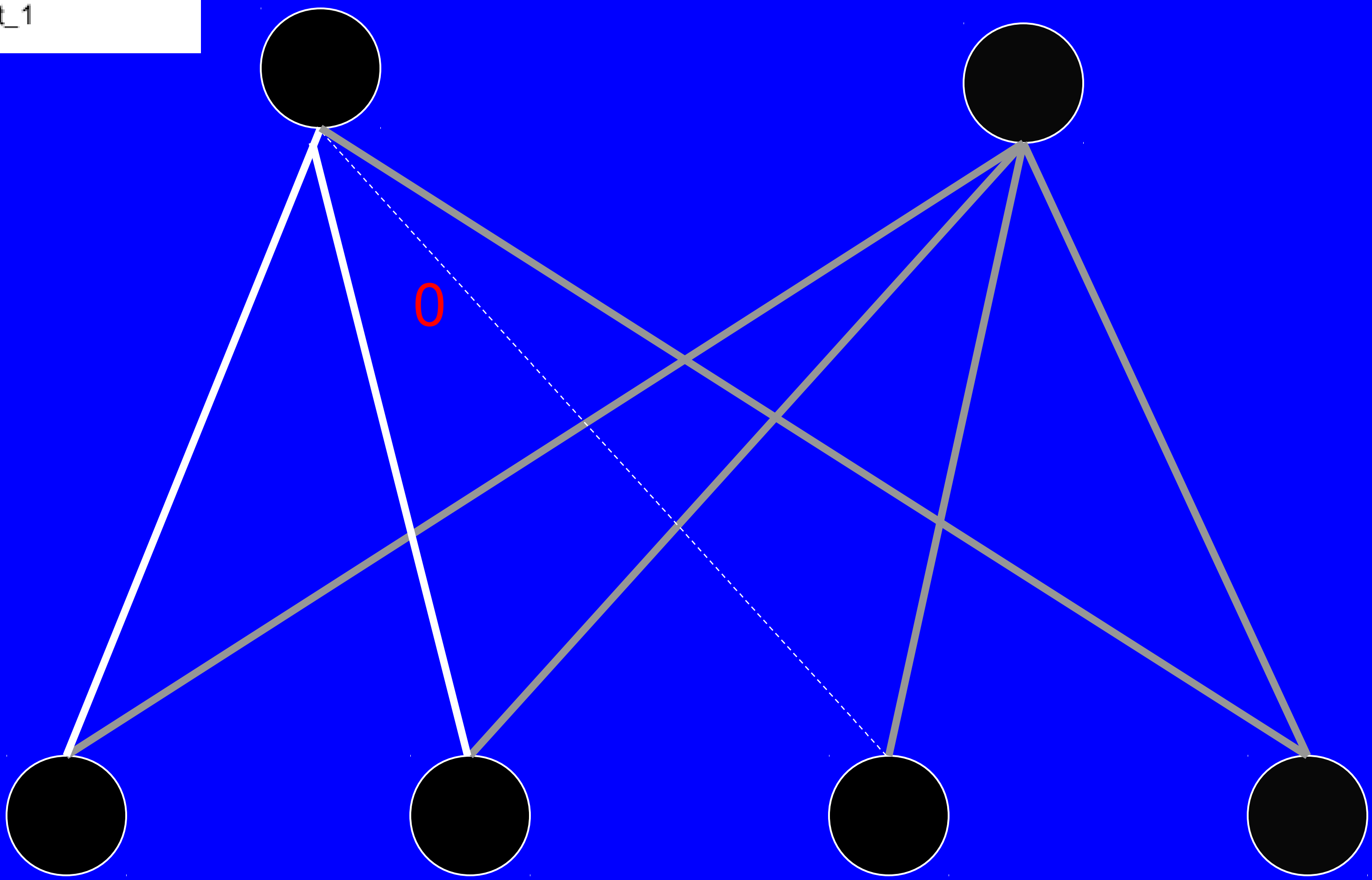
Hebbian learning:
weight ~ P(receiver active | sender active)

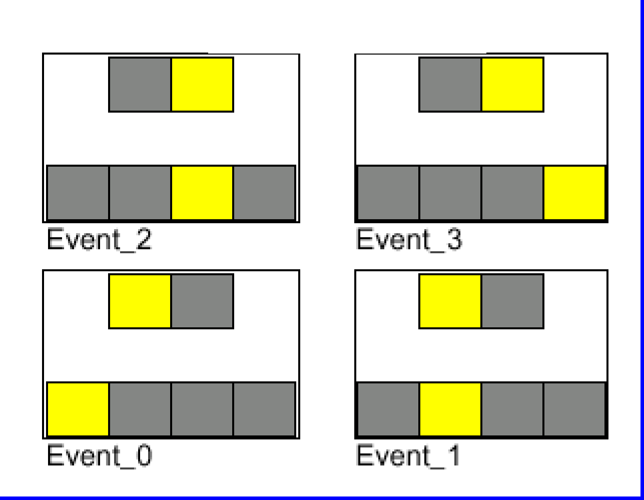




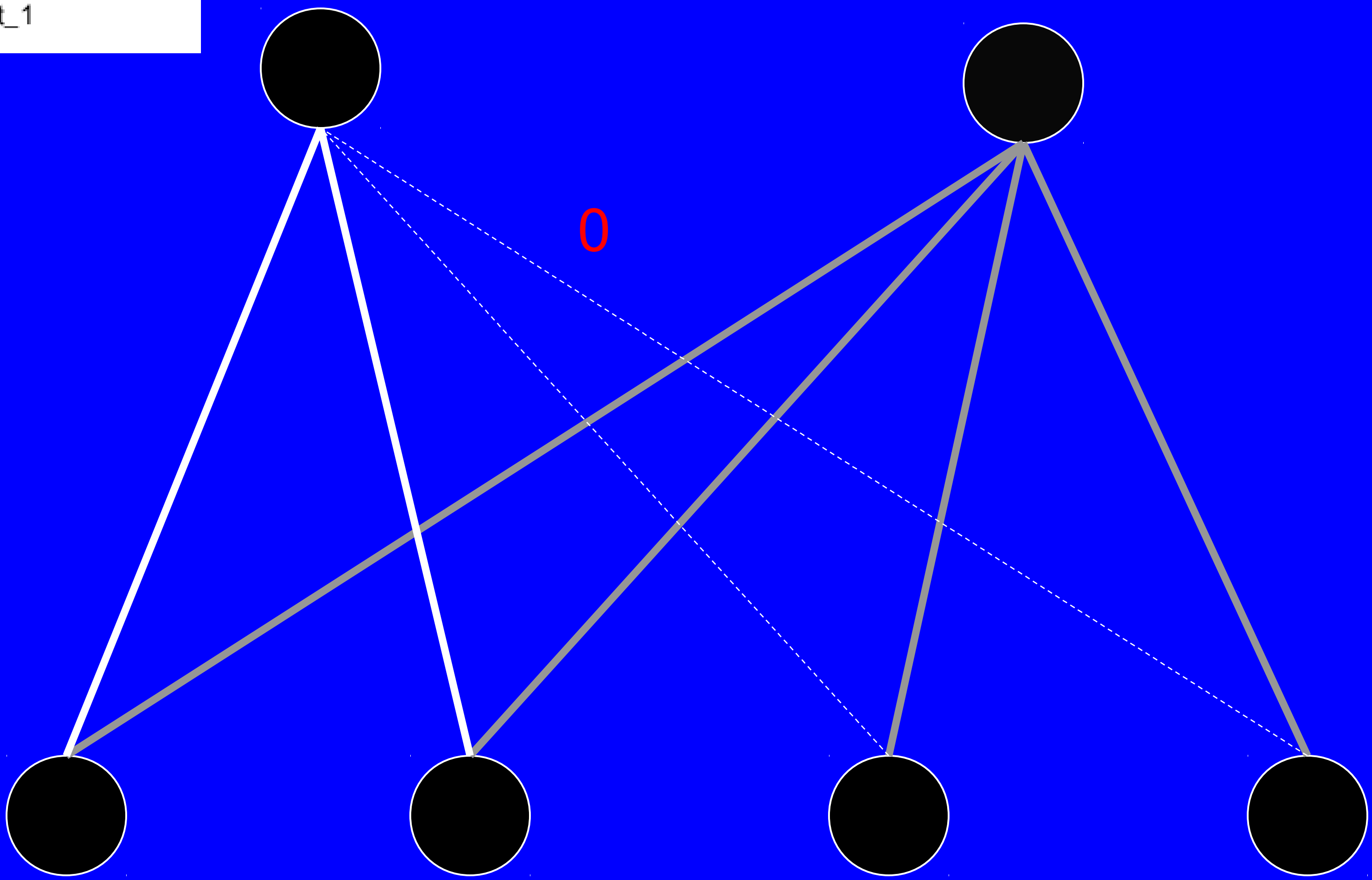


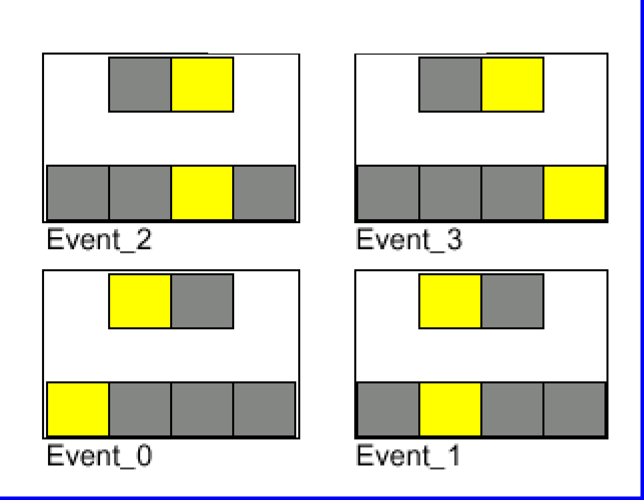
Hebbian learning:
weight ~ P(receiver active | sender active)



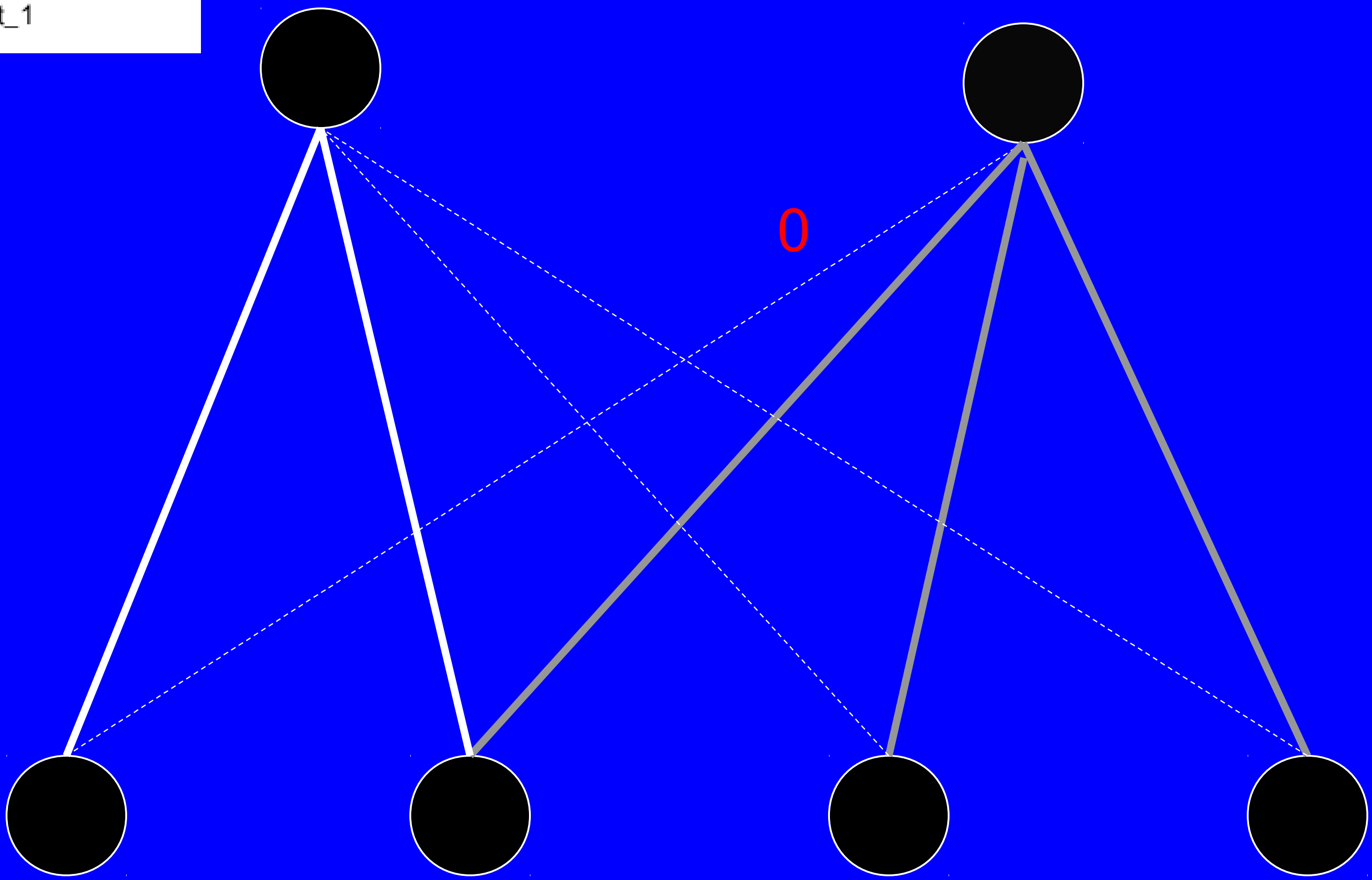


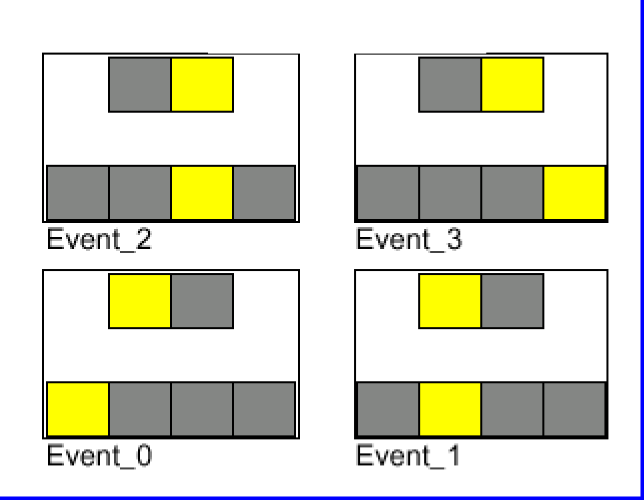
Hebbian learning:
weight ~ P(receiver active | sender active)



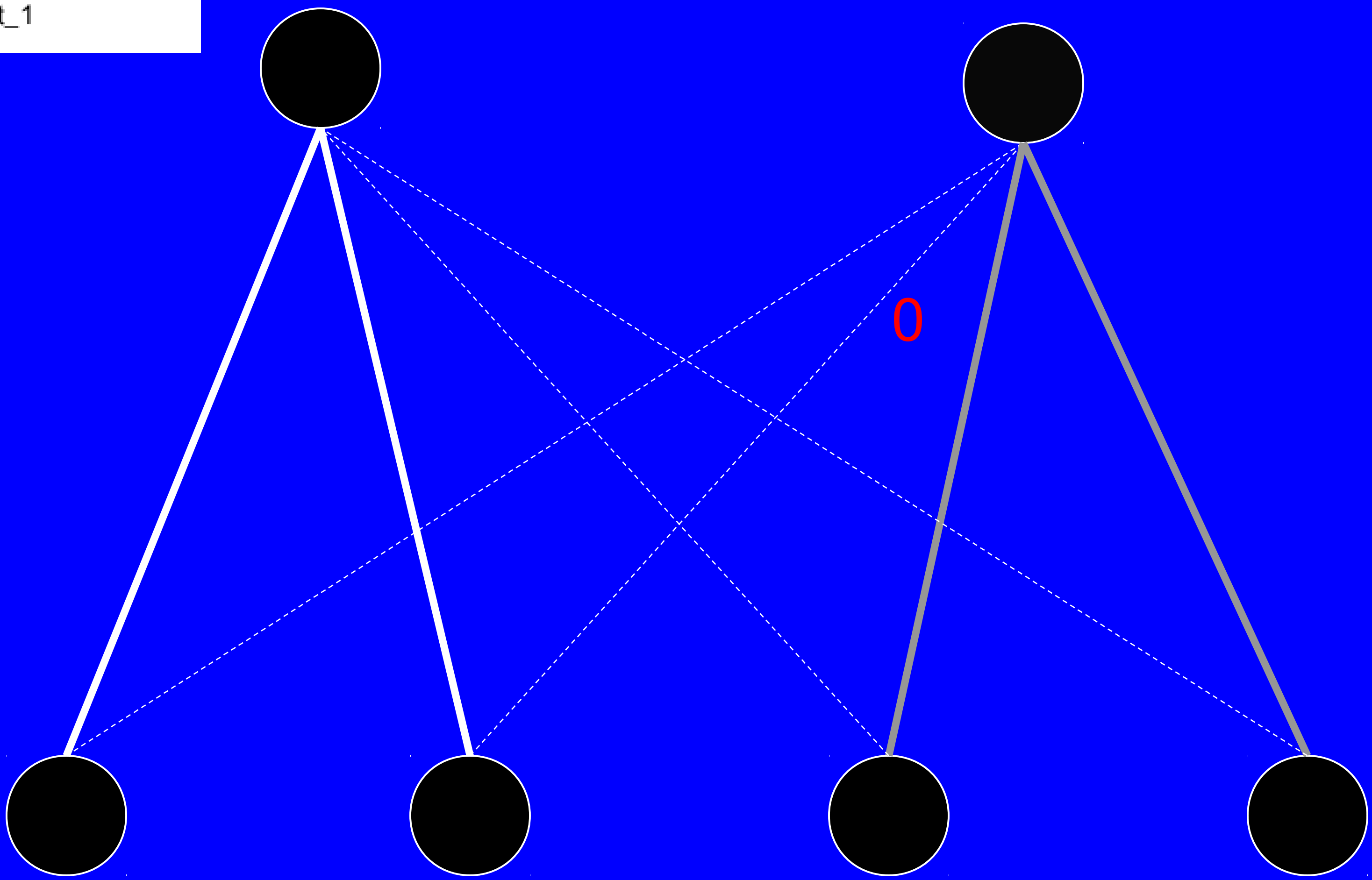


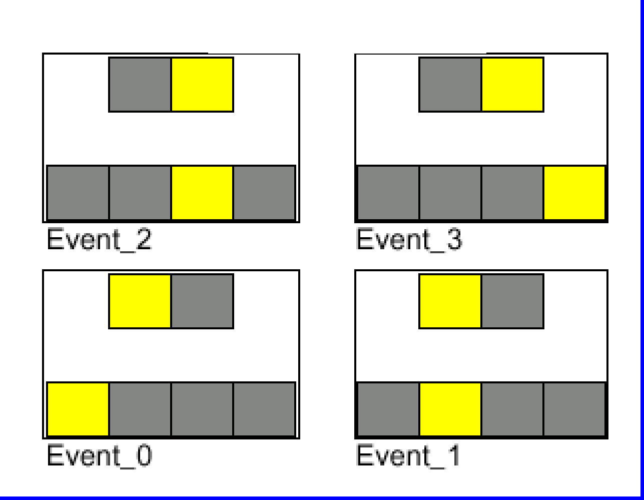
Hebbian learning:
weight ~ P(receiver active | sender active)



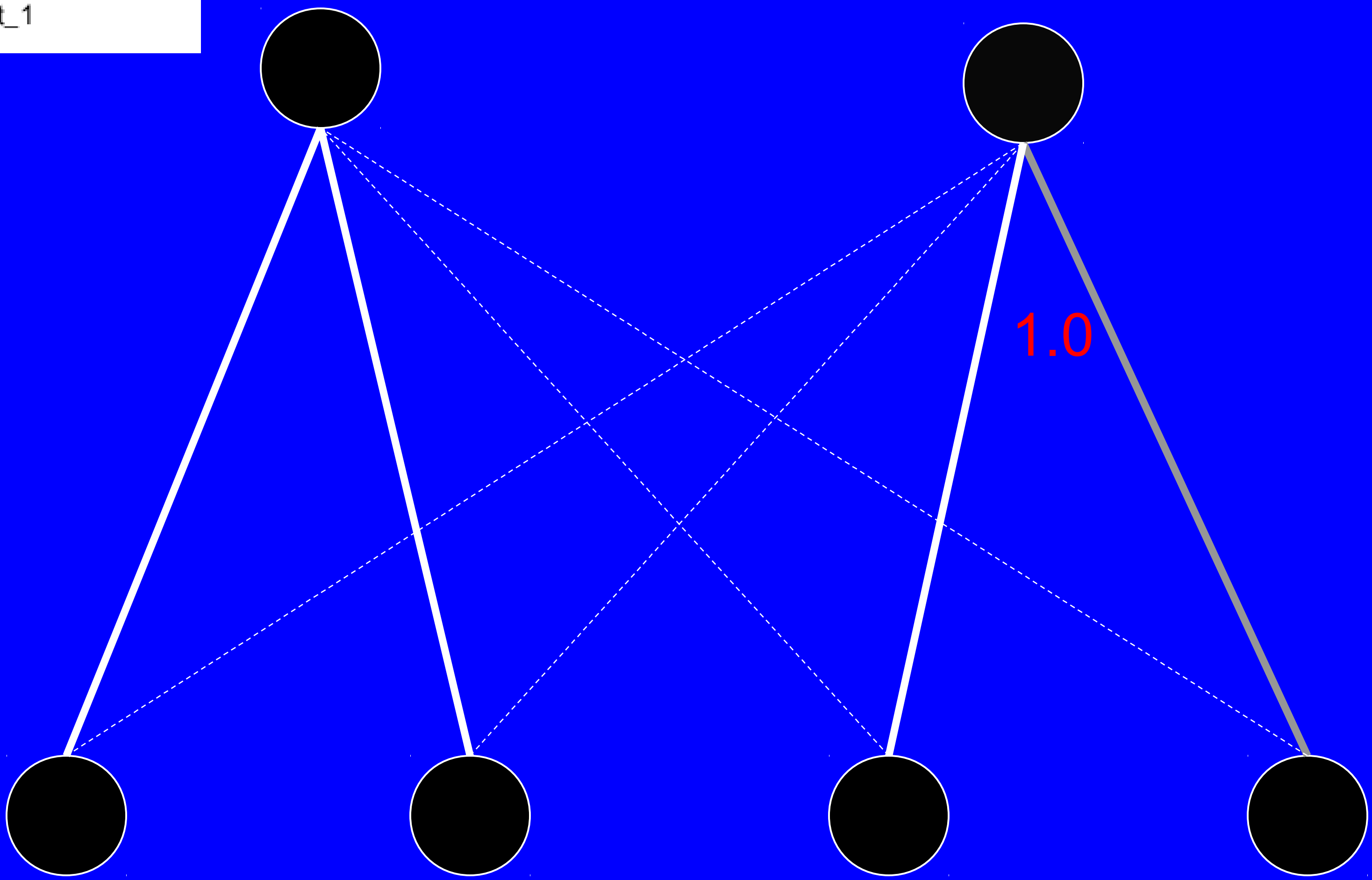


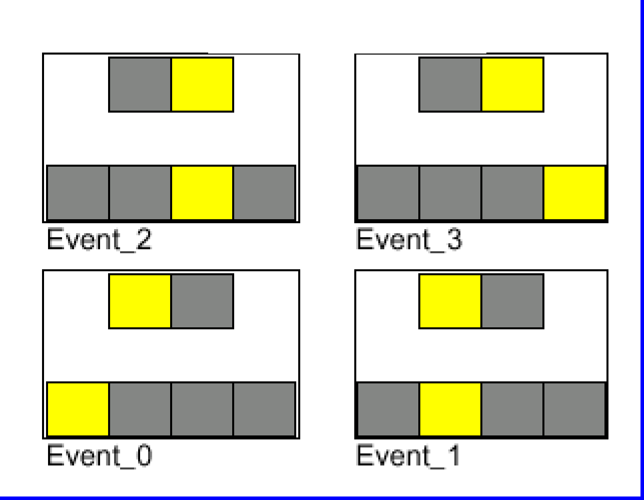
Hebbian learning:
weight ~ P(receiver active | sender active)



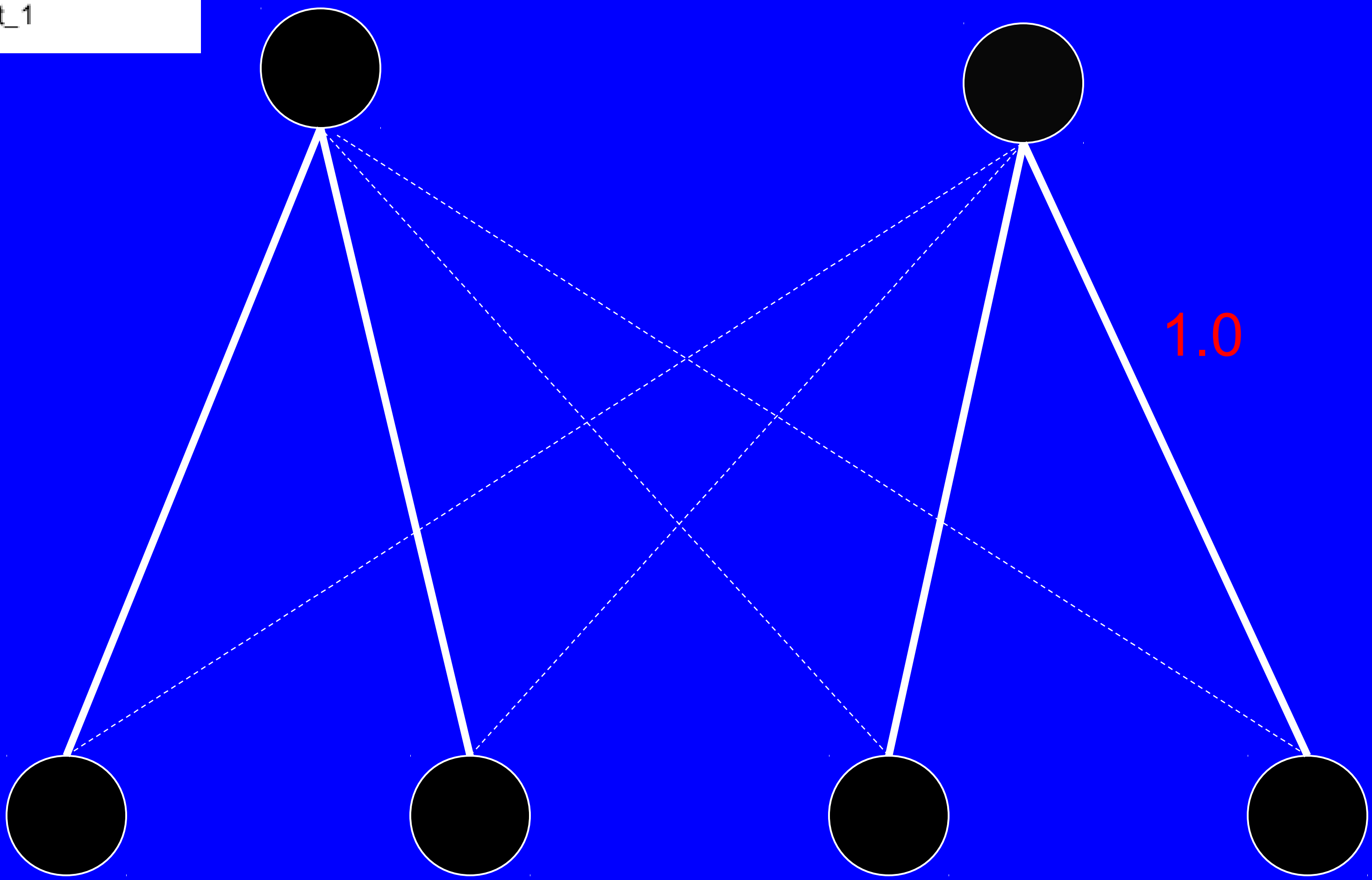


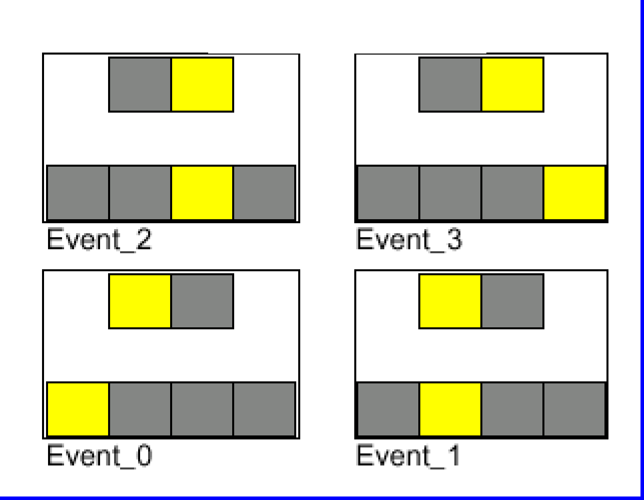
Hebbian learning:
weight ~ P(receiver active | sender active)



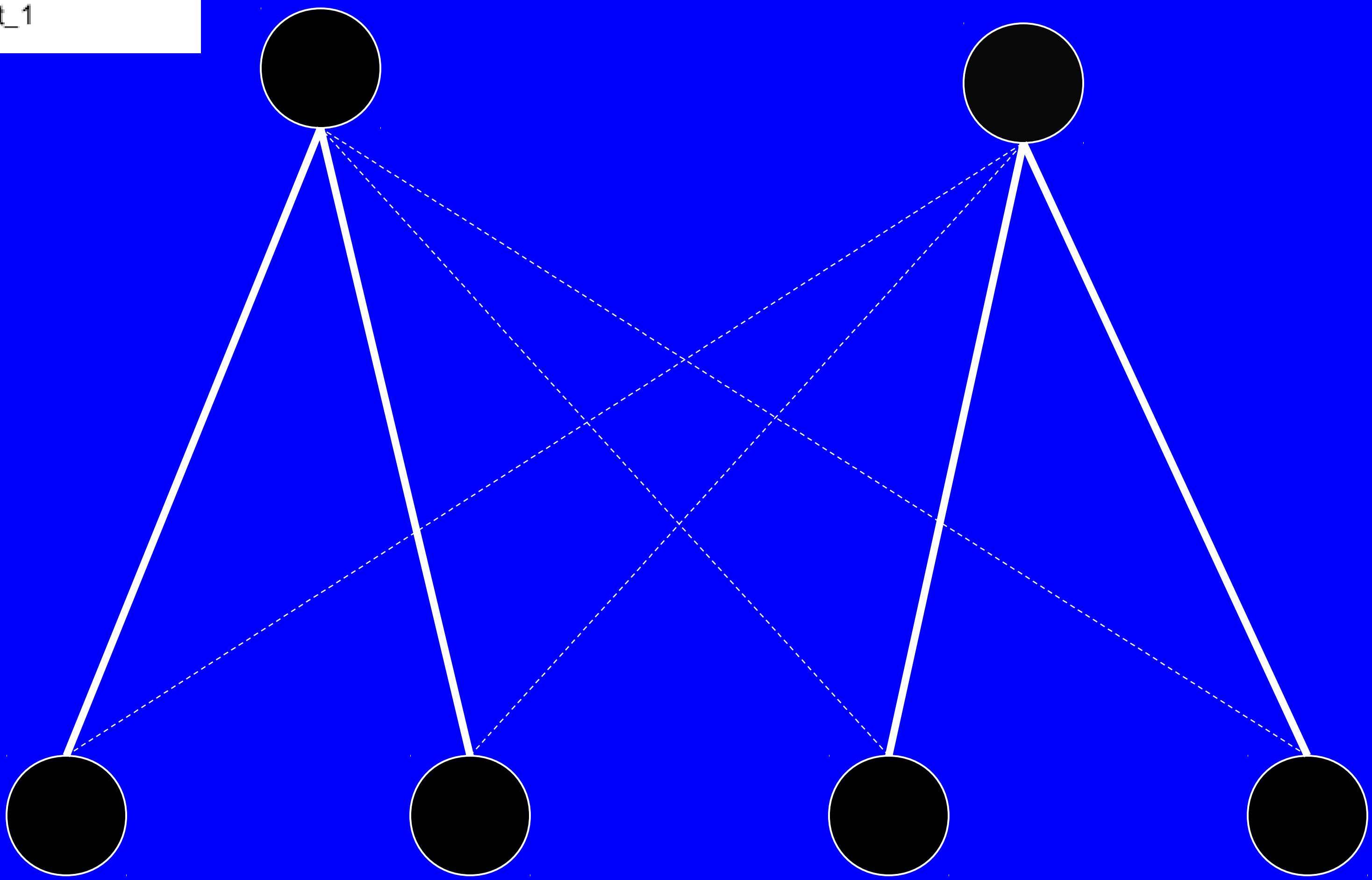


Hebbian learning:
weight ~ P(receiver active | sender active)



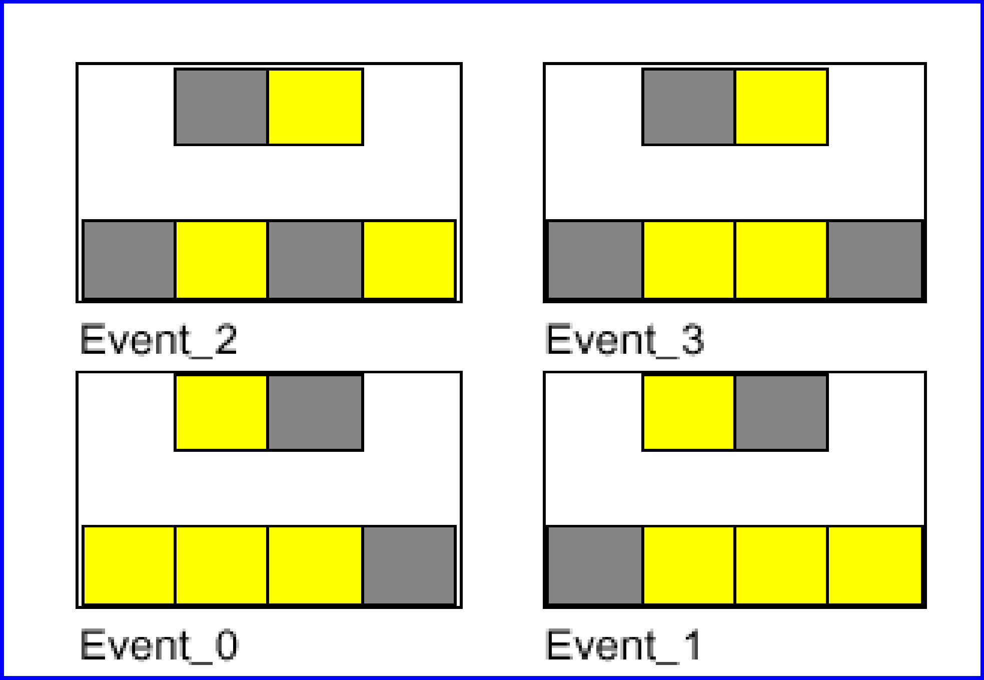


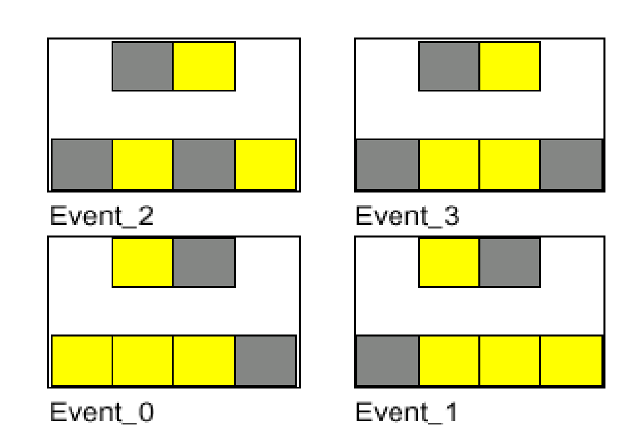
Hebb can solve the task!



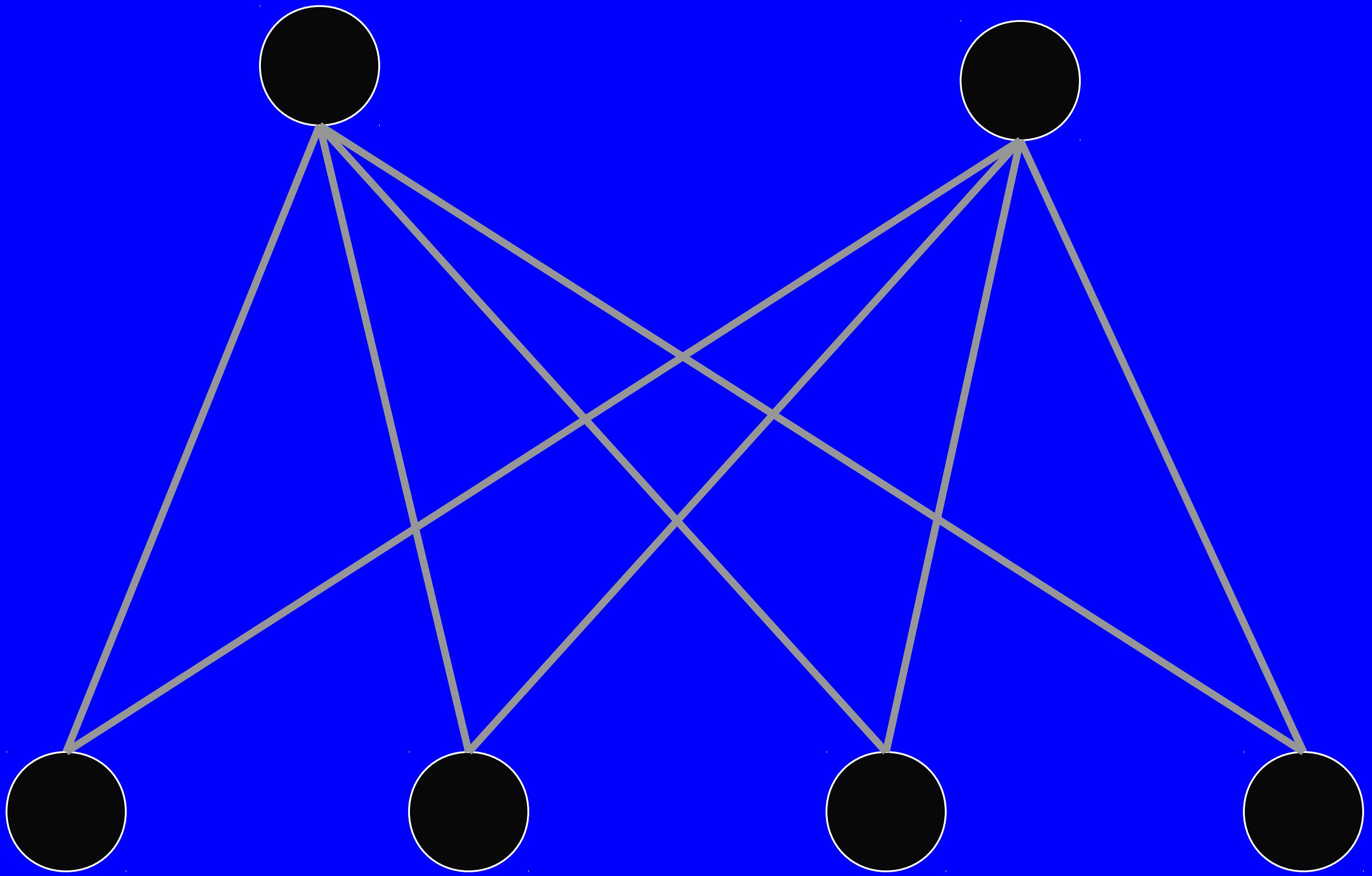
Another (Harder) Mapping

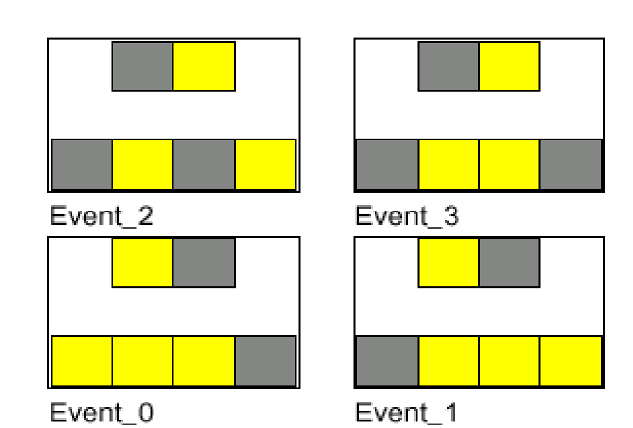
- overlap between inputs
- input units associated with multiple outputs



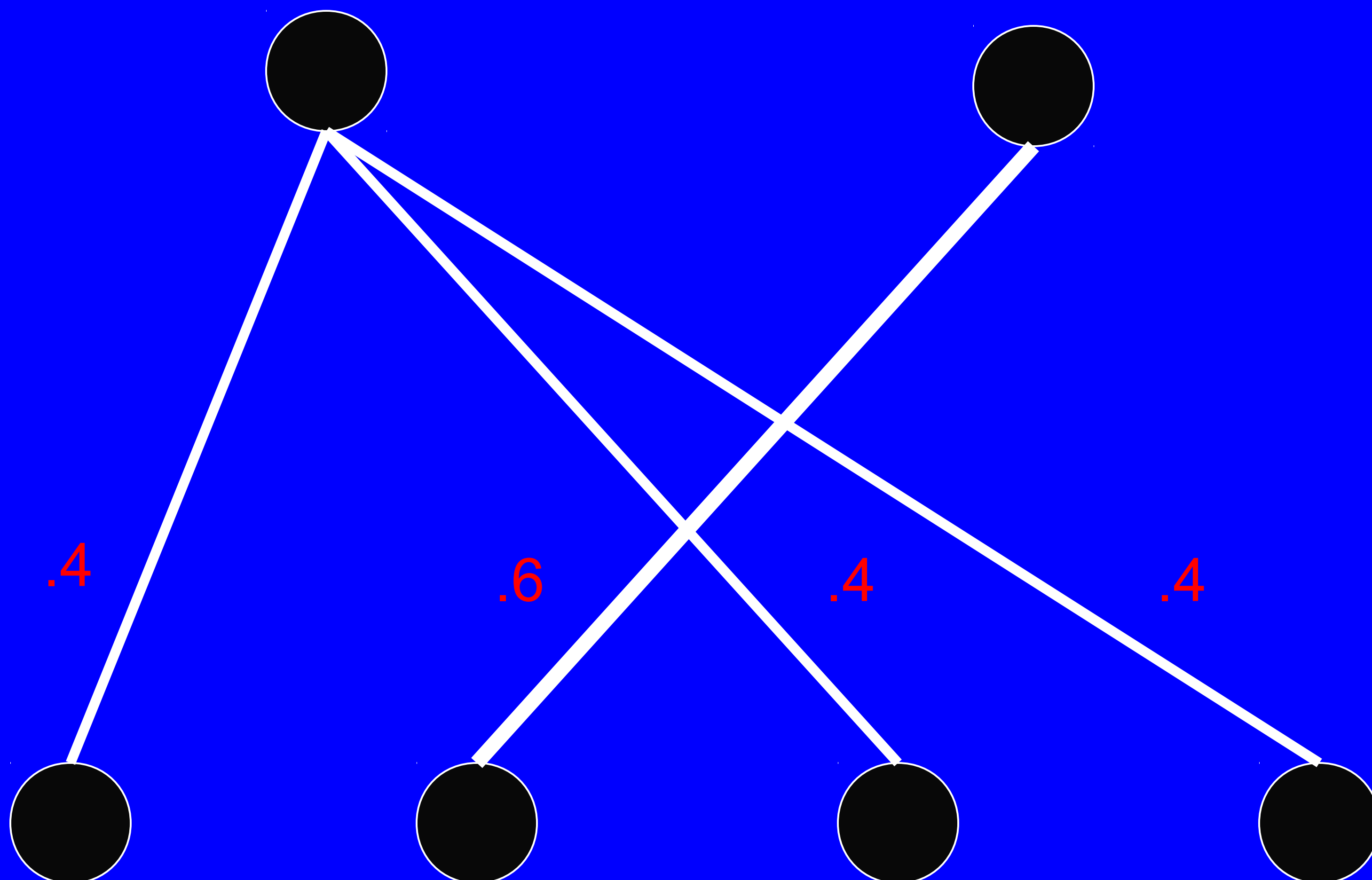


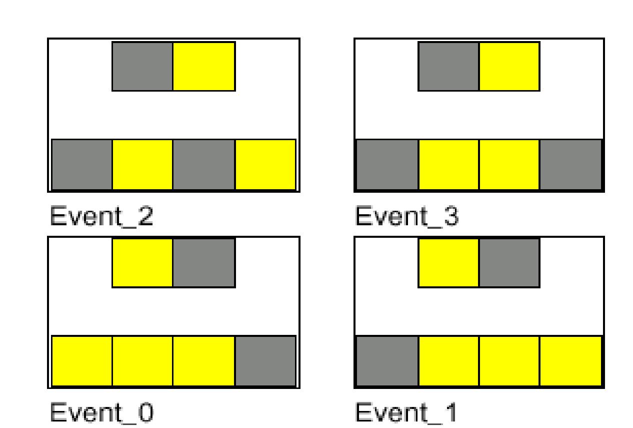
The mapping is solvable!



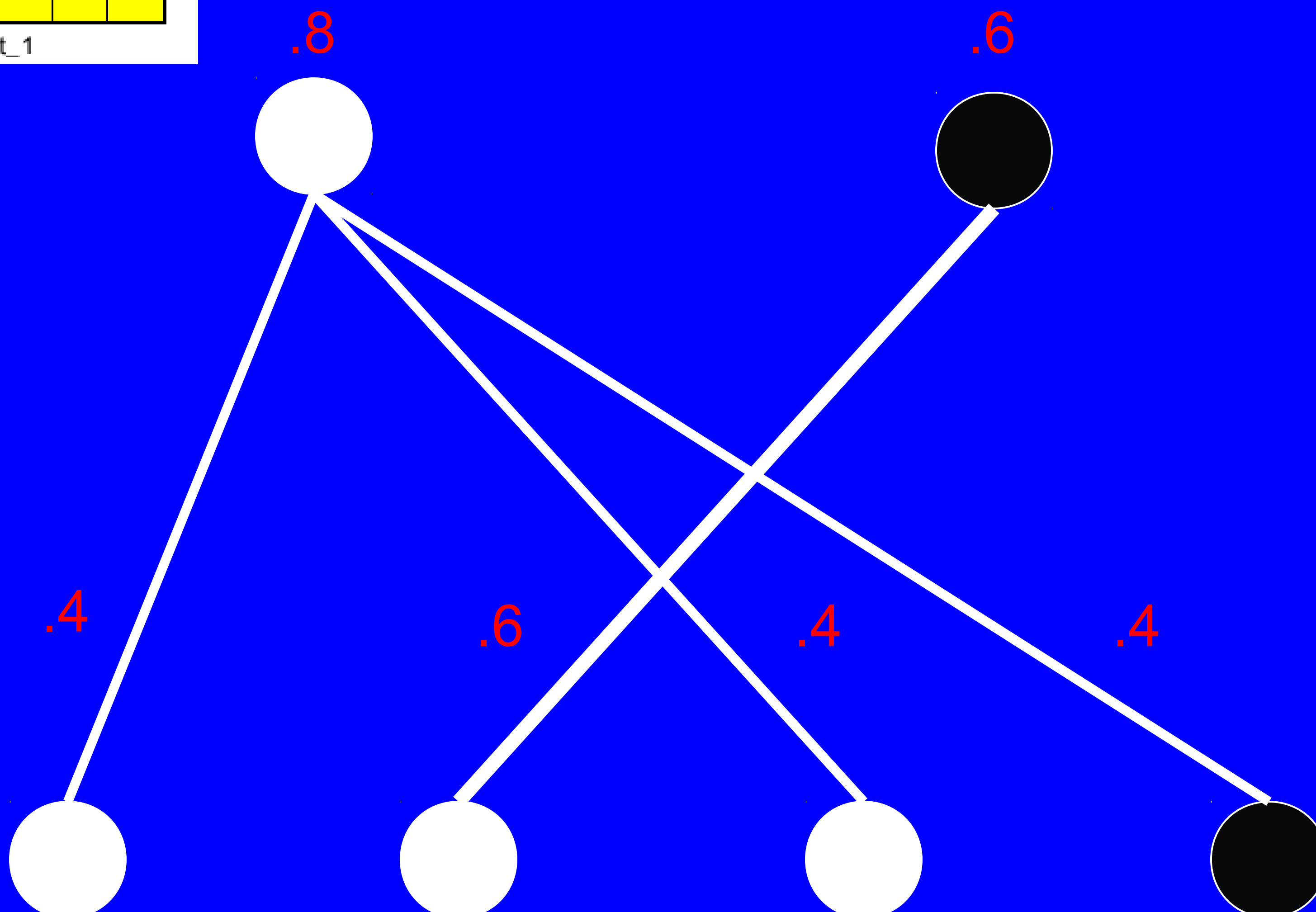


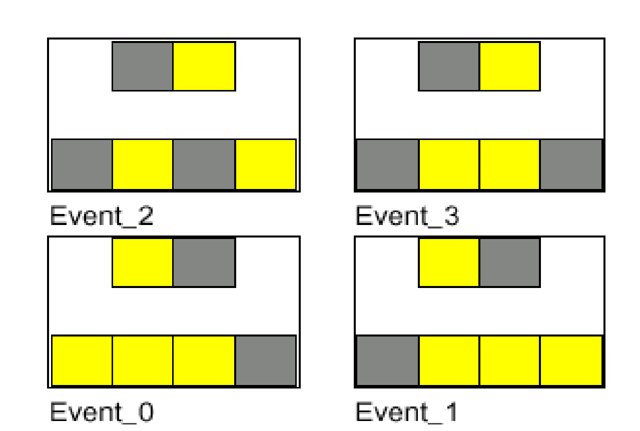
The mapping is solvable!



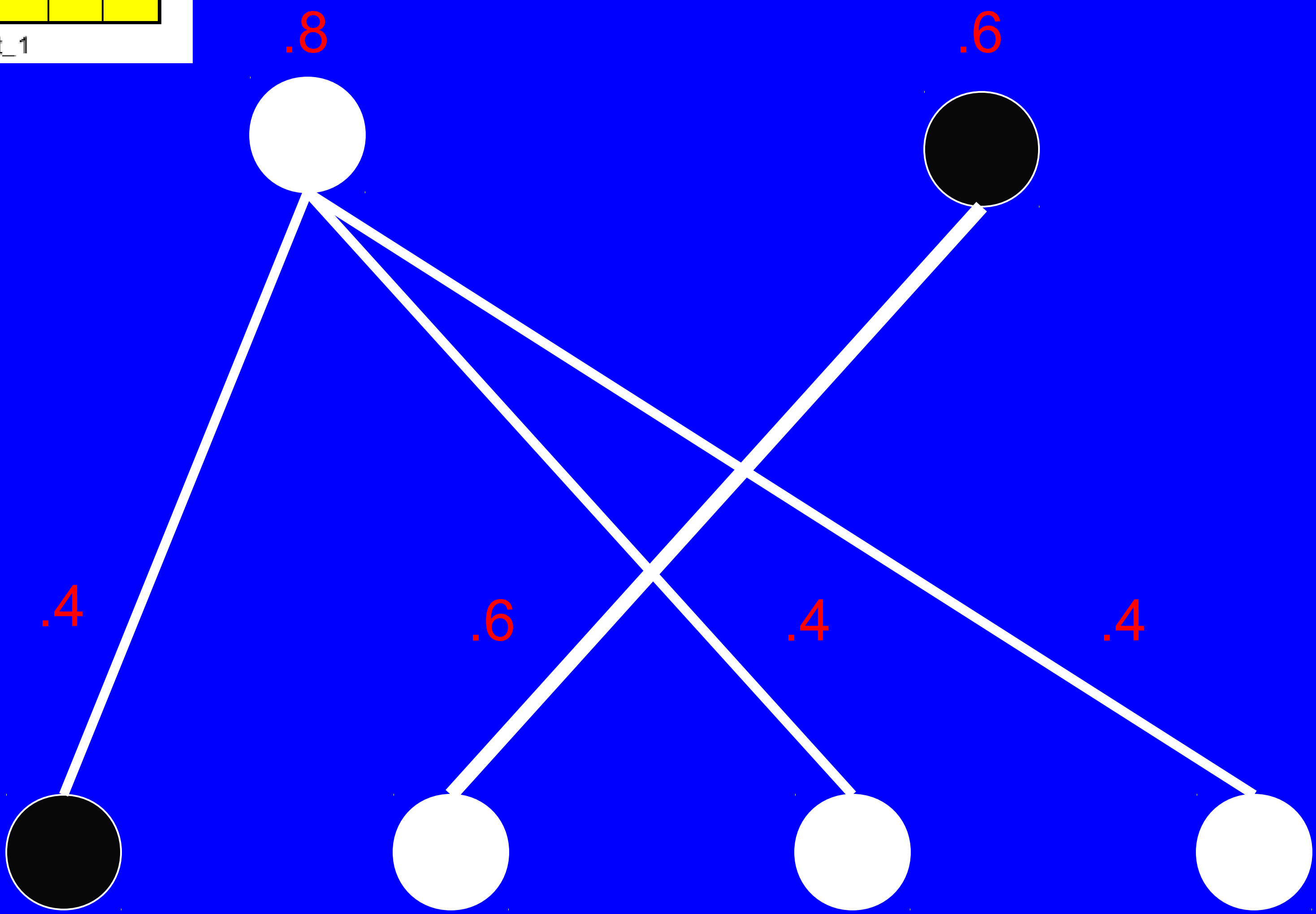


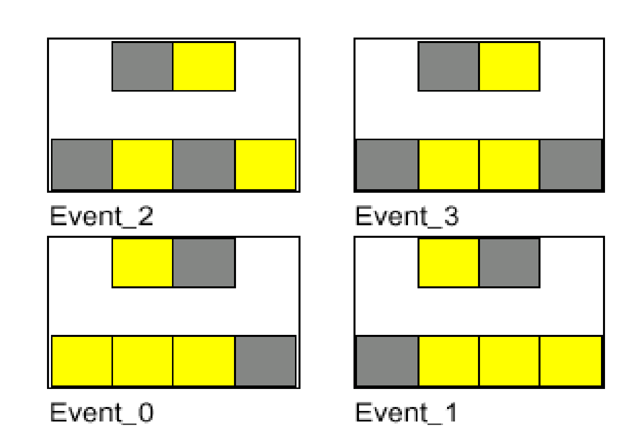
The mapping is solvable!



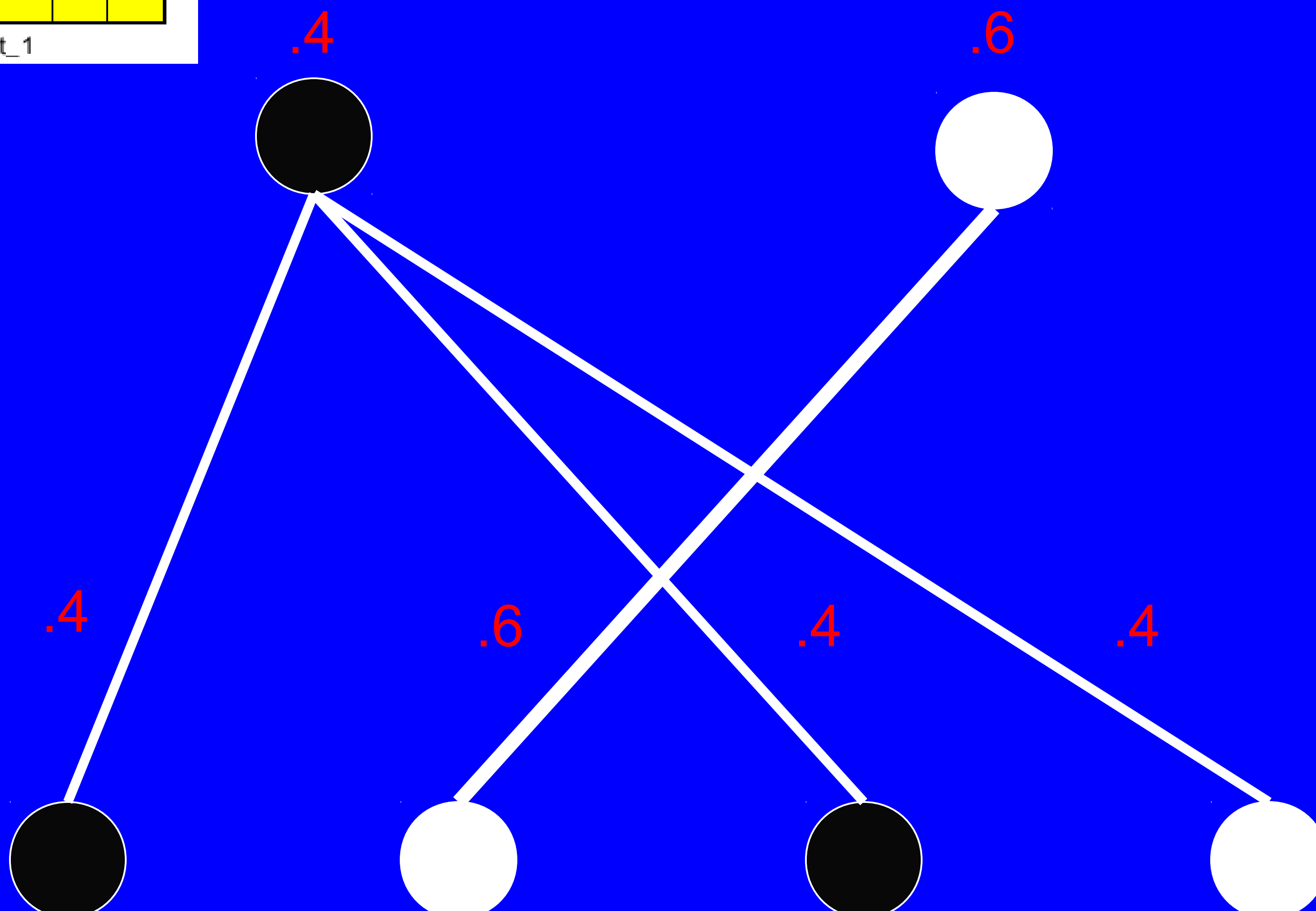


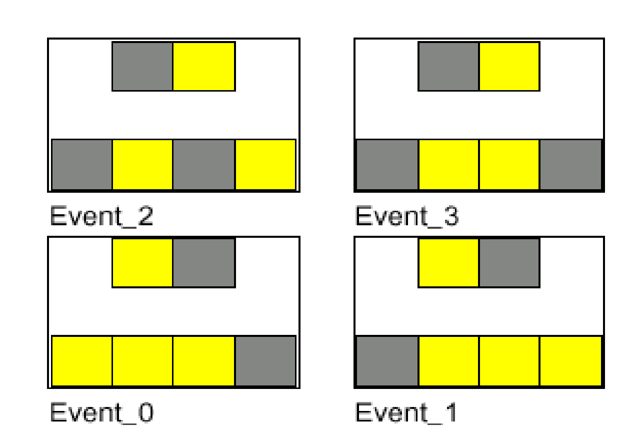
The mapping is solvable!



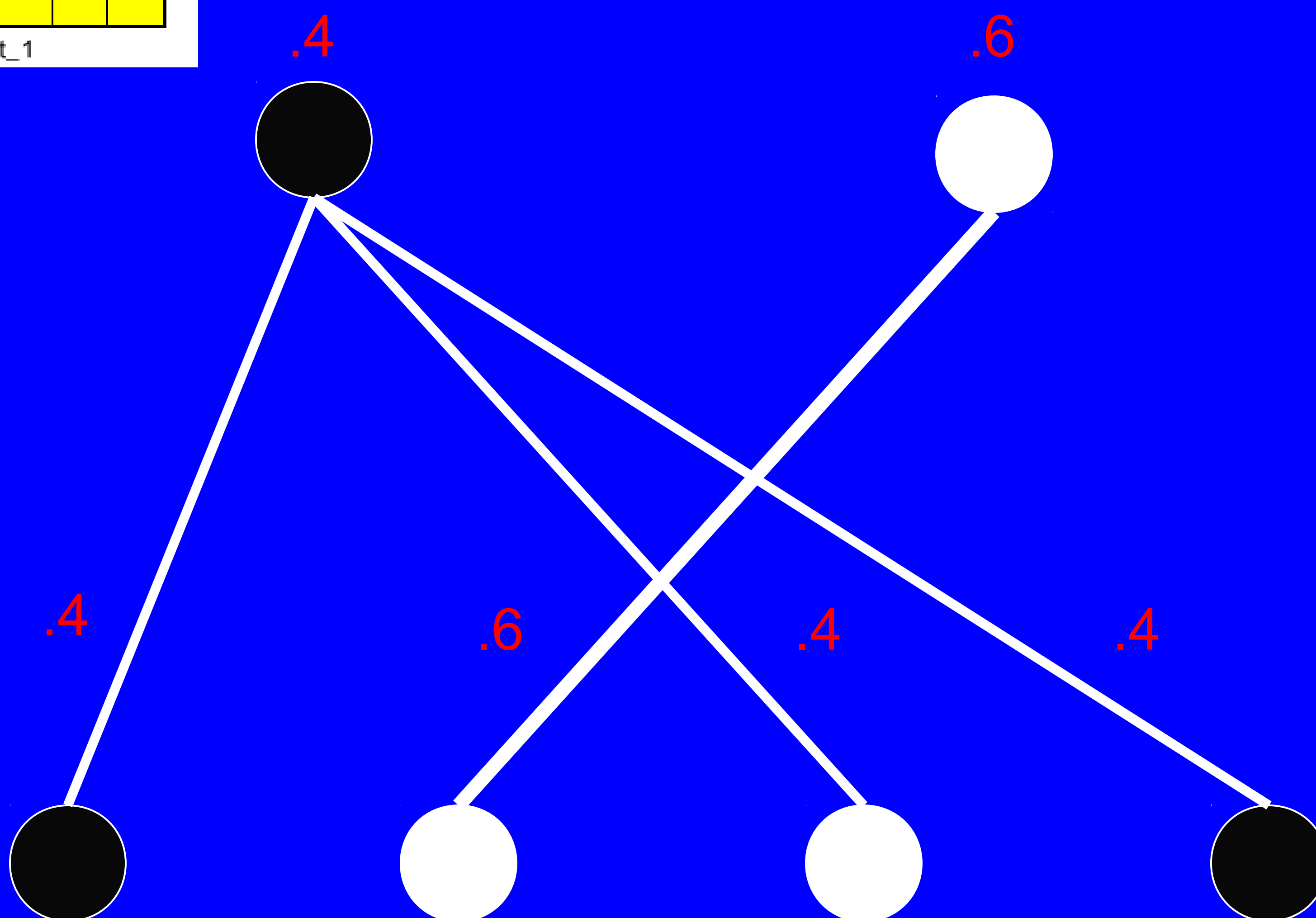


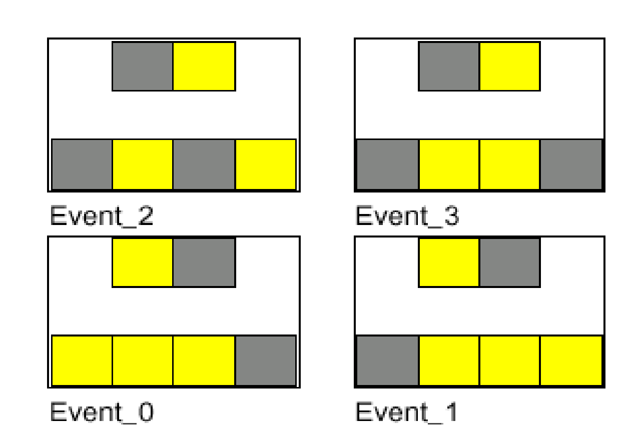
The mapping is solvable!



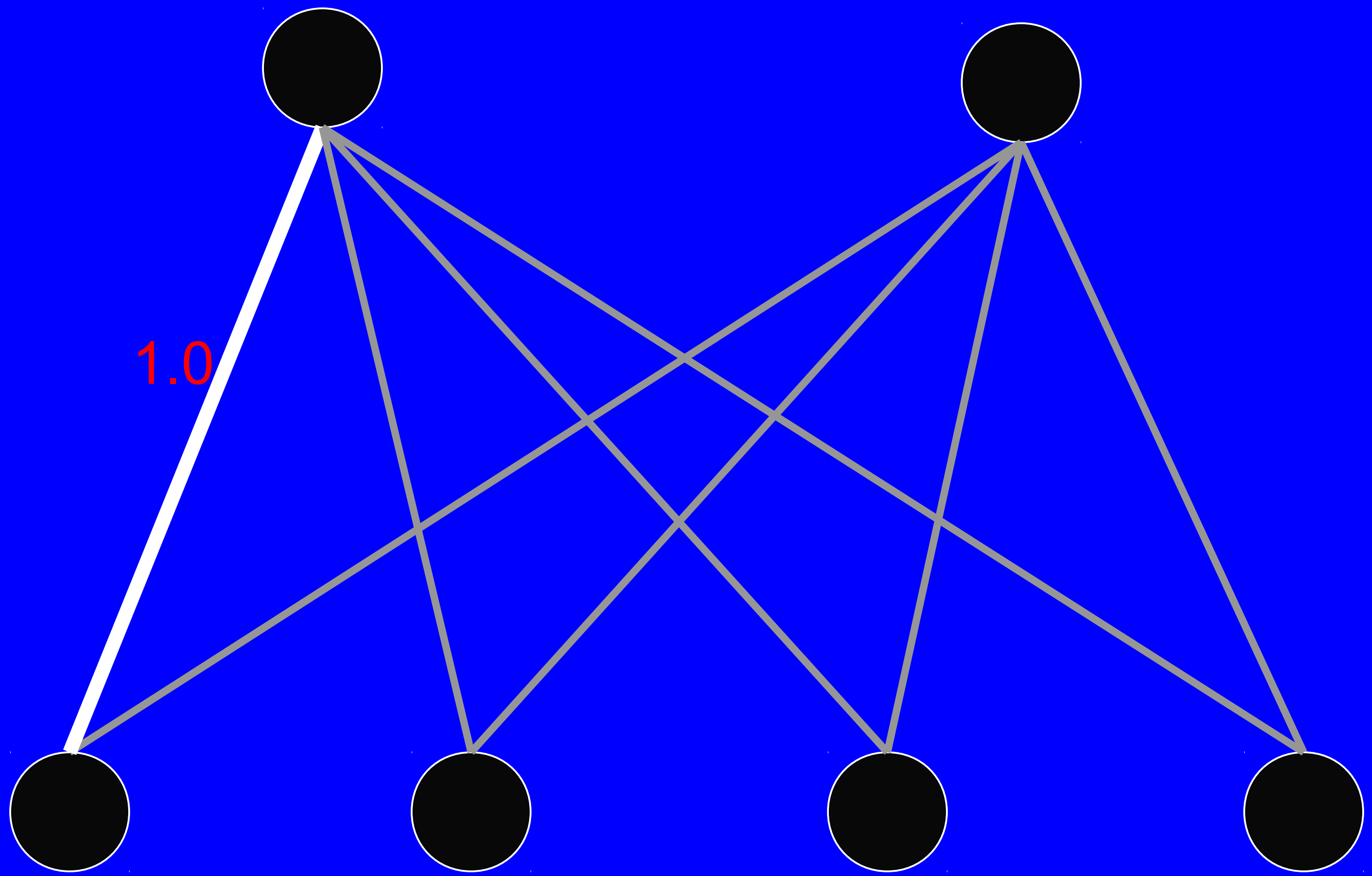


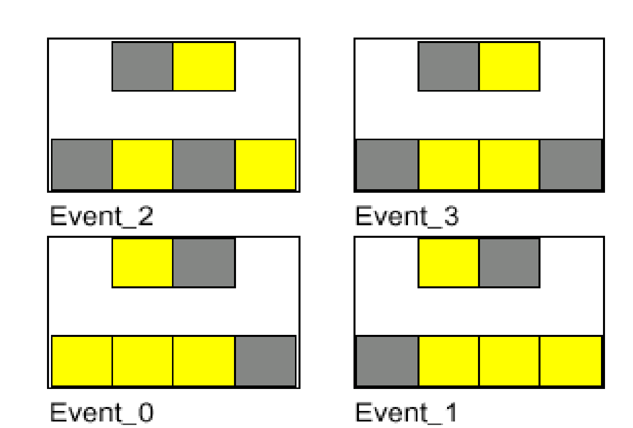
The mapping is solvable!



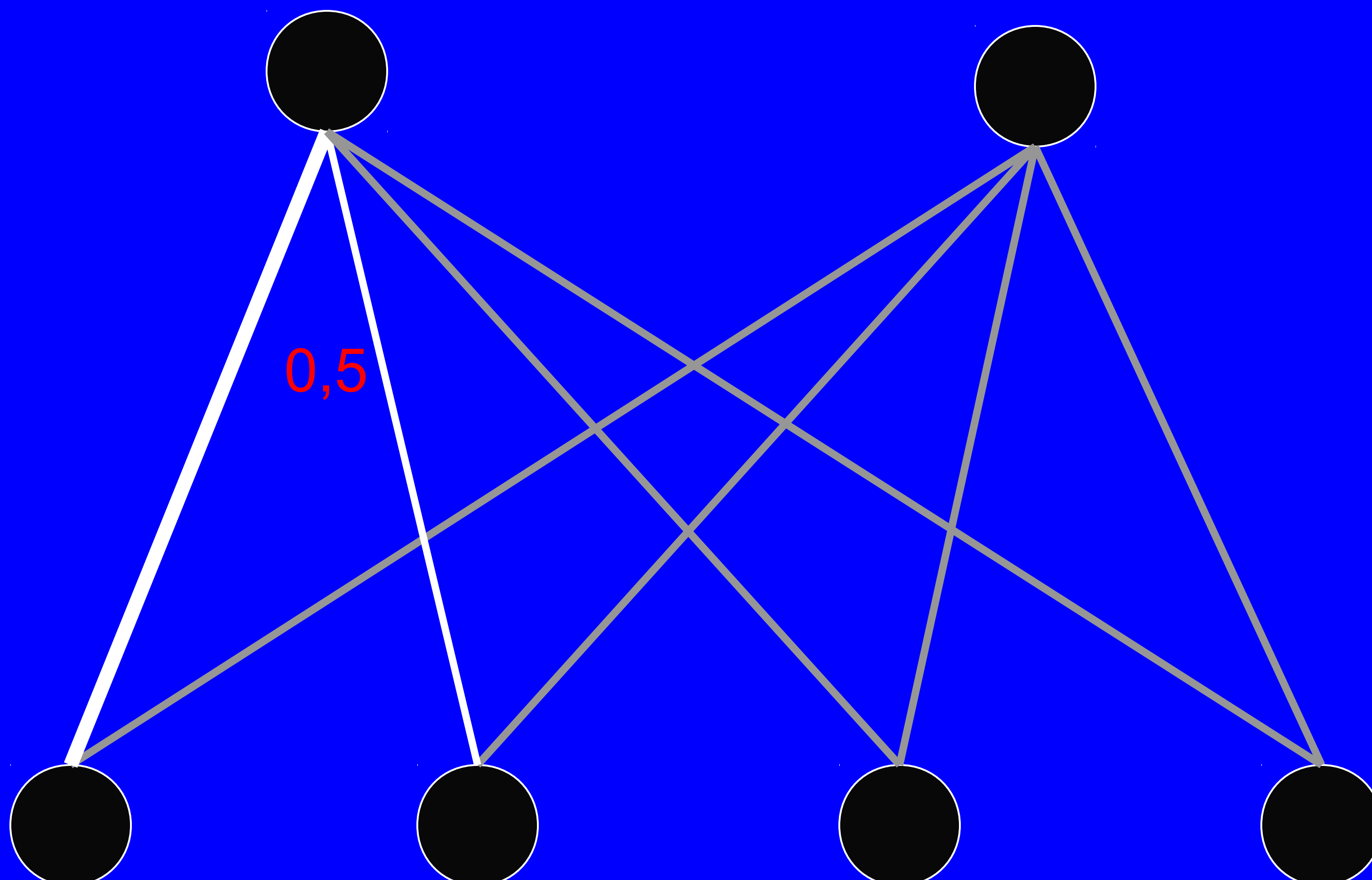


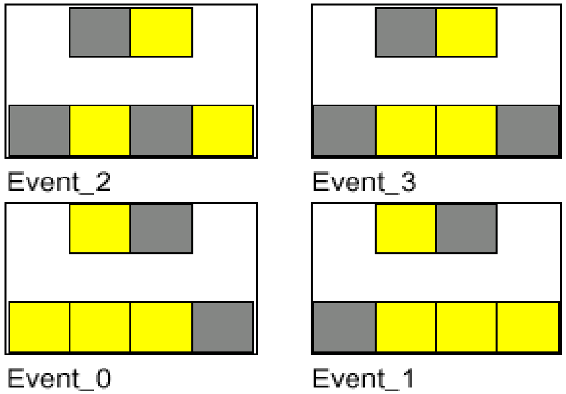
Hebbian learning:
weight ~ P(receiver active | sender active)



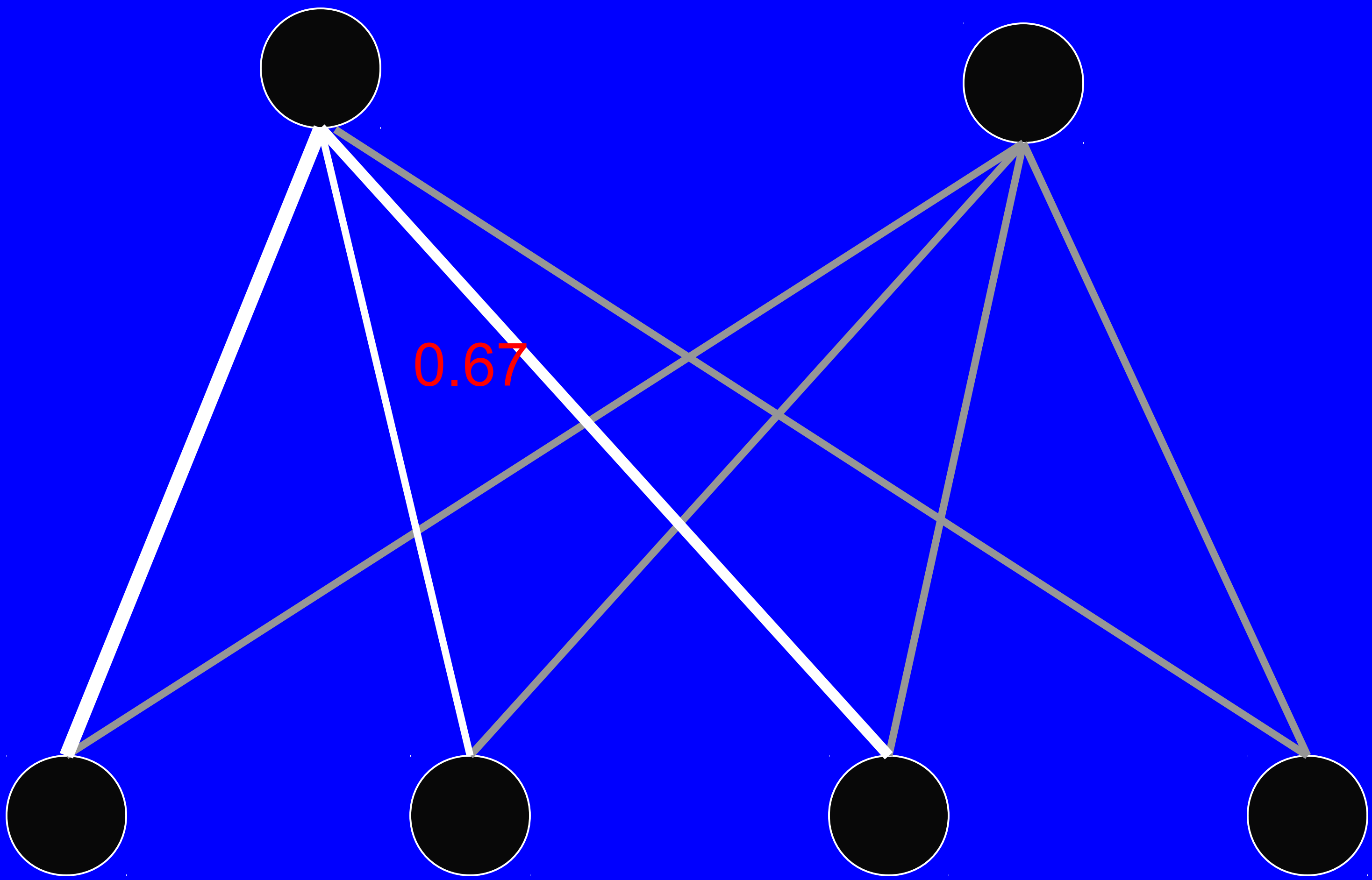


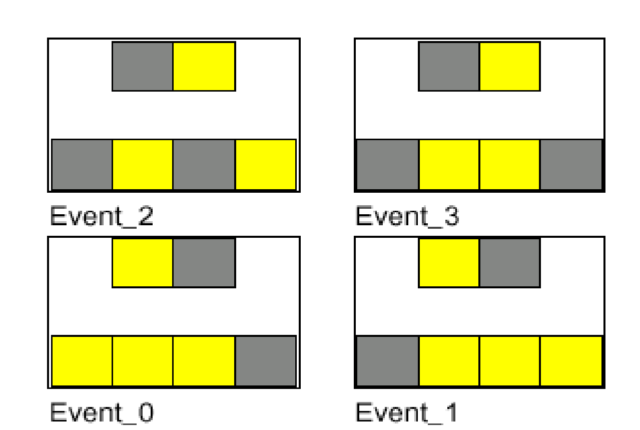
Hebbian learning:
weight ~ P(receiver active | sender active)



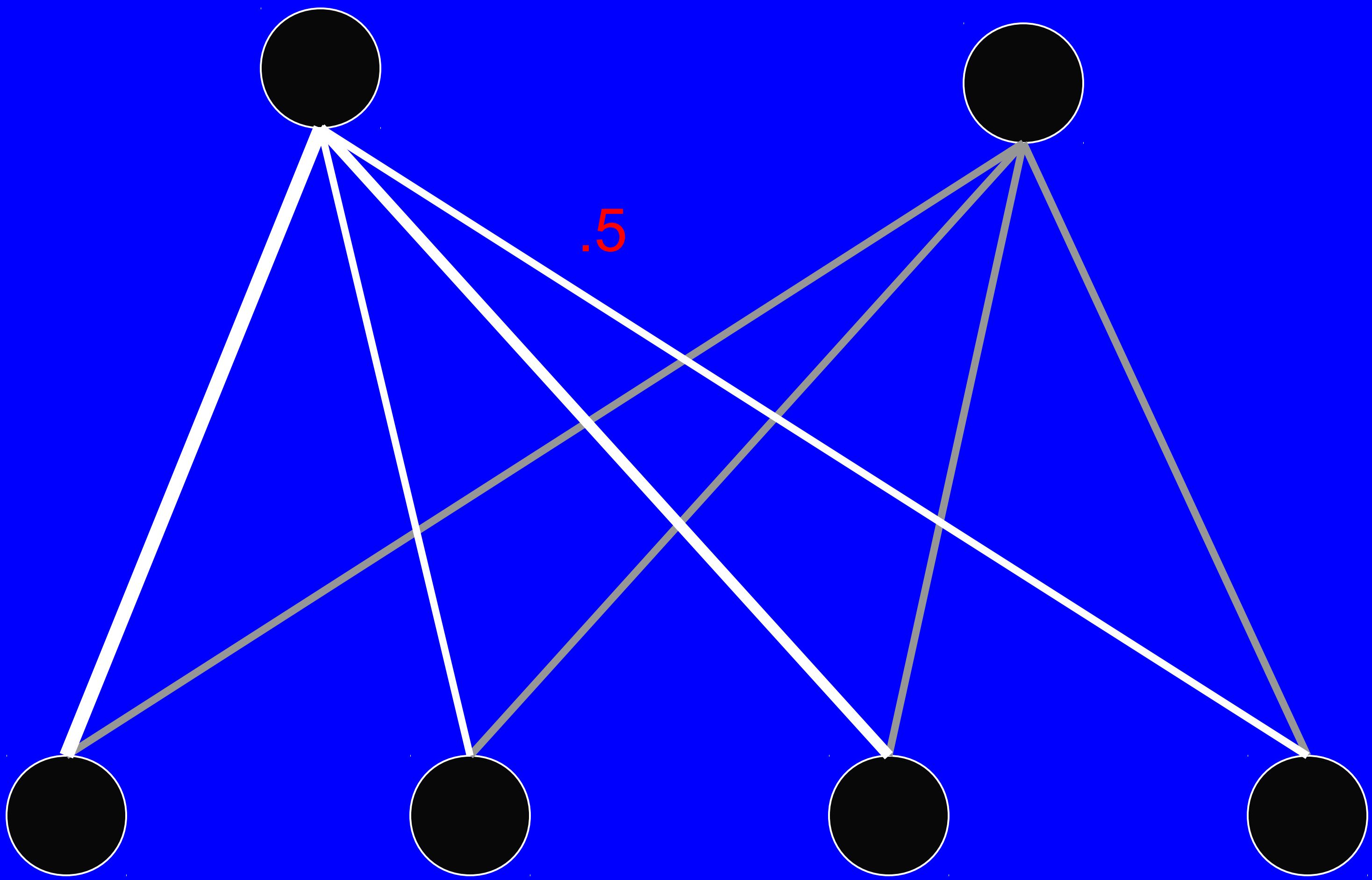


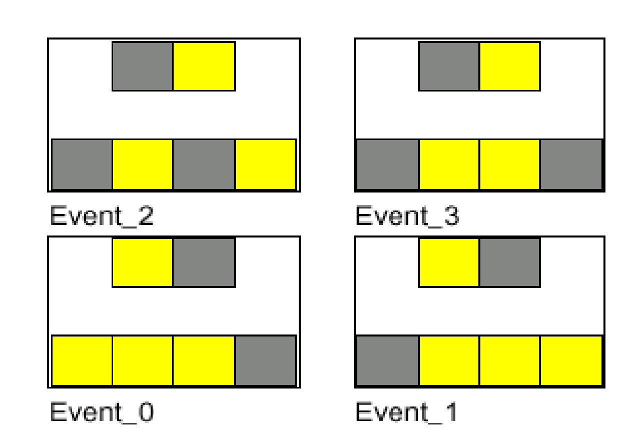
Hebbian learning:
weight ~ P(receiver active | sender active)



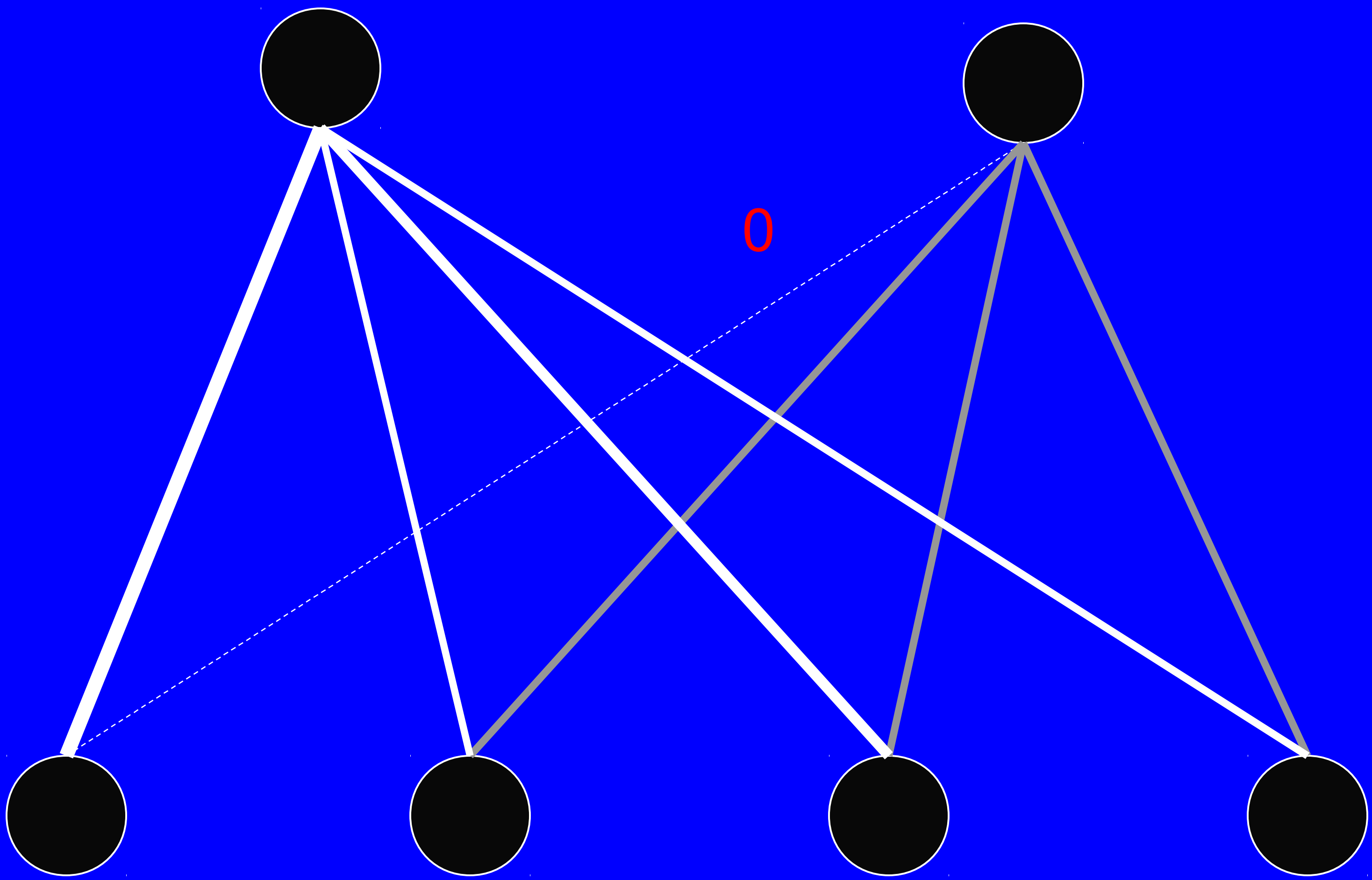


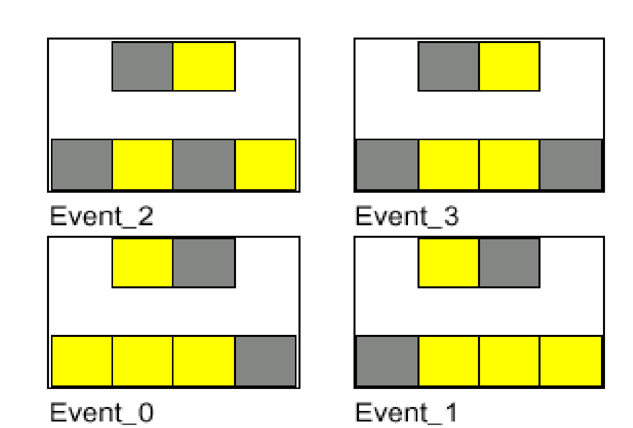
Hebbian learning:
weight ~ P(receiver active | sender active)



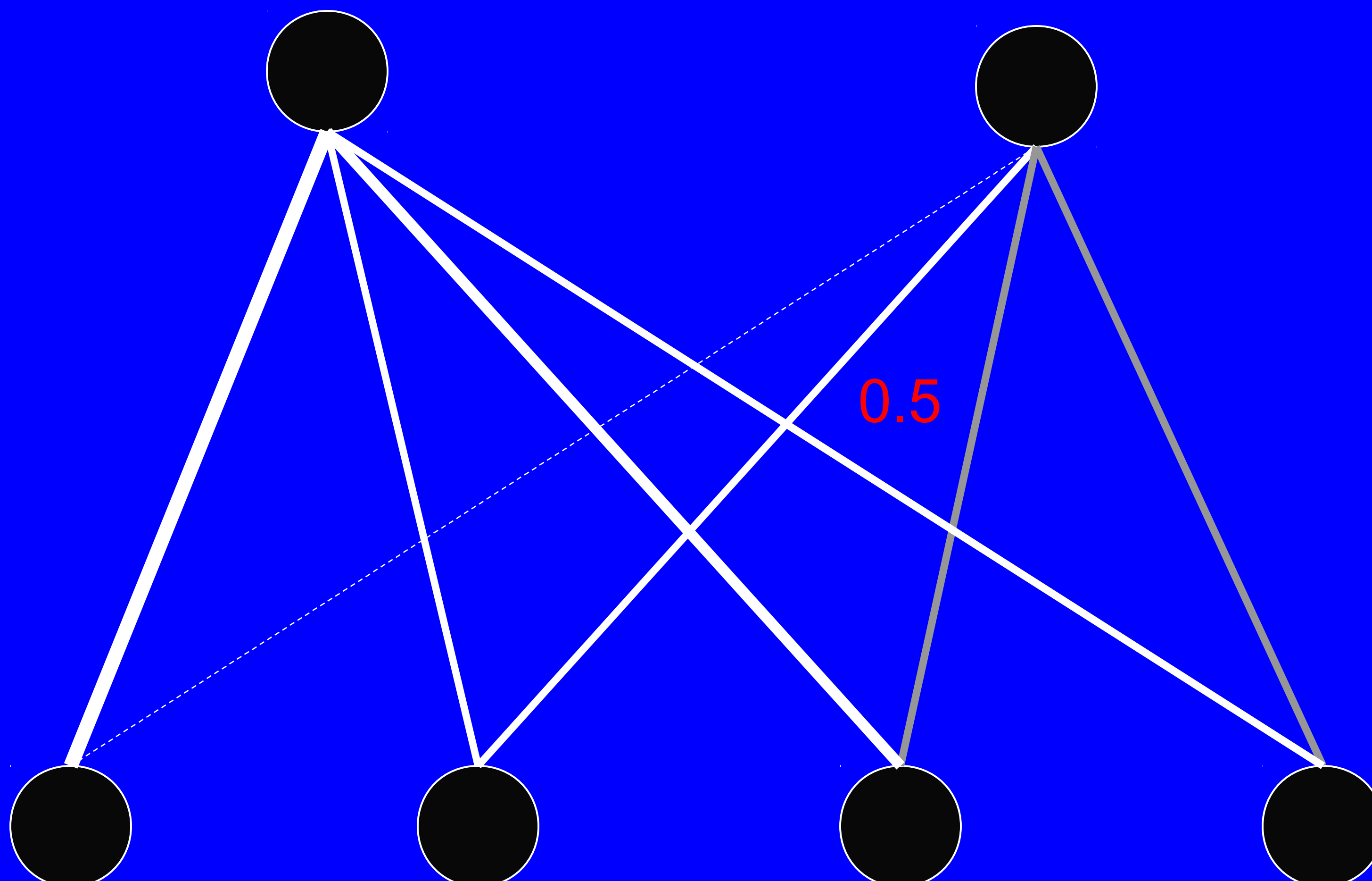


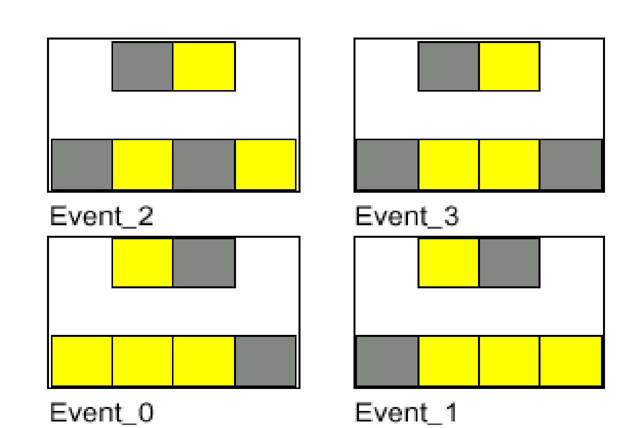
Hebbian learning:
weight ~ P(receiver active | sender active)



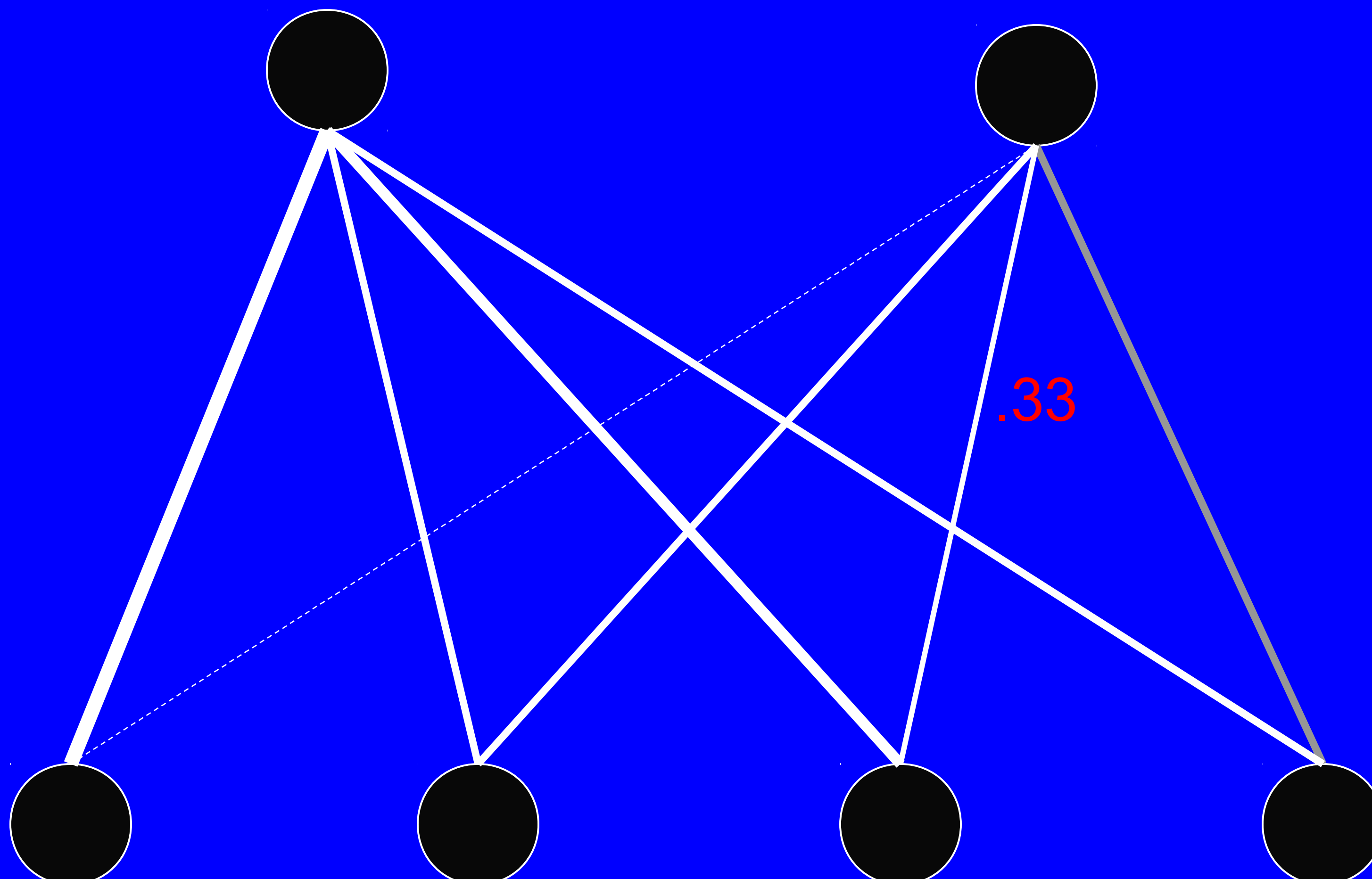


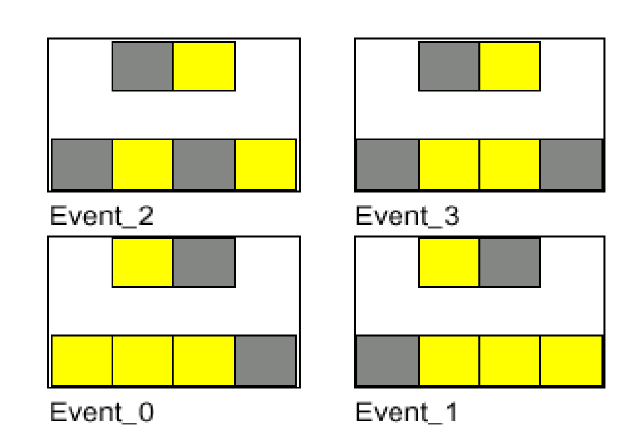
Hebbian learning:
weight ~ P(receiver active | sender active)



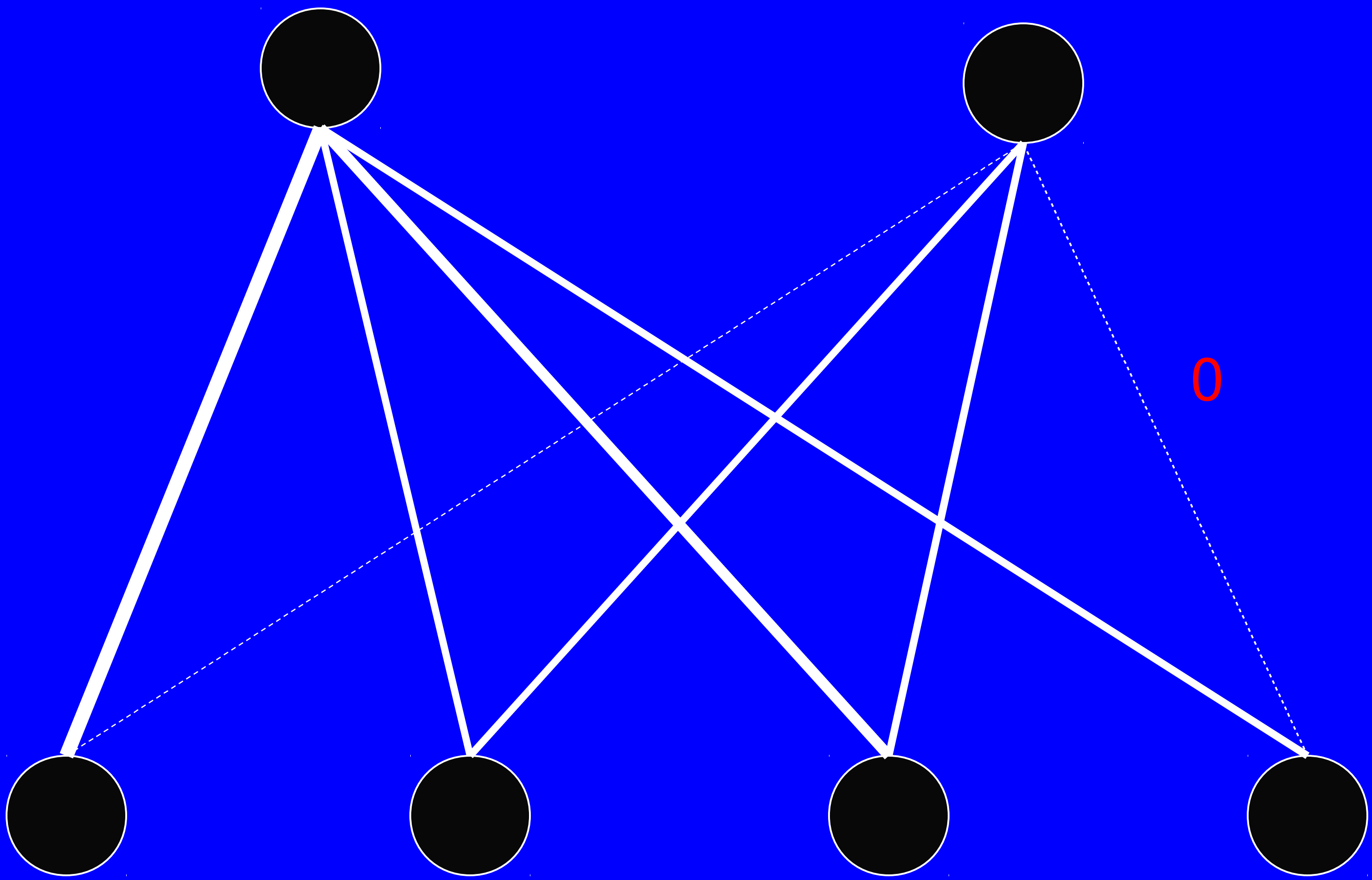


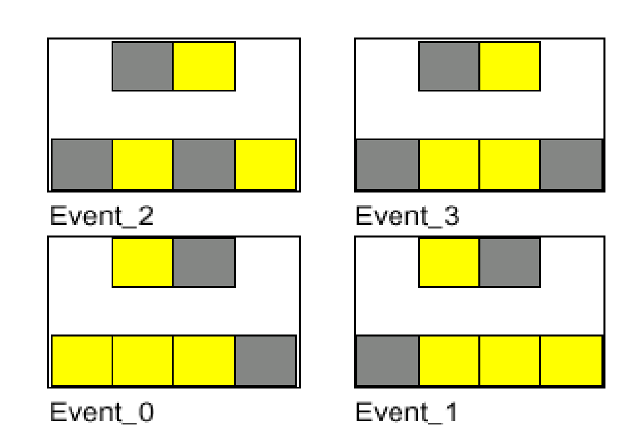
Hebbian learning:
weight ~ P(receiver active | sender active)



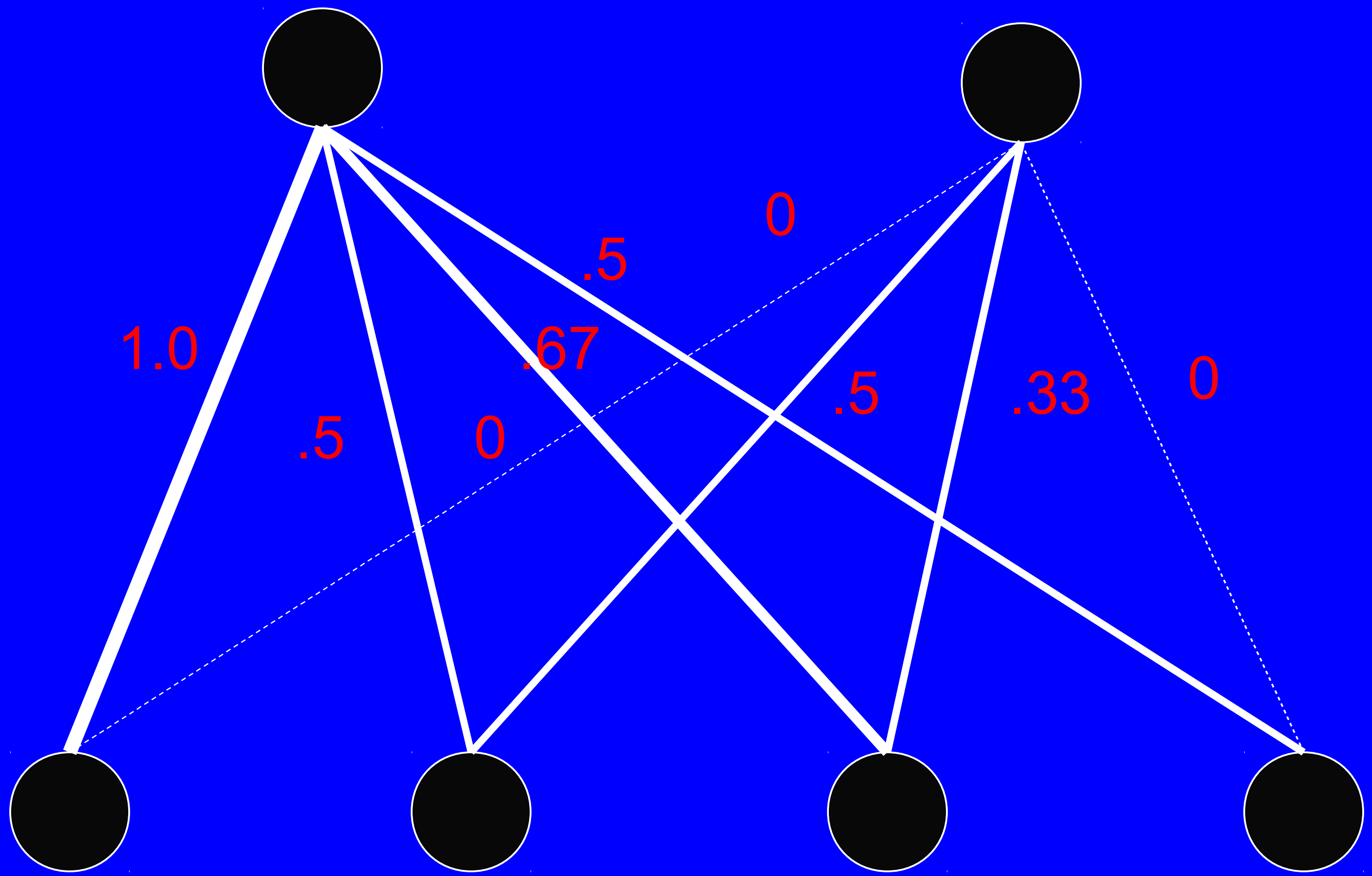


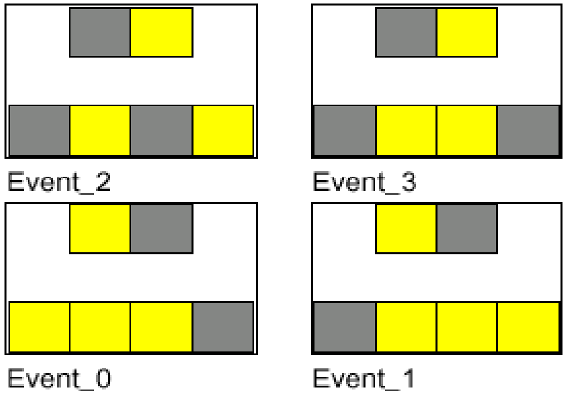
Hebbian learning:
weight ~ P(receiver active | sender active)



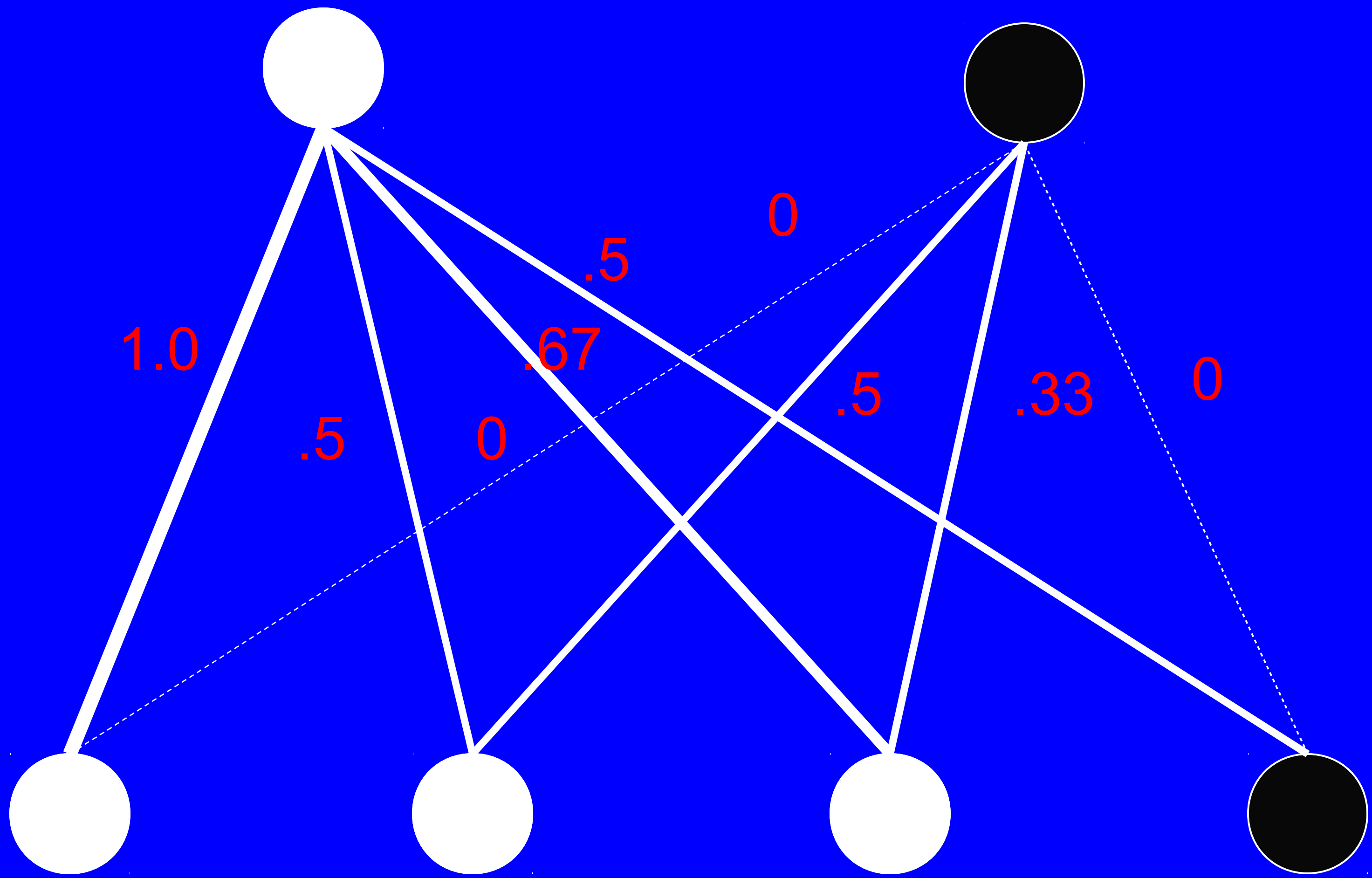


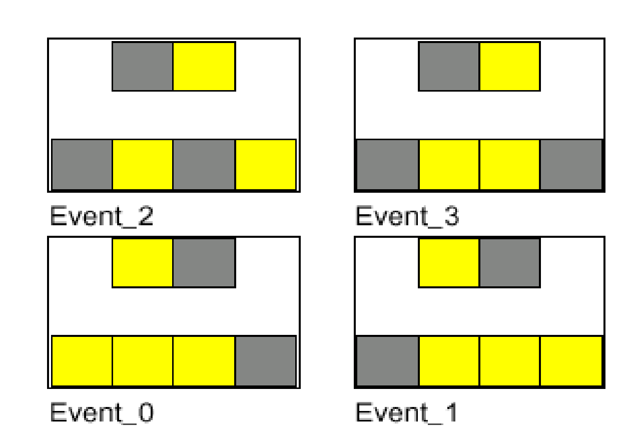
Can these weights solve the task?



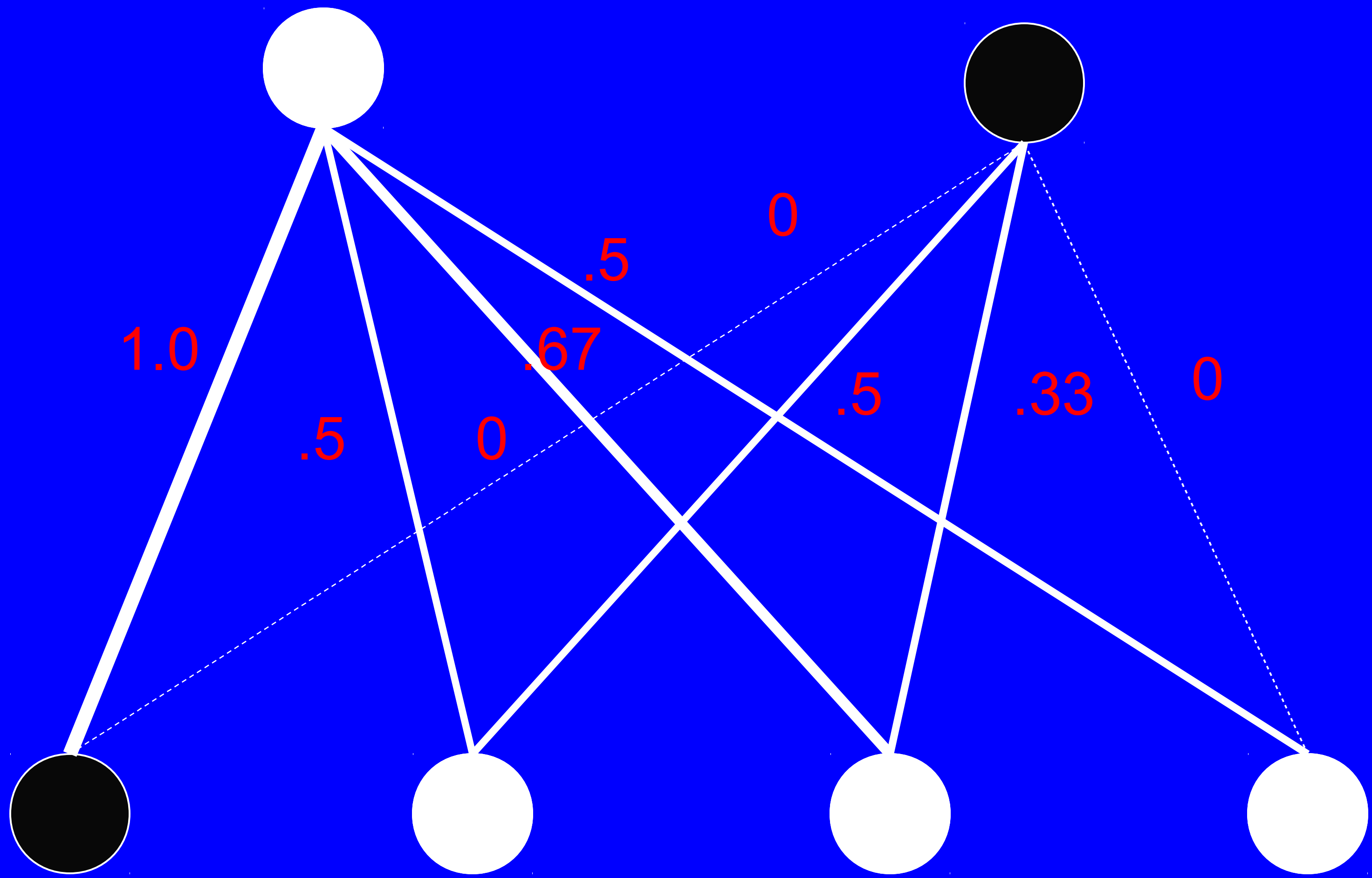


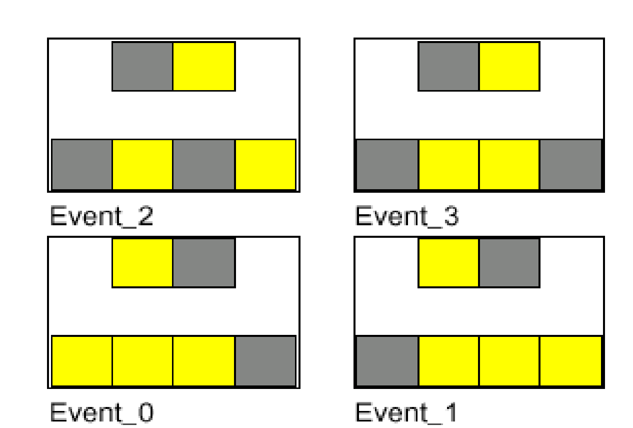
Can these weights solve the task?
Event 0 OK!



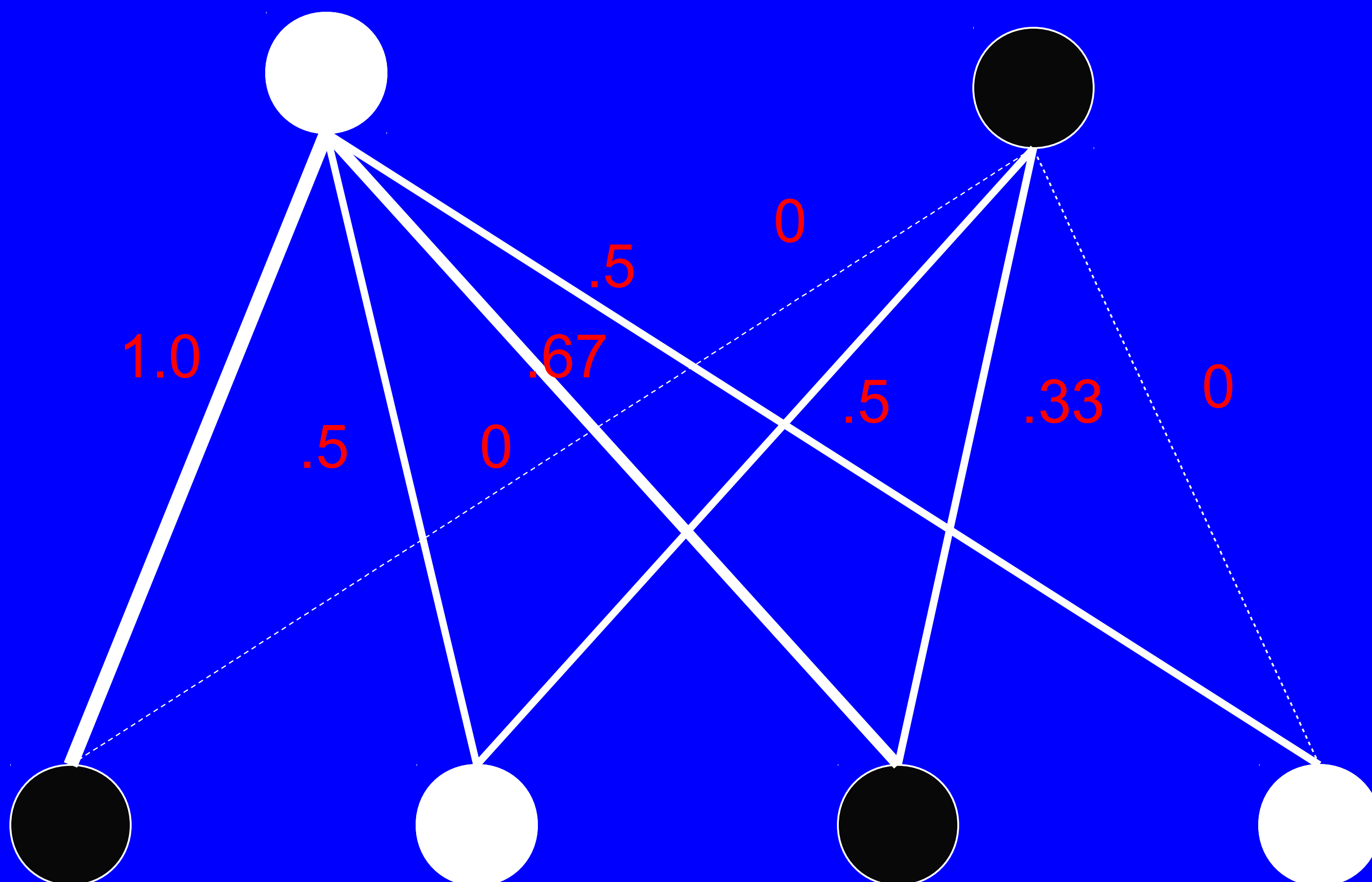


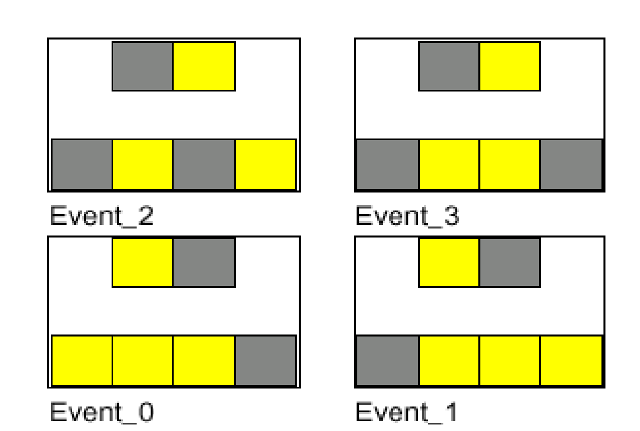
Can these weights solve the task?
Event 1 OK!



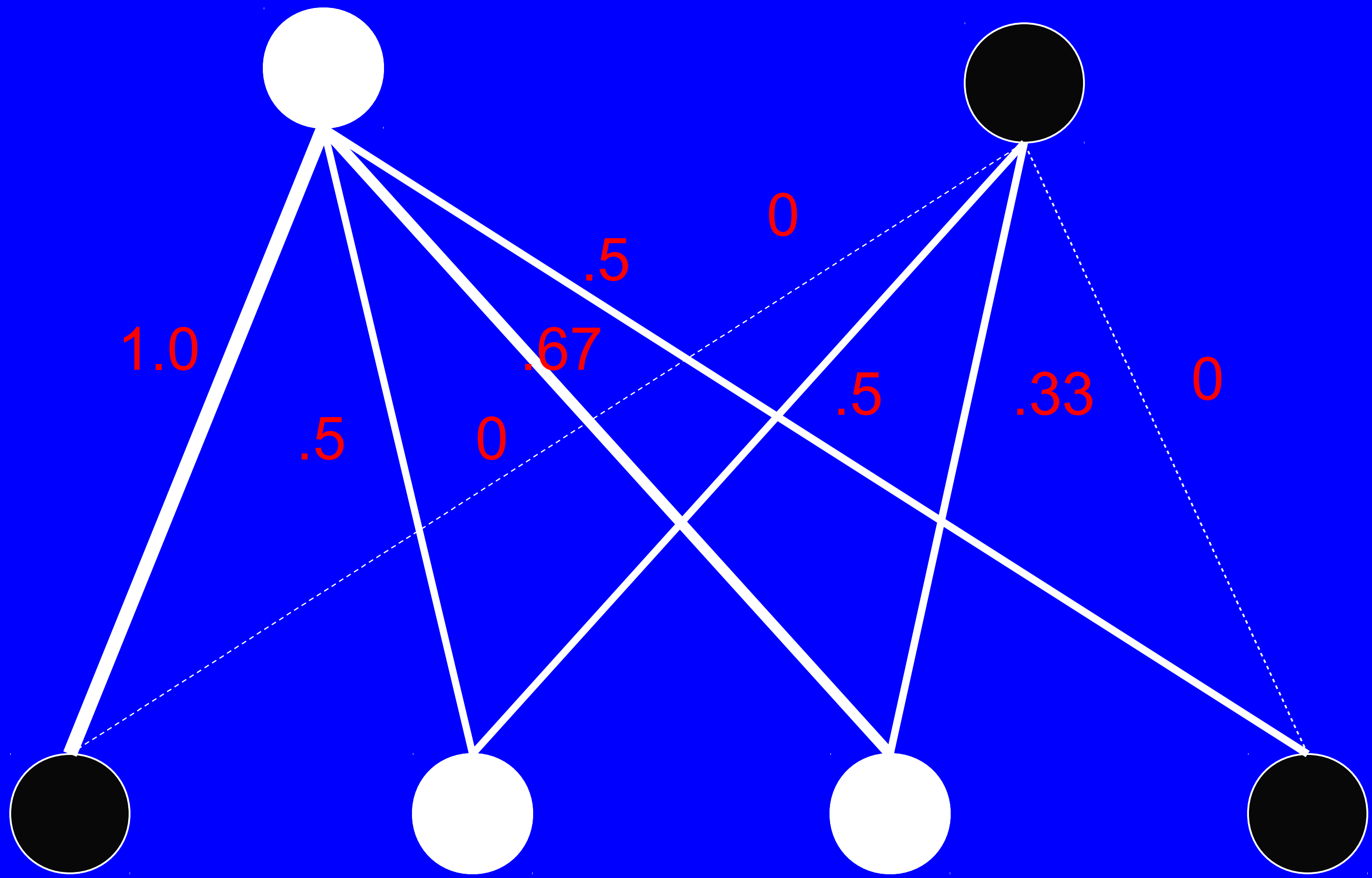


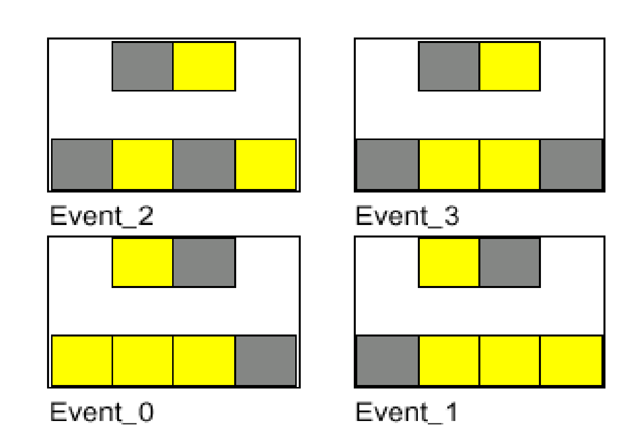
Can these weights solve the task?
Event 2 not OK....



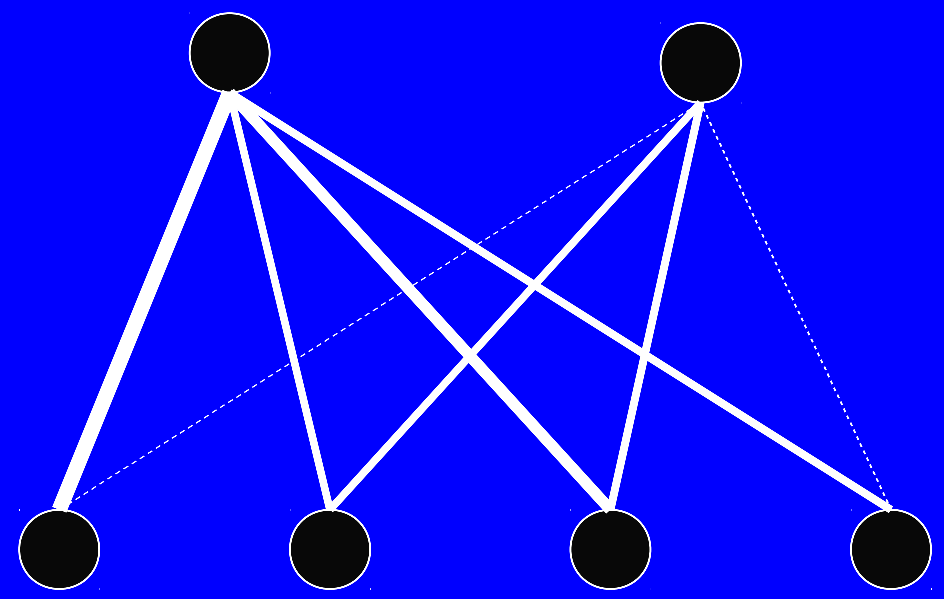


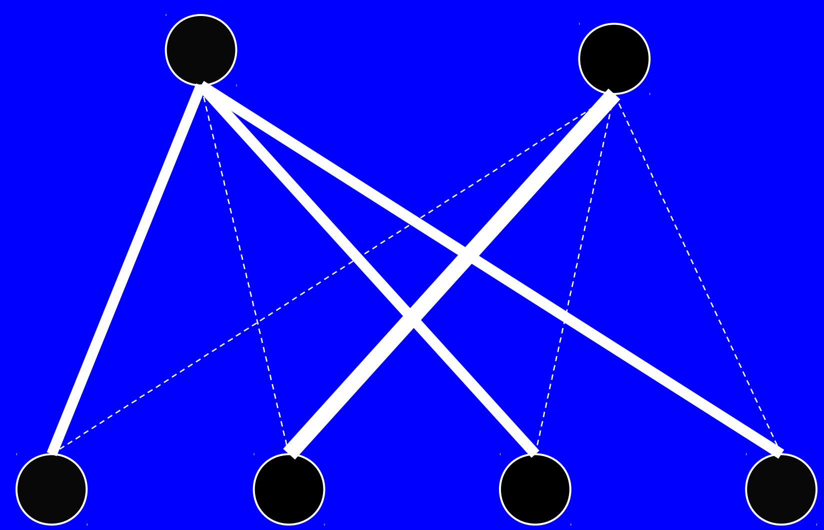
Can these weights solve the task?
Event 3 not OK....





Weights learned by Hebb =>





<= (one set of)
Weights that solve the task

Solution: Error-Driven Learning

First, we will consider how to do this and later come back to biology and more realistic implementation

- Instead of learning based on correlations, learn based on **error**: The difference between what the network is **supposed** to do, and what it **actually does**

- Error can be indexed using sum squared error (SSE)

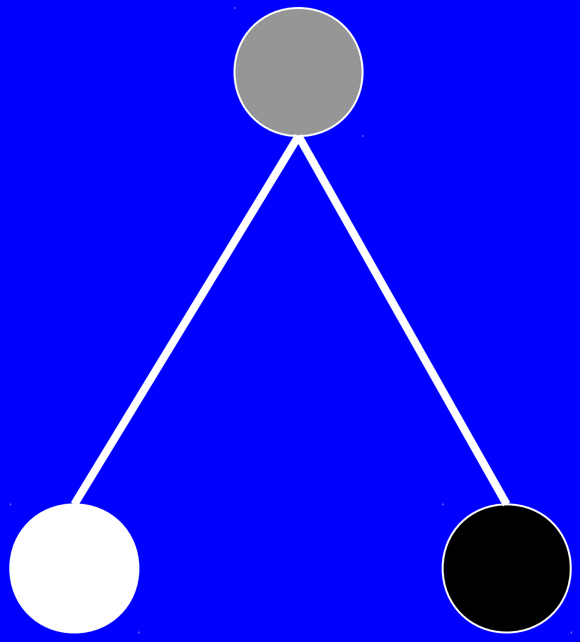
$$SSE = \sum_{p} \sum_{k} \left(t_{kp} - o_{kp}\right)^2$$

- *t* = target output value (what activation is supposed to be) over all output units *k*, summed across all input patterns *p*
- *o* = actual output values for each *k* unit and input pattern *p*

Adjusting Weights to Minimize Error

- Say that we want hidden activity = 1 for this input pattern.

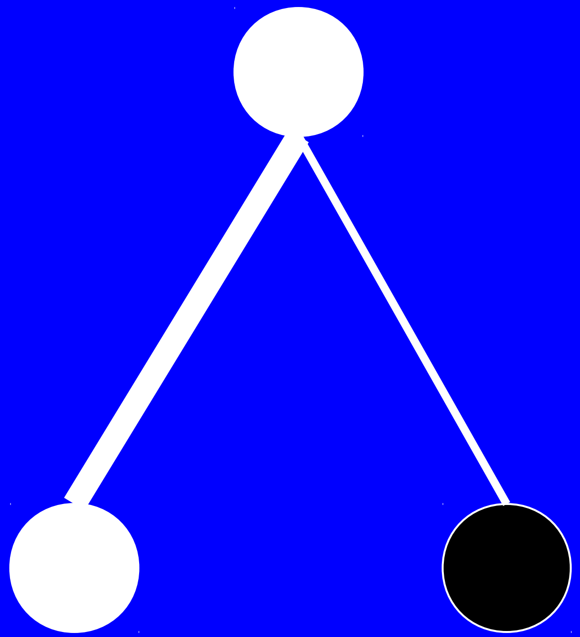
- If you could pick one (of the two) weights to increment, which would you change?



Adjusting Weights to Minimize Error

- Say that we want hidden activity = 1 for this input pattern.

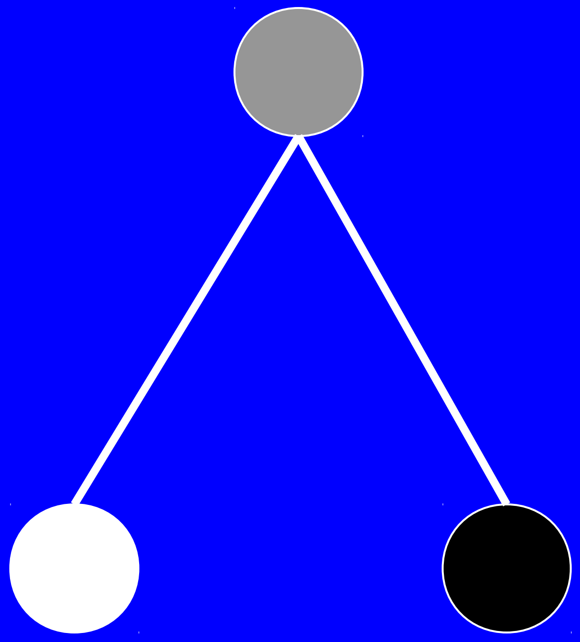
- If you could pick one (of the two) weights to increment, which would you change?



Adjusting Weights to Minimize Error

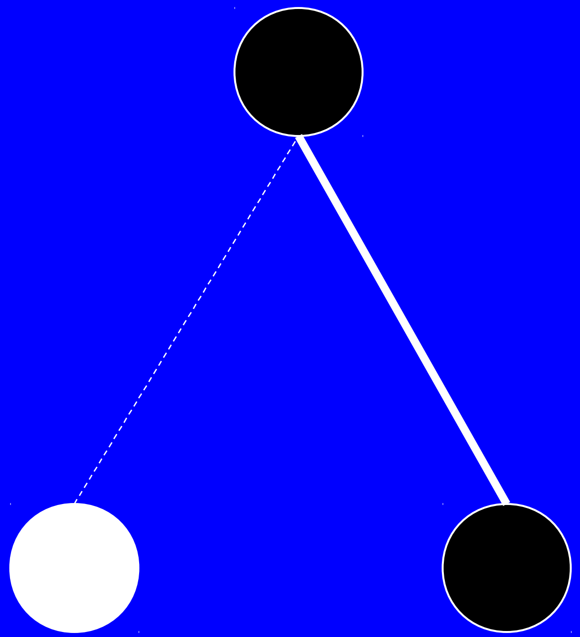
- Say that we want hidden activity = 0 for this input pattern.

- If you could pick one (of the two) weights to decrement, which would you change?



Adjusting Weights to Minimize Error

- Say that we want hidden activity = 0 for this input pattern.
- If you could pick one (of the two) weights to decrement, which would you change?



Credit/Blame Assignment

- Error-driven learning is all about figuring out who to **blame** for mistakes

- If the network makes an error, you should change weights from **active input units**

- Changing weights from inactive inputs has no effect

The Delta Rule

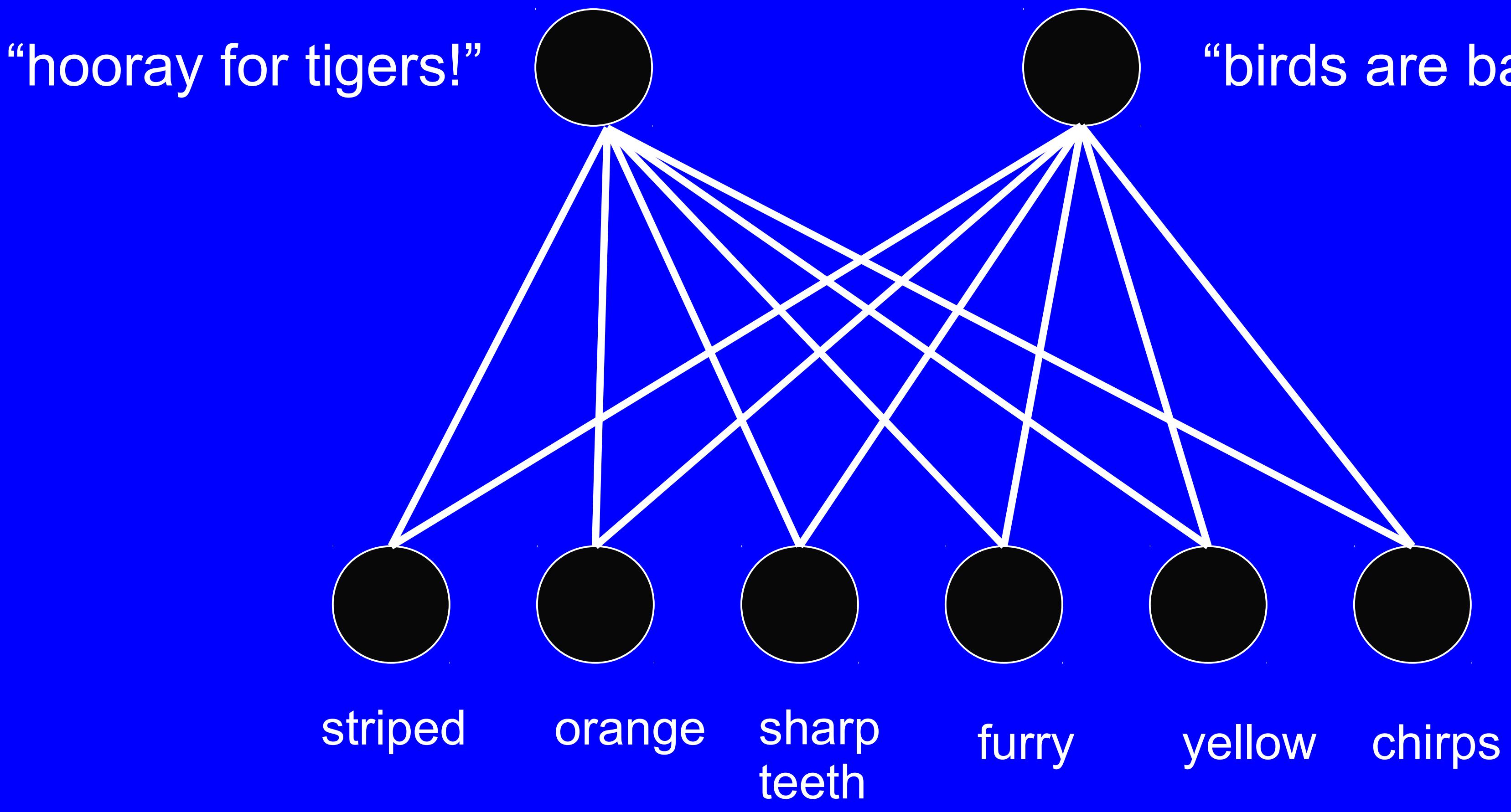
- The *delta rule* meets the criteria we have outlined for error-driven learning:

$$\Delta w_{ik} = \epsilon(t_k - o_k)s_i$$

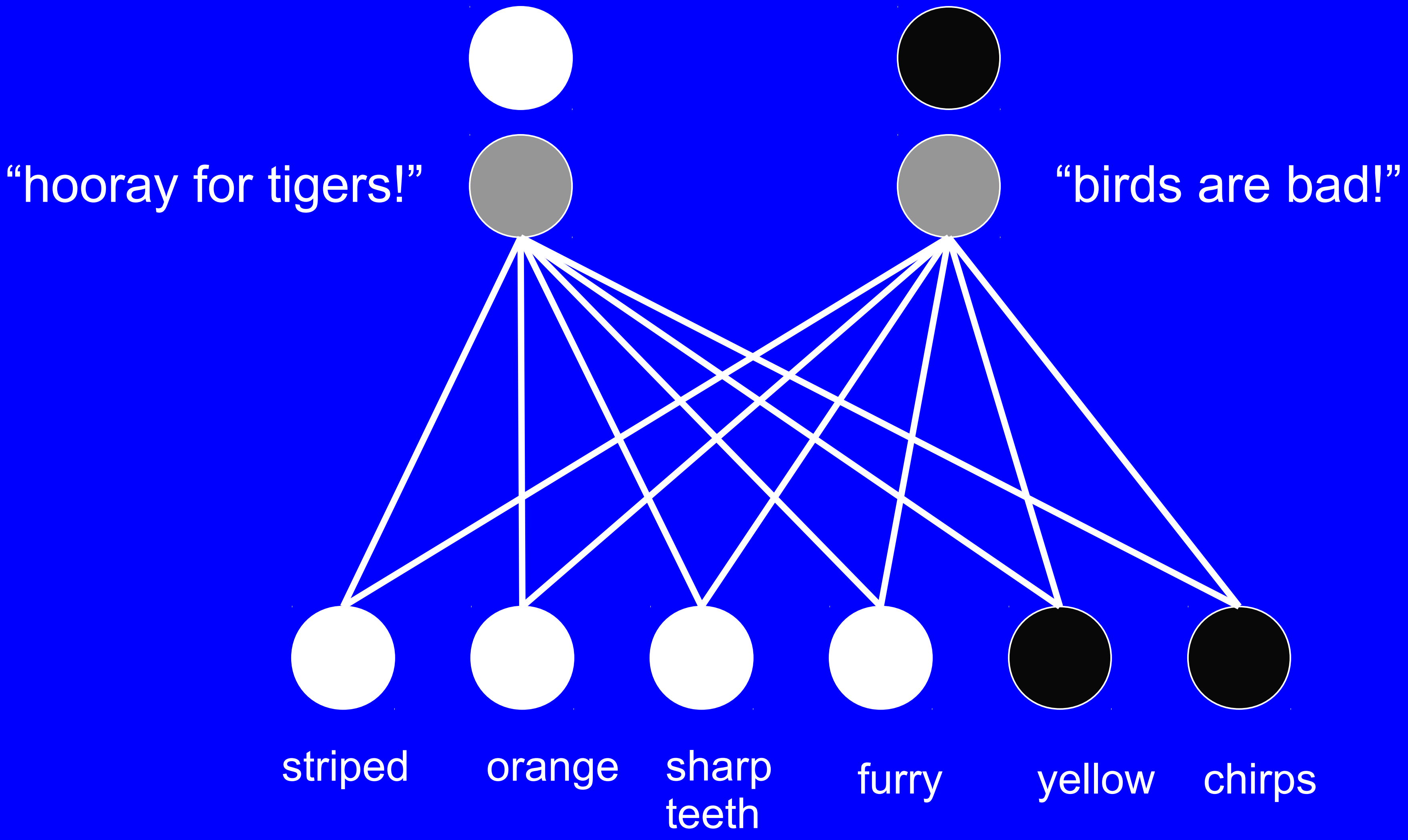
∀ Δw_{ik} = change in weight
- t_k = target output value (what activation is supposed to be)
- o_k = actual output value
- s_i = input unit activity

- Weight change is proportional to error, and it is also proportional to sending unit activity

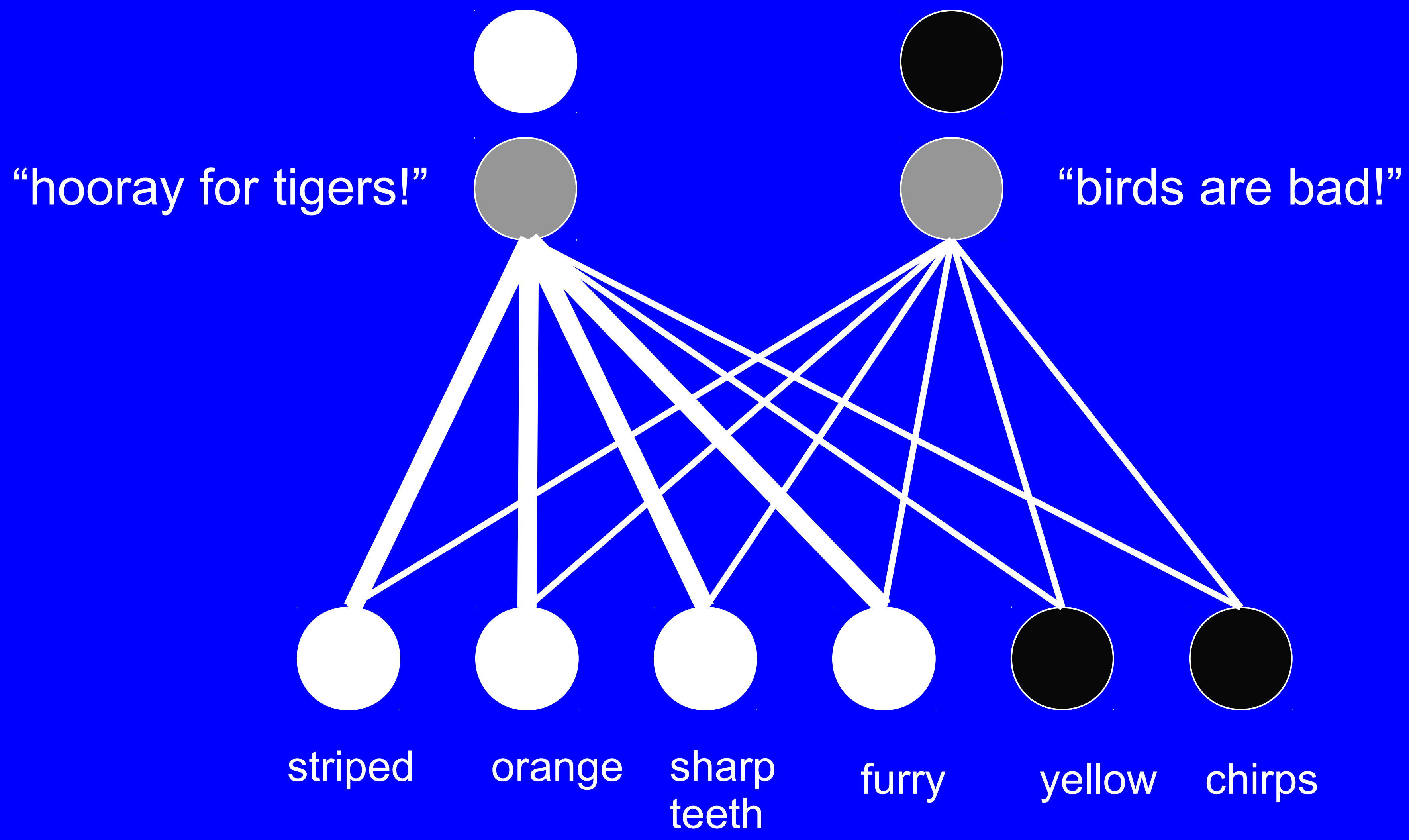
Error-driven learning



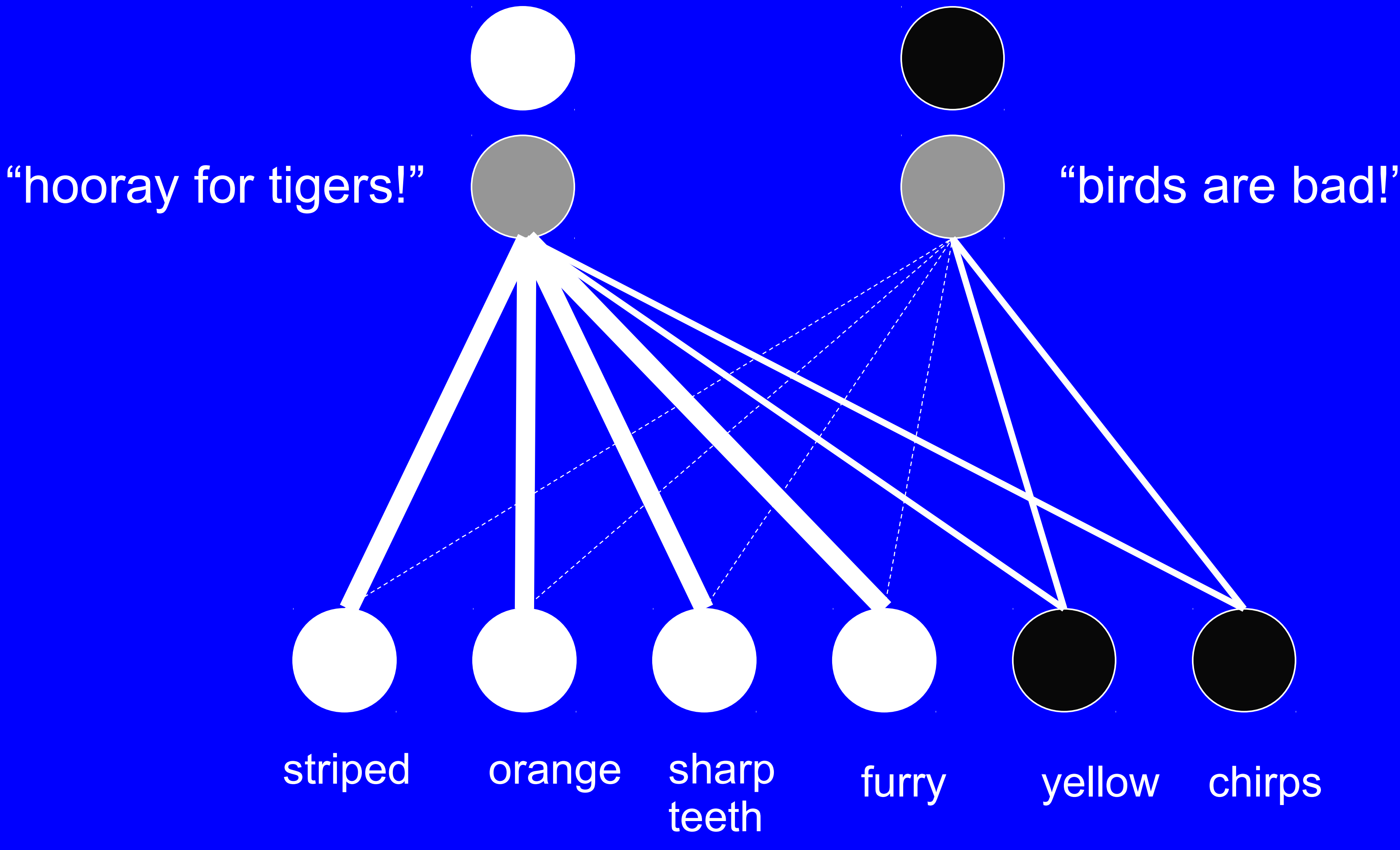
Error-driven learning



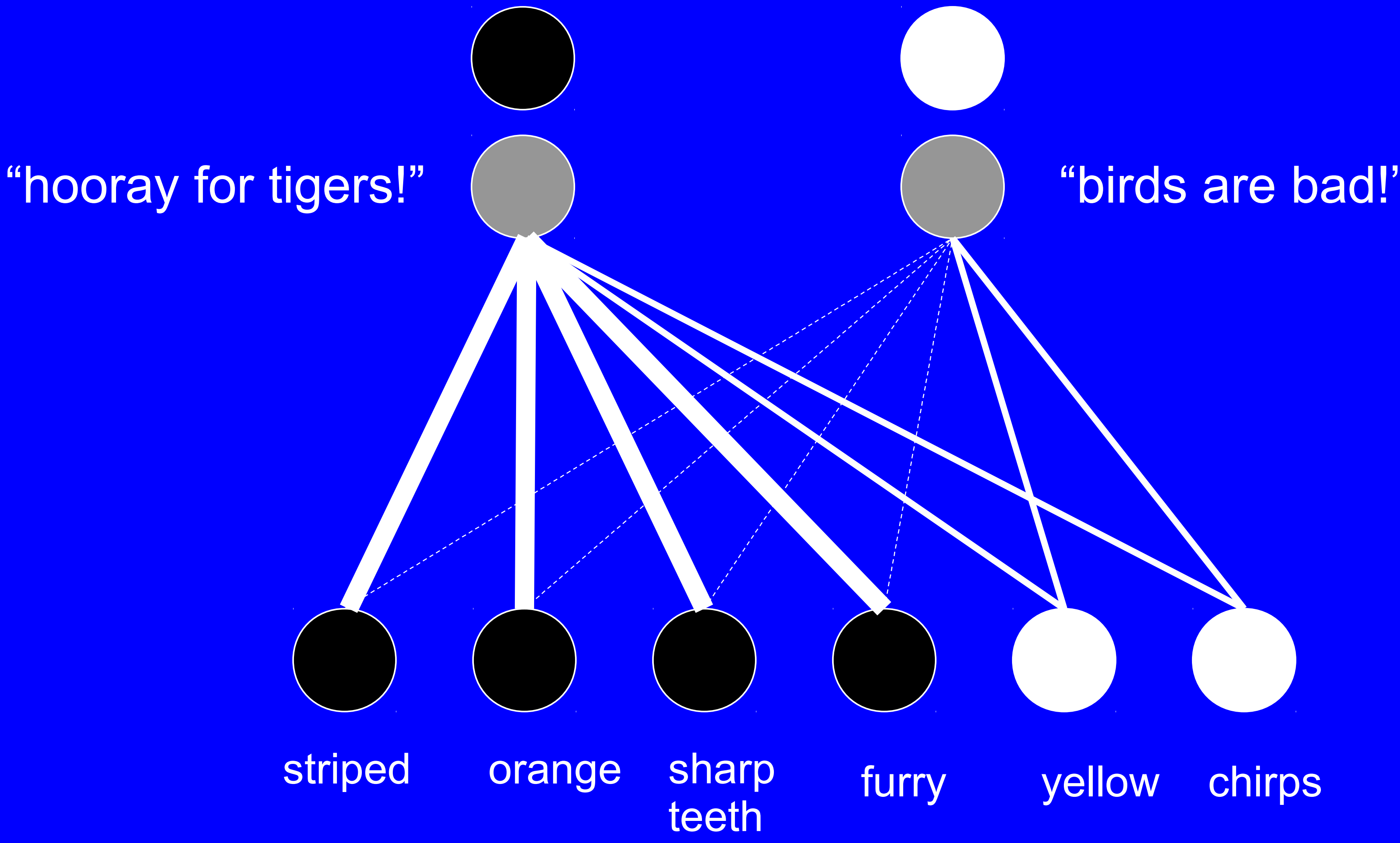
Error-driven learning



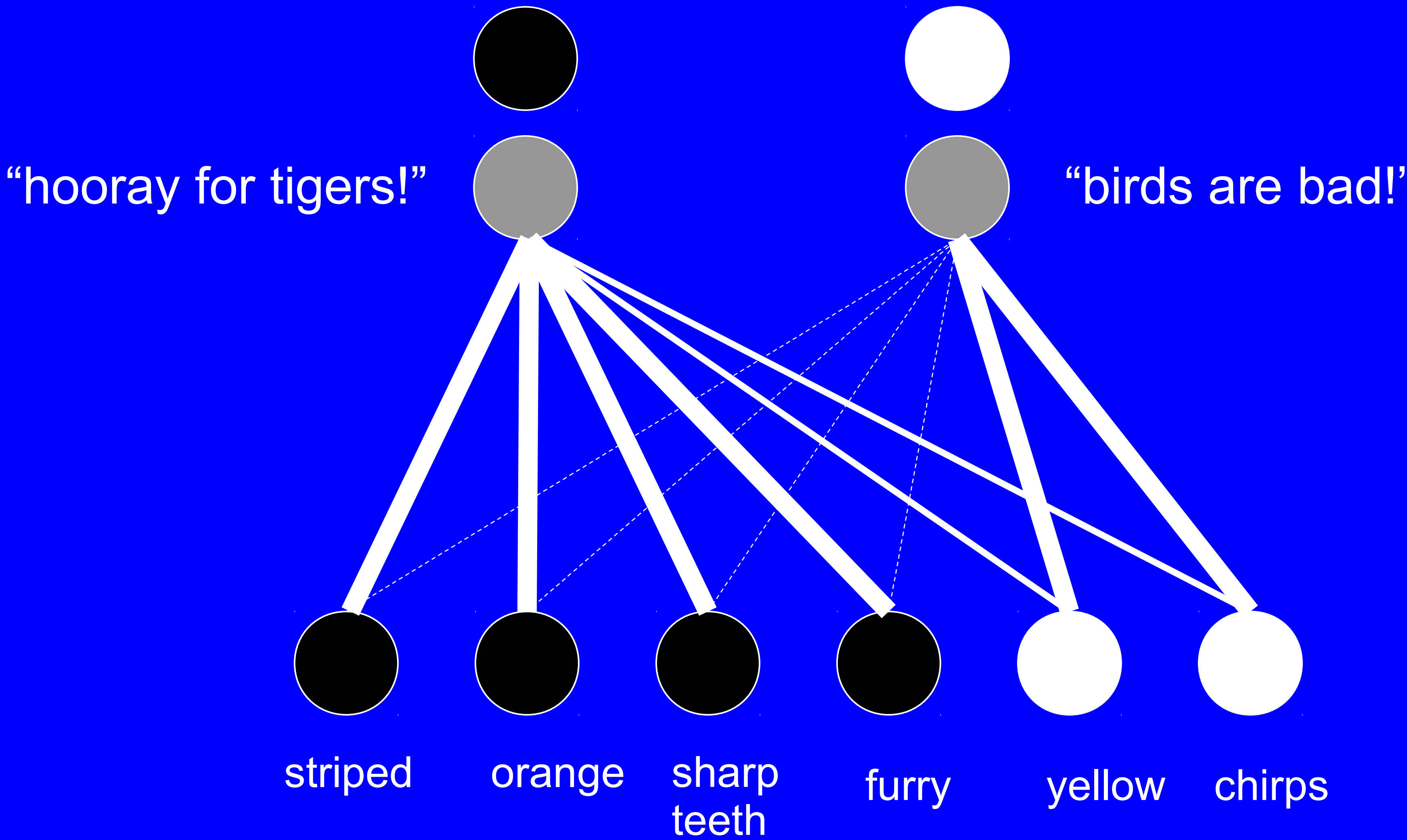
Error-driven learning



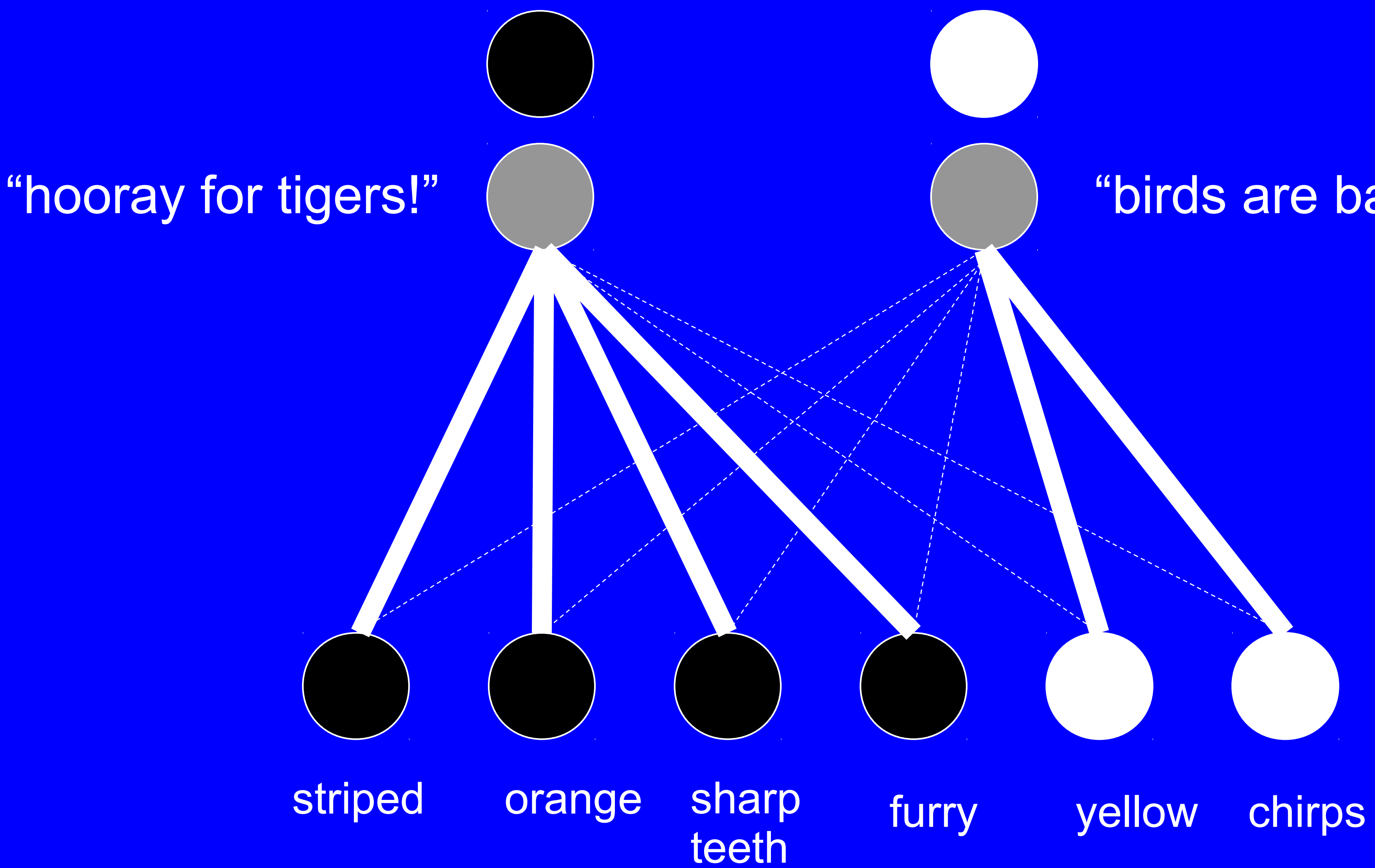
Error-driven learning



Error-driven learning

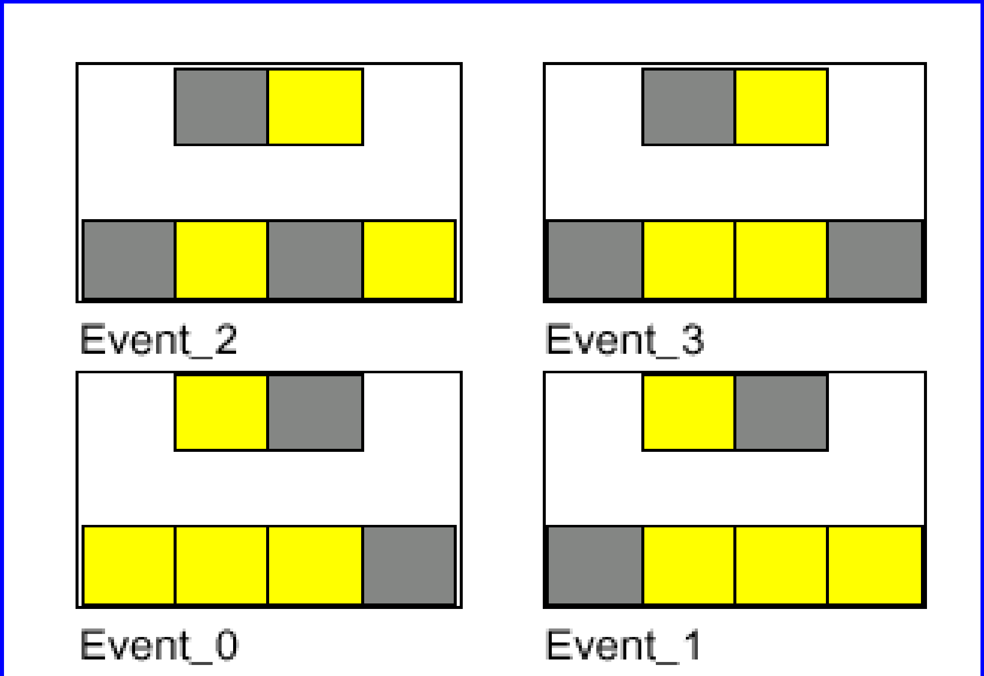


Error-driven learning

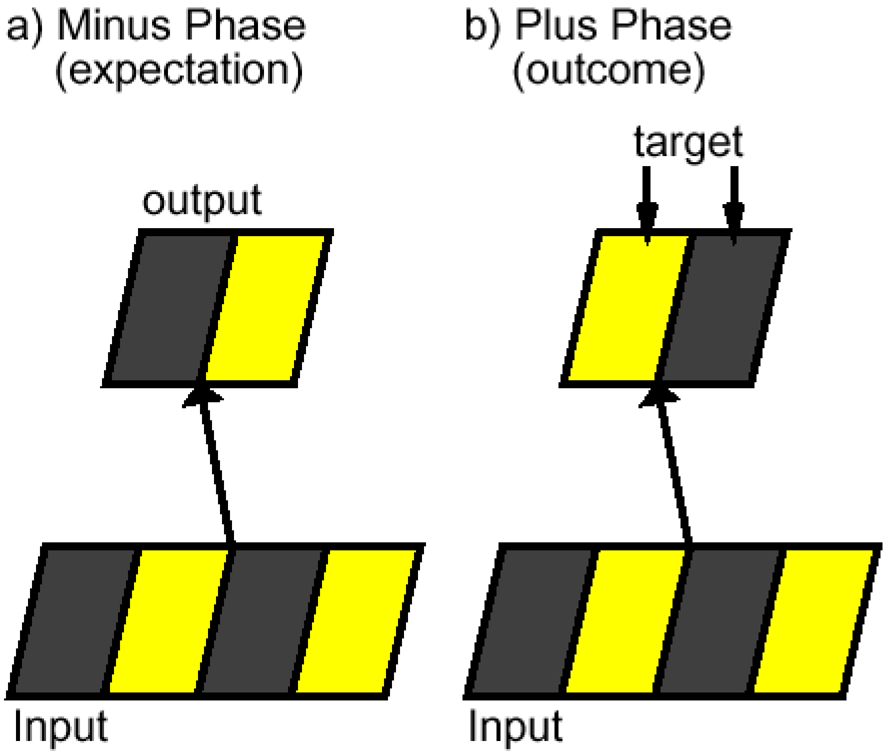


The Delta Rule and the "Hard" Problem

- The delta rule can learn the "hard" mapping that thwarted the Hebb rule



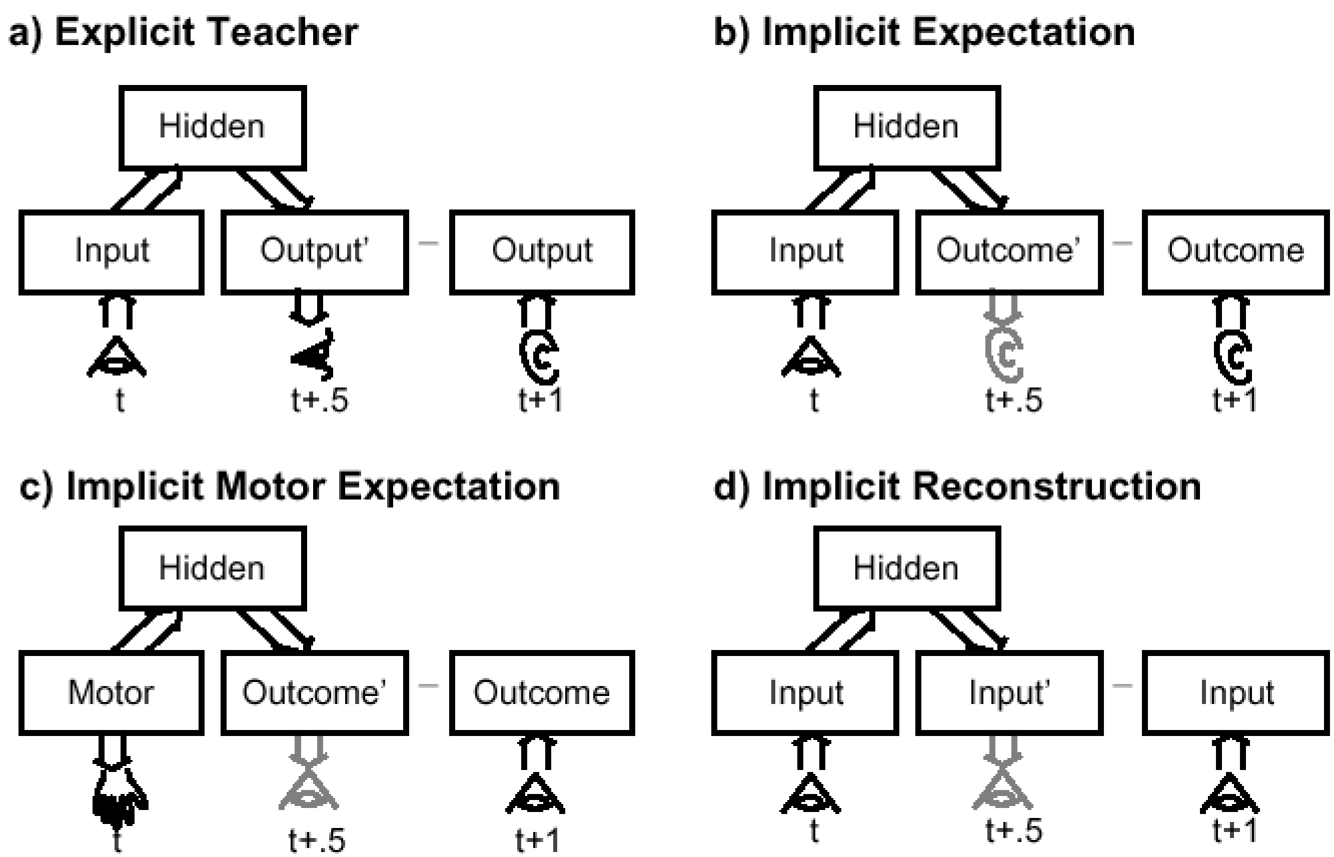
What is Target? Activation Phases



$$\Delta w_{ik} = \epsilon(o_k^+ - o_k^-)s_i$$

[pat_assoc.proj]

Nature of the Training Signals



Soft Weight Bounding

Keep weights bounded between 0-1 by exponentially slowing increases, decreases as they approach bounds:

$$\Delta w_{ik} = [\Delta_{ik}]_{+}(1 - w_{ik}) + [\Delta_{ik}]_{-} w_{ik}$$

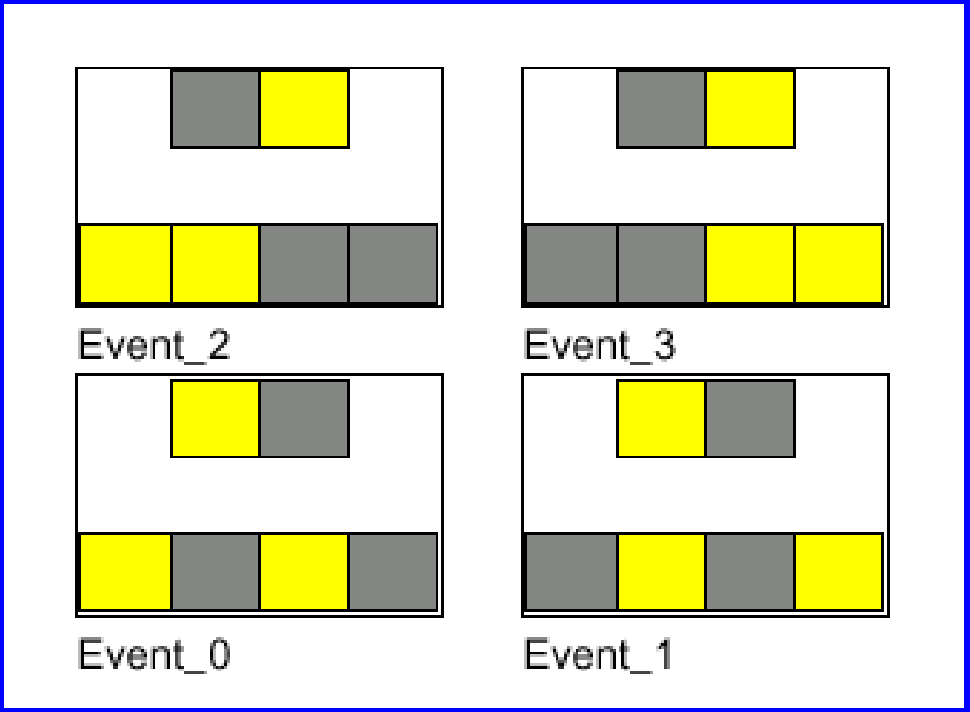
$[\Delta_{ik}]_{+}$ = computed weight change if positive (else 0).

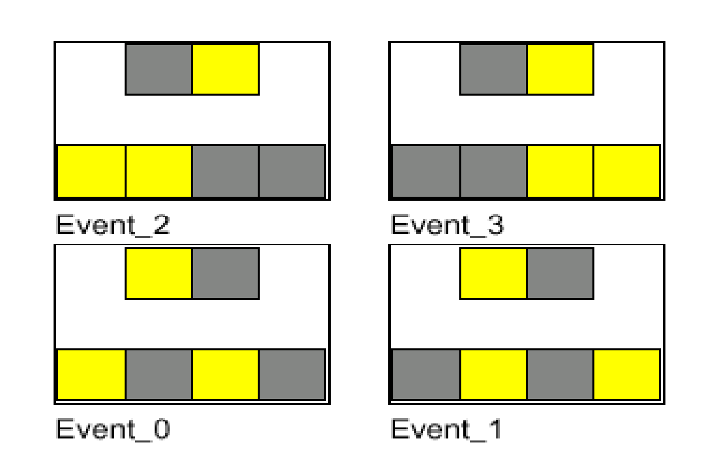
$[\Delta_{ik}]_{-}$ = computed weight change if negative (else 0).

* reflects biological constraints on number of receptors, etc. (weight can only go so high, low)

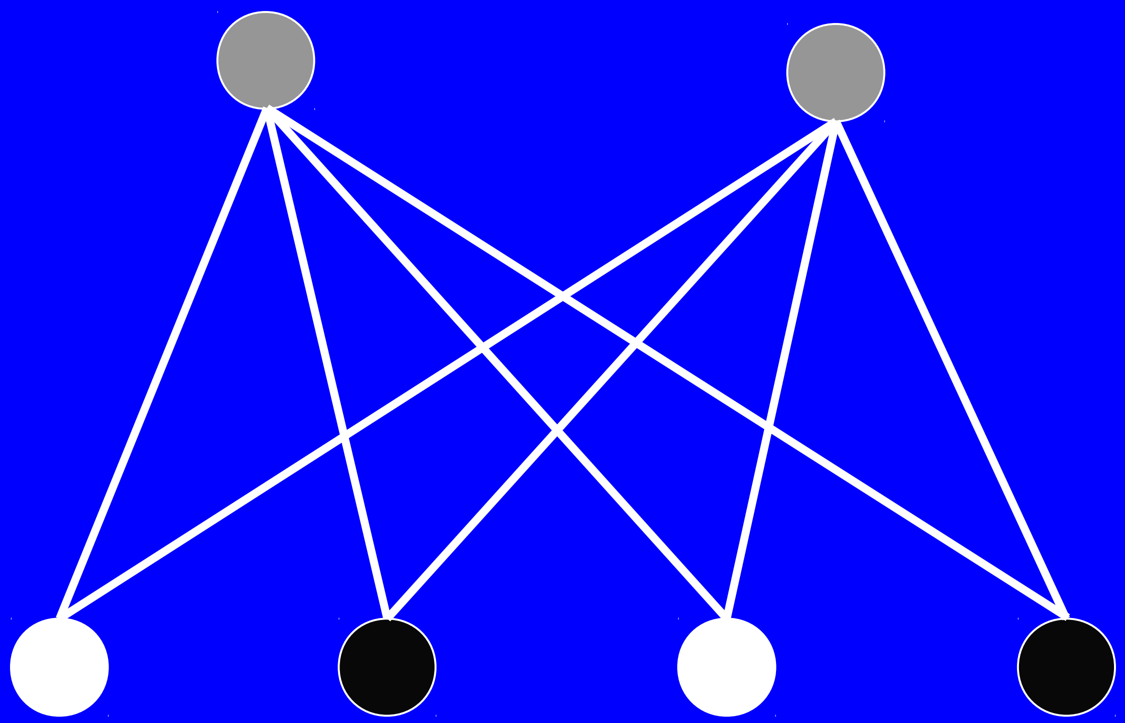
"Impossible" Mapping

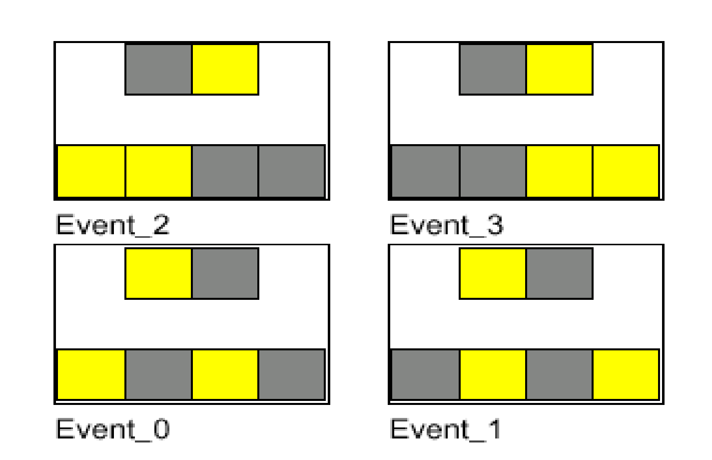
- Each input unit is linked equally often to each output unit
- Two layer networks using the delta rule can not solve this!





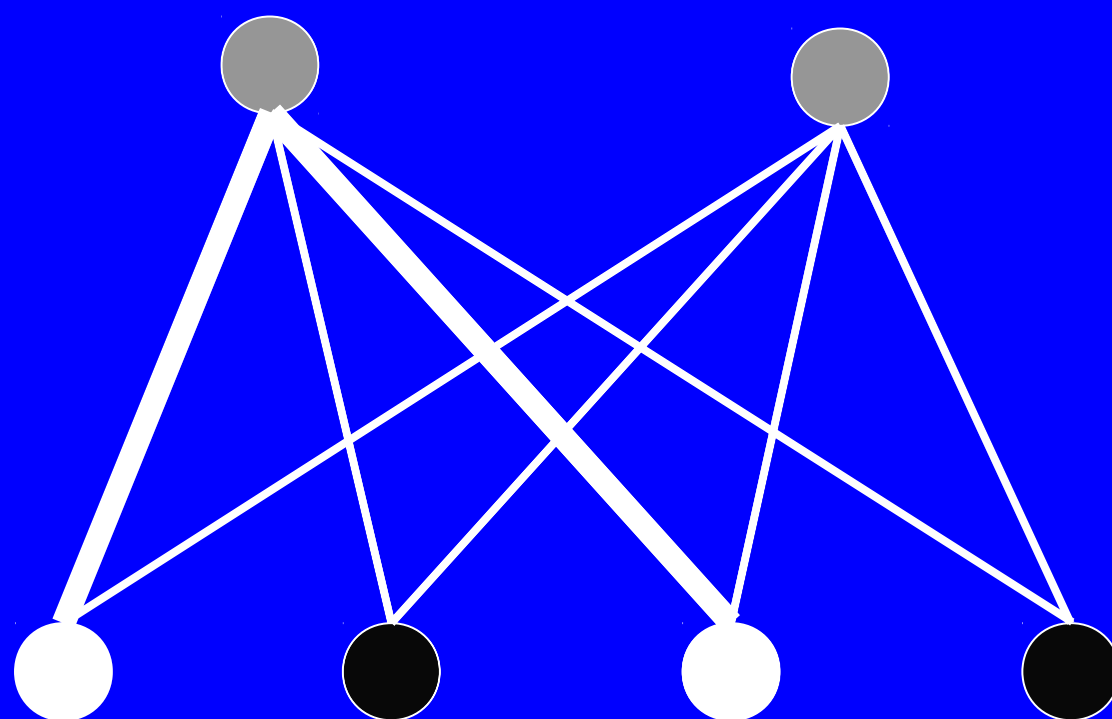
Changing weights to learn Event_0...



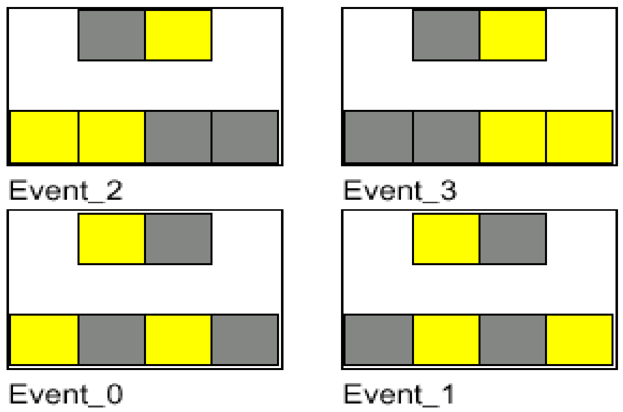


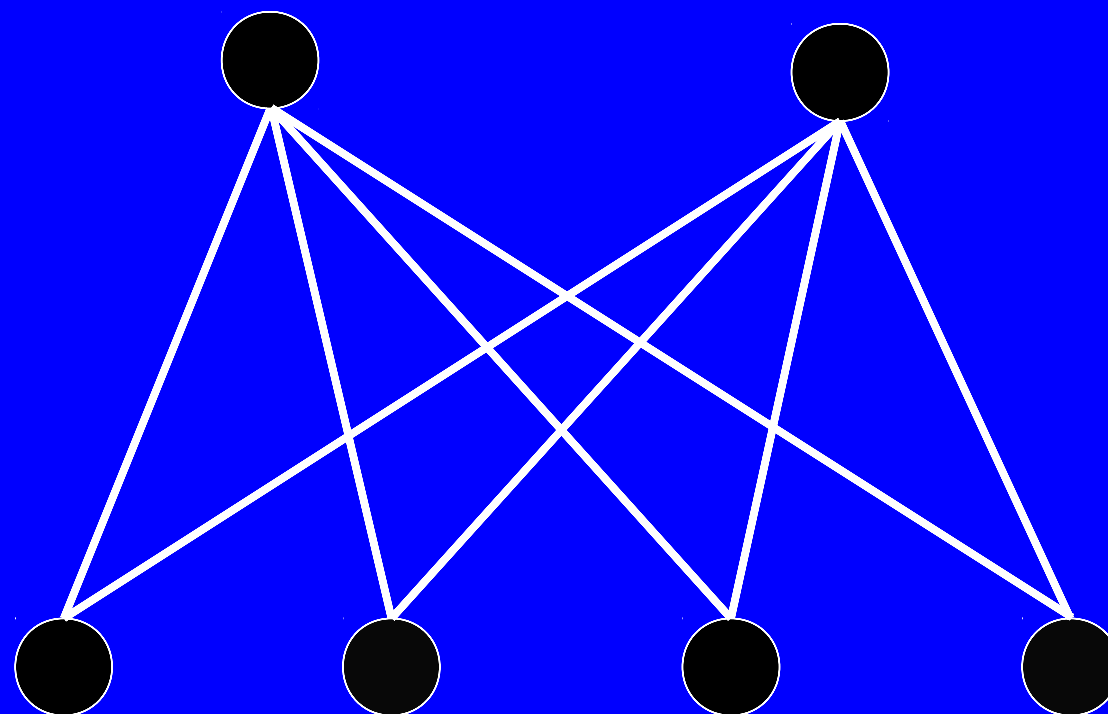
Changing weights to learn Event_0...

... hurts performance for Event_2 and Event_3

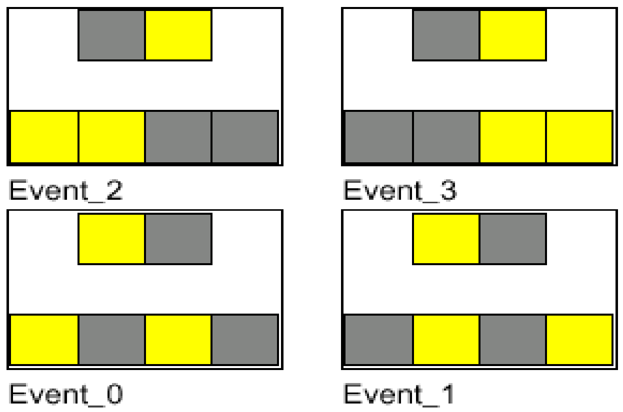


[pat_assoc.proj]

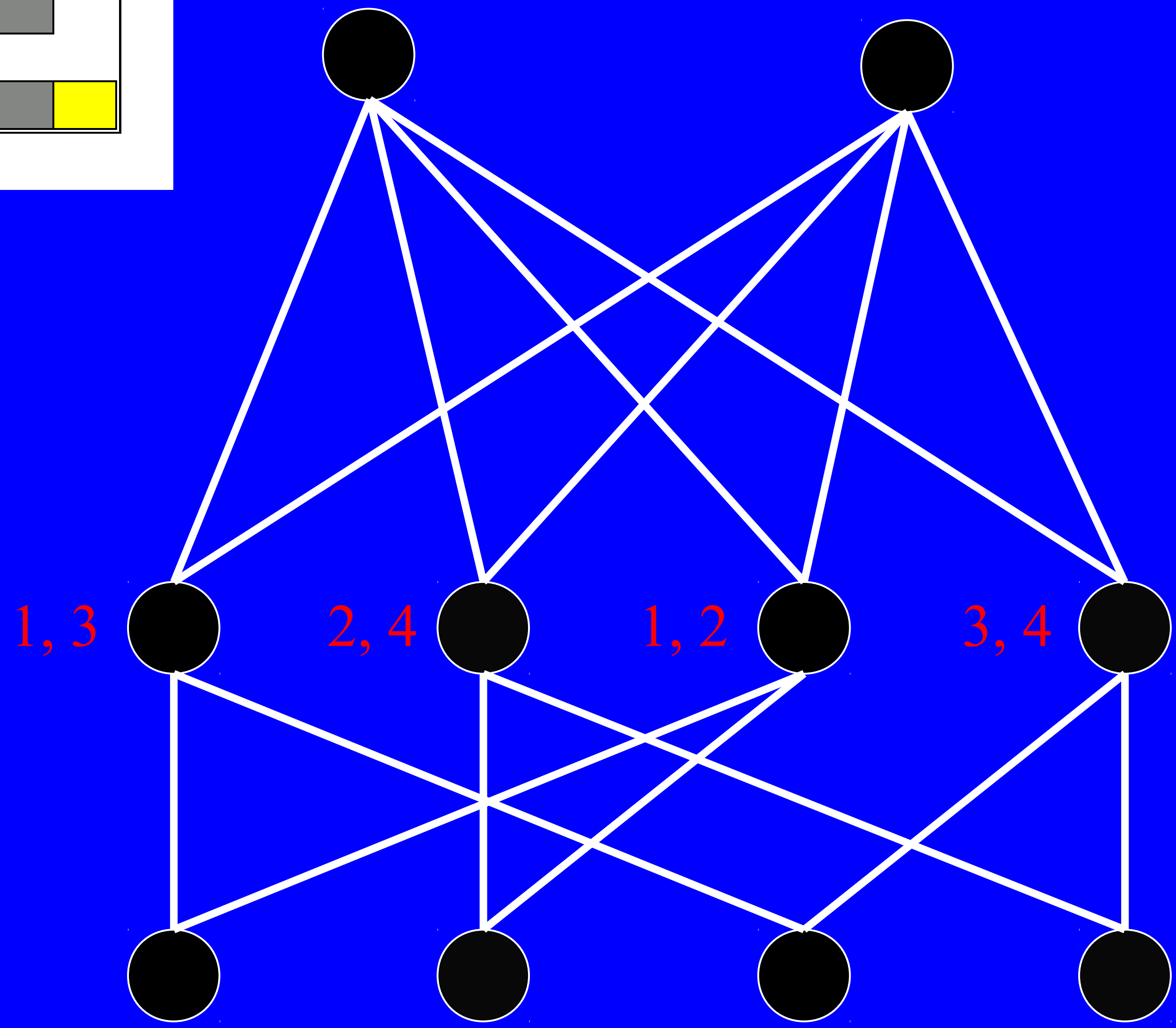

Event_2
Event_3
Event_0
Event_1

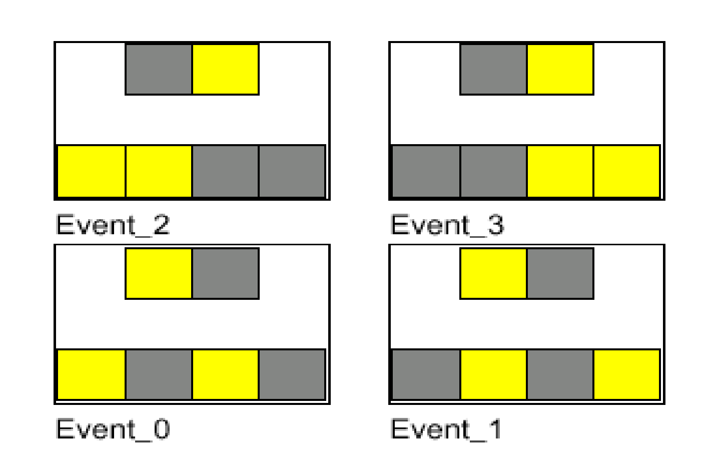


Add a hidden layer that represents feature **conjunctions** ...

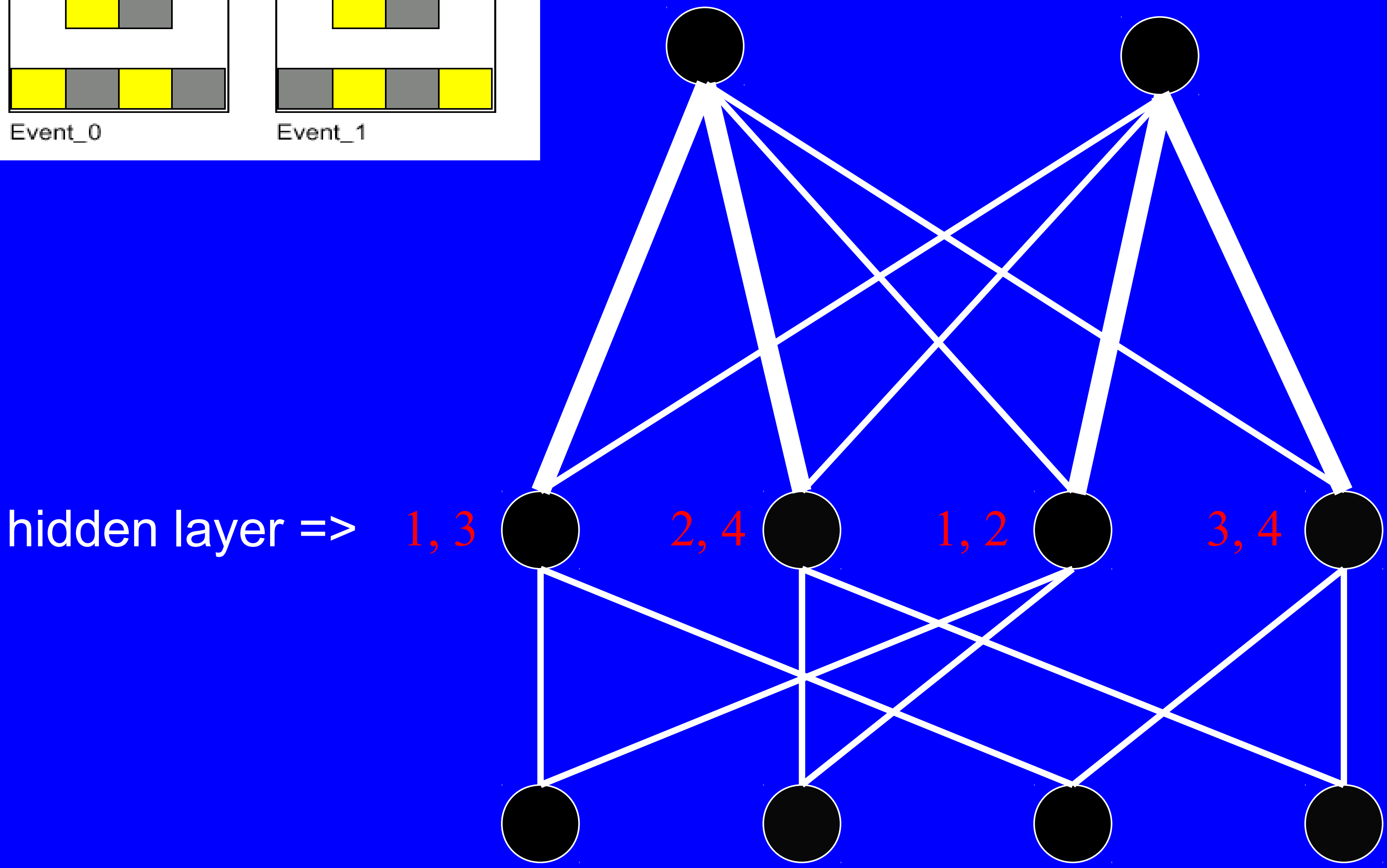


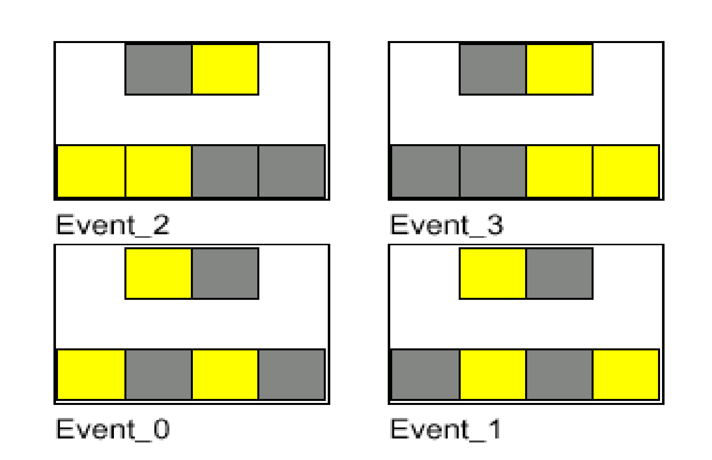
hidden layer =>





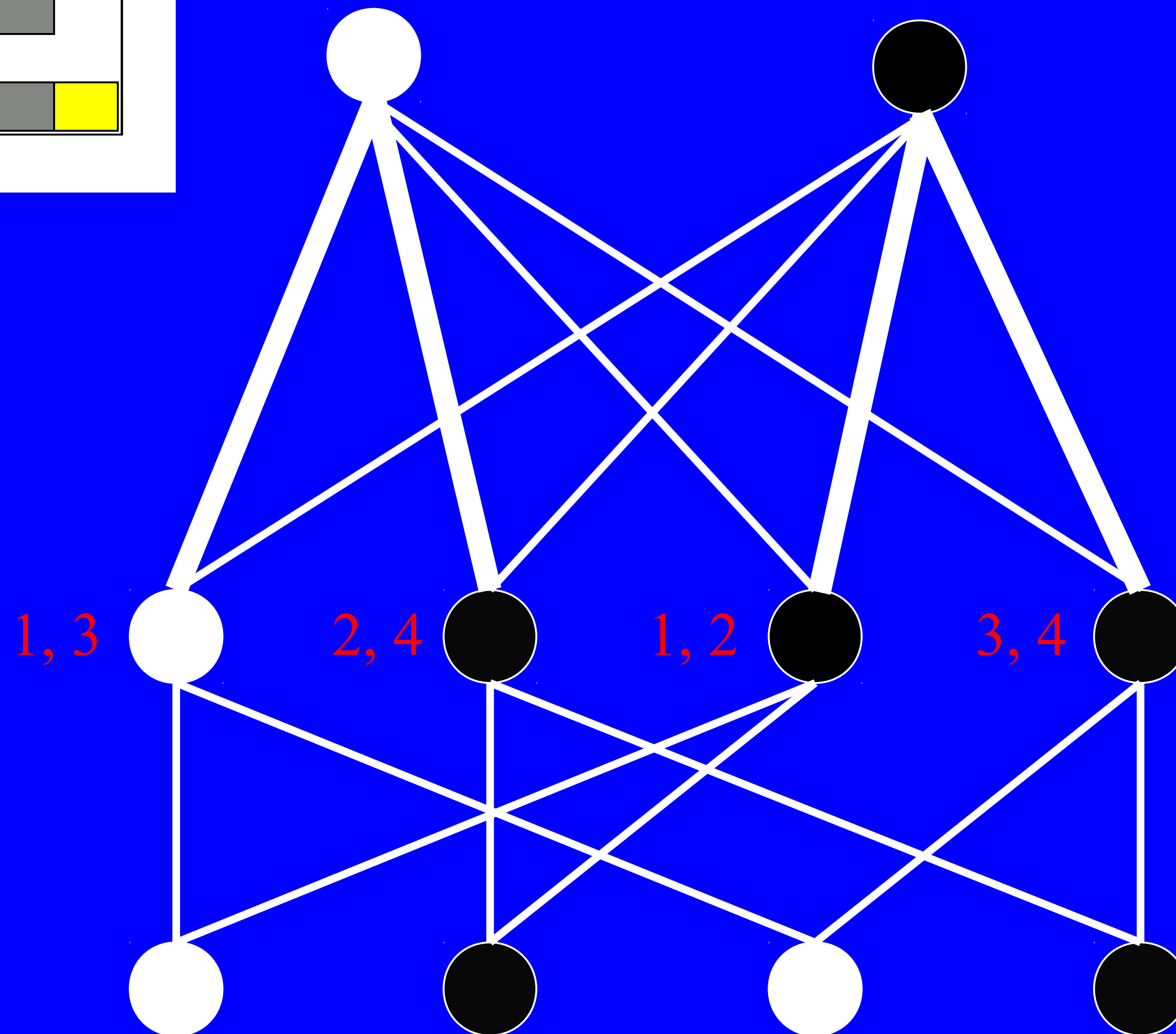
Add a hidden layer that represents feature **conjunctions** ...



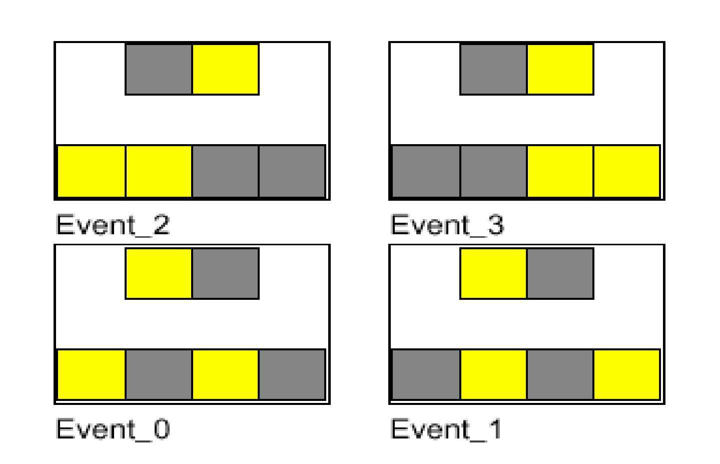


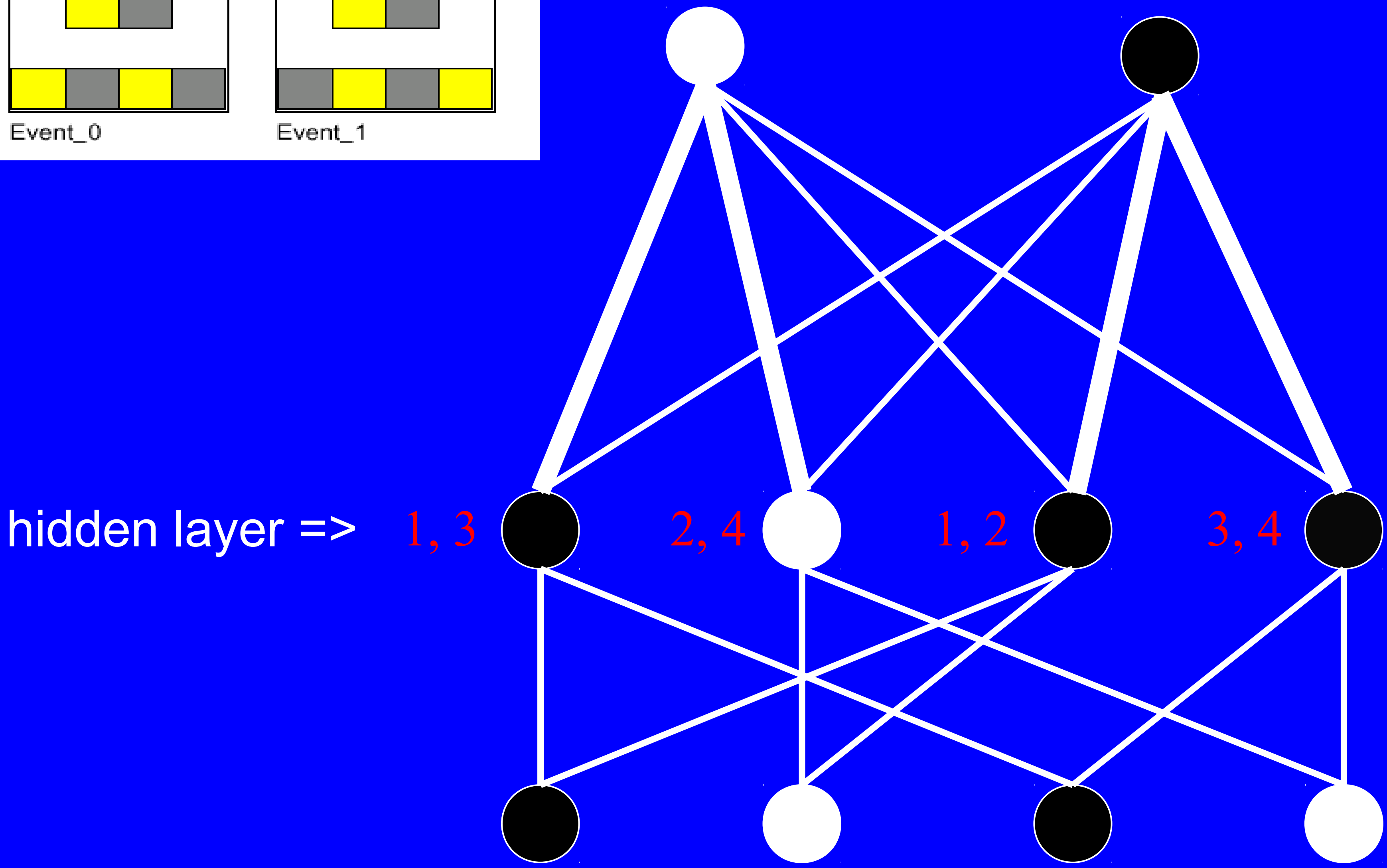
Add a hidden layer that represents feature **conjunctions** ...

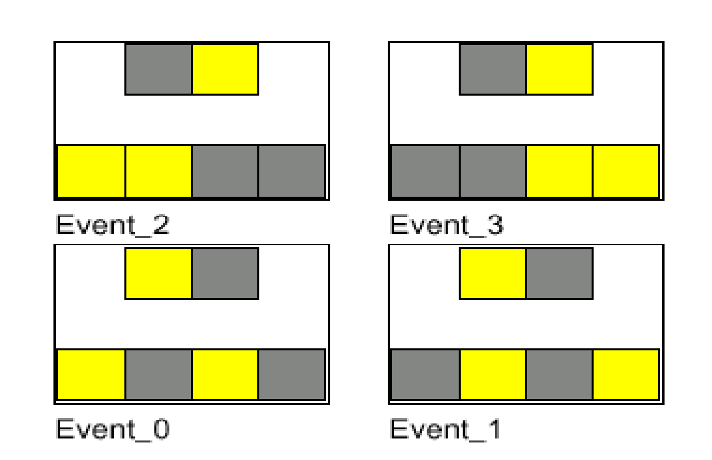
hidden layer =>



Add a hidden layer that represents feature **conjunctions** ...

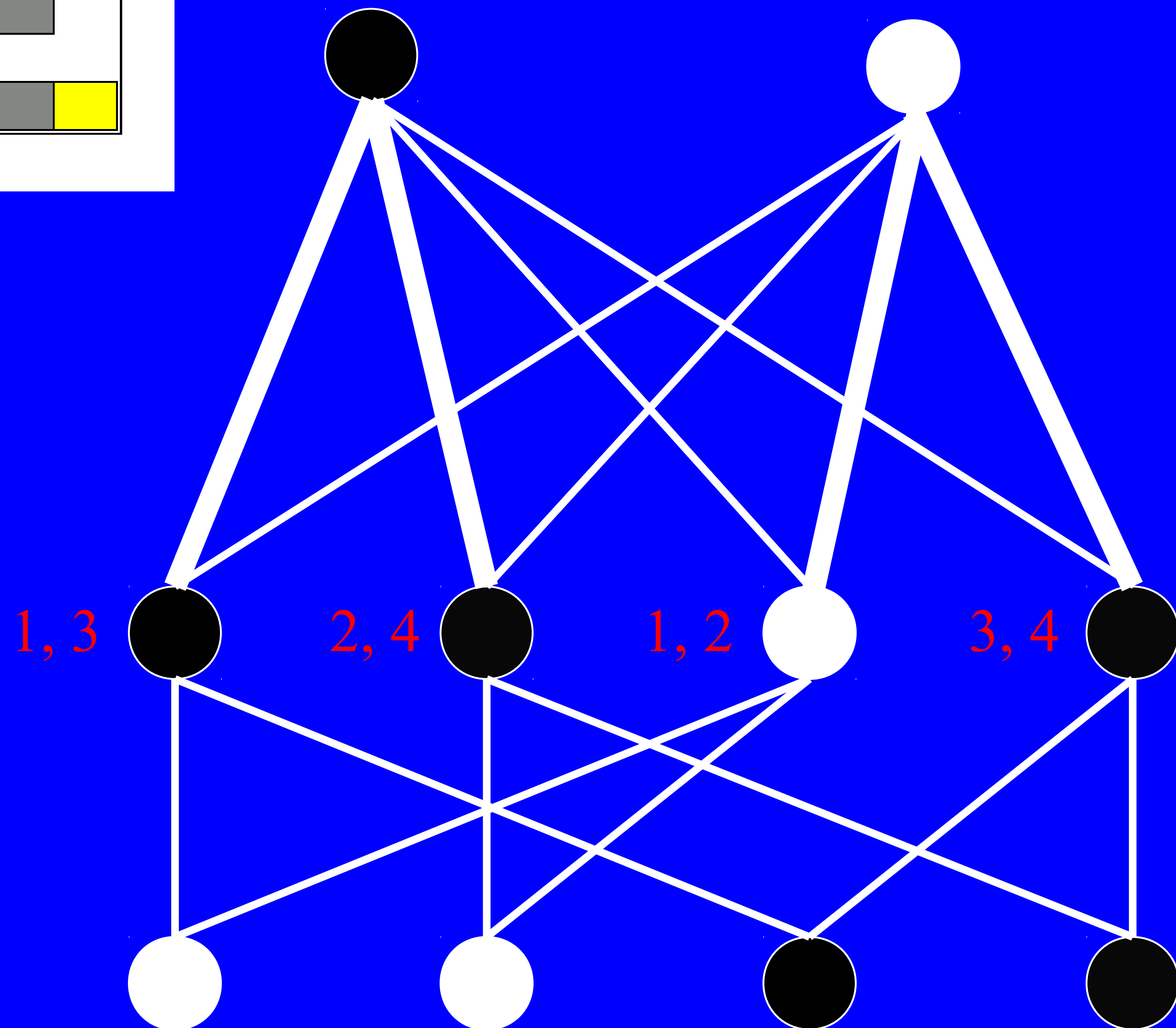


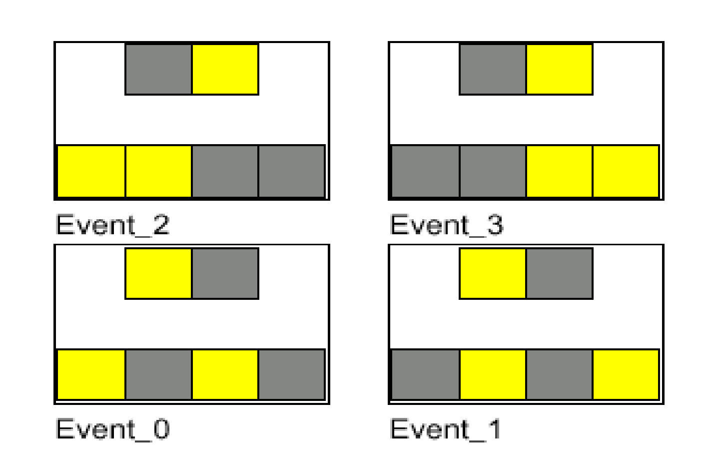




Add a hidden layer that represents feature **conjunctions** ...

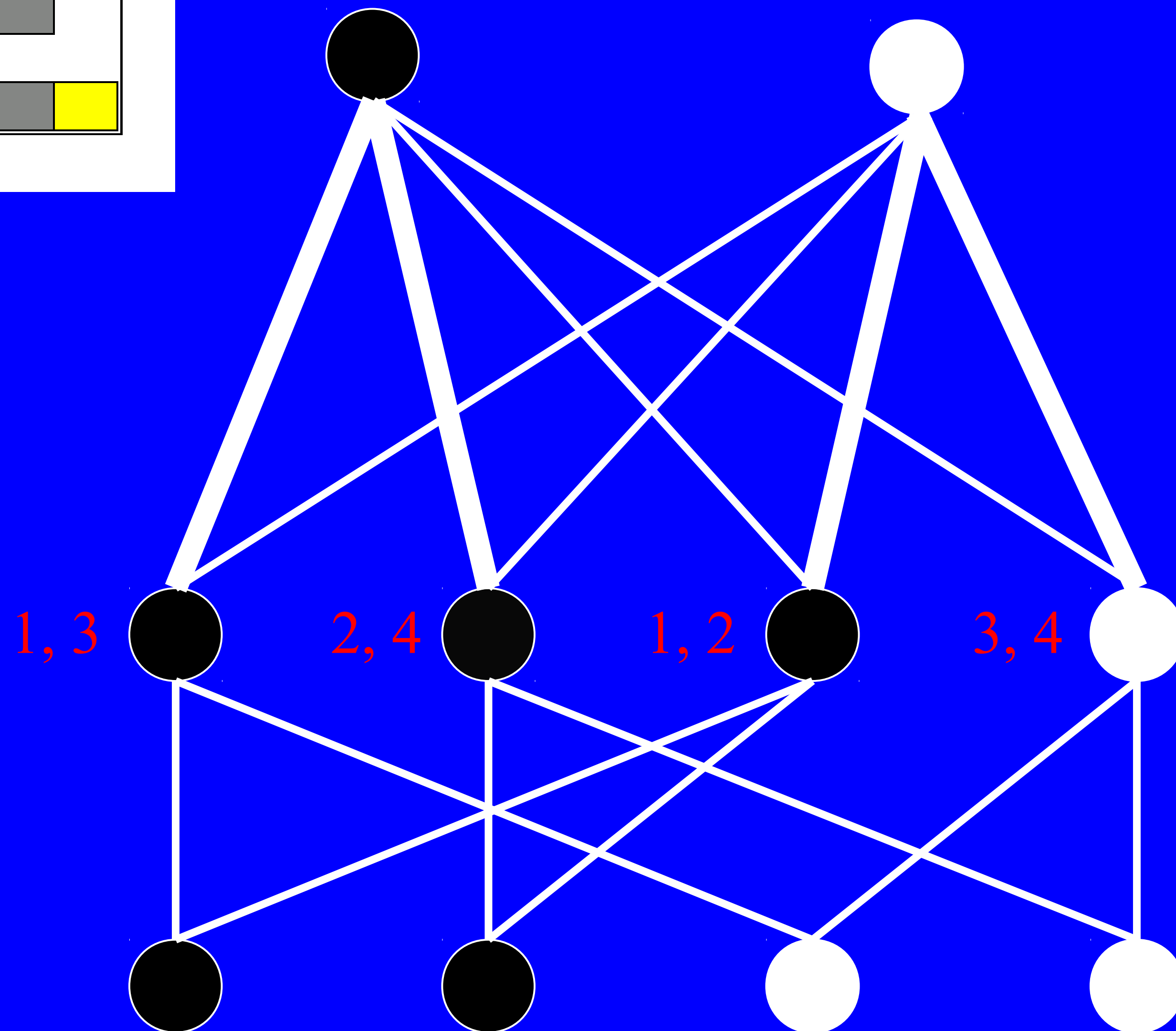
hidden layer =>





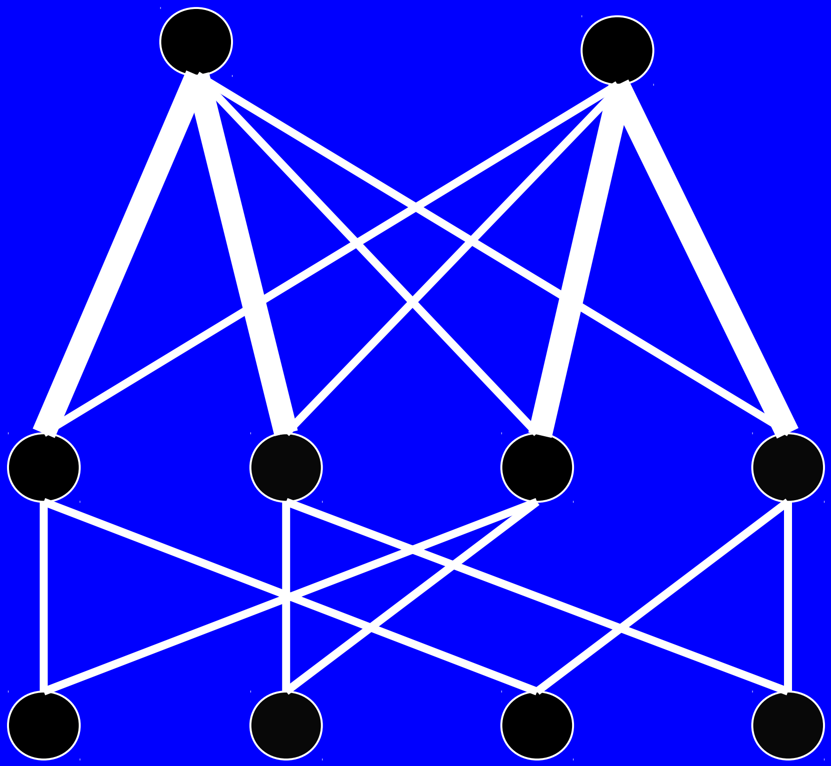
Add a hidden layer that represents feature **conjunctions** ...

hidden layer =>

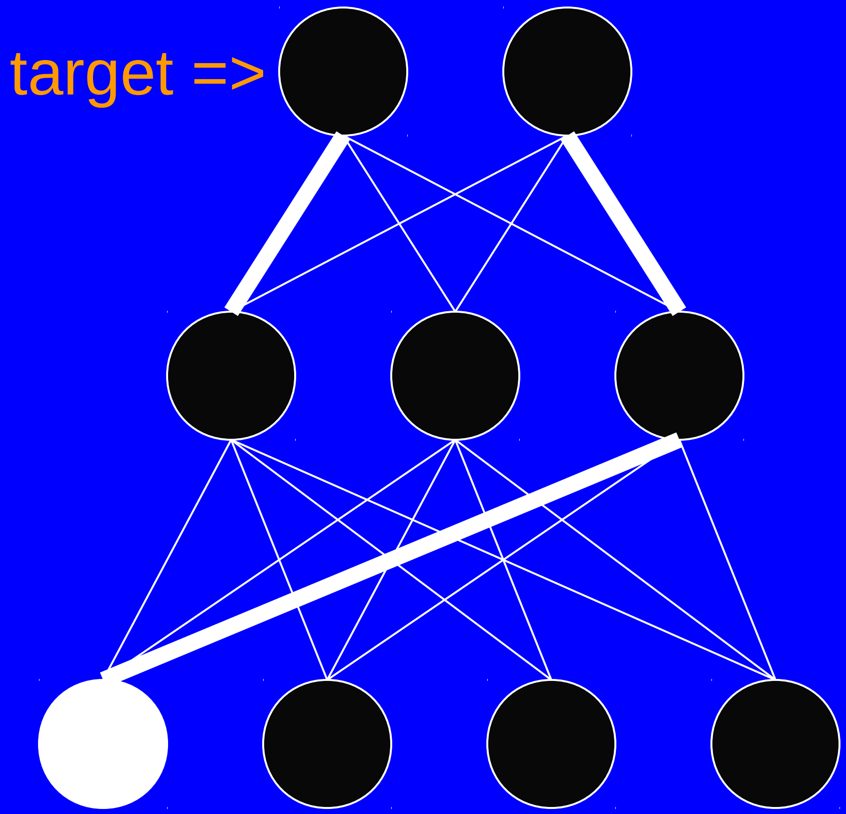


Error-Driven Learning in Multilayer Networks

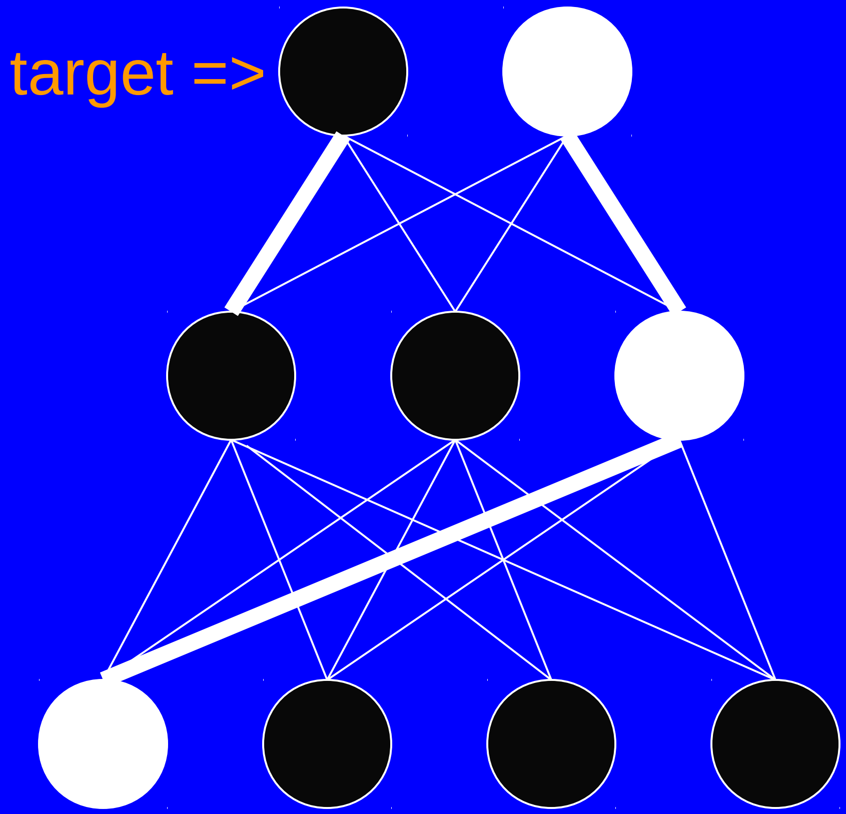
- We established that networks with hidden layers can solve problems that two-layer networks can not solve, by **re-representing the input patterns**
- How do we train multi-layer networks?



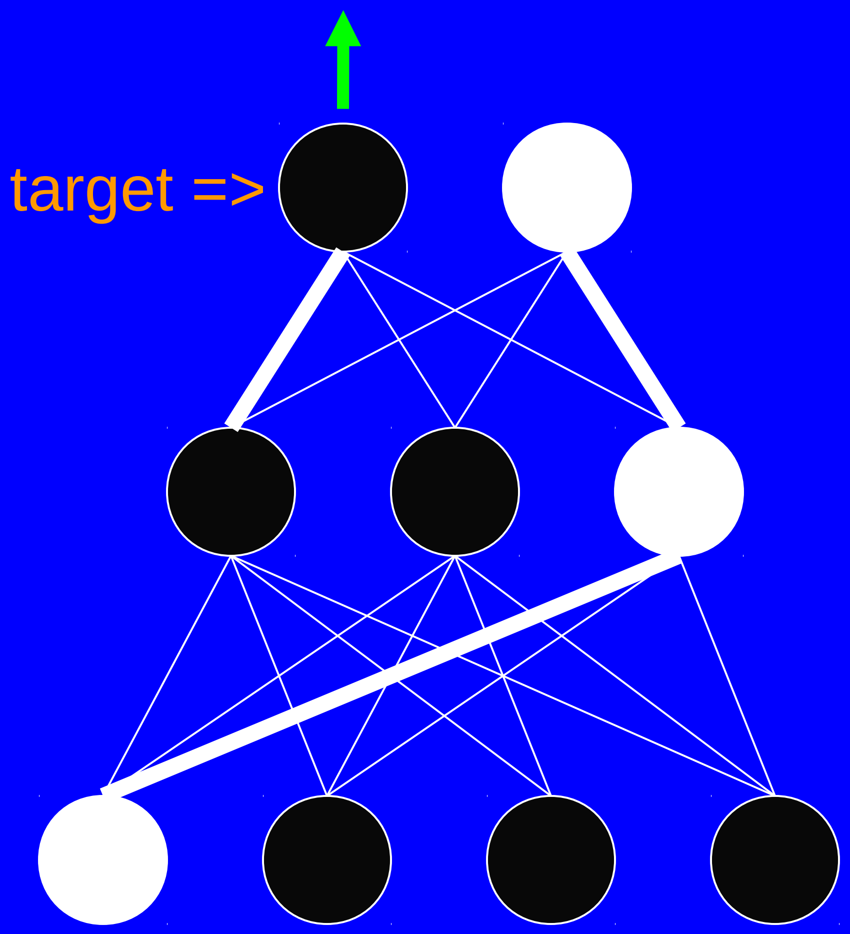
Learning in Multilayer Networks



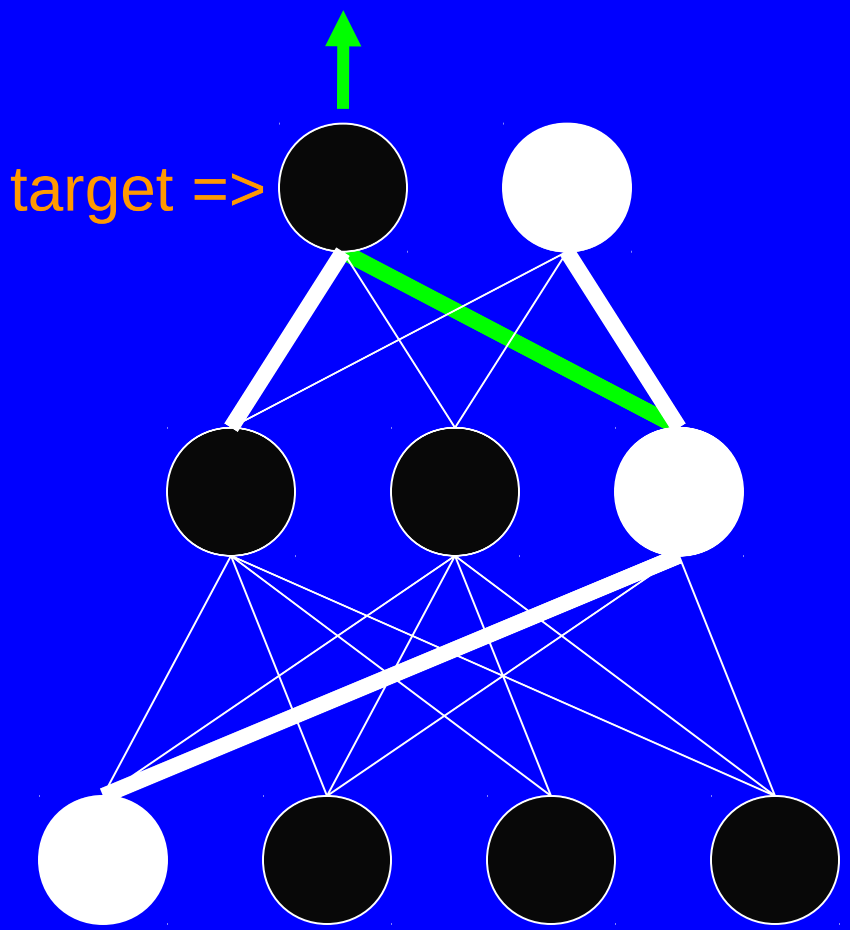
Learning in Multilayer Networks



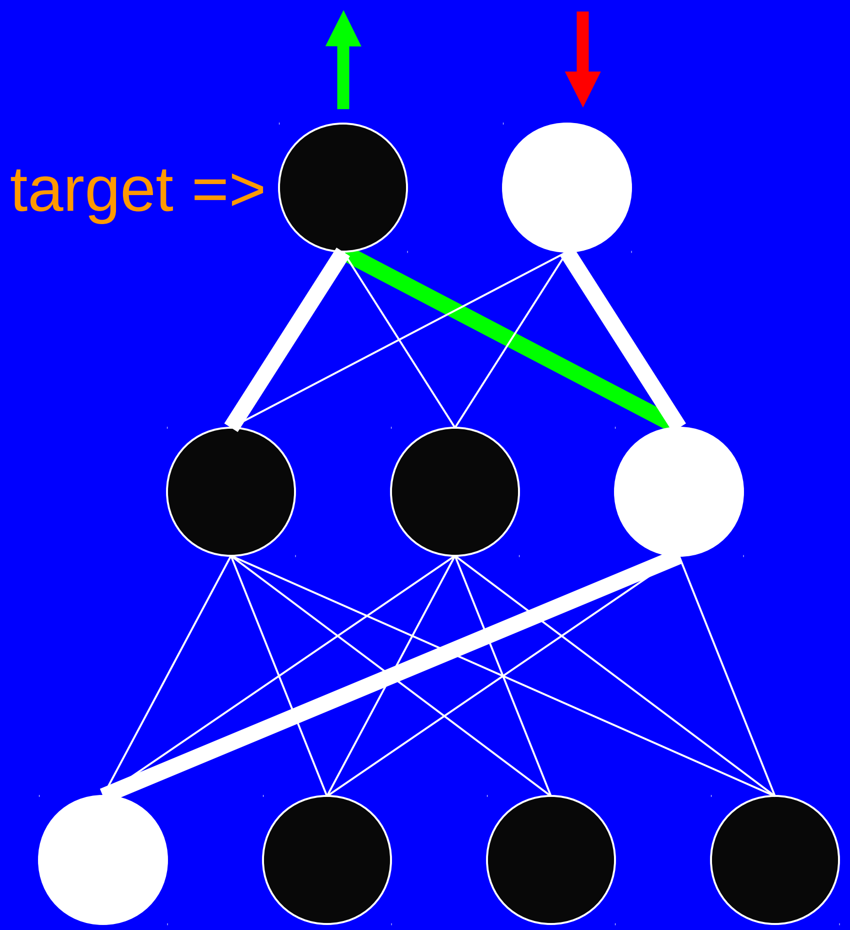
Learning in Multilayer Networks



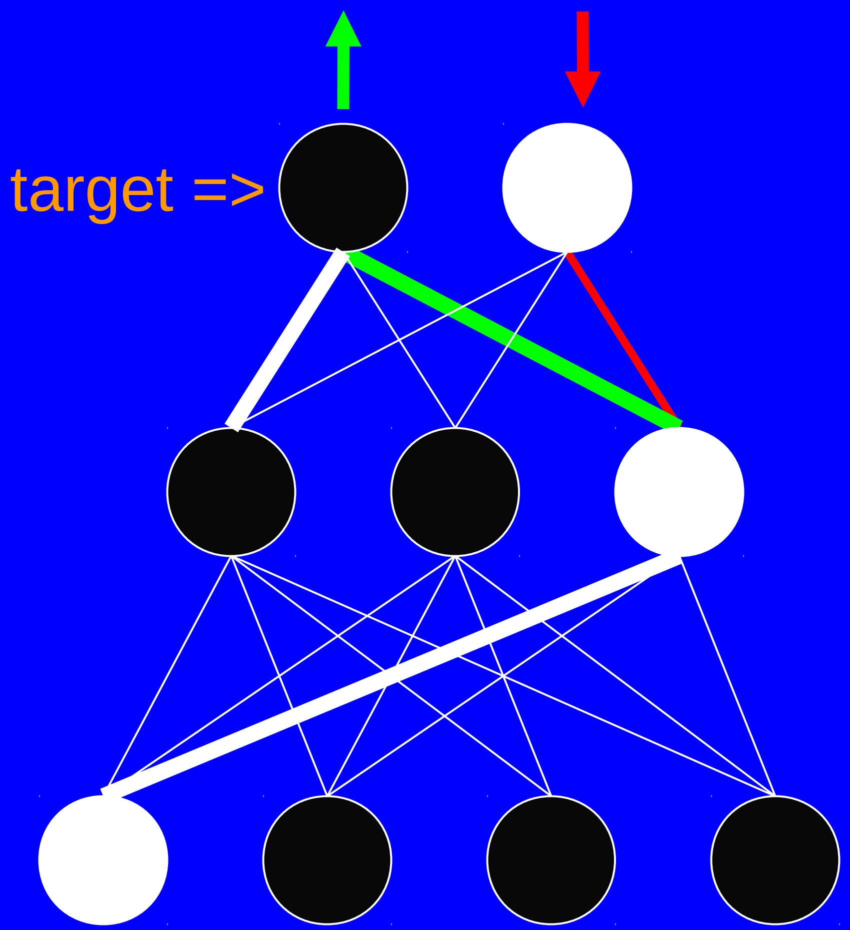
Learning in Multilayer Networks



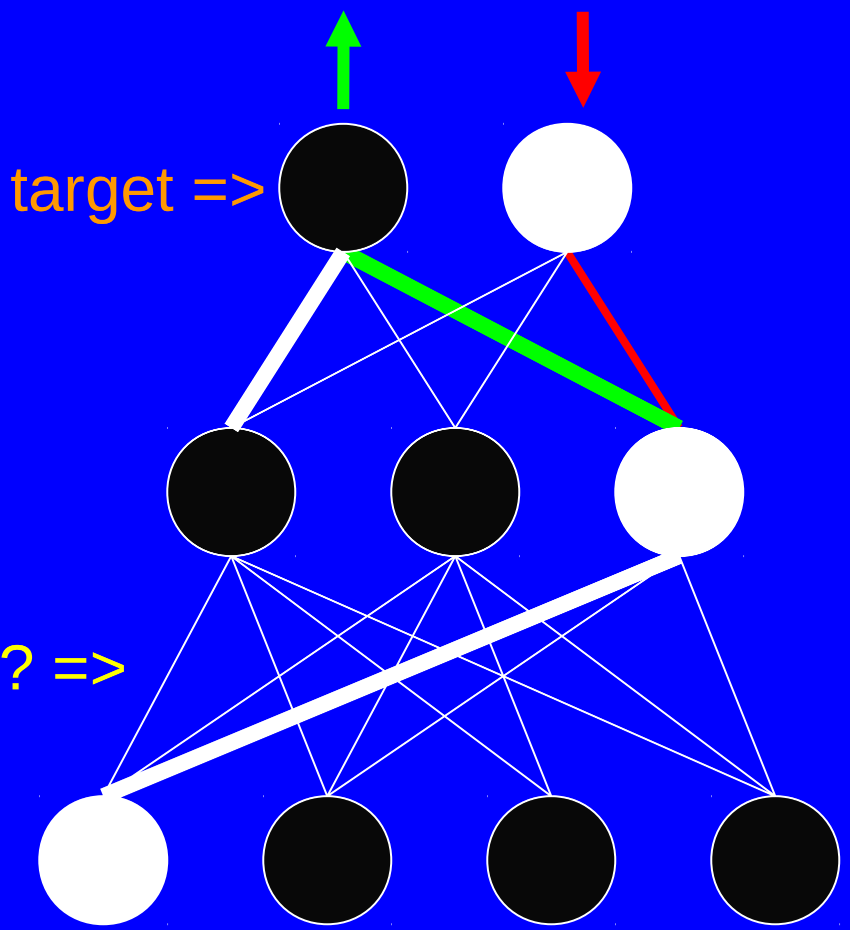
Learning in Multilayer Networks



Learning in Multilayer Networks

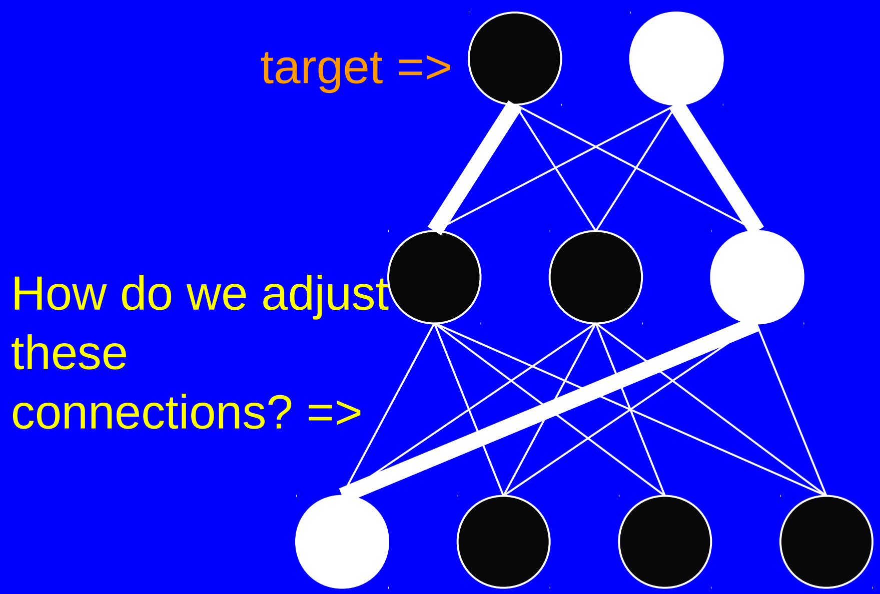


Learning in Multilayer Networks



How do we
adjust these
connections? =>

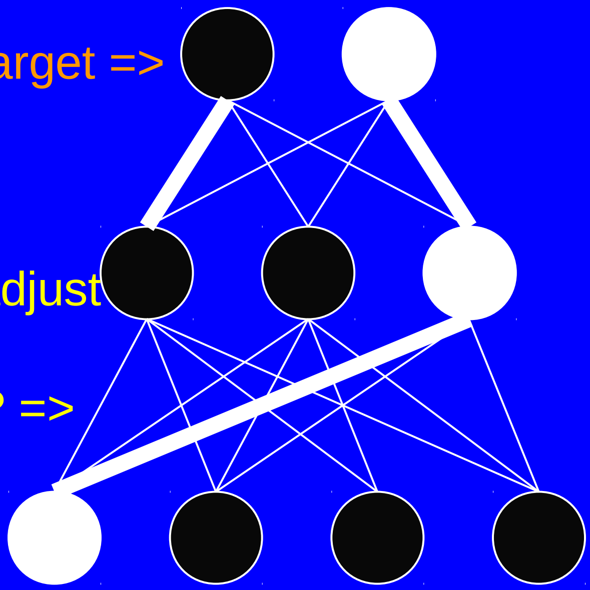
Learning in Multilayer Networks



Intuitively, you want to boost the activity of the middle guys that are well connected to the target unit

Learning in Multilayer Networks

target =>



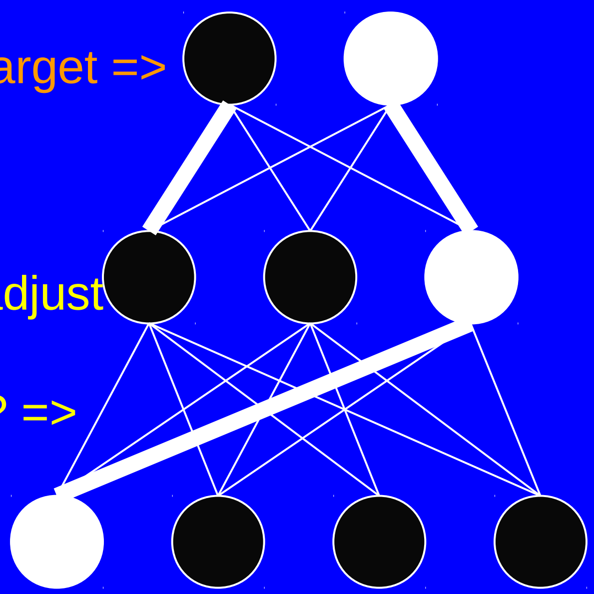
How do we adjust these connections? =>

Intuitively, you want to boost the activity of the middle guys that are well connected to the target unit

How do we identify units that are well connected to the target unit?

Learning in Multilayer Networks

target =>



How do we adjust these connections? =>

Intuitively, you want to boost the activity of the middle guys that are well connected to the target unit

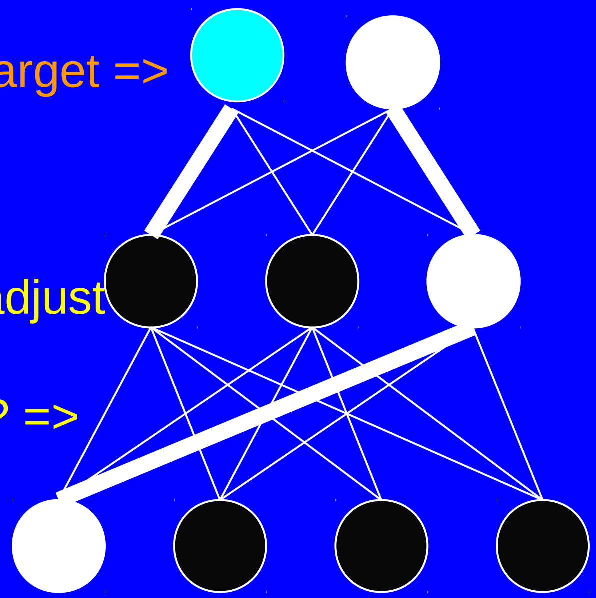
How do we identify units that are well connected to the target unit?

Solution: Propagate activity backwards from the target

Learning in Multilayer Networks

target =>

How do we adjust these connections? =>

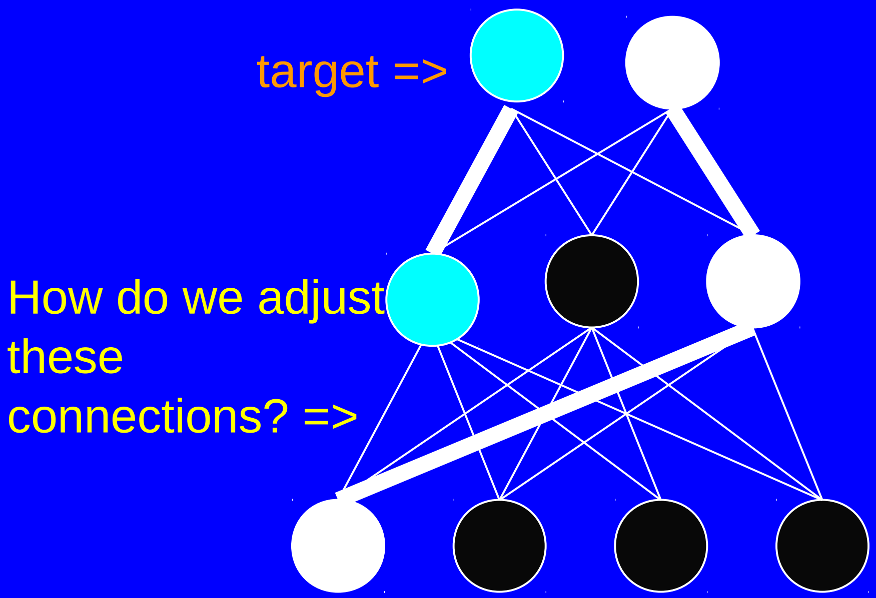


Intuitively, you want to boost the activity of the middle guys that are well connected to the target unit

How do we identify units that are well connected to the target unit?

Solution: Propagate activity backwards from the target

Learning in Multilayer Networks

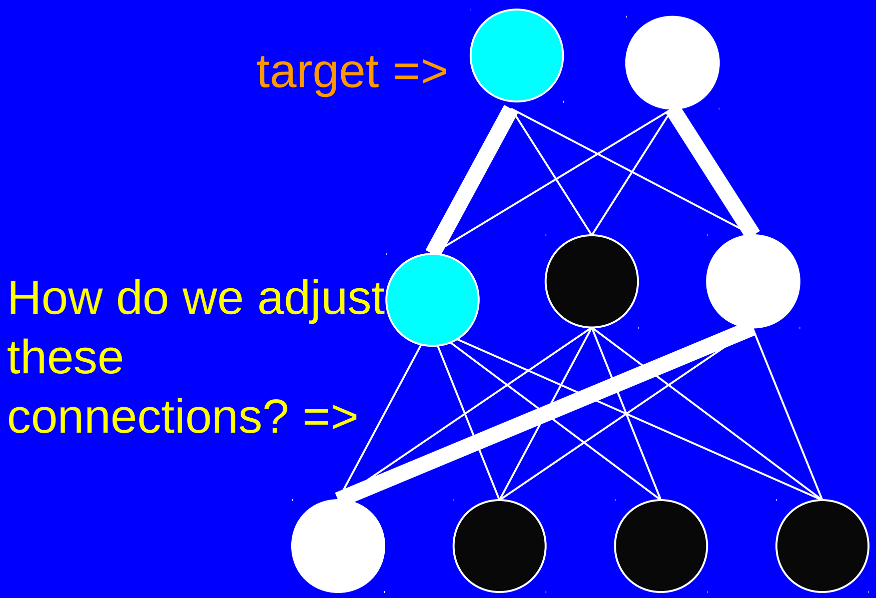


Intuitively, you want to boost the activity of the middle guys that are well connected to the target unit

How do we identify units that are well connected to the target unit?

Solution: Propagate activity backwards from the target

Learning in Multilayer Networks



Intuitively, you want to boost the activity of the middle guys that are well connected to the target unit

How do we identify units that are well connected to the target unit?

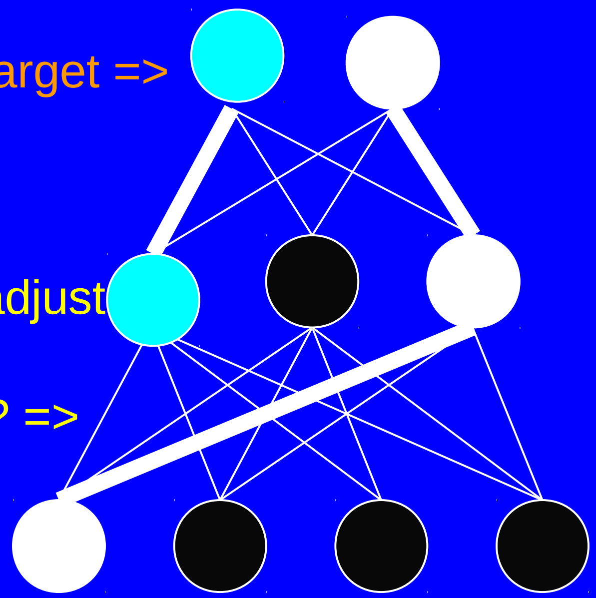
Solution: Propagate activity backwards from the target

Intuition: Backward-spreading activity from the target can help us identify *pathways* to the target (if weights are symmetric)

Learning in Multilayer Networks

target =>

How do we adjust these connections? =>



Intuitively, you want to boost the activity of the middle guys that are well connected to the target unit

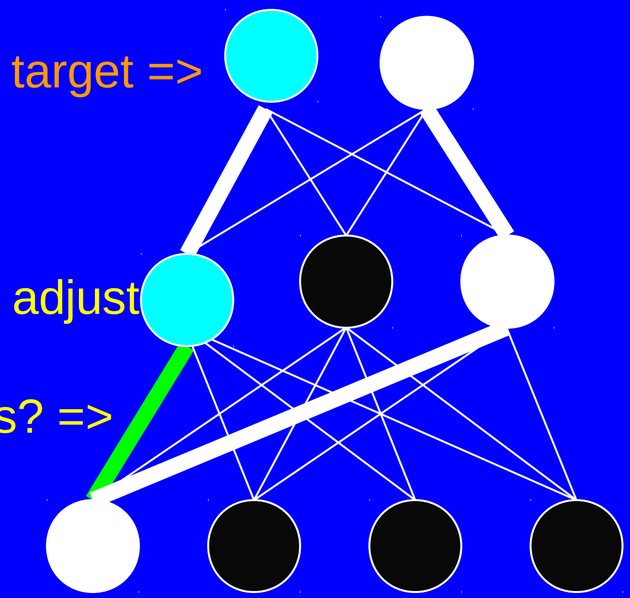
How do we identify units that are well connected to the target unit?

Solution: Propagate activity backwards from the target

Intuition: Backward-spreading activity from the target can help us identify *pathways* to the target (if weights are symmetric)

Then: change weights to strengthen these pathways

Learning in Multilayer Networks



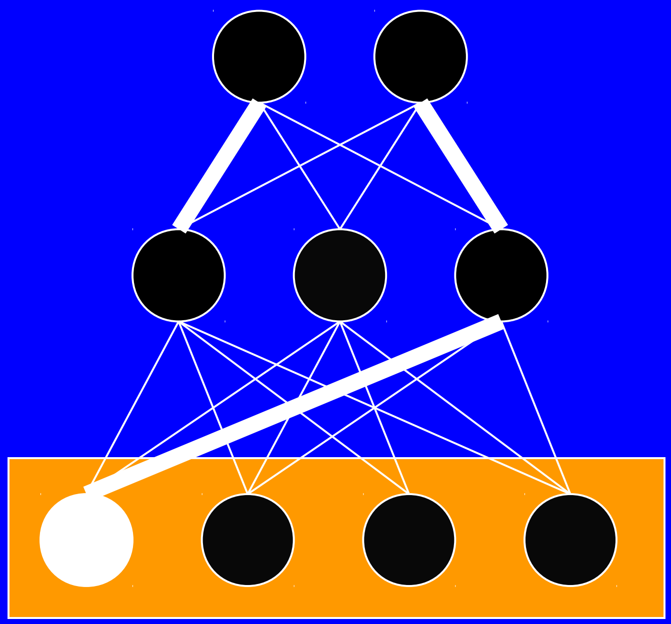
Intuitively, you want to boost the activity of the middle guys that are well connected to the target unit

How do we identify units that are well connected to the target unit?

Solution: Propagate activity backwards from the target
Intuition: Backward-spreading activity from the target can help us identify *pathways* to the target (if weights are symmetric)
Then: change weights to strengthen these pathways

GeneRec Learning Rule

Compare two conditions:

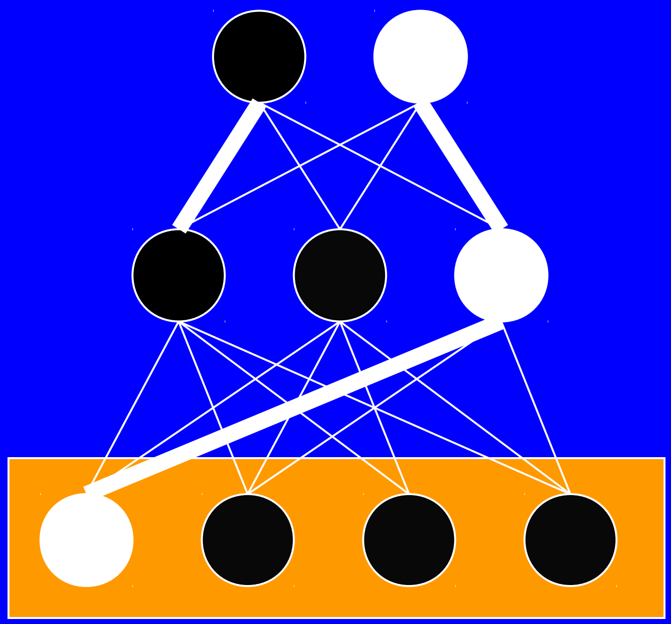


Minus Phase:

Clamp input

GeneRec Learning Rule

Compare two conditions:

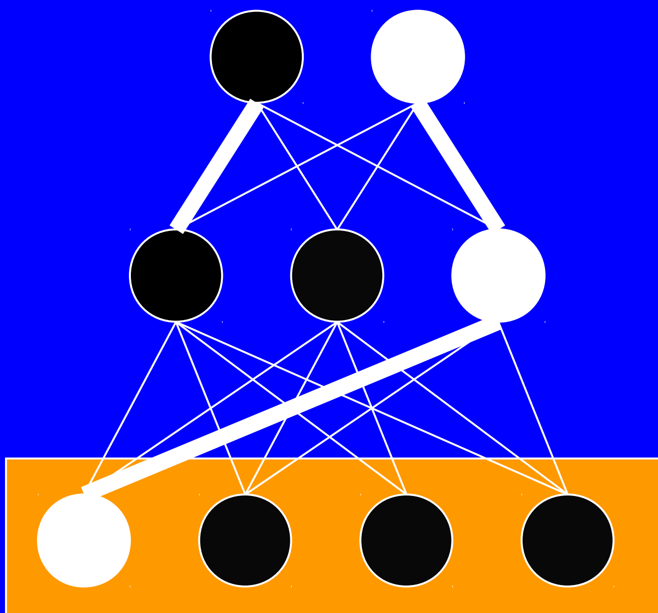


Minus Phase:

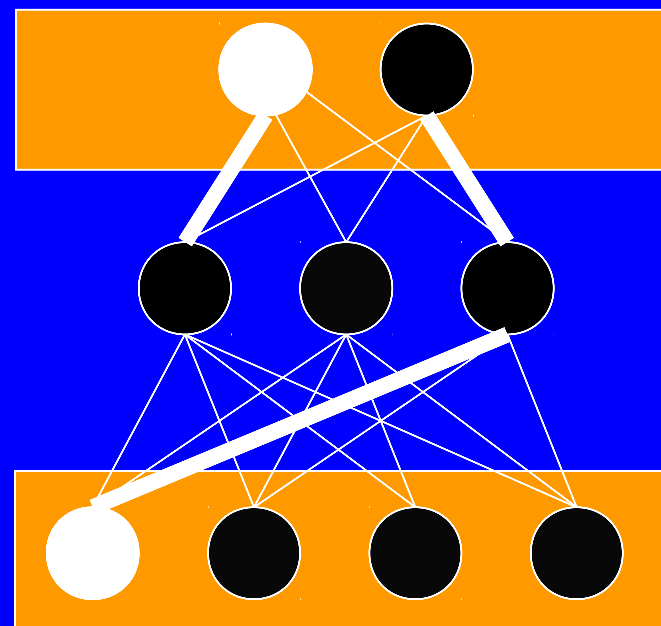
Clamp input

GeneRec Learning Rule

Compare two conditions:



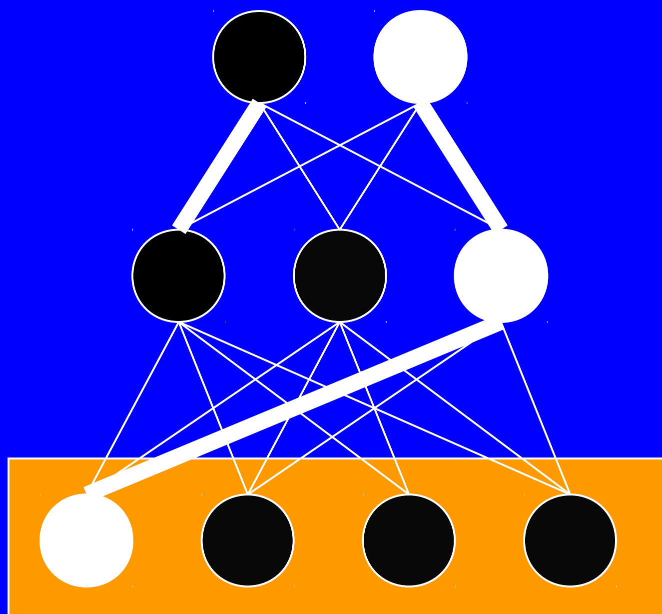
Minus Phase:
Clamp input



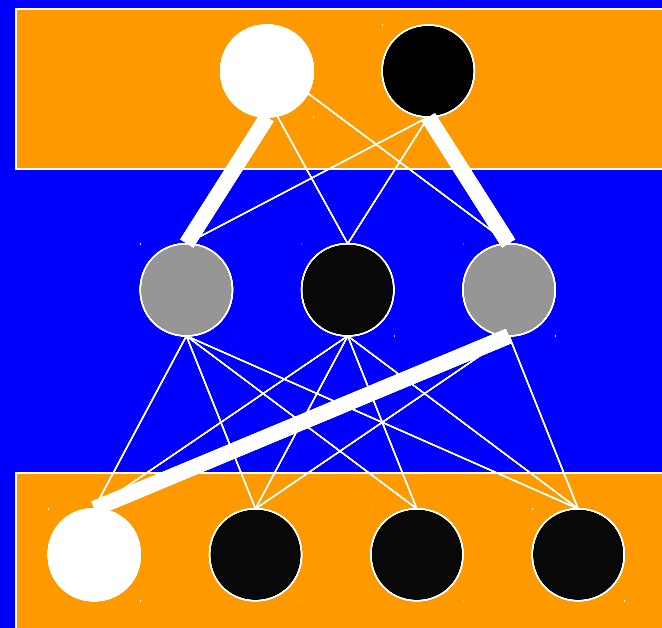
Plus Phase:
Clamp input and target output

GeneRec Learning Rule

Compare two conditions:



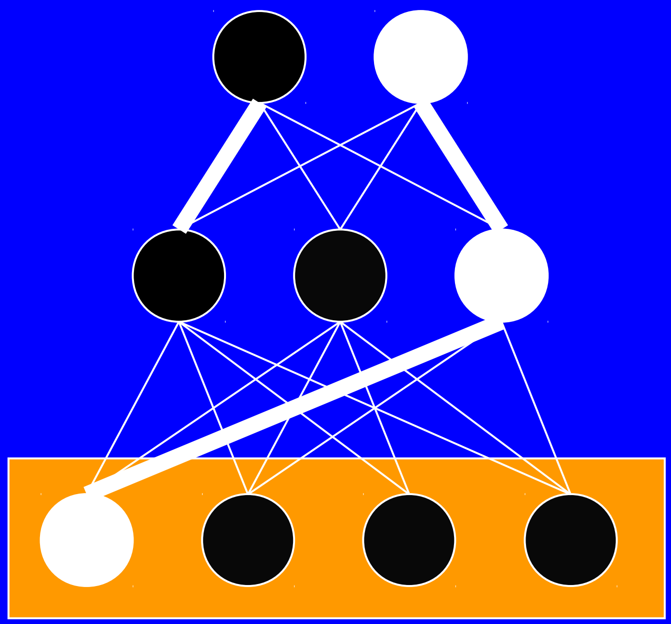
Minus Phase:
Clamp input



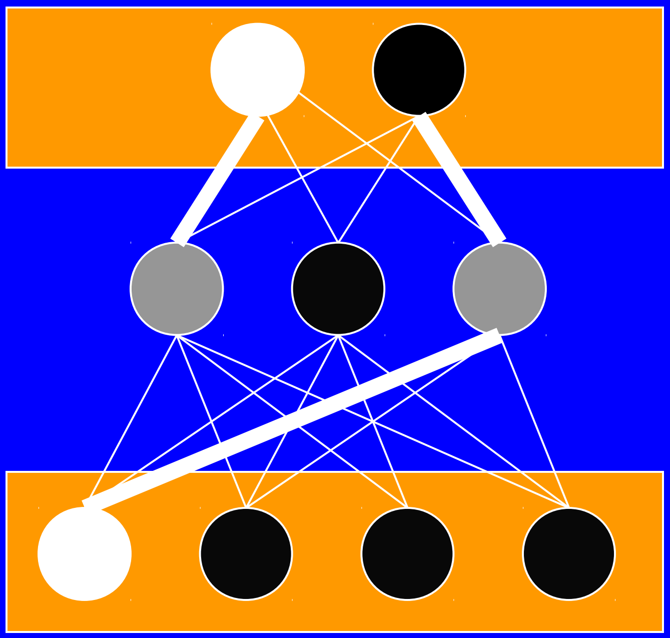
Plus Phase:
Clamp input and target output

GeneRec Learning Rule

Compare two conditions:



Minus Phase:
Clamp input

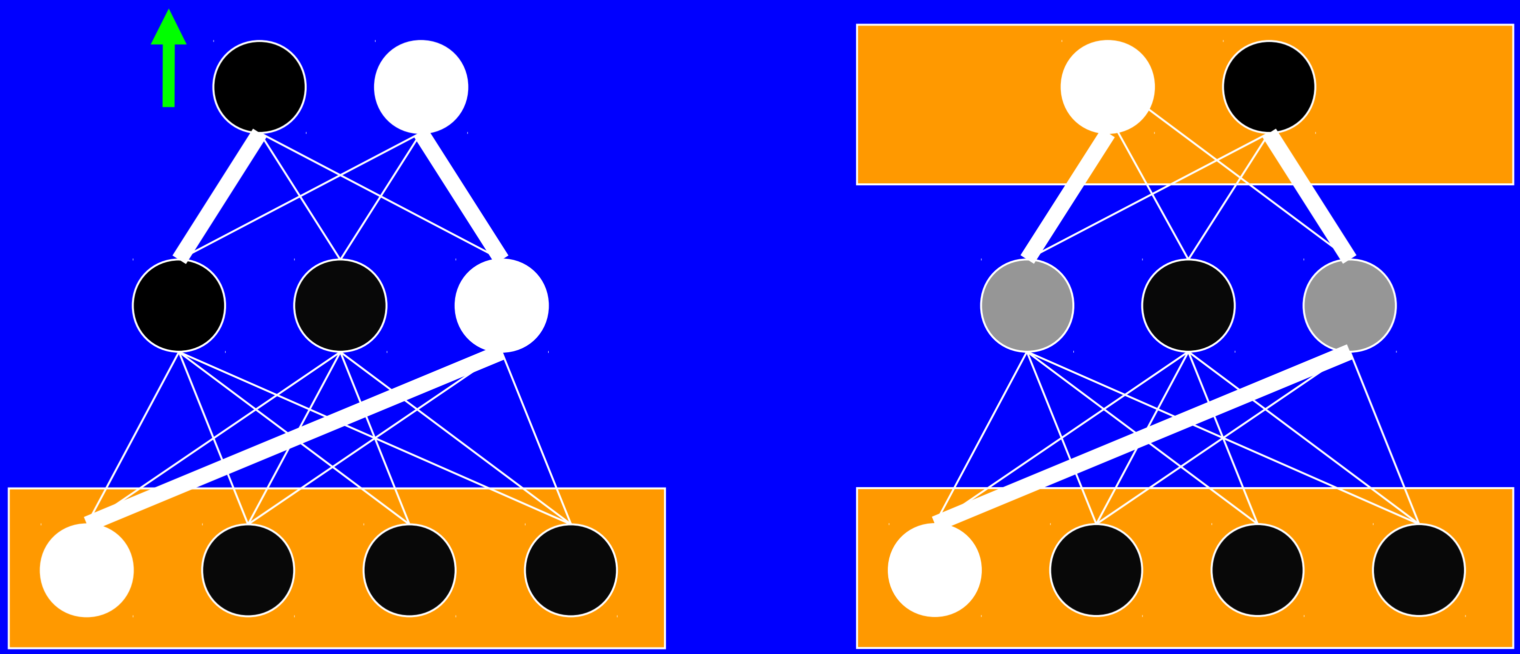


Plus Phase:
Clamp input and target output

For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec Learning Rule

Compare two conditions:



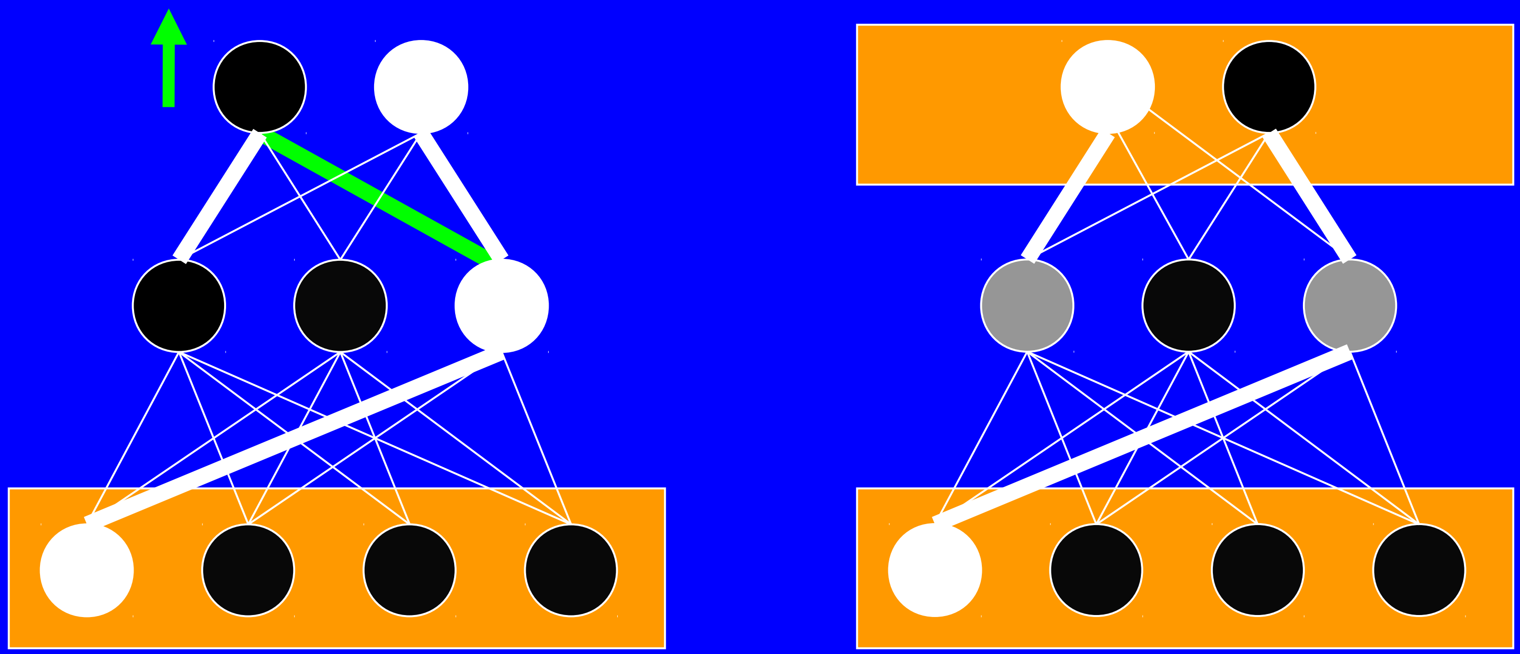
Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec Learning Rule

Compare two conditions:



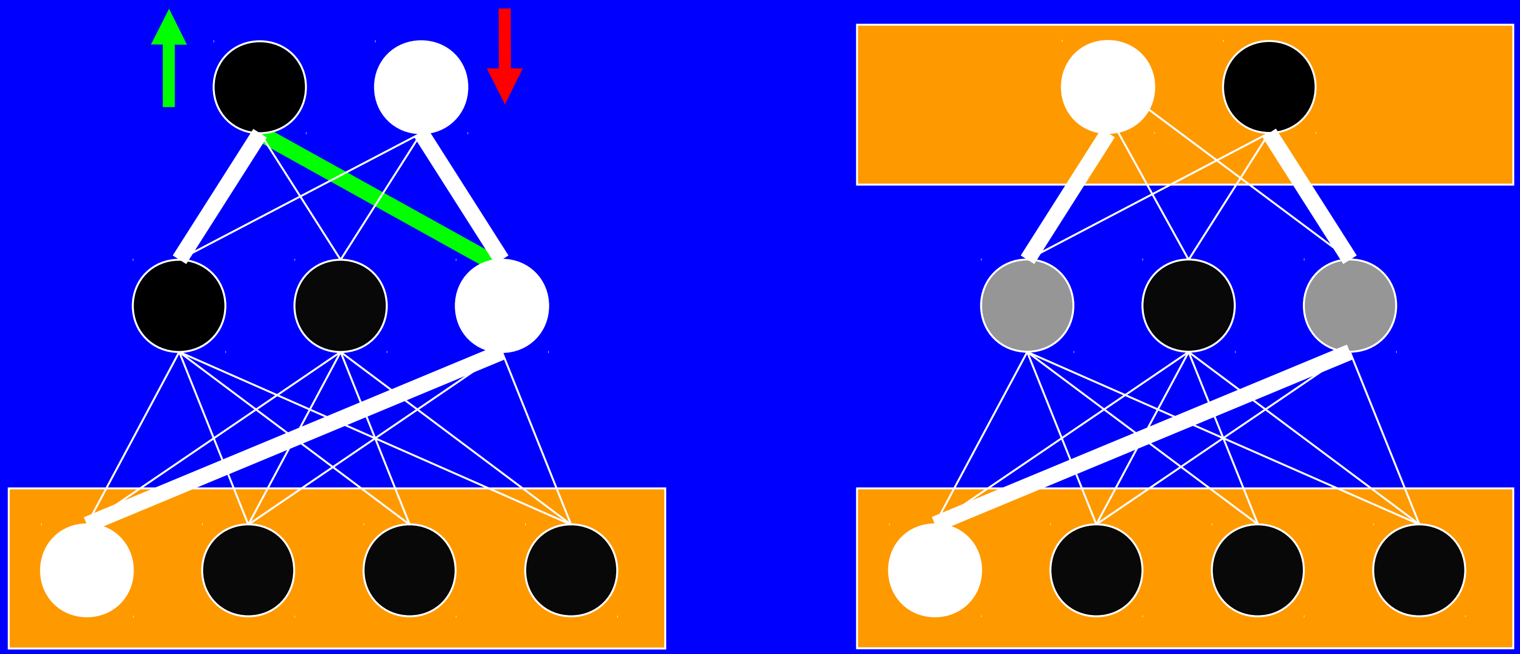
Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec Learning Rule

Compare two conditions:



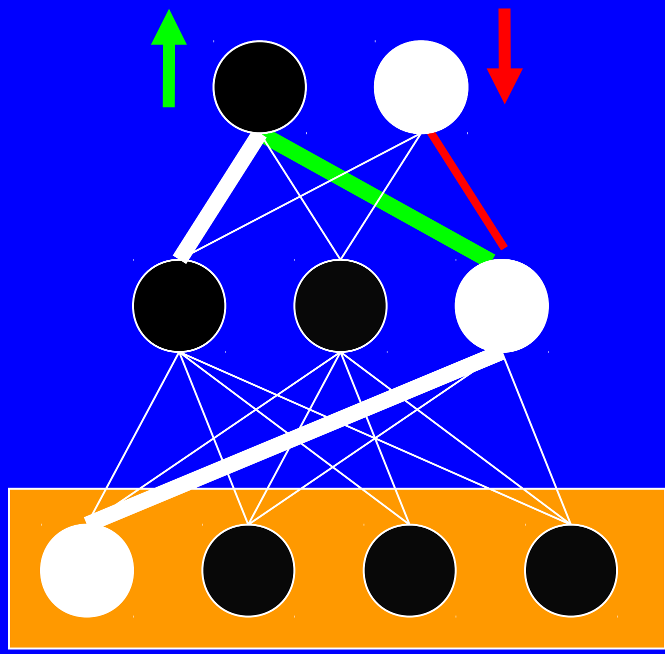
Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

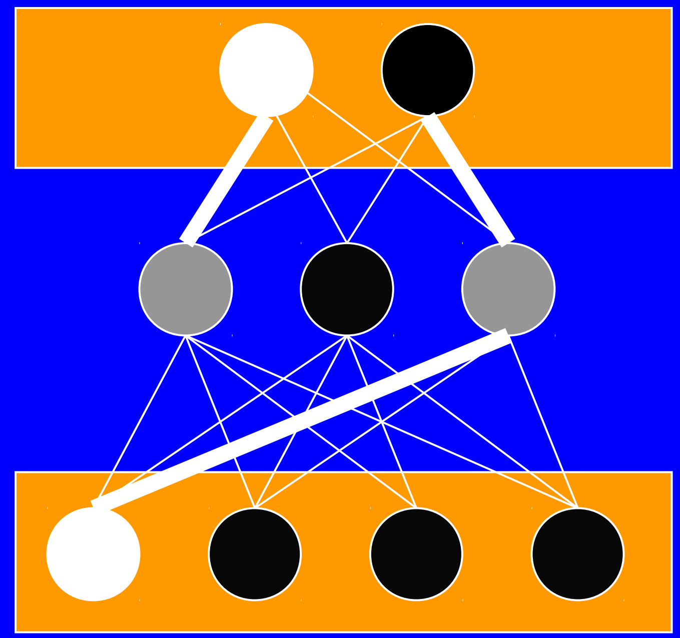
For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec Learning Rule

Compare two conditions:



Minus Phase:
Clamp input

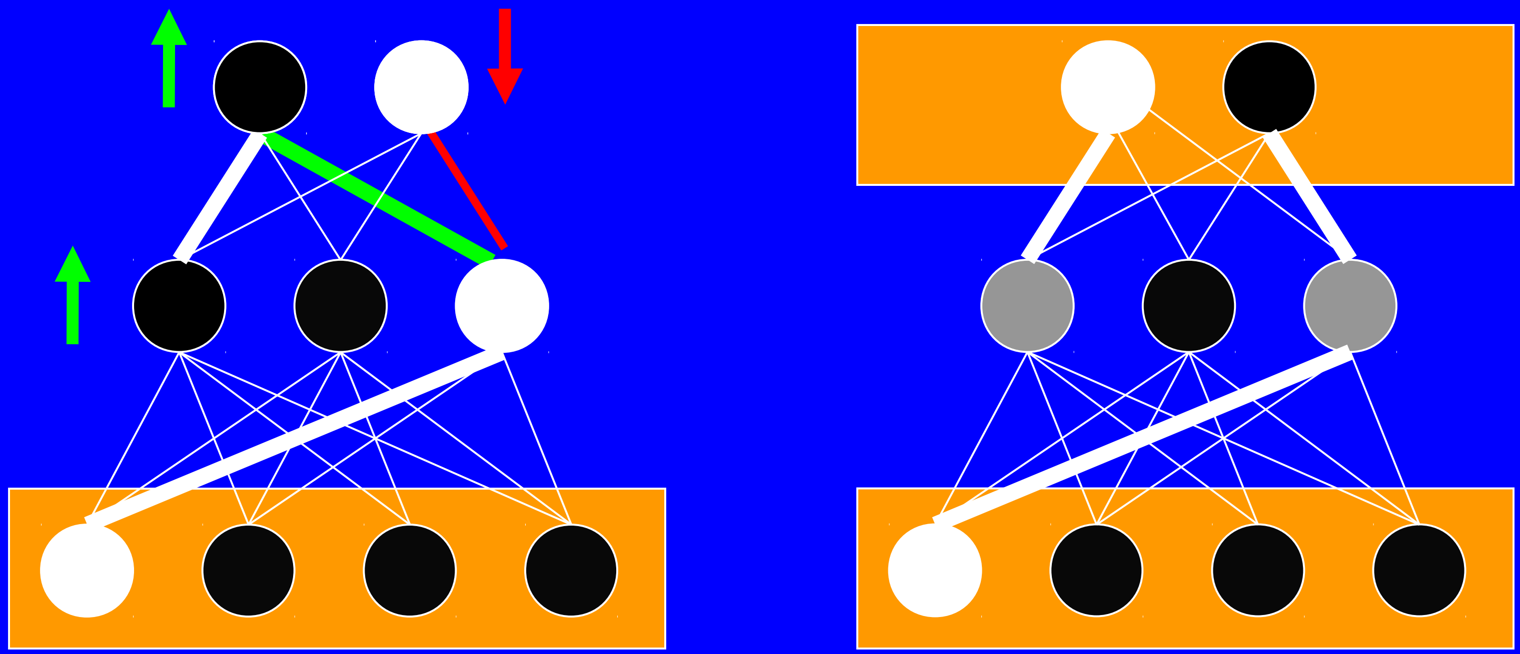


Plus Phase:
Clamp input and target output

For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec Learning Rule

Compare two conditions:



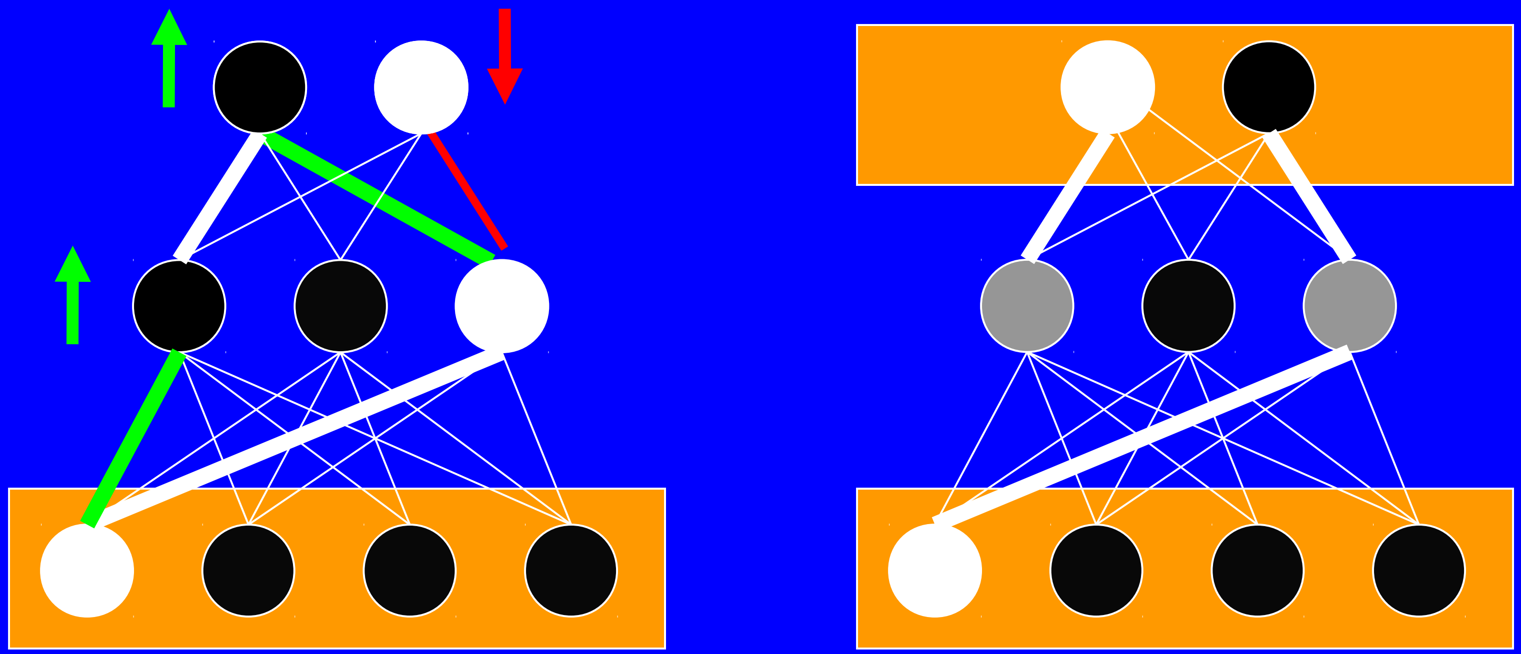
Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec Learning Rule

Compare two conditions:



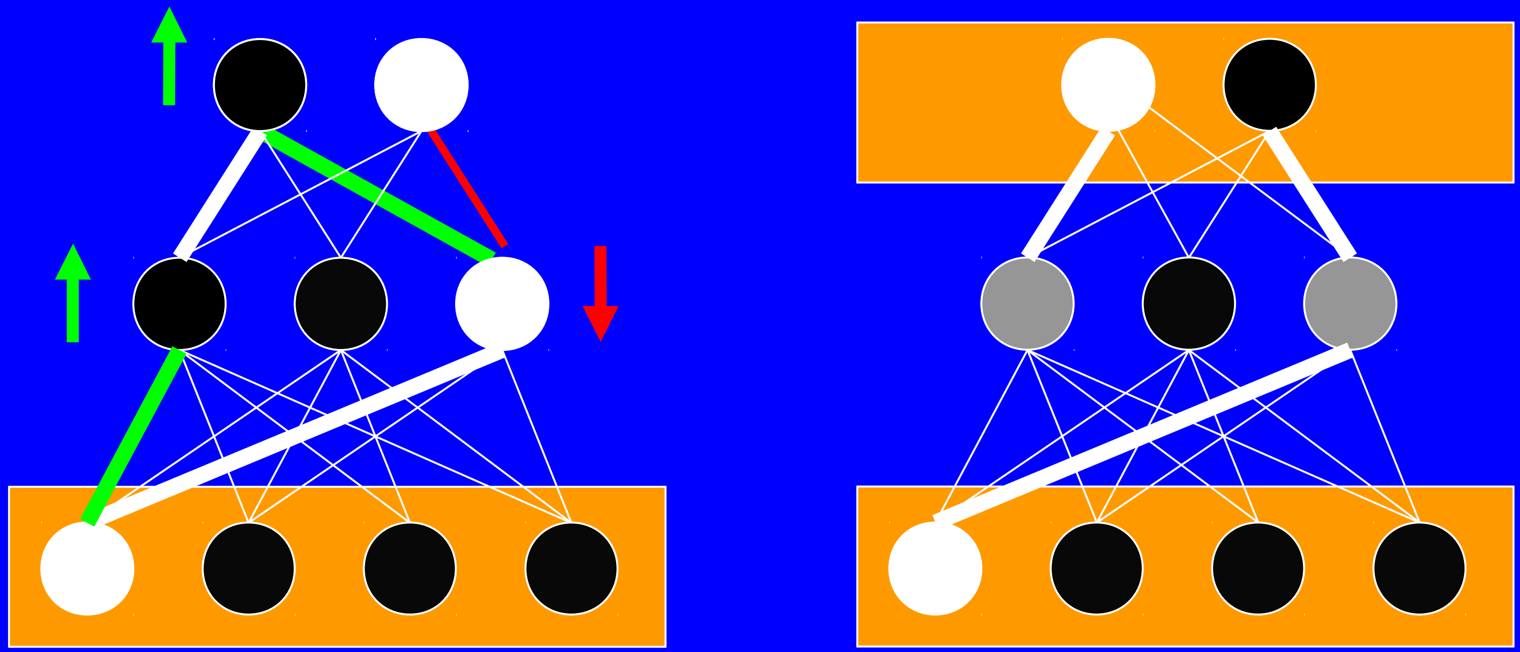
Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec Learning Rule

Compare two conditions:



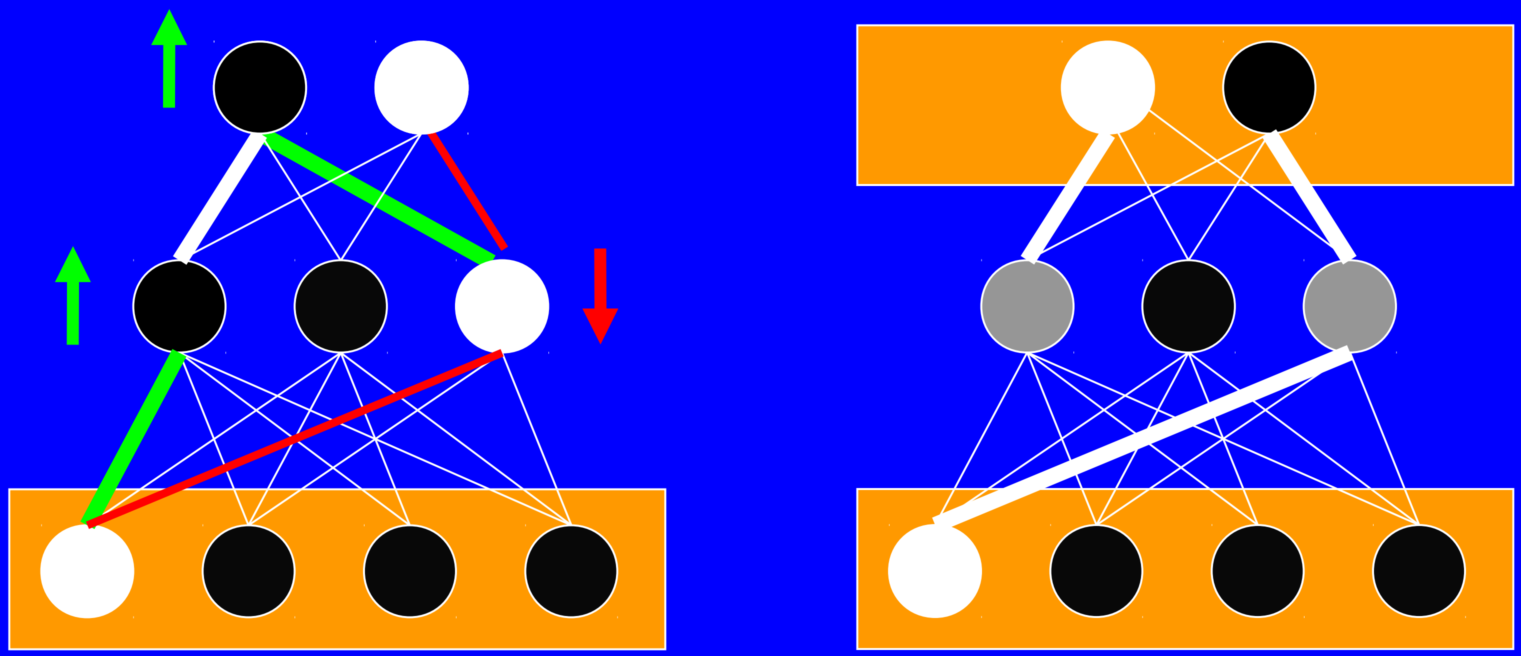
Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec Learning Rule

Compare two conditions:



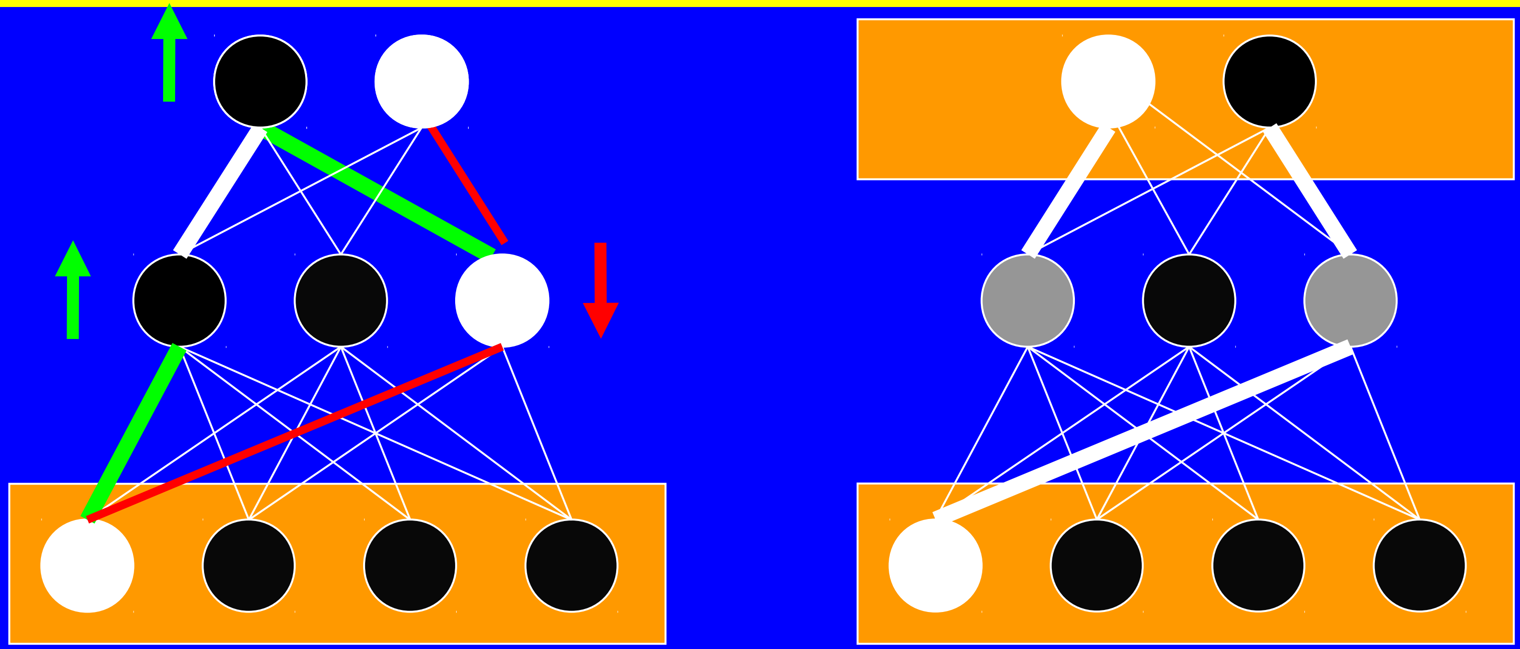
Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

For each layer, use the **difference between minus and plus activations** as an error signal and learn using the delta rule

GeneRec: Summary

The goal of error-driven learning is to construct a **path** from the input to the target output

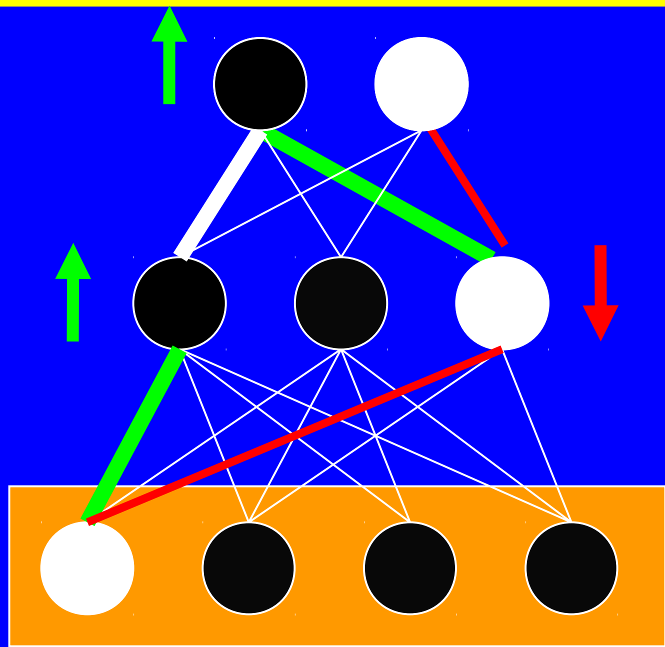


Minus Phase:
Clamp input

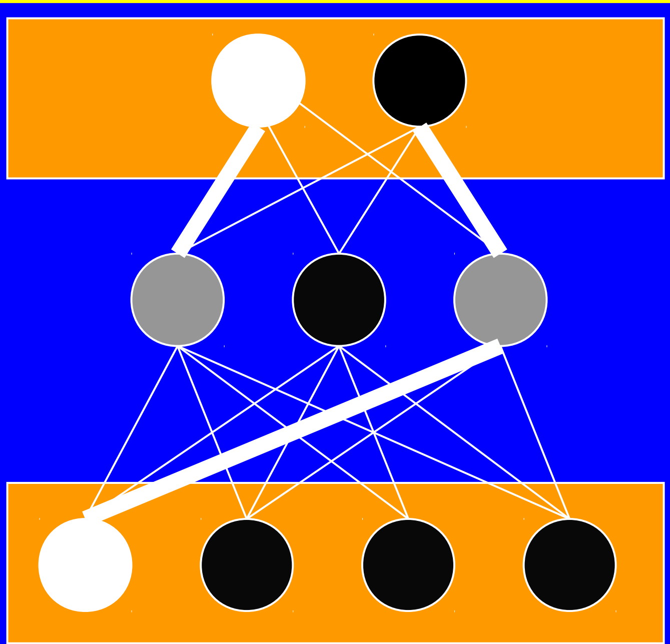
Plus Phase:
Clamp input and target output

GeneRec: Summary

The goal of error-driven learning is to construct a **path** from the input to the target output



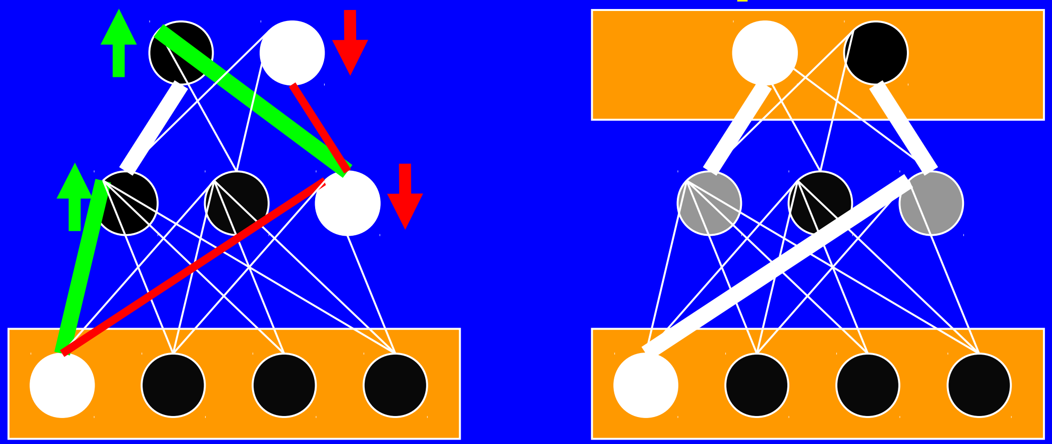
Minus Phase:



Plus Phase:

The Plus Phase helps identify **bridging units** that are well connected to both the input and the target output, and GeneRec adjusts weights to maximize the activity of these units

GeneRec: Equations

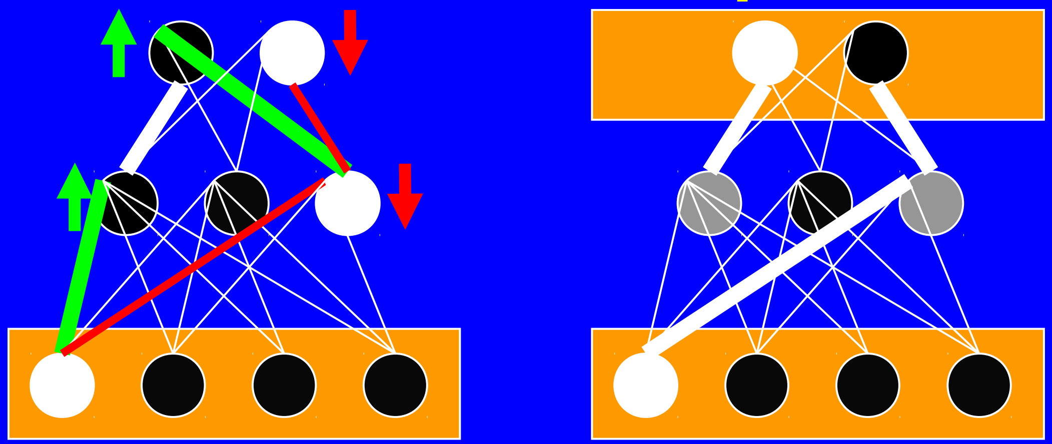


Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

Basic GeneRec: $\Delta w_{ij} = \epsilon(y_j^+ - y_j^-)x_i^-$

GeneRec: Equations



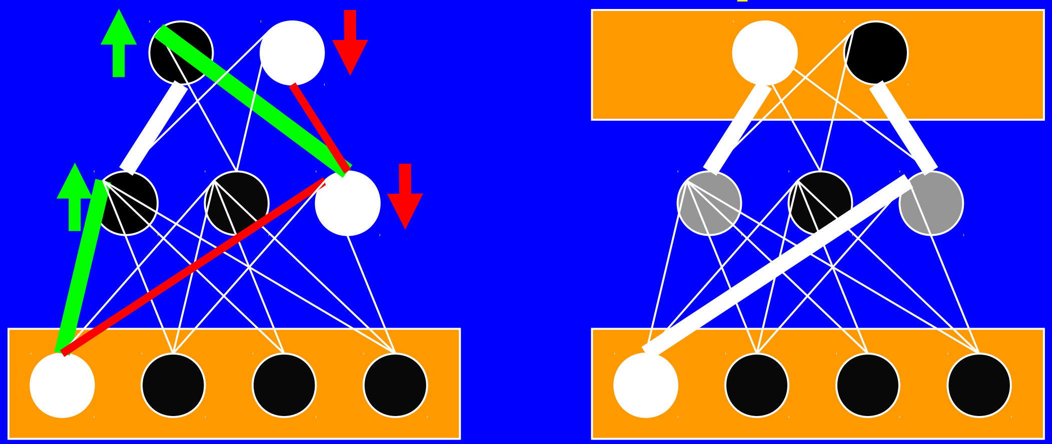
Minus Phase:
Clamp input

Plus Phase:
Clamp input and target output

Basic GeneRec: $\Delta w_{ij} = \epsilon(y_j^+ - y_j^-)x_i^-$

Two issues: Need weights to be symmetric, and why should we use minus phase sending activity instead of plus phase?

GeneRec: Equations



Minus Phase:
Clamp input

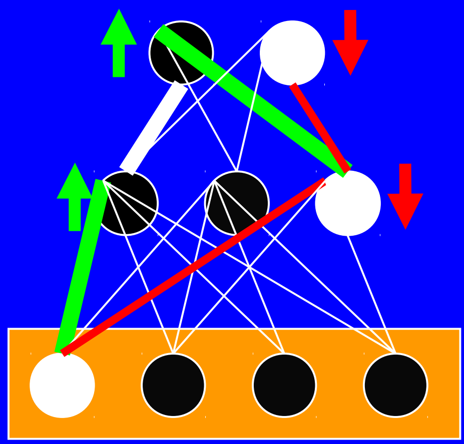
Plus Phase:
Clamp input and target output

Basic GeneRec: $\Delta w_{ij} = \epsilon(y_j^+ - y_j^-)x_i^-$

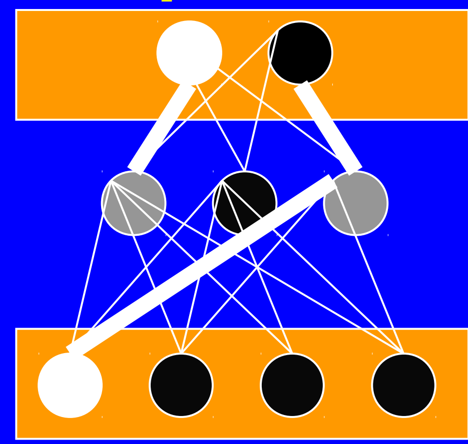
Two issues: Need weights to be symmetric, and why should we use minus phase sending activity instead of plus phase?

Solution: Average together plus and minus phase sending activation, and average together feedforward and feedback weight changes

GeneRec: Equations



Minus Phase:
Clamp input

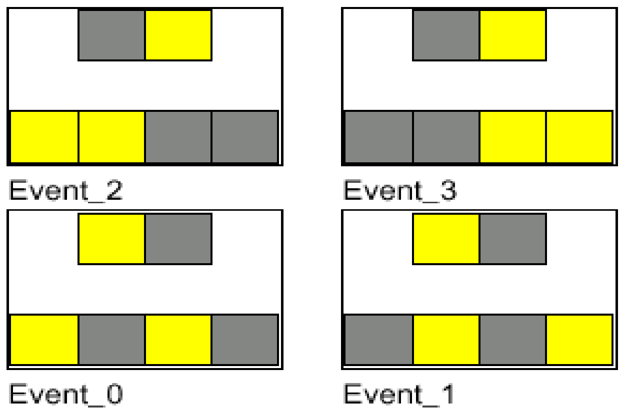


Plus Phase:
Clamp input and target output

Solution: Average together plus and minus phase sending activation, and average together feedforward and feedback weight changes

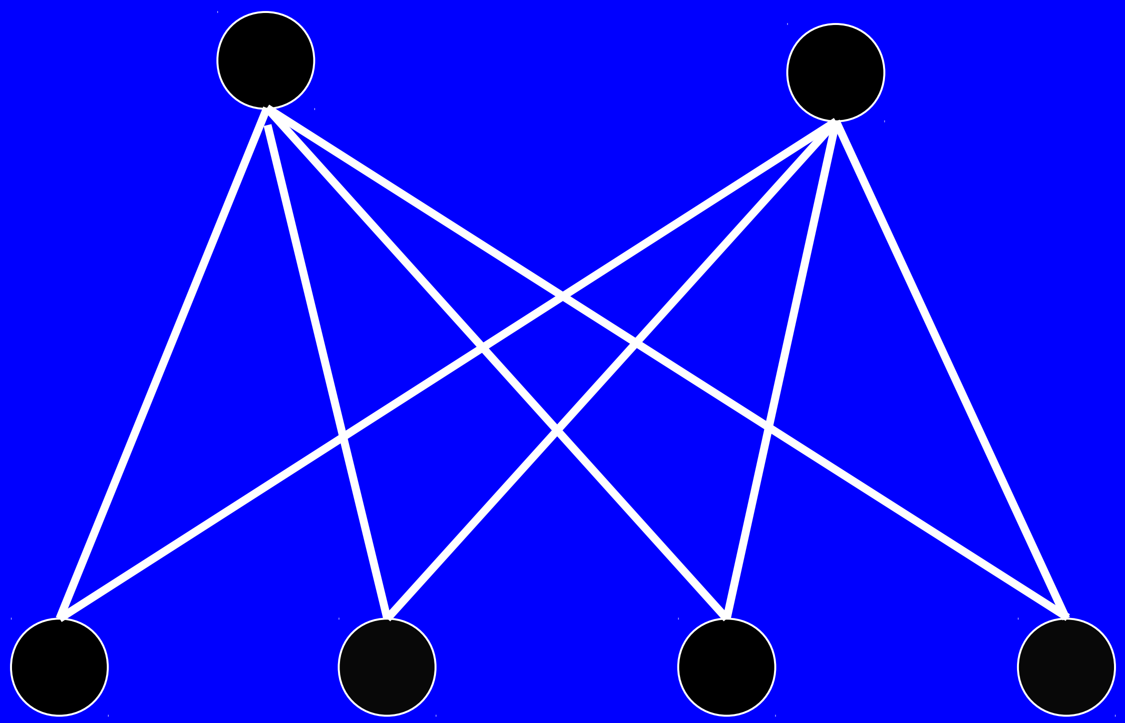
New and improved GeneRec: (CHL)

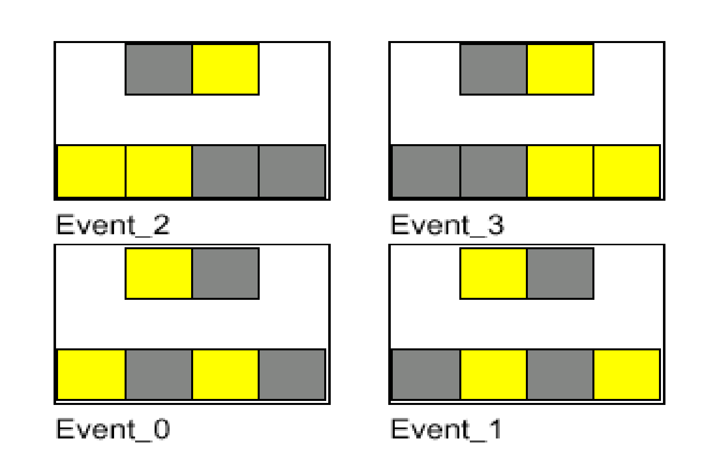
$$
\begin{aligned}
\Delta w_{ij} &= \epsilon \frac{1}{2}\left[(x_i^+ + x_i^-)(y_j^+ - y_j^-) + (y_j^+ + y_j^-)(x_i^+ - x_i^-)\right] \\
&= \epsilon \left[(x_i^+ y_j^+) - (x_i^- y_j^-)\right]
\end{aligned}
$$



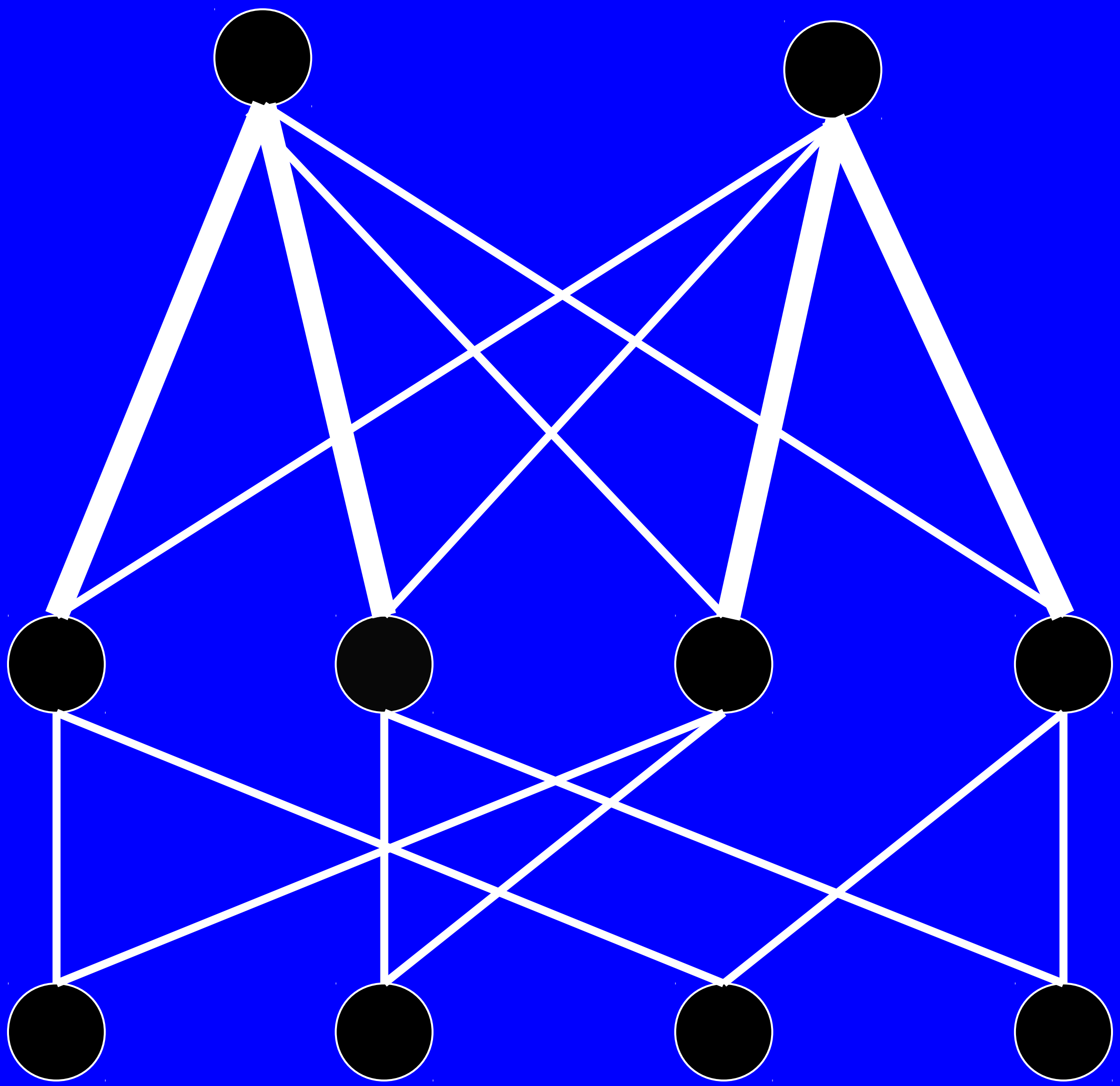
Remember the “impossible” problem?

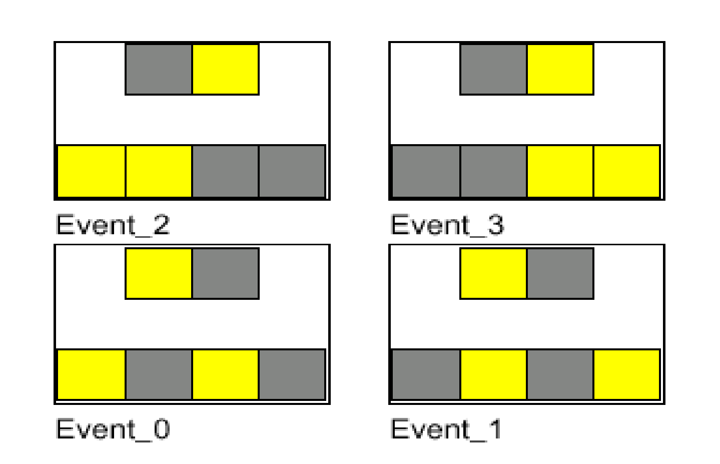
It can’t be solved by two-layer networks using the delta rule...





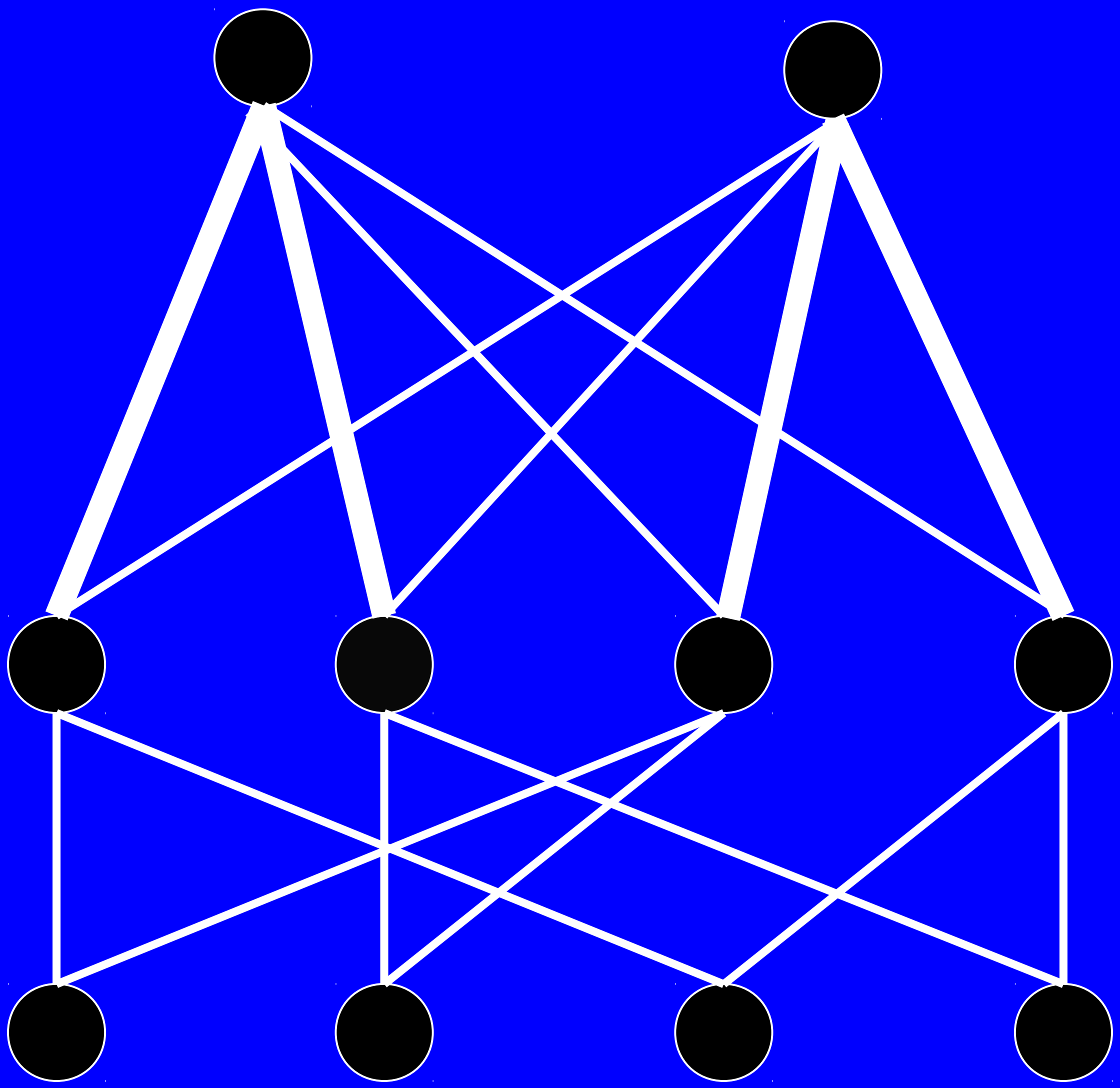
But it can be solved by three layer networks where hidden units represent feature conjunctions....





But it can be solved by three layer networks where hidden units represent feature conjunctions....

Does error-driven learning learn the correct set of weights?



Task Learning: Summary

- Hebbian learning alone is very limited in its ability to learn input-output mappings
  - If the input-output mapping happens not to coincide with the correlational structure of the inputs, Hebbian learning fails
- **Error-driven learning rules** (that leverage the difference between what the network was supposed to do, and what it actually did) do better at learning input-output mappings

Task Learning: Summary

- The **delta rule** can learn a wide variety of input-output mappings (including some that Hebb can not learn) in two-layer networks, but:
  - There are some mappings it can not learn (e.g., the “impossible” mapping)
  - It does not apply to networks with more than two layers

Task Learning: Summary

- The **GeneRec** rule remedies the deficiences of the simple delta rule
  - It applies to networks with hidden layers
  - It can solve tasks that can not be solved by the simple delta rule; this is accomplished by **re-representing** input patterns...
  - The rule is biologically plausible! Key prerequisites: Bidirectional connectivity, (approximate) symmetry, two “phases” (expectation and outcome)
- Next lecture: Synergies between Error and Hebb => Error + Hebb leads to better learning than Error alone!